

vSphere Web Services SDK Programming Guide

vSphere Web Services SDK 4.1

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see <http://www.vmware.com/support/pubs>.

EN-000304-00

vmware[®]

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

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

The VMware Web site also provides the latest product updates.

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

docfeedback@vmware.com

Copyright © 2004–2010 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at <http://www.vmware.com/go/patents>.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

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

Contents

About This Book	11
1 VMware vSphere and vSphere Management APIs	13
Virtualization and VMware vSphere Components	13
vSphere Development Tools	14
vSphere Web Services SDK	14
CIM APIs	14
vSphere SDK for Perl	14
vSphere PowerCLI	14
VIX API	14
SDK Developer Setup	15
SDK Samples	15
UML Diagrams Used in This Guide	15
2 vSphere API Programming Model	17
vSphere Client-Server Architecture	17
vSphere API as a Web Service	18
WSDL Files and the Client-Side Proxy Interface	18
Network Access to the vSphere Web Service	19
Language-Specific Classes and Methods	19
Mapping XML Data Types to Java and C# Data Types	20
Access to Managed Objects	21
Access to vSphere Server Data	21
Obtaining Information from a Server	21
Working with Data Structures	22
Accessing Property Values	22
Escape Character in Name and Path Properties	23
3 Client Applications	25
Basic Client Application Capability	25
Overview of a Java Sample Application	25
Web Server Session Token	28
Multiple Versions of the vSphere API	29
Identifying the API Version Supported by the Server	30
Helper Classes for Sample Applications	30
Java Helper Classes	31
C# Helper Classes	31
4 Datacenter Inventory	33
Inventory Overview	33
Inventory Hierarchies and ServiceInstance	33
ESX/ESXi Inventory Hierarchy	33
vCenter Server Inventory Hierarchy	34
Folders in the Hierarchy	35
Accessing Inventory Objects	35
Creating Inventory Objects	36

Privileges Required for Inventory Management	36
Privileges	36
Permissions	37
Managed and Standalone ESX/ESXi Hosts	37
5 Property Collector	39
Introduction to the PropertyCollector	39
Data Retrieval	40
Inventory Traversal and Object Selection	40
vSphere Data Objects for Property Collection	40
vSphere Methods for Property Collection	40
PropertyCollector Example (RetrievePropertiesEx)	41
Inventory Traversal	45
TraversalSpec Traversal	45
SelectionSpec Traversal	48
Simple Reference SelectionSpec	48
Recursive Traversal	48
Client Data Synchronization (WaitForUpdatesEx)	50
Property Filters	50
WaitForUpdatesEx	50
Server Data Transmission	52
PropertyCollector Performance	52
SearchIndex	52
6 Authentication and Authorization	53
Objects for Authentication and Authorization Management	53
Authentication and Authorization for ESX/ESXi and vCenter Server	54
ESX/ESXi User Model	54
vCenter Server User Model	55
vSphere Security Model	55
Privileges	55
Roles	56
Permissions	56
Setting Up Users, Groups, and Permissions	56
Obtaining User and Group Information from UserDirectory	57
Managing ESX/ESXi Users and Groups with HostLocalAccountManager	57
Managing Roles and Permissions with AuthorizationManager	58
Using Roles to Consolidate Sets of Privileges	60
Modifying Sample Roles to Create New Roles	61
Granting Privileges Through Permissions	61
Obtaining Information About Permissions	61
Setting, Changing, or Deleting Permissions	61
Impact of Group Membership on Permissions	62
Applying Permission to a Managed Entity	62
Authenticating Users Through SessionManager	63
Using the Credential Store for Automated Login	63
Credential Store Methods	64
Credential Store Backing File	64
Credential Store Samples	65
Specifying Roles and Users with the Credential Store	65
Managing Licenses with LicenseManager	66

- 7 Hosts 67
 - Host Management Objects 67
 - Retrieving Host Information 67
 - Configuring and Reconfiguring Hosts 68
 - Managing the Host Lifecycle 69
 - Reboot and Shutdown 69
 - Using Standby Mode 69
 - Disconnecting and Reconnecting Hosts 69
 - Querying and Changing the Host Time 70
 - Querying Virtual Machine Memory Overhead 70

- 8 Storage 71
 - Storage Management Objects 71
 - Introduction to Storage 72
 - How Virtual Machines Access Storage 72
 - Datstores 73
 - Choosing the Storage API to Use 74
 - Configuring Disk Partitions 75
 - Multipath Management 76
 - Configuring iSCSI Storage 76
 - Creating and Managing Datstores 78
 - Accessing Datstores 79
 - Creating and Modifying a VMFS Datastore 79
 - Removing and Updating Datstores 81
 - Managing VMFS Datstores with HostStorageSystem 81
 - Managing VMFS Volume Copies (Resignaturing) 81
 - Managing Diagnostic Partitions 82
 - Retrieving Diagnostic Partition Information 82
 - Creating a Diagnostic Partition 83
 - Sample Code Reference 83

- 9 Networking 85
 - Network Management Objects and Methods 85
 - Introduction to Networking 86
 - vNetwork Standard Switch Environment 86
 - Virtual Switches 87
 - Port Group 87
 - Virtual Machine Network Interface 87
 - VMkernel Network Interfaces 87
 - Physical Network Adapter (pnic) 88
 - Setting Up Networking with vSS 88
 - Retrieving Information About the Network Configuration 88
 - Adding a Virtual Switch 88
 - Adding a Virtual Port Group 89
 - Adding a VMkernel Network Interface 89
 - Defining the Host Network Policies 90
 - NIC Teaming 90
 - Setting Up IPv6 Networking 91
 - Adding Networking Services 92
 - Adding an NTP Service 92
 - Setting Up the IP Route Configuration 92
 - Setting Up SNMP 92
 - Sample Code Reference 93

10 Virtual Machine Configuration 95

- VirtualMachine Management Objects and Methods 95
- Creating Virtual Machines and Virtual Machine Templates 96
 - Creating a Virtual Machine Using VirtualMachineConfigSpec 96
 - Calling the CreateVM_Task Method 96
 - Specifying Virtual Machine Attributes with VirtualMachineConfigSpec 97
 - Creating Virtual Machine Templates 97
 - Cloning a Virtual Machine 97
 - Converting a Template to a Virtual Machine 98
 - Accessing Information About Virtual Machine 98
- Configuring a Virtual Machine 99
 - Name and Location 99
 - Hardware Version 99
 - Boot Options 100
 - Operating System 100
 - CPU and Memory Information 100
 - CPU and Memory Resource Allocation 100
 - CPU and Memory Modification for Running Virtual Machines 101
 - Number of CPUs 101
 - CPU Processors and Memory Affinity 101
 - CPU Features 101
 - Networks 101
 - Virtual Network Interfaces 101
 - Virtual Machine MAC Address 101
 - Fibre Channel NPIV Settings 102
 - File Locations 102
- Adding Devices to Virtual Machines 102
- Performing Virtual Machine Power Operations 104
- Registering and Unregistering Virtual Machines 104
- Customizing the Guest Operating System 105
- Installing VMware Tools 105
- Upgrading a Virtual Machine 106

11 Virtual Machine Management 107

- Virtual Machine Migration 107
 - Cold Migration 107
 - Migration with VMotion 108
 - Using Storage VMotion 108
- Snapshots 108
 - Creating a Snapshot 109
 - Reverting to a Snapshot 109
 - Deleting a Snapshot 110
- Linked Virtual Machines 110
 - Linked Virtual Machines and Disk Backings 110
 - Creating a Linked Virtual Machine 110
 - Creating a Linked Virtual Machine From a Snapshot 110
 - Creating a Linked Virtual Machine From the Current Running Point 111
 - Removing Snapshots and Deleting Linked Virtual Machine 112
 - Relocating a Virtual Machine in a Linked Virtual Machine Group 112
 - Promoting a Virtual Machine's Disk 112
 - Performing Advanced Manipulation of Delta Disks 113

12	Virtual Applications	115
	About Virtual Applications	115
	Management Overview	115
	Direct and Linked Children	116
	OVF Packages	116
	Creating a VirtualApp	117
	Managing VirtualApp Children	117
	Exporting a Virtual Application	118
	Acquiring the Export Lease	118
	Downloading the Disks	118
	Generating the OVF Descriptor	118
	Completing the Lease	118
	Importing an OVF Package	119
	Virtual Application Life Cycle	119
	Powering a Virtual Application On or Off	119
	Unregistering a Virtual Application	120
	Suspending a Virtual Application	120
	Destroying a Virtual Application	120
13	Resource Management	121
	Resource Management Objects	121
	Introduction to Resource Management	122
	Resource Allocation	122
	Resource Pool Hierarchies	122
	Resource Pool Management Guidelines	123
	Cluster Overview	123
	Creating and Configuring Resource Pools	124
	Understanding Expandable Reservation	124
	Expandable Reservation Example 1	124
	Expandable Reservation Example 2	124
	Deleting Child Resource Pools	125
	Moving Resource Pools or Virtual Machines Into a Resource Pool	126
	Introduction to VMware DRS and VMware HA Clusters	126
	VMware DRS	126
	VMware HA	126
	Creating and Configuring Clusters	127
	Creating a Cluster	127
	Adding a Host to a Cluster	127
	Reconfiguring a Cluster	128
	Managing DRS Clusters	128
	Managing HA Clusters	129
	Primary and Secondary Hosts	129
	Failure Detection and Host Network Isolation	129
	Using VMware HA and DRS Together	129
14	Tasks and Scheduled Tasks	131
	Task Management Objects	131
	Accessing and Manipulating Tasks	132
	Examining Recent Tasks with TaskManager	133
	Using TaskInfo to Monitor Task Information	133
	Monitoring TaskInfo Properties	134
	Cancelling a Task	135
	Understanding ScheduledTaskManager	135
	Scheduling Tasks	136

Defining the Schedule and Action	137
Scheduling Recurring Operations	137
Cancelling a Scheduled Task	139
Using a TaskHistoryCollector	139
Creating a TaskHistoryCollector Filter	140
Managing the HistoryCollector	140
Sample Code Reference	140
15 Events and Alarms	141
Event and Alarm Management Objects	141
Understanding Events	141
Managing Events with EventManager	142
Event Data Objects	143
Formatting Event Message Content	143
Creating Custom Events	144
Using an EventHistoryCollector	144
Creating an EventHistoryCollector Filter	144
Managing the HistoryCollector	145
Using Alarms	145
Obtaining a List of Alarms	145
Creating an Alarm	145
Defining Alarms Using the AlarmSpec Data Object	146
Specifying Alarm Trigger Conditions with AlarmExpression	147
AlarmExpression Types	147
Using MetricAlarmExpression	147
Specifying Alarm Actions	148
Deleting or Disabling an Alarm	149
Sample Code Reference	149
16 Performance	151
PerformanceManager Objects and Methods	151
Performance Counters	152
Sampling Periods and Intervals	153
Obtaining Statistics	154
vCenter Server and PerformanceManager	155
Modifying Historical Intervals	156
Optimizing Query Response Time	156
Sample Code Reference	157
A Diagnostics and Troubleshooting	159
Troubleshooting Best Practices	159
Overview of Configuration Files and Log Files	160
ESX/ESXi Log File	160
Virtual Machine Log Files	161
vCenter Server Log Files	162
Modifying the Log Level to Obtain Detailed Information	162
Setting the Log Level on ESX/ESXi Systems	162
Generating Logs	163
Setting the Log Level on vCenter Server Systems	163
Using DiagnosticManager	163
Using the MOB to Explore the DiagnosticManager	165
Generating Diagnostic Bundles	166

B	Managed Object Browser	167
	Using the MOB to Explore the Object Model	167
	Accessing the MOB	167
	Using the MOB to Navigate the VMware Infrastructure Object Model	168
	Using the MOB to Invoke Methods	168
	Passing Primitive Datatypes to Method	168
	Passing Arrays of Primitives to Methods	169
	Passing Complex Structures to Methods	169
	Simple Content	169
	Complex Content	170
C	HTTP Access to vSphere Server Files	173
	Introduction to HTTP Access	173
	URL Syntax for HTTP Access	174
	Datastore Access (/folder)	174
	Host File Access (/host)	175
	Update Package Access (/tmp)	176
	Privilege Requirements for HTTP Access	176
D	Privileges Reference	177
	Privileges Required to Invoke Operations	177
	Privileges Required to Read Properties	185
	Privileges Defined for the Administrator Role	186
E	Sample Program Overview	191
	Java Sample Programs	191
	C# DotNet Sample Programs	194
	Index	197

About This Book

The *vSphere Web Services SDK Programming Guide* explains client application development using the VMware® vSphere Web Services SDK 4.1.

VMware provides different APIs and SDKs for different applications and goals. The vSphere Web Services SDK targets developers who create client applications for managing VMware® vSphere components available on VMware ESX/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 guide is revised with each release of the product or when necessary. A revised version can contain minor or major changes. [Table 1](#) summarizes the significant changes in each version of this guide.

Table 1. Revision History

Revision	Description
13JUL2010	Restructured manual and added chapters about host, storage, and networking. Revised property collector chapter and added appendix about HTTP access.
07MAY2009	First version of the 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 or C# (in the Microsoft .NET environment) targeting VMware vSphere. An understanding of Web Services technology and some programming background in one of the stub languages (C# or Java) is required.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation, go to <http://www.vmware.com/support/pubs>.

Document Feedback

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

Technical Support and Education Resources

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

Online and Telephone Support

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

Support Offerings

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

VMware Professional Services

VMware Education Services courses offer extensive hands-on labs, case study examples, and course materials designed to be used as on-the-job reference tools. Courses are available onsite, in the classroom, and live online. For onsite pilot programs and implementation best practices, VMware Consulting Services provides offerings to help you assess, plan, build, and manage your virtual environment. To access information about education classes, certification programs, and consulting services, go to <http://www.vmware.com/services>.

VMware vSphere and vSphere Management APIs

1

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

The chapter includes the following topics:

- [“Virtualization and VMware vSphere Components”](#) on page 13
- [“vSphere Development Tools”](#) on page 14
- [“SDK Developer Setup”](#) on page 15
- [“SDK Samples”](#) on page 15
- [“UML Diagrams Used in This Guide”](#) on page 15

Virtualization and VMware vSphere Components

VMware software products virtualize computing resources, including CPU, memory, storage, and networks. Virtualization provides an abstraction layer between computing resources, physical storage, and networking hardware, and the applications that use the resources.

VMware vSphere includes ESX/ESXi, vCenter Server, and several additional server products. The base products support running and managing virtual machines. With additional licenses, you can take advantage of the vSphere distributed resource management (DRS), disaster recovery, and high availability (HA) features.

- The ESX/ESXi hypervisor is capable of supporting multiple virtual machines and other virtual components, such as storage and networks.
- vCenter Server is a database-backed Windows service that provides central management for multiple ESX/ESXi systems.
- vSphere Client is a GUI to manage vSphere. vSphere Client runs on Windows systems and uses the same interface as the vSphere Web Services SDK to communicate with vSphere servers. You can perform any task the vSphere Client supports using the Web Services SDK. In fact, the SDK supports some tasks you cannot perform with the vSphere Client.

For more information about ESX/ESXi and vCenter Server, see the VMware vSphere documentation page on the VMware Web site. If you are new to VMware vSphere or new to the vSphere Web Services SDK, see one of these vSphere administrator documents for background information about vSphere:

- *Introduction to VMware vSphere*
- *Virtual Machine Management Guide*
- *Server Management Guide*
- *ESX Configuration Guide* and *ESXi Configuration Guide*
- *vSphere Resource Management Guide*

vSphere Development Tools

VMware supports SDKs and scripting tools for managing vSphere.

vSphere Web Services SDK

The vSphere Web Services SDK is the most comprehensive of the available management APIs. The SDK works against both ESX/ESXi and vCenter Server systems. As a Web Services SDK, the SDK is language neutral. The SDK includes stubs and examples for Java and C# and a comprehensive documentation set including an *API Reference* generated from the source.

CIM APIs

The VMware CIM APIs provide a CIM (Common Information Model) interface for developers building management applications. With the VMware CIM APIs, developers can use standards-based CIM-compliant applications to manage ESX/ESXi hosts.

The CIM APIs include:

- CIM SMASH/Server Management API – profiles compatible with the DMTF System Management Architecture for Server Hardware (SMASH) initiative. SMASH profiles allow CIM clients to monitor system health of a managed server.
- CIM Storage Management API – profiles compatible with the Storage Management Initiative Specification (SMI-S) of the Storage Network Industry Association. SMI-S profiles allow CIM clients to explore the virtual machines on an ESX/ESXi host, and associated storage resources.

vSphere SDK for Perl

The vSphere SDK for Perl is an easy-to-use Perl scripting interface to the vSphere API. Administrators and developers can work with vSphere API objects using vSphere SDK for Perl subroutines. Administrators can use the utility applications included with vSphere SDK for Perl.

The vSphere SDK for Perl also includes the Web Services for Management component for writing scripts that retrieve CIM data from the ESX/ESXi host using CIMOM, a service that provides standard CIM management functions. The vSphere SDK for Perl also includes subroutines for managing the VMware Credential Store and an example application that illustrates credential store use.

vSphere SDK for Perl is bundled with the vSphere Command-Line Interface (vSphere CLI). The vSphere CLI command set allows you to run common system administration commands against ESX/ESXi systems from an administration server of your choice.

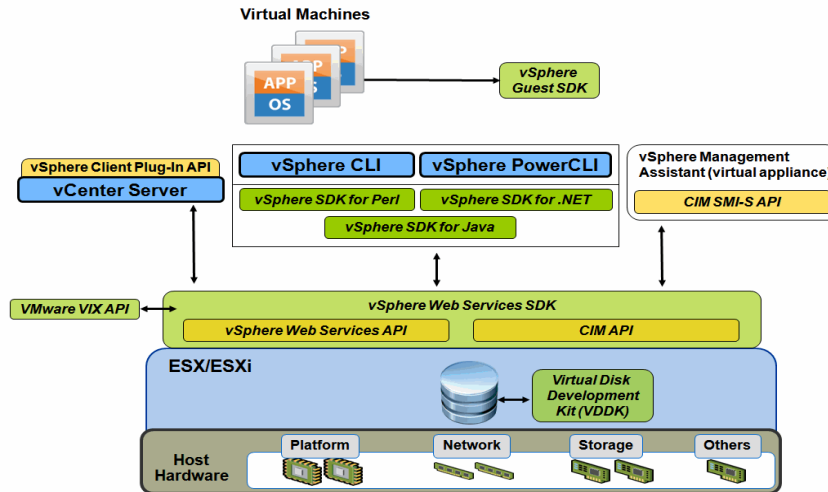
vSphere PowerCLI

VMware vSphere PowerCLI provides a Windows PowerShell interface to the vSphere API. vSphere PowerCLI includes PowerShell Cmdlets for administering vSphere components. In addition, the vSphere PowerCLI package includes the vSphere SDK for .NET for developers who want to create their own applications.

VIX API

The VIX API is a library for writing scripts and programs to manipulate virtual machines. It is high-level, easy to use, and practical for both script developers and application programmers. This API is well suited for dedicated IT personnel in an organization building their own in-house tools. It might also be used by software vendors using VIX to integrate VMware products with their own products or to build management products for virtual machines.

[Figure 1-1](#) gives an overview of the different vSphere APIs and CLIs and illustrates how they fit into the virtual infrastructure.

Figure 1-1. vSphere APIs and CLIs

SDK Developer Setup

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

SDK Samples

The SDK includes a set of samples that illustrate much of the SDK features. Two sample sets are available:

- Java samples use the generated Java stubs that are shipped with the SDK.
- C# samples use the generated C# stubs that are shipped with the SDK.

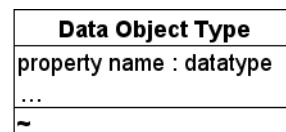
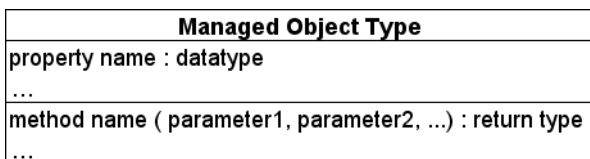
Both sample sets include a set of utility applications that are used by the sample code.

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 [Appendix E, “Sample Program Overview,”](#) on page 191 for lists of samples for the two languages and a brief explanation of what each sample does.

UML Diagrams Used in This 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](#) 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

vSphere API Programming Model

The vSphere 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 ESX, ESXi, and vCenter Server systems.

This chapter includes the following topics:

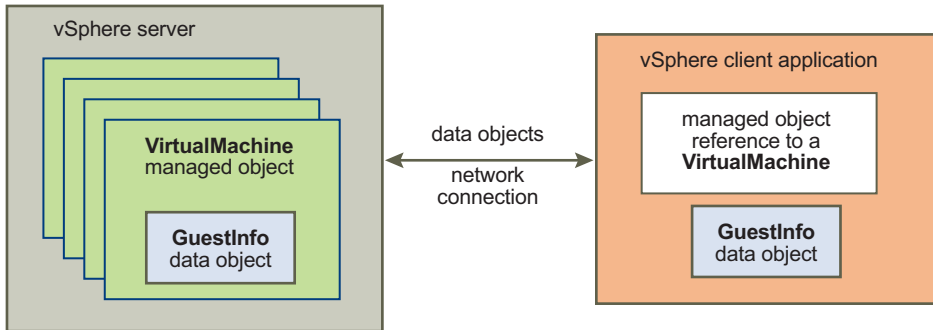
- [“vSphere Client-Server Architecture”](#) on page 17
- [“vSphere API as a Web Service”](#) on page 18
- [“Access to Managed Objects”](#) on page 21
- [“Access to vSphere Server Data”](#) on page 21

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`.

[Figure 2-1](#) shows a vSphere server and client application. The client has a managed object reference to a virtual machine on the server, and a copy of the `GuestInfo` data object for the virtual machine. A client must maintain its copy of a data object because, depending on the type of client request, a vSphere server might send property data for a data object as a set of name-value pairs associated with a managed object reference. See the description of the `ObjectContent` data object in the *vSphere API Reference*.

Figure 2-1. vSphere Server and Client

The VMware vSphere application model uses an asynchronous client-server communication model in most cases. Methods are nonblocking and return a reference to a Task managed object. See [Chapter 14, “Tasks and Scheduled Tasks,”](#) on page 131.

vSphere API as a Web Service

The vSphere API is a language-neutral Web service that runs on ESX/ESXi and vCenter Server systems. The vSphere API complies with the Web Services Interoperability Organization (WS-I) Basic Profile 1.0. The WS-I Basic Profile 1.0 includes support for:

- XML Schema 1.0
- SOAP 1.1
- WSDL 1.1

For information about the WS-I Basic Profile 1.0, go to the Web Services Interoperability Organization (WS-I) Web site at <http://www.ws-i.org>.

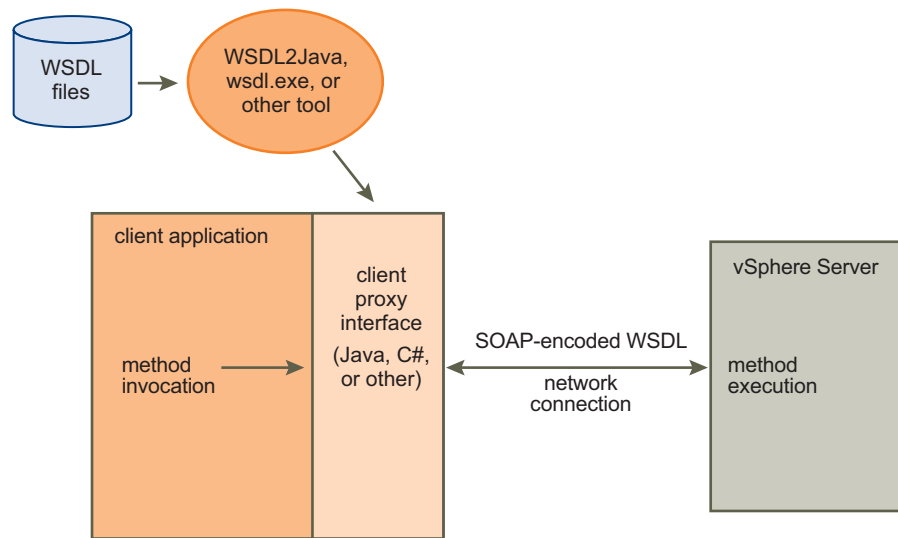
Web services technology supports operations, which are the same as methods in other programming languages. The vSphere API Web service provides access to all operations necessary for monitoring and managing vSphere components, such as compute resources, virtual machines, networks, storage, and so on.

WSDL Files and the Client-Side Proxy Interface

The vSphere Web Services SDK provides a set of WSDL (Web Services Description Language) files that define the vSphere Web Services API. Web-services development tools such as Axis WSDL2Java, or Microsoft .NET `wsdl.exe` use these WSDL files to generate client-side proxy code (stubs).

The client proxy provides a language-specific vSphere API, for example, using Java or C#. 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](#) 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 API client libraries in to your client application using the following Java and C# statements.

```
C#      using VimApi;
Java    import com.vmware.vim25.*;
```

IMPORTANT The vSphere Web Services SDK includes Java client-side proxy code that was generated using the Axis toolkit. If the versions of Java and Axis 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 configuring a development environment for the vSphere Web Services SDK.

Network Access to the vSphere Web Service

Your client application can use the vSphere 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. See the *Developer's Setup Guide* for details.

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 (`VimServiceLocator`, `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.

[Table 2-1](#) identifies client proxy definitions for the vSphere Web Services SDK WSDL.

Table 2-1. Client Proxy Definitions

Element Access	Java	C#
Access to vSphere Web service (HTTPS/HTTP)	VimServiceLocator class	AppUtil class
Access to vSphere API methods	VimPortType class	VimService class
Access to vSphere API properties	<i>getPropertyName</i> and <i>setPropertyName</i> methods defined for data objects	get and set methods defined for properties
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 declarations for the `AfterStartupTaskScheduler.minute` property.

Java

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

C#

```
public int minute {
    set; get; }
```

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 `setDisplayname` method to assign a string value to the property, and obtain the string value by using the `getDisplayname` method. The vSphere Web Services SDK includes Java and C# sample code that illustrates using the proxy interfaces. See [Chapter 3, “Client Applications”](#) on page 25.

Mapping XML Data Types to Java and C# 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.

[Table 2-2](#) lists some of the more common XML primitive data type mappings.

Table 2-2. Standard XML Schema Primitives to Java and .NET Data Type Mappings

XML Schema	Java	.NET Data Type
<code>xsd:base64binary</code>	<code>byte[]</code>	<code>Byte[]</code>
<code>xsd:boolean</code>	<code>boolean</code>	<code>Boolean</code>
<code>xsd:byte</code>	<code>byte</code>	<code>SByte</code>
<code>xsd:dateTime</code>	<code>java.util.Calendar</code>	<code>DateTime</code>
<code>xsd:decimal</code>	<code>java.math.BigDecimal</code>	<code>Decimal</code>
<code>xsd:double</code>	<code>double</code>	<code>Double</code>
<code>xsd:float</code>	<code>float</code>	<code>Single</code>
<code>xsd:int</code>	<code>int</code>	<code>Int32</code>
<code>xsd:string</code>	<code>java.lang.String</code>	<code>String</code>

Access to Managed Objects

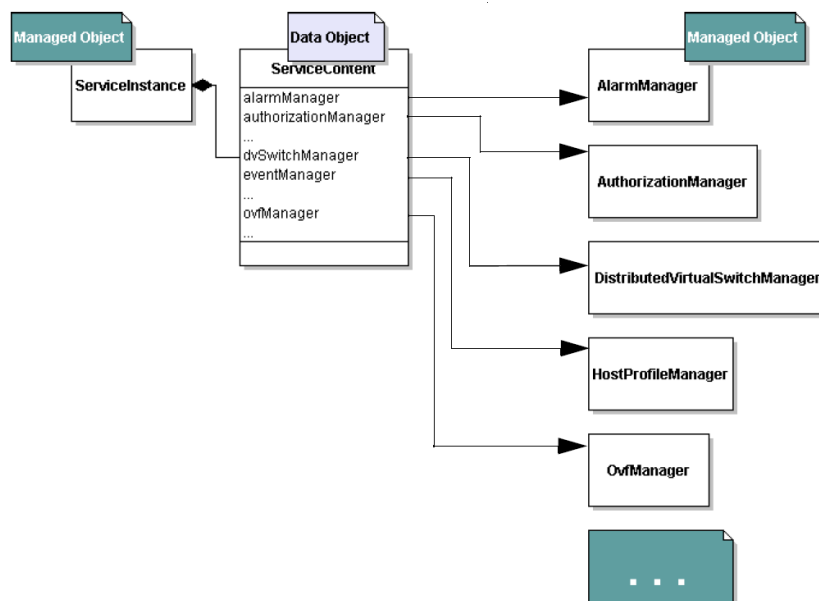
Your client application obtains access to managed objects through the `ServiceInstance` managed object and its associated `ServiceContent` data object. The `ServiceContent` data object contains managed object references to services and manager entities, and to the root folder of the inventory.

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](#)” on page 25 for an example of connecting to a server and using the `ServiceInstance` reference to retrieve the `ServiceContent` object.

[Figure 2-3](#) 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](#)” on page 19 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 5](#) 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.

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) audiotape. 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

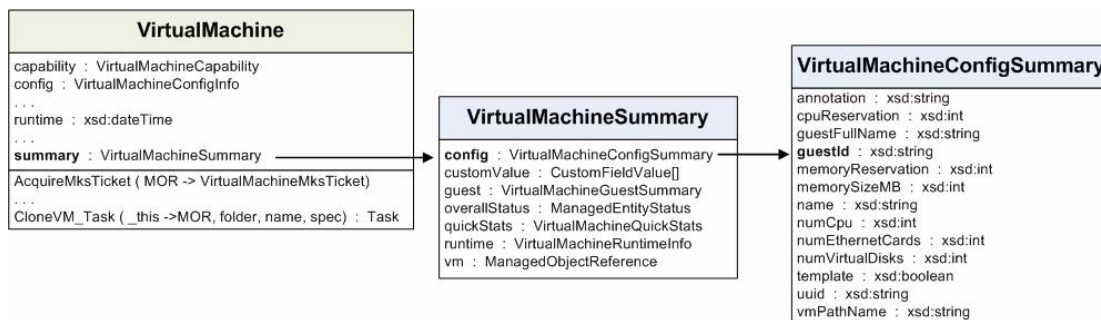
Working with composite data structures requires an understanding of nested properties and how to access the data they contain.

Nested Properties and Property Paths

The VMware vSphere object model consists of composite data structures. Object types can include properties defined as primitive data types, such as an `xsd:int` or `xsd:string`, or as complex data types, such as data objects. Properties can nest to several levels.

For example, [Figure 2-4](#) 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` 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).

[Table 2-3](#) 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](#).

Table 2-3. Nested Properties and Data Types

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

Key-Based Array and Indexed 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 position in the array does 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) 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. 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`.

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 the components of an array.

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.

Table 2-4. Special Characters

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

Client Applications

Any client application that you create using the vSphere Web Services SDK 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.

This chapter includes the following topics:

- [“Basic Client Application Capability”](#) on page 25
- [“Overview of a Java Sample Application”](#) on page 25
- [“Web Server Session Token”](#) on page 28
- [“Multiple Versions of the vSphere API”](#) on page 29
- [“Identifying the API Version Supported by the Server”](#) on page 30
- [“Helper Classes for Sample Applications”](#) on page 30

Basic Client Application Capability

Every client application that uses the vSphere API performs the following operations:

- 1 Create a connection to the vSphere server Web service.
- 2 Instantiate a local proxy object for reference to `ServiceInstance`. The `ServiceInstance` object provides access to the `ServiceContent` data object. `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

[Example 3-1](#) is a complete client application that demonstrates the basic client capability. The sample application accepts command-line arguments for the vSphere server name (DNS name or IP address), user name, and password.

The sample does not use the helper classes discussed in [“Helper Classes for Sample Applications”](#) on page 30 nor does it follow best practices. For example, it does not handle exceptions, and it uses the `SunFakeTrustSocketFactory` class instead of a local Java SSL certificate to establish the secure sockets layer (SSL) handshake. Using `SunFakeTrustSocketFactory` is appropriate for development environments only. Do not use this technique for production applications.

The following procedure uses code fragments from [Example 3-1](#).

To connect to a server and obtain a managed object reference to the ServiceInstance

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

- `ManagedObjectReference` for the `ServiceInstance`.
- `VimServiceLocator` 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:

```
private ManagedObjectReference siRef;
private VimServiceLocator locator;
private VimPortType service;
private ServiceContent siContent;
```

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

```
siRef = new ManagedObjectReference();
siRef.setType("ServiceInstance");
siRef.set_value("ServiceInstance");
```

- 3 Connect to the server.

To identify the vSphere Web Service, the following code fragment creates a URL that uses the target server host name and the path `/sdk/vimService`.

```
String url = "https://" + hostName + "/sdk/vimService";
...
locator = new VimServiceLocator();
locator.setMaintainSession(true);
service = locator.getVimPort(new URL(url));
```

The Axis SOAP toolkit generates the `VimServiceLocator` and `VimPortType` objects to provide access to vSphere servers. The `getVimPort` method returns a `VimPortType` object that provides access to the vSphere API methods. Use the following general syntax to call a `VimPortType` method:

```
webservice.method(mor[, param...])
```

- *webservice* represents the `VimPortType` variable.
- *method* is the name of a `VimPortType` method.
- *mor* is the managed object reference to the object that defines method.
- *param...* is one or more parameters that might be defined for the method.

- 4 Obtain `ServiceInstance` content (the `ServiceContent` data object).

Pass the managed object reference for the `ServiceInstance` object (`siRef`) to the `VimPortType.retrieveServiceContent` method. The method corresponds to the managed object method `ServiceInstance.RetrieveServiceContent`.

```
siContent = service.retrieveServiceContent(siRef);
```

- 5 Log in to the server. If an instance of `SessionManager` exists for the current `ServiceContent` object, the example code calls the `SessionManager.login` method.

The call to the `VimPortType.login` method specifies the `SessionManager` managed object reference (returned by `siContent.getSessionManager`), along with the user name and password.

```
if (siContent.getSessionManager() != null) {
    service.login(siContent.getSessionManager(),
                 userName,
                 password,
                 null);
}
```

The `null` value in the argument list is for the `Login` method's `locale` parameter.

- 6 Obtain the root folder of the inventory, obtain a reference to one or more of the managers (PerformanceManager, for example), and perform other tasks required by your application.

In [Example 3-1](#), the application only obtains some information about the server. The code fragment uses the Axis SOAP toolkit accessor methods (`getAbout`, `getFullName`, `getApiType`, and `getApiVersion`) to retrieve the `AboutInfo` data from the `ServiceContent` data object.

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

- 7 Close the connection to maintain security.

```
service.logout(siContent.getSessionManager());
service = null;
siContent = null;
```

Use the following command to execute the compiled class:

```
C:\>java TestClient server01.vmware.com tooluser password
```

[Table 3-1](#) shows the output from using the sample application to connect to ESX/ESXi and vCenter Server systems.

Table 3-1. Sample Runs of Simple Client Application

ESX	vCenter Server
VMware ESX 4.0.0 build-139526 API type is HostAgent API version is 4.0	VMware vCenter Server 4.0.0 build-139524 API type is VirtualCenter API version is 4.0

[Example 3-1](#) shows the complete sample client application code.

Example 3-1. Simple Java Client Application

```
import com.vmware.vim25.*;
import java.net.URL;

public class TestClient {

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

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

        // variables for access to the server
        ManagedObjectReference siRef;
        VimServiceLocator locator;
        VimPortType service;
        ServiceContent siContent;

        // skip the certificate
        System.setProperty("axis.socketSecureFactory",
            "org.apache.axis.components.net.SunFakeTrustSocketFactory");

        // set up the MOR for the ServiceInstance
        siRef = new ManagedObjectReference();
        siRef.setType("ServiceInstance");
        siRef.setValue("ServiceInstance");

        // connect to the server
        locator = new VimServiceLocator();
        locator.setMaintainSession(true);
        service = locator.getVimPort(new URL(url));
```

```

// get the ServiceContent object and login
siContent = service.retrieveServiceContent(siRef);
if (siContent.getSessionManager() != null) {
    service.login(siContent.getSessionManager(),
        userName,
        password,
        null);
}

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

// close the connection
service.logout(siContent.getSessionManager());
service = null;
siContent = null;
}
}
}

```

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.

In [Example 3-1](#), the client application does not explicitly handle the session token. However, you can extract the session token from the SOAP message context and save it to a local file, for reuse later. For example, if your client application is multithreaded, save the token and use it for each thread in the application. See [“Using the Credential Store for Automated Login”](#) on page 63.

[Example 3-2](#) shows an example that uses Java and Axis client libraries to obtain the session token from the current SOAP message context. The example casts the token to a `String`. With the session token in `String` form, you can save it locally to a file on the client for subsequent reuse. In your connection setup code, you can check for an existing session file and use that session file as needed.

Example 3-2. Obtaining a Session Token from the Web Services Context

```

org.apache.axis.client.Stub clientStub = (org.apache.axis.client.Stub)_service;
org.apache.axis.client.Call callObj = clientStub._getCall();
org.apache.axis.MessageContext msgContext = callObj.getMessageContext();
String sessionString = (String)msgContext.getProperty(
    org.apache.axis.transport.http.HTTPConstants.HEADER_COOKIE);

```

Session tokens returned from connections to VirtualCenter Server 2.5 and ESX/ESXi 3.5, for example, might look like the following:

```
vCenter Server    vmware_soap_session="63869259-F0FF-4DB5-9B3B-6493212AB9CD"
```

```
ESX/ESXi         vmware_soap_session="52b1910a-31ad-df35-95d1-210f86c55efb"
```

With every connection, the identifier is passed from the client to the server in the HTTP header. Session tokens can be passed across multiple connections to the Web service. By default, sessions expire after 30 minutes of inactivity, but you can change the value. For example, you can set the timeout value to 45 minutes (2700 seconds), as follows:

```
...
_locator = new VimServiceLocator();
_locator.setMaintainSession(true);
_service = _locator.getVimPort(new URL(url));
((org.apache.axis.client.Stub)_service).setTimeout(2700);
...
```

The sample applications included with the vSphere Web Services SDK include both C# and Java helper classes that handle the details of creating sessions, obtaining session tokens, saving the session token as a string to a file, and reusing the session. The Microsoft .NET Web services implementation uses the `Cookie` class to handle the session information from the server.

Multiple Versions of the vSphere API

When a client application connects to a Web service running on an vSphere server (ESX/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 ([Example 3-3](#)).

Example 3-3. 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>4.0</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 (ESX/ESXi 4.0 and ESX/ESXi 3.5, 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.

The vSphere Web Services SDK package includes several sample applications demonstrating how to work with multiple server versions (Table 3-2).

Table 3-2. Sample Applications that Demonstrate Working with Multiple API Versions

Java (SDK\samples\Axis\java\com\vmware\samples\version)	C# SDK\samples\DotNet\cs\
displaynewproperties	DisplayNewProperties
displaynewpropertieshost	DisplayNewPropertiesHost
displaynewpropertiesvm	DisplayNewPropertiesVM
getvirtualdiskfiles	GetVirtualDiskFiles
hostpowerops	HostPowerOps
installhostpatch	~
querymemoryoverhead	QueryMemoryOverhead
recordsession	RecordSession
apputils\version	AppUtil\VersionUtil.cs

Identifying the API Version Supported by the Server

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 not ESX/ESXi 4.x or vCenter Server 4.x.

Example 3-4 is an excerpt from the `VersionUtil.java` sample that demonstrates this general approach. See the `getTargetNameSpaceAndVersion` method of `VersionUtil.java` located in the vSphere Web Services SDK package for more information.

Example 3-4. Determining the URL from Which to Obtain the API Version Information

```
try {
    String wsdlUrlString = "";
    String vimServiceXmlUrlString = "";

    if((urlString.indexOf("https://") !=-1) || (urlString.indexOf("http://") !=-1)){
        wsdlUrlString = urlString.substring(0,urlString.indexOf("/sdk")+4)
            + "/vimService?wsdl";
        vimServiceXmlUrlString = urlString.substring(0,urlString.indexOf("/sdk")+4)
            + "/vimServiceVersions.xml";
    }
    else{
        wsdlUrlString = "https://" + urlString + "/sdk/vimService?wsdl";
        vimServiceXmlUrlString = "https://" + urlString
            + "/sdk/vimServiceVersions.xml";
    }
}
```

Helper Classes for Sample Applications

The vSphere Web Services SDK includes sample applications, written in C# and in Java, that demonstrate features of the vSphere API and object model (see Appendix E, “Sample Program Overview,” on page 191). The samples rely on several utility classes (helper classes). The helper classes (listed in Table 3-3 and Table 3-4) handle command-line input such as common parameters, server name, and other details. These helper classes are located in the unpacked SDK download, in these locations:

Java Helper Classes %SDKHOME%\samples\Axis\java\apputils

C# Helper Classes %SDKHOME%\samples\DotNet\cs\AppUtil

Java Helper Classes

Table 3-3 summarizes the Java versions of the client helper classes.

Table 3-3. Helper Classes for Java Sample Applications

Package	Java Class	Functional Description
com.vmware.apputils	ClientUtil.java	Convenience methods for handling user input from command line. Catches errors (faults). Logs output to console.
com.vmware.apputils.vim	ServiceUtil.java	Wrapper methods for the vimService methods (the local proxy code methods). Creates instances of ServiceConnection and AppUtil and handles connection to the service. Supports obtaining initial managed object references for root folder and all its descendents.
com.vmware.apputils.vim	ServiceConnection.java	Creates an instance of local proxy for connecting to the server, and obtains managed object references to several managed objects (ServiceInstance, ServiceContent, rootFolder) that serve as starting points for obtaining objects. Includes methods for saving sessions for reuse and loading saved sessions. Includes ignoreCert method for circumventing the local SSL certificate handling.
com.vmware.apputils	AppUtil.java	Sets up logging, instantiates local ClientUtil. Uses the credential store.
com.vmware.apputils.version	VersionUtil.java	Identifies the version of the WSDL on the server. Extracts prior API version information from the vimServiceVersions.xml file on the server (if it exists).
com.vmware.apputils.version	ExtendedAppUtil.java	Extends the AppUtil class with additional functionality for handling multiple API versions.
com.vmware.apputils.vim	VMUtils.java	Convenience methods for working with virtual machine objects, including provisioning and life-cycle management.

C# Helper Classes

Table 3-4 lists the helper classes available for C#.

Table 3-4. Helper Classes for C# Sample Applications

AppUtils	Functional Description
AppUtil.cs	Convenience methods for handling user input from command line. Catches errors (faults). Logs output to console.
ClientUtil.cs	Convenience methods for handling user input from command line. Catches errors (faults). Logs output to console.
OptionSpec.cs	Helper class for handling default and custom command-line arguments.
ServiceUtil.cs	Wrapper methods for the vimService methods (the local proxy code methods) for API 2.0 and prior releases.
ServiceUtilV25.cs	Wrapper methods for the vimService methods (the local proxy code methods) for API 4.0, API 3.5U2, API 3.5 U1, API 3.5, and other recent releases.
TrustAllCertificatePolicy.cs	Creates an instance of local proxy for connecting to the server, and obtains managed object references to several needed managed objects— ServiceInstance, ServiceContent, rootFolder.
VersionUtil.cs	Helper class for identifying API version.

Datacenter Inventory

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.

The chapter includes the following topics:

- [“Inventory Overview”](#) on page 33
- [“Inventory Hierarchies and ServiceInstance”](#) on page 33
- [“Accessing Inventory Objects”](#) on page 35
- [“Creating Inventory Objects”](#) on page 36
- [“Privileges Required for Inventory Management”](#) on page 36
- [“Managed and Standalone ESX/ESXi Hosts”](#) on page 37

Inventory Overview

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 5, “Property Collector,”](#) on page 39. How you access objects depends on whether your client application is connected to a vCenter Server or an ESX/ESXi system.

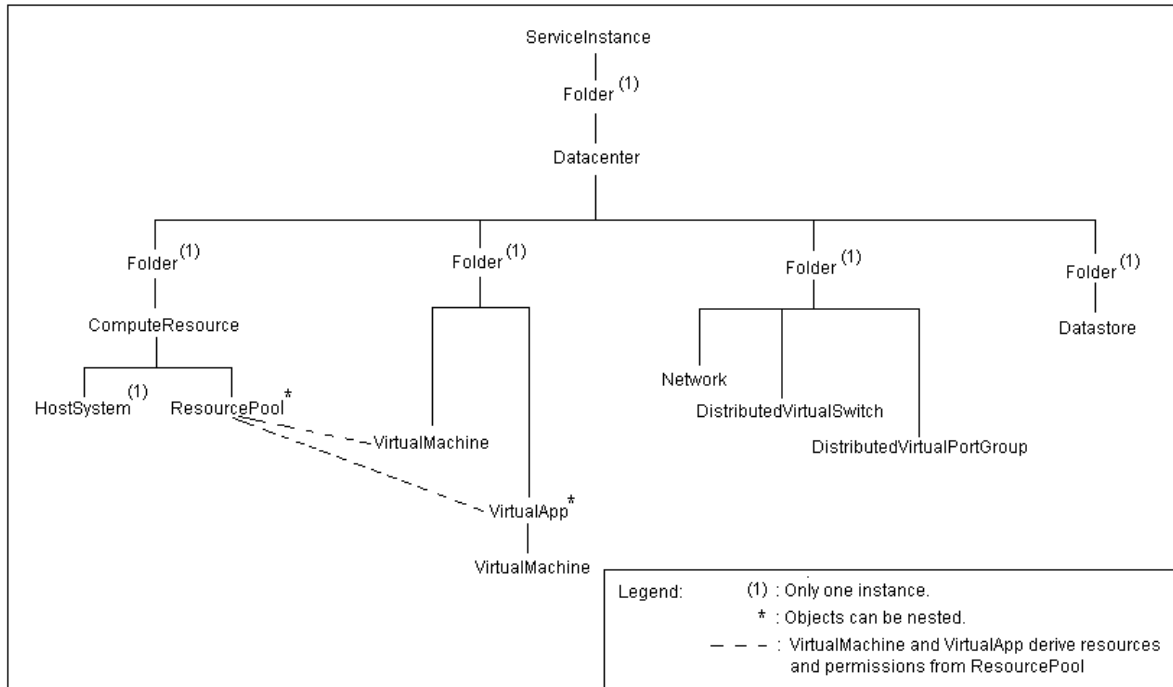
Inventory Hierarchies and ServiceInstance

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.

ESX/ESXi Inventory Hierarchy

When you access an ESX/ESXi host directly, rather than accessing the host through a vCenter Server system, the hierarchy shown in [Figure 4-1](#) allows you to traverse the inventory.

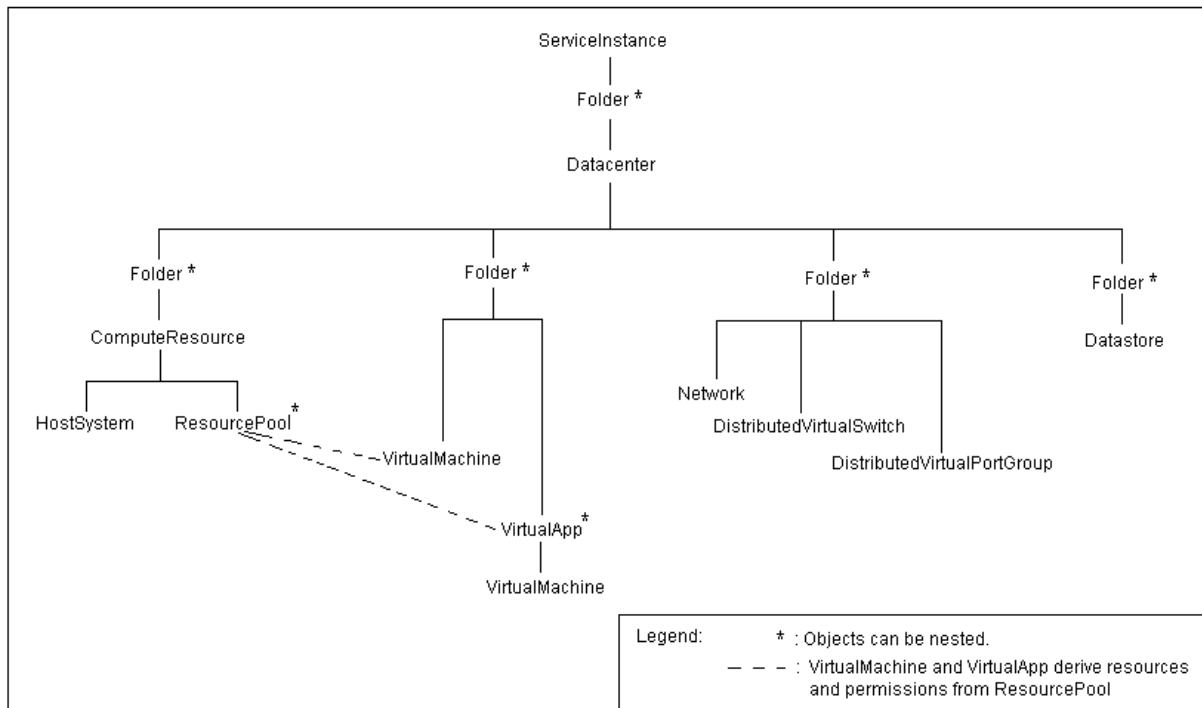
Figure 4-1. ESX/ESXi Inventory Hierarchy



vCenter Server Inventory Hierarchy

When you access a vCenter Server System, the hierarchy shown in [Figure 4-2](#) allows you to traverse the inventory.

Figure 4-2. vCenter Server Inventory Hierarchy



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

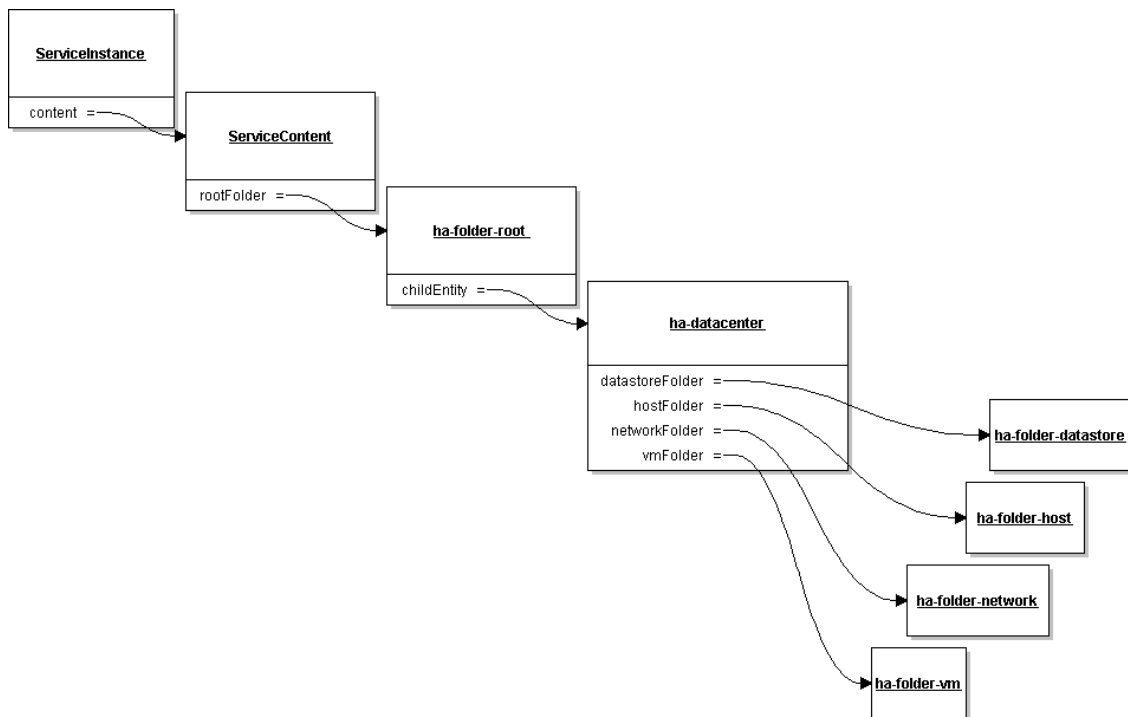
Accessing Inventory Objects

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 4-3 shows the `childEntity` and `folder` properties that define the default objects in the inventory of a standalone ESX/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 4-3. 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 are automatically created 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 ESX/ESXi. For example, the `Alarm.Create` privilege associated with `AlarmManager` is available only through vCenter Server systems.

See [Chapter 6, "Authentication and Authorization,"](#) on page 53 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

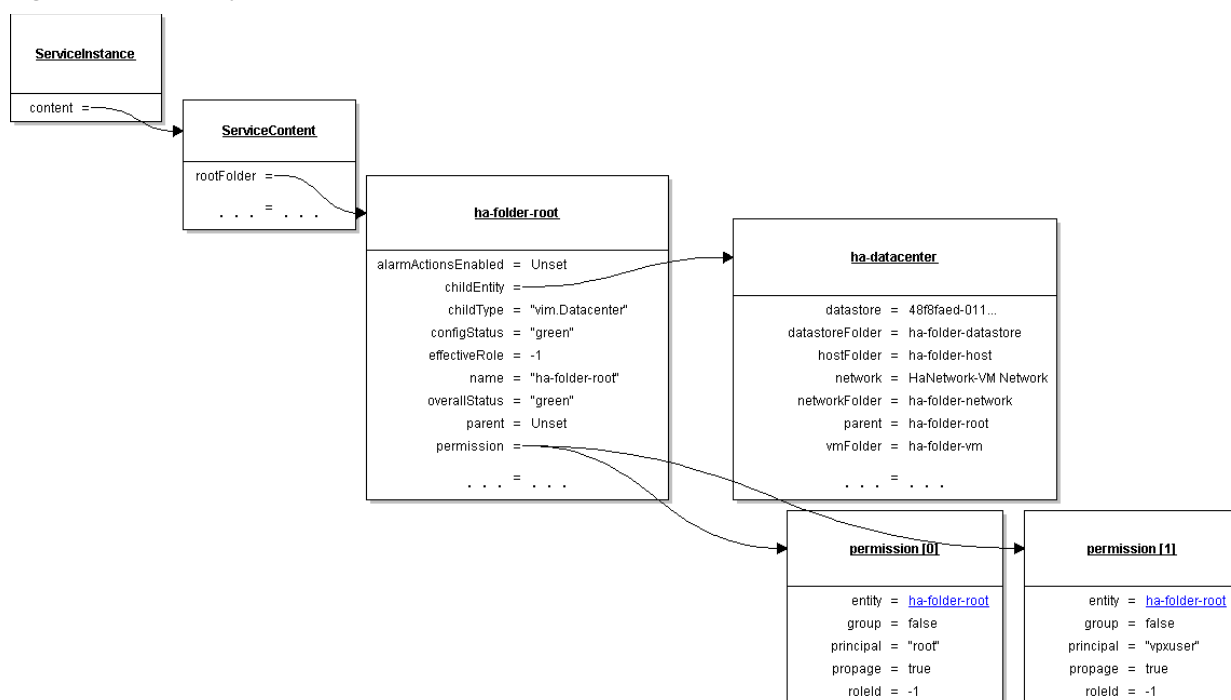
Permissions are the associations of roles with privileges on a specified managed entity. You use permissions to specify which users can access which managed entity.

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 4-4 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 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 6, "Authentication and Authorization,"](#) on page 53 for a detailed discussion of privileges, permissions, and user management.

Figure 4-4. Inventory and Permissions



Managed and Standalone ESX/ESXi Hosts

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

- Standalone ESX/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 ESX/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.

Table 4-1 summarizes the differences between the number of objects that an inventory can contain. See also [Figure 4-1, "ESX/ESXi Inventory Hierarchy,"](#) on page 34 and [Figure 4-2, "vCenter Server Inventory Hierarchy,"](#) on page 34.

Table 4-1. Standalone ESX/ESXi and vCenter Server Inventories

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

vSphere servers provide the `PropertyCollector` service for accessing data. You 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”](#) on page 39
- [“vSphere Data Objects for Property Collection”](#) on page 40
- [“vSphere Methods for Property Collection”](#) on page 40
- [“PropertyCollector Example \(RetrievePropertiesEx\)”](#) on page 41
- [“Inventory Traversal”](#) on page 45
- [“Client Data Synchronization \(WaitForUpdatesEx\)”](#) on page 50
- [“Server Data Transmission”](#) on page 52
- [“PropertyCollector Performance”](#) on page 52
- [“SearchIndex”](#) on page 52

Introduction to the PropertyCollector

The `PropertyCollector` service interface supports traversing the inventory to retrieve data from a vSphere server. You can retrieve information about managed objects, such as whether a virtual machine is powered on or whether a host in a cluster is offline. You can also retrieve data to keep the client representation of the virtual environment synchronized with the state of server objects.

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` and supports the use of multiple `PropertyCollector` objects. You can create additional `PropertyCollector` objects, using one per thread to perform mutually independent 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 and return data objects containing sets of name-value pairs for managed object properties.
- Incremental property retrieval 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. You can use either polling or notification to obtain incremental changes. Use `WaitForUpdatesEx` to maintain a client-side synchronized representation of server objects.

Inventory Traversal and Object Selection

`PropertyCollector` filter properties identify objects and paths that define inventory traversal. For example, you can retrieve the properties for a `VirtualMachine` object and specify a traversal path using the `VirtualMachine.network` property to obtain the properties for the associated `Network` objects.

You can use vSphere view objects (for example, `ContainerView`) in filters to simplify traversal specification and to support more efficient communication between client and server during `PropertyCollector` operations. A view maintains a subset of inventory objects. By using a view, you can easily specify a set of objects that the `PropertyCollector` can use for data collection. Also, the lifetime of a view is independent of the lifetime of a filter.

For information about the vSphere inventory, see [Chapter 4, “Datacenter Inventory,”](#) on page 33.

vSphere Data Objects for Property Collection

[Table 5-1](#) provides an overview of the `PropertyCollector` data objects. For more detailed descriptions, see the *vSphere API Reference*.

Table 5-1. PropertyCollector Data Objects

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 one <code>PropertySpec</code> .
<code>ObjectSpec</code>	Identifies the starting object for property collection. An <code>ObjectSpec</code> also identifies additional objects for collection.
<code>TraversalSpec</code>	Identifies the type of object for property collection. It also provides one or more paths for inventory traversal.
<code>SelectionSpec</code>	Acts as a placeholder reference to a <code>TraversalSpec</code> .
<code>PropertySpec</code>	Identifies properties for collection.
View objects	Identify a subset of the vSphere inventory 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.

[Table 5-2](#) shows the `PropertyCollector` methods organized by the context in which you use them. For more information about these methods, see the *vSphere API Reference*.

Table 5-2. `PropertyCollector` Methods

Method Context	Method	Description
Thread support	<code>CreatePropertyCollector</code>	Creates a new <code>PropertyCollector</code> object for use in a separate thread. 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 specified managed objects.
	<code>ContinueRetrievePropertiesEx</code>	Retrieves additional property data for an operation started by <code>RetrievePropertiesEx</code> .
	<code>CancelRetrievePropertiesEx</code>	Cancels a <code>RetrievePropertiesEx</code> or <code>ContinueRetrievePropertiesEx</code> operation.
Incremental collection operation	<code>WaitForUpdatesEx</code>	Retrieves changes to property data since the last <code>WaitForUpdatesEx</code> cycle. <code>WaitForUpdatesEx</code> blocks until it can satisfy the request or until the request times out. <code>WaitForUpdatesEx</code> supports chunked data transmission (see “Server Data Transmission” on page 52), polling, and notification.
	<code>CancelWaitForUpdatesEx</code>	Cancels a <code>WaitForUpdatesEx</code> operation.
General	<code>CreateFilter</code>	Creates a new instance of a <code>PropertyFilter</code> managed object.

PropertyCollector Example (RetrievePropertiesEx)

[Example 5-1](#) 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:

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

The following procedure uses code fragments from [Example 5-1](#). The complete example includes server connection code; this procedure only describes the task of using the `PropertyCollector`. For a description of server connection, see [“To connect to a server and obtain a managed object reference to the ServiceInstance”](#) on page 26.

To use the `PropertyCollector` for a single retrieval operation

- 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 `container` (the inventory root folder, returned by the method `sContent.getRootFolder`) and `type` ("Virtual Machine") direct the `ViewManager` to select virtual machines, starting at the root folder. The value `true` for the `recursive` parameter extends the selection beyond the root folder so that the `ViewManager` will follow child folder paths to add virtual machines to the view. The container view provides references to all virtual machines in the inventory.

```
String[] vm = {"VirtualMachine"};
ManagedObjectReference cViewRef =
    methods.createContainerView(viewMgrRef, sContent.getRootFolder(), vm, 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 an object type. `TraversalSpec.path` identifies a property in the `type` object. 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.

```
oSpec.setSelectSet(new TraversalSpec[] {tSpec});
```

- 6 Identify the properties to be retrieved.

The example program creates a `PropertySpec` data object to specify the properties to be collected. The `type` property is set to `VirtualMachine` to match the object selections 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.setPathSet(new String[] {"name"});
```

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

A `PropertyFilterSpec` must have at least one `ObjectSpec` and one `PropertySpec`.

```
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.setObjectSet(new ObjectSpec[] {oSpec});
fSpec.setPropSet(new PropertySpec[] {pSpec});
```

8 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 that no maximum for the number of objects that can be returned is set. The `PropertyCollector` can impose a maximum. If the number of collected objects is greater than the maximum, the `PropertyCollector` returns a token value in the `RetrieveResult` data object. For more information, see [“Server Data Transmission”](#) on page 52.

```
PropertyFilterSpec[] pfs = {fSpec};
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl,pfs,ro);
```

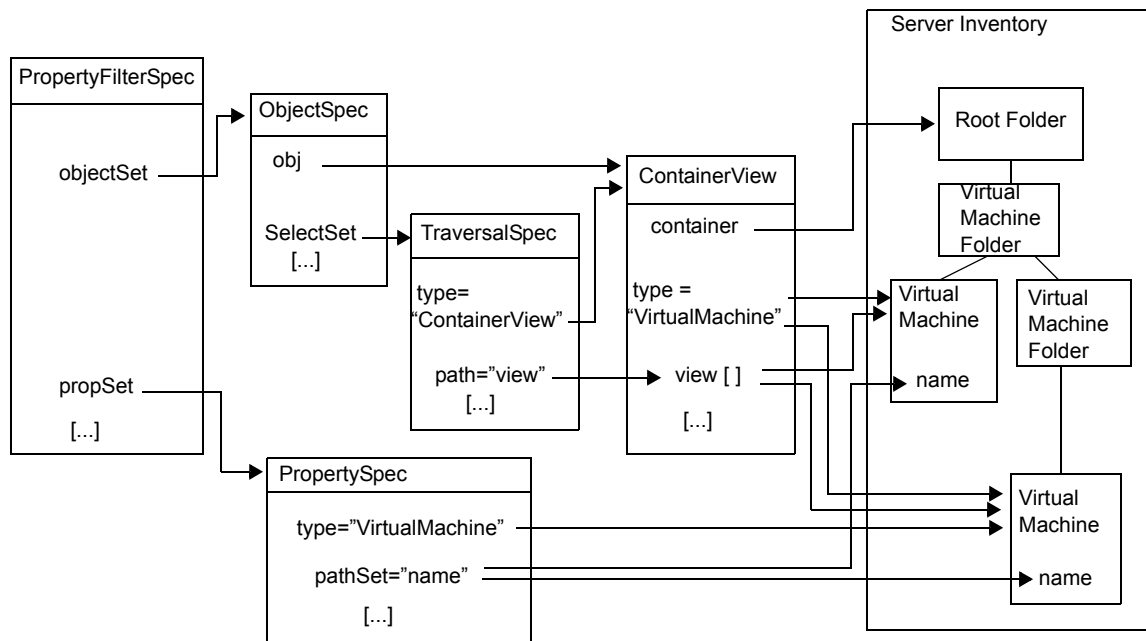
9 Print the virtual machine names.

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.

```
for (ObjectContent oc : props.getObjects()) {
    for (DynamicProperty dp : oc.getPropSet()) {
        System.out.println(dp.getName() + " = " + dp.getVal());
    }
}
```

Figure 5-1 shows the objects used in Example 5-1. The figure represents properties that identify inventory elements directly or indirectly. It does not show all the properties for the different objects.

Figure 5-1. Property Filter Specification



Example 5-1. Simple `PropertyCollector` Example (Java)

```
import com.vmware.vim25.*;
import java.net.URL;

// 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
String[] vm = {"VirtualMachine"};
ManagedObjectReference cViewRef =
    methods.createContainerView( viewMgrRef,sContent.getRootFolder(),vm,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");
oSpec.setSelectSet(new TraversalSpec[] {tSpec});

// specify the property for retrieval (virtual machine name)
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.setPathSet(new String[] {"name"});

// add the object and property specs to the property filter spec
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.setObjectSet(new ObjectSpec[] {oSpec});
fSpec.setPropSet(new PropertySpec[] {pSpec});

// get the data
PropertyFilterSpec[] pfs = {fSpec};
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl,pfs,ro);

for (ObjectContent oc : props.getObjects()) {
    for (DynamicProperty dp : oc.getPropSet()) {
        System.out.println(dp.getName() + "=" + dp.getVal());
    }
}
}

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 for the server connection
    ManagedObjectReference siRef;
    VimServiceLocator locator;
    VimPortType service;
    ServiceContent siContent;

    // ignore certificates
    System.setProperty("axis.socketSecureFactory",
        "org.apache.axis.components.net.SunFakeTrustSocketFactory");

    // ServiceInstance MOR
    siRef = new ManagedObjectReference();
    siRef.setType("ServiceInstance");
    siRef.set_value("ServiceInstance");

```

```

// allocate a service locator, connect, and login
locator = new VimServiceLocator();
locator.setMaintainSession(true);
service = locator.getVimPort(new URL(url));
siContent = service.retrieveServiceContent(siRef);
if (siContent.getSessionManager() != null) {
    service.login(siContent.getSessionManager(),
                userName,
                password,
                null);
}

// retrieve data
collectProperties( service, siContent );

service.logout(siContent.getSessionManager());
service = null;
siContent = null;
}
}

```

Inventory Traversal

[Example 5-1](#) 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 select objects from the inventory, a filter includes `TraversalSpec` and possibly `SelectionSpec` objects. Use these objects to make object selections based on the references in a view, and to extend inventory traversal beyond those objects (or beyond the object specified in `ObjectSpec.obj`).

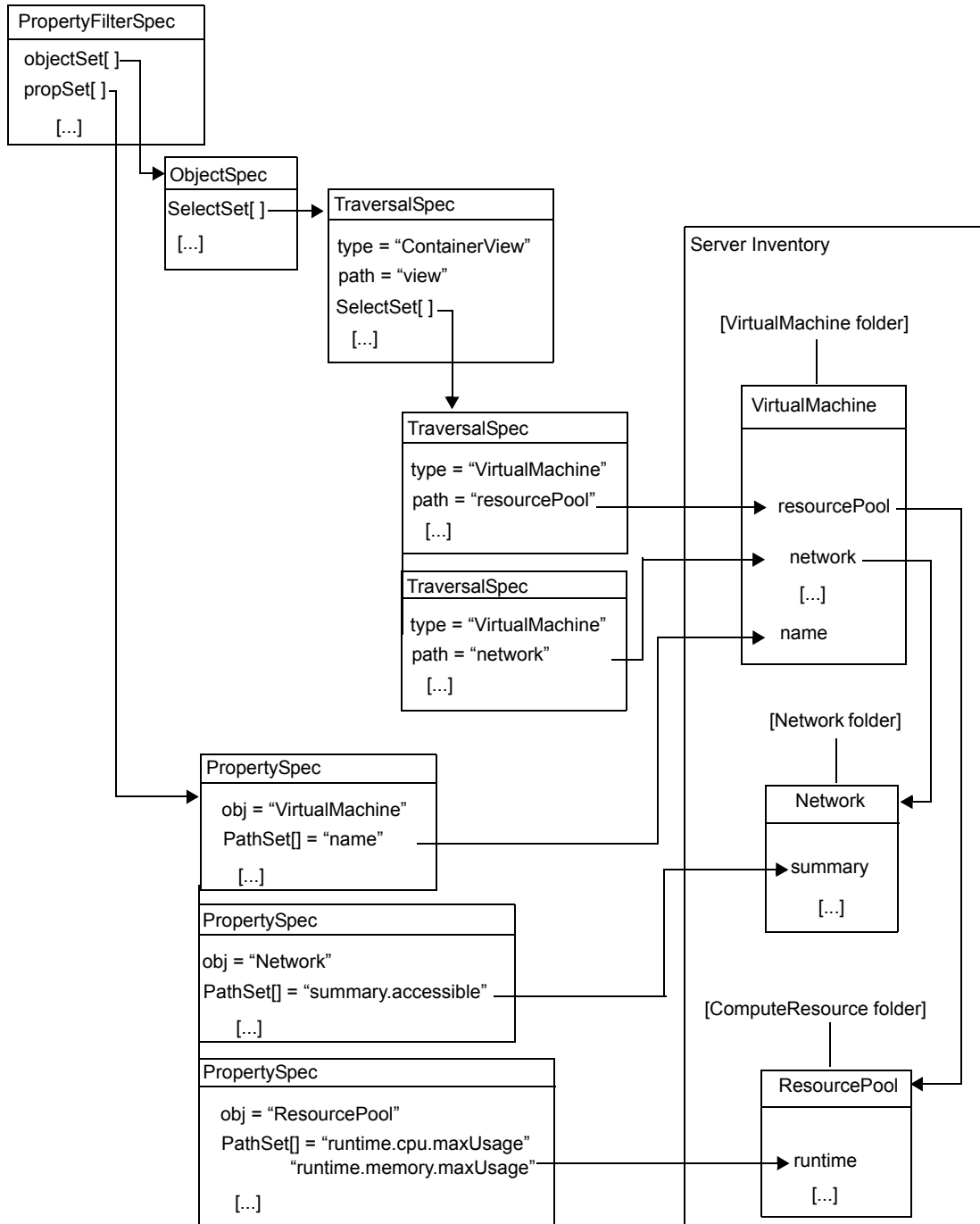
TraversalSpec Traversal

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` applies the `TraversalSpec` objects in the `selectSet` array to the result of the traversal (the target of `TraversalSpec.path`). The `selectSet` array can also contain `SelectionSpec` objects; a `SelectionSpec` is a reference to a `TraversalSpec`. See [“SelectionSpec Traversal”](#) on page 48.
- `skip` – indicates whether to collect properties for the path object.

During inventory traversal, the `PropertyCollector` applies the `PropertySpec` object or objects (`PropertyFilterSpec.propSet`) to objects. Inventory traversal begins with the object identified by `ObjectSpec.obj` and continues by following `TraversalSpec` paths. If `PropertySpec.type` matches the current object type, and the `skip` property is `false`, the `PropertyCollector` sends the `PropertySpec.pathSet` properties to your client.

[Figure 5-2](#) is a representation of a `PropertyFilterSpec` that defines traversal of `VirtualMachine` 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 5-2. Inventory Navigation

Example 5-2 shows a Java code fragment, based on Example 5-1, that implements the inventory traversal shown in Figure 5-2.

To define inventory traversal

- 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.

Example 5-2. Inventory Traversal

```
// start with a container view
String[] vm = {"VirtualMachine"};
ManagedObjectReference cViewRef =
    methods.createContainerView( viewMgrRef, sContent.getRootFolder(), vm,true );

// create an object spec for 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 it to the object spec
oSpec.setSelectSet(new TraversalSpec[] {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 to virtual machine traversal
tSpec.setSelectSet(new TraversalSpec[] {tSpecVmN,tSpecVmRp});

// specify the properties for retrieval
PropertySpec pSpec = new PropertySpec();
pSpec.setType("VirtualMachine");
pSpec.setPathSet(new String[] {"name"});

PropertySpec pSpecNs = new PropertySpec();
pSpecNs.setType("Network");
pSpecNs.setPathSet(new String[] {"summary.accessible"});

PropertySpec pSpecRPr = new PropertySpec();
pSpecRPr.setType("ResourcePool");
pSpecRPr.setPathSet(new String[] {"runtime.cpu.maxUsage",
    "runtime.memory.maxUsage"});
```

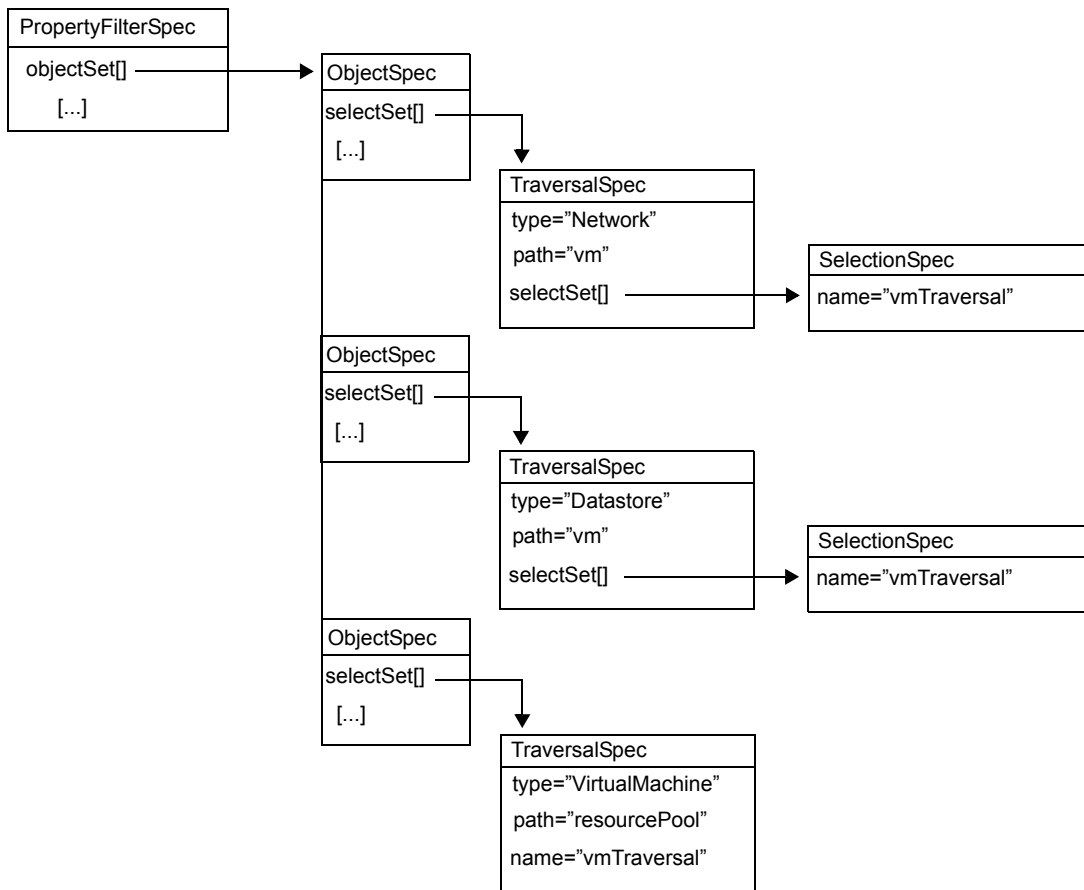
SelectionSpec Traversal

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 5-3](#) shows the use of `SelectionSpec` references to a virtual machine `TraversalSpec`. The `SelectionSpec` references are associated with `Network` and `Datastore` traversals.

Figure 5-3. SelectionSpec Reference



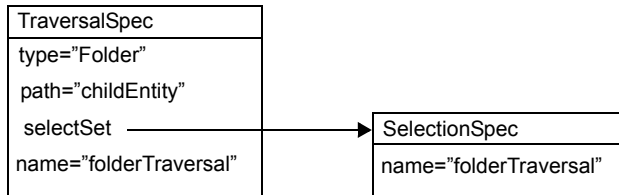
If the `ObjectSpec.selectSet` array contains a `SelectionSpec`, the referenced `TraversalSpec` must identify the same object type. `TraversalSpec.type` must match the type of the object specified in `ObjectSpec.obj`. The `PropertyCollector` applies the `TraversalSpec` to the object and use the `TraversalSpec.path` 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”](#) on page 33 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 5-4](#) shows a representation of a `TraversalSpec` and its associated `SelectionSpec` for nested folder traversal.

Figure 5-4. Recursive `TraversalSpec` and `SelectionSpec`



[Example 5-3](#) shows a Java code fragment that creates a recursive filter for nested folder traversal.

To define recursive inventory traversal

- 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”](#) on page 52.

- 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.

Example 5-3. Nested Folder Traversal

```

// Get reference to PropertyCollector, root vm folder
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(false);

// 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");

// add the SelectionSpec for recursive folder traversal
tSpecF.setSelectSet(new SelectionSpec[] {sSpecF});

// add folder traversal to object spec
oSpec.setSelectSet(new TraversalSpec[] {tSpecF});

// specify the properties for retrieval (folder name)
PropertySpec pSpec = new PropertySpec();
pSpec.setType("Folder");
pSpec.setPathSet(new String[] {"name"});

// add the object and property specs to the property filter spec
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.setObjectSet(new ObjectSpec[] {oSpec});
fSpec.setPropSet(new PropertySpec[] {pSpec});

// get the data
PropertyFilterSpec[] pfs = {fSpec};
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = methods.retrievePropertiesEx(propColl,pfs,ro);

```

Client Data Synchronization (WaitForUpdatesEx)

To maintain a client-side representation of server object state, use the `CreateFilter` and `WaitForUpdatesEx` methods. The `WaitForUpdatesEx` method supports an incremental retrieval model.

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 both polling and notification mechanisms for property collection. When you call `WaitForUpdatesEx`, specify the following parameters:

- 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. For more information about data versions, see [“Server Data Transmission”](#) on page 52.
- 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 a polling or notification model. Using a polling model, your client calls the `WaitForUpdatesEx` method and the server returns data immediately. Using a notification model, your client calls the method and waits for a response. `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.
- `maxWaitSeconds` set to zero specifies polling. 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 notification. The `PropertyCollector` returns null if no updates are available within `maxWaitSeconds`.

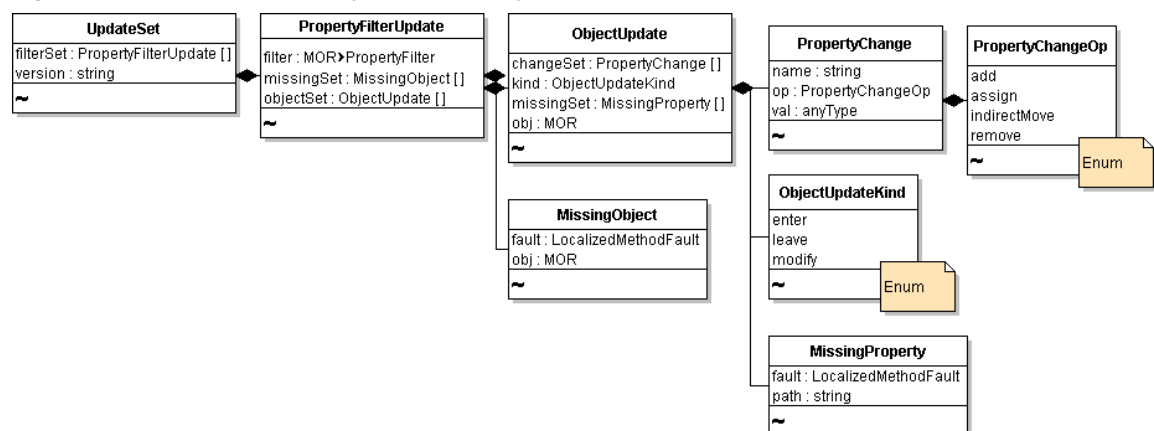
Table 5-3 lists some of the advantages and disadvantages of these two operations.

Table 5-3. `WaitForUpdatesEx` Operations Compared

Operation	Advantages	Disadvantages
<code>MaxWaitSeconds=0</code>	Polling mechanism that 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>	Notification mechanism that blocks 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.

The `WaitForUpdatesEx` method returns an `UpdateSet` data object, the composite data structure shown in Figure 5-5.

Figure 5-5. `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.

For VirtualCenter Server 2.5, for an inventory of 2,000 virtual machines supporting about 25 client applications simultaneously, performance is acceptable up to about 12 moderately complex filters monitoring updates on all 2,000 virtual machines. If the filters include complex traversal objects, performance might degrade appreciably unless you reduce the number of filters across the system.

To minimize `PropertyCollector` overhead and the amount of network traffic for your client application, use `View` objects with the `PropertyCollector`. [Example 5-1](#) 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

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 ESX/ESXi systems and vCenter Server systems.

The chapter includes the following topics:

- [“Objects for Authentication and Authorization Management”](#) on page 53
- [“Authentication and Authorization for ESX/ESXi and vCenter Server”](#) on page 54
- [“Setting Up Users, Groups, and Permissions”](#) on page 56
- [“Obtaining User and Group Information from UserDirectory”](#) on page 57
- [“Managing ESX/ESXi Users and Groups with HostLocalAccountManager”](#) on page 57
- [“Managing Roles and Permissions with AuthorizationManager”](#) on page 58
- [“Authenticating Users Through SessionManager”](#) on page 63
- [“Using the Credential Store for Automated Login”](#) on page 63
- [“Managing Licenses with LicenseManager”](#) on page 66

See the *vSphere Datacenter Administration Guide* for a list of required privileges for common tasks and best practices for roles and permissions. See [Appendix D, “Privileges Reference,”](#) on page 177 for lists of privileges required to invoke operations and to read properties, and privileges defined for the administrator role.

Objects for Authentication and Authorization Management

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

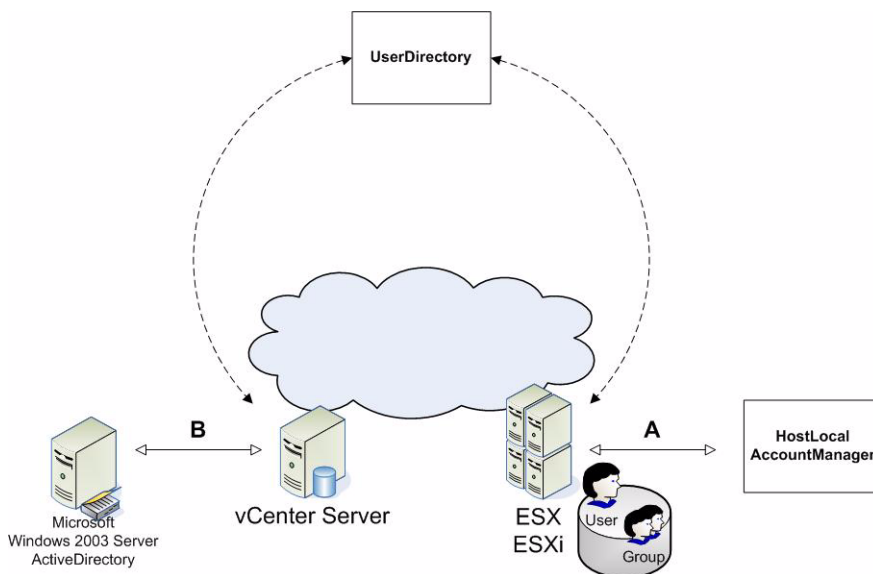
- `HostLocalAccountManager` is used to create and manage user accounts and groups on ESX/ESXi systems. Authenticated users can view objects or invoke operations on the server depending on the permissions associated with their account. See [“Managing ESX/ESXi Users and Groups with HostLocalAccountManager”](#) on page 57.
- `AuthorizationManager` protects vSphere components from unauthorized access. Access to components is role-based: Users and groups 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”](#) on page 57.

- `SessionManager` provides an interface to the authentication infrastructure on the target server system (see [“Authenticating Users Through SessionManager”](#) on page 63).
 - For vCenter Server systems, `SessionManager` supports integration with Microsoft SSPI (Security Service Provider Interface), which enables single sign-on using Microsoft Windows local accounts or Active Directory.
 - For ESX/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”](#) on page 66.

Authentication and Authorization for ESX/ESXi and vCenter Server

Several server-side mechanisms interactively 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 ESX/ESXi uses Linux-based authentication, and vCenter Server is a Windows service, the two systems use different approaches for handling user accounts. [Figure 6-1](#) shows the two different user management mechanisms associated with the VMware vSphere server.

Figure 6-1. Managed Objects for Handling User Accounts



These services work together to ensure that only authenticated users can connect to ESX/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”](#) on page 63.

ESX/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.

ESX/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 ESX/ESXi system by using `HostLocalAccountManager`.

vCenter Server User Model

vCenter Server is a Windows-based service that uses 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 the vCenter Server installation process. This vCenter Server administrator account must be a member of the local Windows Administrator group on the machine. VMware recommends creating a dedicated Windows user account for installing and managing the vCenter Server system.

Other vCenter Server users who connect to the Web service must also have a Windows account on the local Administrator group.

IMPORTANT Even if a user with the same name exists on an ESX/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 2003 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.

VMware vSphere Web Services SDK client applications can use SSPI to authenticate to the local machine and establish a remote connection. See the *vSphere API Reference* for more information.

vSphere Security Model

While the details of authentication and authorization differ between ESX/ESXi and vCenter Server, the model itself is the same for both system. 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 ESX/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.

Table 6-1. Privileges Required for Datastore Objects Apply Regardless of Access Mechanism

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 (see [“Using Roles to Consolidate Sets of Privileges”](#) on page 60). When you assign a 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.

[Table D-3, “Privileges Granted to the Administrator Role,”](#) on page 186 shows a complete list of privileges encompassed by the Administrator role as defined on a vCenter Server 4.0 system.

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 ESX/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 ESX/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 [Table 6-2, “System and Sample Roles,”](#) on page 60.
- 2 If necessary, create additional roles (sets of privileges). See [“Modifying Sample Roles to Create New Roles”](#) on page 61.

- 3 Retrieve information about existing users and groups (see “[Obtaining User and Group Information from UserDirectory](#)” on page 57) and create additional groups if needed (see “[Managing ESX/ESXi Users and Groups with HostLocalAccountManager](#)” on page 57).
- 4 Associate users or groups with roles using permissions. See “[Managing Roles and Permissions with AuthorizationManager](#)” on page 58.

At runtime, use `SessionManager` to log in to the server. The `SessionManager.Login` method requires a user ID and password and determines what that session can access on the server.

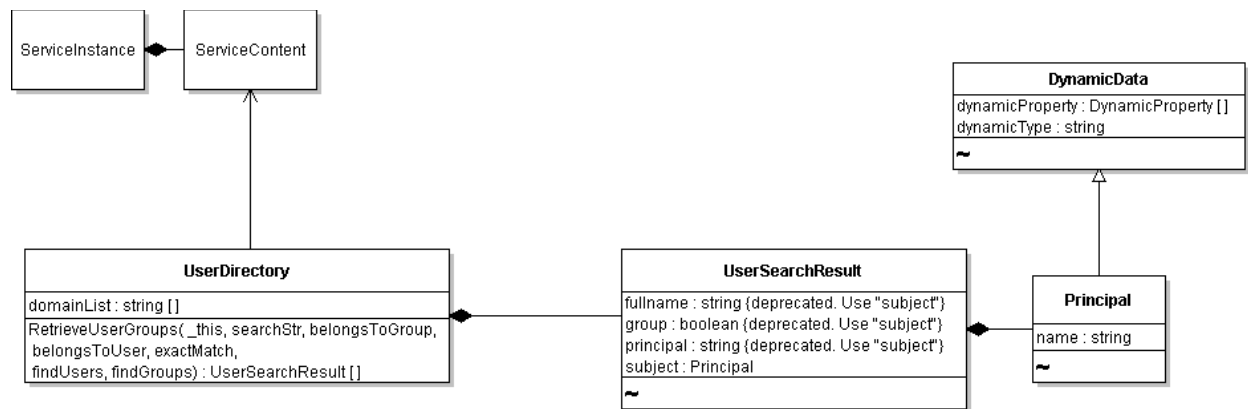
Obtaining User and Group Information from UserDirectory

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 ESX/ESXi system.

- **vCenter Server system.** Domain controller, Active Directory, or local Windows account repository.
- **ESX/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 ESX/ESXi systems, the `domainList` property is empty.

Figure 6-2. UserDirectory Managed Object



`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).

For ESX/ESXi systems, search returns all users and groups from the `passwd` file. If this file contains Network Information System (NIS) or NIS+ users and groups, `RetrieveUserGroups` returns these accounts as well.

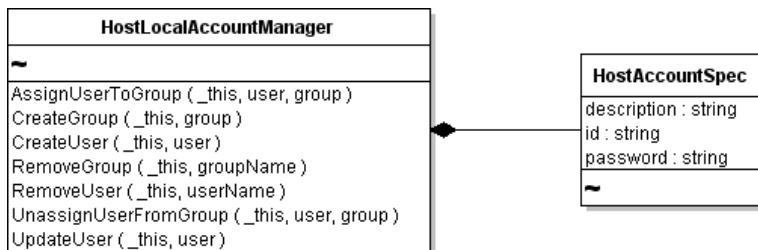
For vCenter Server, search is limited to the specified Windows domain. If the domain is omitted, the search is performed on local users and groups.

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

Managing ESX/ESXi Users and Groups with HostLocalAccountManager

The `HostLocalAccountManager` managed object supports user- and group-administration tasks. `HostLocalAccountManager` is available only on ESX/ESXi system.

IMPORTANT vCenter Server systems use the Microsoft Windows user management facilities. See “[vCenter Server User Model](#)” on page 55.

Figure 6-3. HostLocalAccountManager Managed Object

HostLocalAccountManager allows you manage uses and groups by using the following methods:

- **User Management.** CreateUser, RemoveUser, UnassignUserFromGroup, AssignUserToGroup, UpdateUser
- **Group Management.** CreateGroup, RemoveGroup

For those methods, you must define the HostAccountSpec data object following the requirements on the target system. Requirements include password length requirements, restrict use of dictionary words, and so on.

To create a user account on an ESX/ESXi system

- 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 ESX/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).

To assign a user to a group on an ESX/ESXi system

- 1 Create a HostAccountSpec data object that defines the properties of the group (description, id). No password is needed for the group.
- 2 Call the HostLocalAccountManager.CreateGroup method, passing in the HostAccountSpec data object.

Now that the user and group both exist, you can add the user to the group.

- 3 Call the HostLocalAccountManager.AssignUserToGroup method, passing the managed object reference to the HostLocalAccountManager, the `userId`, and the `groupId` in the invocation.

After creating user accounts and groups on the ESX/ESXi system, you can grant these users access to virtual components by using AuthorizationManager methods. See [“Managing Roles and Permissions with AuthorizationManager”](#) on page 58.

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.

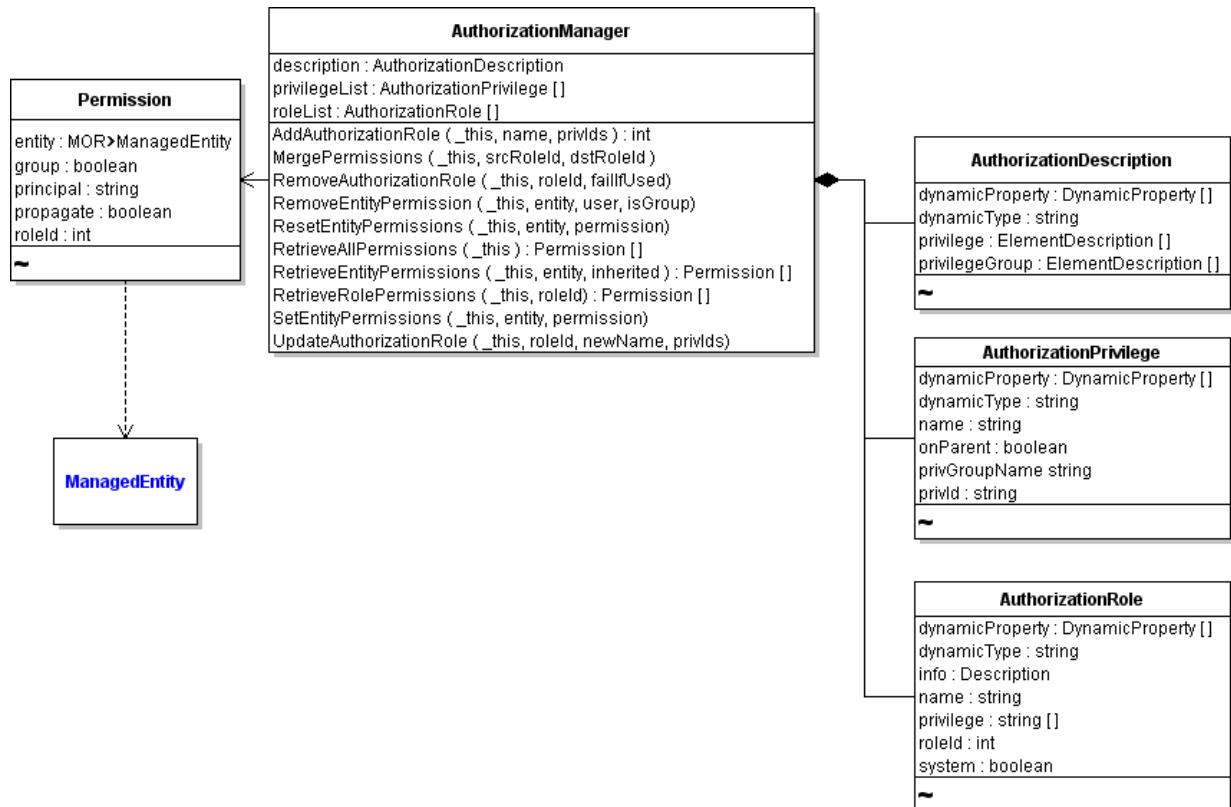
The AuthorizationManager also allows access and prevents access to specific server objects based on the permissions associated with the object.

AuthorizationManager includes methods for managing roles and for managing permissions:

- **Roles Management.** AddAuthorizationRole, RemoveAuthorizationRole, and UpdateAuthorizationRole. See [“Using Roles to Consolidate Sets of Privileges”](#) on page 60 and [“Modifying Sample Roles to Create New Roles”](#) on page 61.
- **Permissions Management.** MergePermissions, RemoveEntityPermission, ResetEntityPermissions, RetrieveAllPermissions, RetrieveEntityPermissions, RetrieveRolePermissions, and SetEntityPermissions. See [“Granting Privileges Through Permissions”](#) on page 61.

Figure 6-4 shows these methods in a UML diagram for AuthorizationManager and some of its associated data objects.

Figure 6-4. 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. See [Appendix D, “Privileges Reference,”](#) on page 177 for lists of privileges.
- 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. ESX/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 ESX/ESXi systems. You can create new roles using these predefined user roles as a starting point.

Table 6-2 describes these two types of roles in more detail and lists currently available roles as examples.

Table 6-2. System and Sample Roles

Type	Role name	Role ID	Description
System roles	Administrator	-1	Superuser access. Encompasses the set of all defined privileges. See Table D-3, “Privileges Granted to the Administrator Role,” on page 186 for an example list from a vCenter Server system. 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 No Access 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. (System.Anonymous, System.Read, and System.View). 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 System.Anonymous and System.View privileges. Cannot be granted.
Sample roles	Virtual Machine Administrator	1	Set of privileges necessary to manage virtual machines and hosts within the system.
	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 listed in [Table 6-2](#) cannot be modified or deleted. However, you can create new roles, or modify the sample roles.

To create new roles using the API

- 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

- 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

[Example 6-1](#) 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 6-1. 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 ESX/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 ESX/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 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 ESX/ESXi host to the password for that user on the server.
- C#, Java, and Perl libraries for managing the credential store programmatically. See [Table 6-4](#) for available methods.
- Java and Microsoft PowerShell-based command-line utilities for managing the credential store.

In addition to the libraries listed in [Table 6-3](#), 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 the credential store client libraries, shown in [Table 6-3](#) in an application, you must set up the credential store on all client machines that run your application.

Table 6-3. Credential Store Client Libraries

Package <code>com.vmware.security.credstore (Java)</code>	Namespace <code>VMware.Security.CredentialStore(C#)</code>
<code>CredentialStore.java</code>	<code>CredentialStoreFactory.cs</code>
<code>CredentialStoreFactory.java</code>	<code>CredentialStore.cs</code>

Several of the helper classes provided with the sample applications use the credential store mechanism.

Credential Store Methods

Table 6-4. Credential Store Client Methods

Java	C#	Description
<code>addPassword(hostname, username, password)</code>	<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>	<code>RemovePassword(hostname, username)</code>	Deletes the password for the specified user from the credential store.
<code>clearPasswords()</code>	<code>ClearPasswords()</code>	Deletes all passwords from the credential store.
<code>getPassword(hostname, username)</code>	<code>GetPassword(hostname, username)</code>	Returns the password for the specified host and user from the credential store.
<code>getHosts()</code>	<code>GetHosts()</code>	Returns the set of hosts contained in the credential store.
<code>getUsernames(hostname)</code>	<code>GetUsernames(hostname)</code>	Returns the collection of all user names that have passwords stored for the specified hostname.
<code>close()</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.** `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.

[Example 6-2](#) shows the XML elements that are read and written to the file.

Example 6-2. 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 ESX/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:

- 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 [Table 6-2, “System and Sample Roles,”](#) on page 60 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 perform tasks in the vSphere environment, you must have licenses to do so. Licensing applies to ESX/ESXi hosts, vCenter Server, and special features such as VMware HA or VMware vMotion.

The *vSphere Datacenter Administration Guide* explains how to manage ESX/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 ESX/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.

Hosts

Many of the operations in your vSphere environment involve setting up the ESX/ESXi hosts on which the virtualization layer runs. You can set up storage (see [Chapter 8, “Storage,”](#) on page 71) and networking (see [Chapter 9, “Networking,”](#) on page 85), and those settings directly affect the virtual machine. You must also manage other aspects of the host, as discussed in this chapter.

The chapter includes the following topics:

- [“Host Management Objects”](#) on page 67
- [“Retrieving Host Information”](#) on page 67
- [“Configuring and Reconfiguring Hosts”](#) on page 68
- [“Managing the Host Lifecycle”](#) on page 69
- [“Querying and Changing the Host Time”](#) on page 70
- [“Querying Virtual Machine Memory Overhead”](#) on page 70

IMPORTANT See the *ESX Configuration Guide* and the *ESXi Configuration Guide* for important information on security considerations, not included here.

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`.

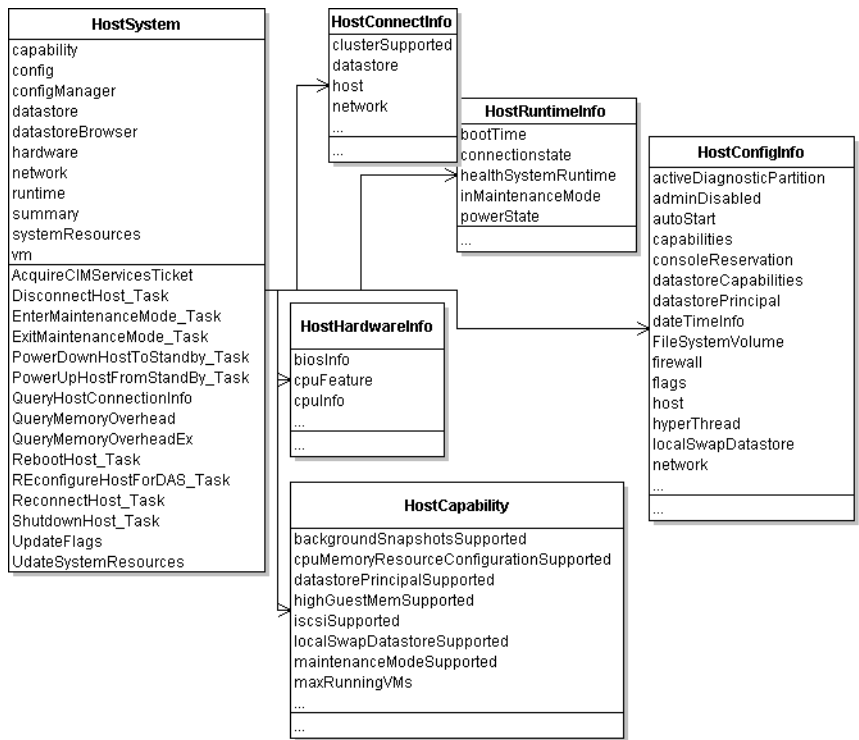
- `HostSystem.capability` is a `HostCapability` object. The `HostCapability` properties indicate the features that are supported by the host, for example, `maintenanceModeSupported` or `recursiveResourcePoolsSupported`.
- `HostSystem.runtimeInfo` is a `HostRuntimeInfo` object that contains several data 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 7-1. 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”](#) on page 69.
- **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 NTP configuration. See also [“Adding an NTP Service”](#) on page 92.

The `HostDateTimeSystem.dateTimeInfo` property allows you to retrieve and set date and time information. The `HostDateTimeInfo` data object’s properties contain two data object for date time management:

- `HostNTPConfig` contains a list of NTP servers for use by the host.
- `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 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 both the NTP 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 methods returns the amount of memory required, in bytes.

Storage

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.

The chapter includes the following topics:

- [“Storage Management Objects”](#) on page 71
- [“Introduction to Storage”](#) on page 72
- [“Choosing the Storage API to Use”](#) on page 74
- [“Configuring Disk Partitions”](#) on page 75
- [“Creating and Managing Datastores”](#) on page 78
- [“Managing VMFS Volume Copies \(Resignaturing\)”](#) on page 81
- [“Managing Diagnostic Partitions”](#) on page 82
- [“Sample Code Reference”](#) on page 83

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 9, “Networking,”](#) on page 85.

Storage Management Objects

You can access the objects that support storage management through the `HostSystem` managed object.

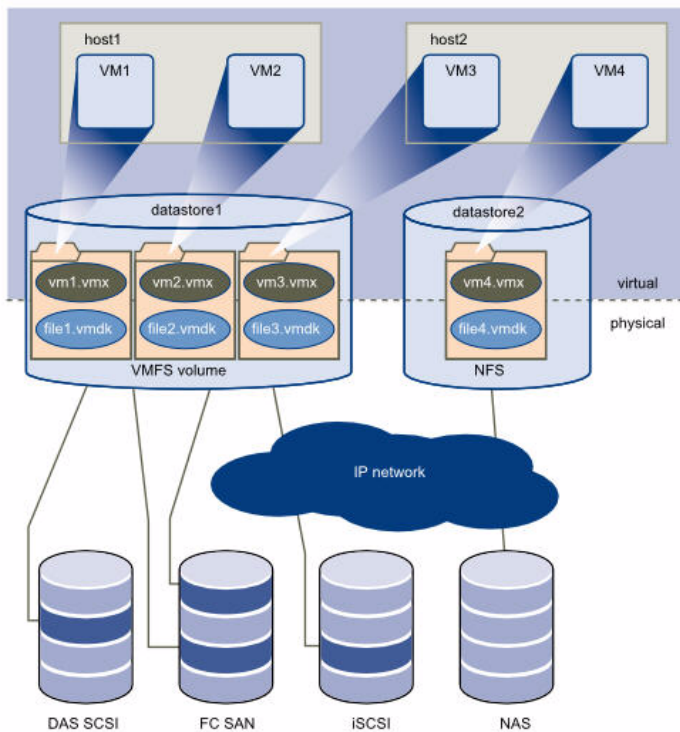
- `HostStorageSystem` – The `HostSystem.storageSystem` property is a managed object reference to the `HostStorageSystem` of the ESX/ESXi system. `HostStorageSystem` is a low-level interface that is used mainly for configuring the physical storage. See [“Configuring Disk Partitions”](#) on page 75.
- `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”](#) on page 78.
- `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”](#) on page 78.
- **HostDiagnosticPartition** – Supports creating and querying diagnostic partitions for your ESX/ESXi host. See [“Managing Diagnostic Partitions”](#) on page 82.

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 [Figure 8-1](#).

Figure 8-1. 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.

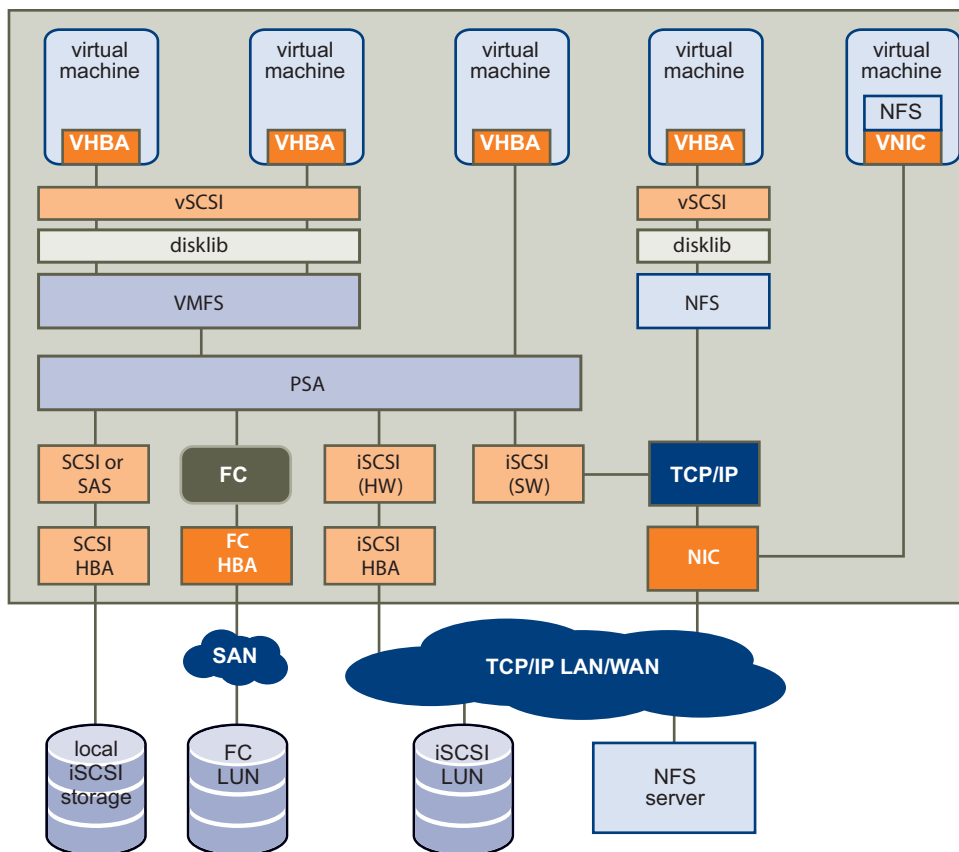
When a virtual machine communicates with its virtual disk stored on a datastore, it issues SCSI commands. Because datastores can exist on different types of physical storage, these commands are encapsulated into other forms, depending on the protocol that the ESX/ESXi host uses to connect to the physical 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 8-1](#). 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 8-2](#) gives an overview of the different possibilities.

Figure 8-2. 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 a 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.

Datstores 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 volume. They don't require an in-depth knowledge of storage systems, and do enforce best practices.

Figure 8-3 gives an overview of the different APIs; Table 8-1 shows which tasks are commonly performed with which API.

Figure 8-3. Storage APIs

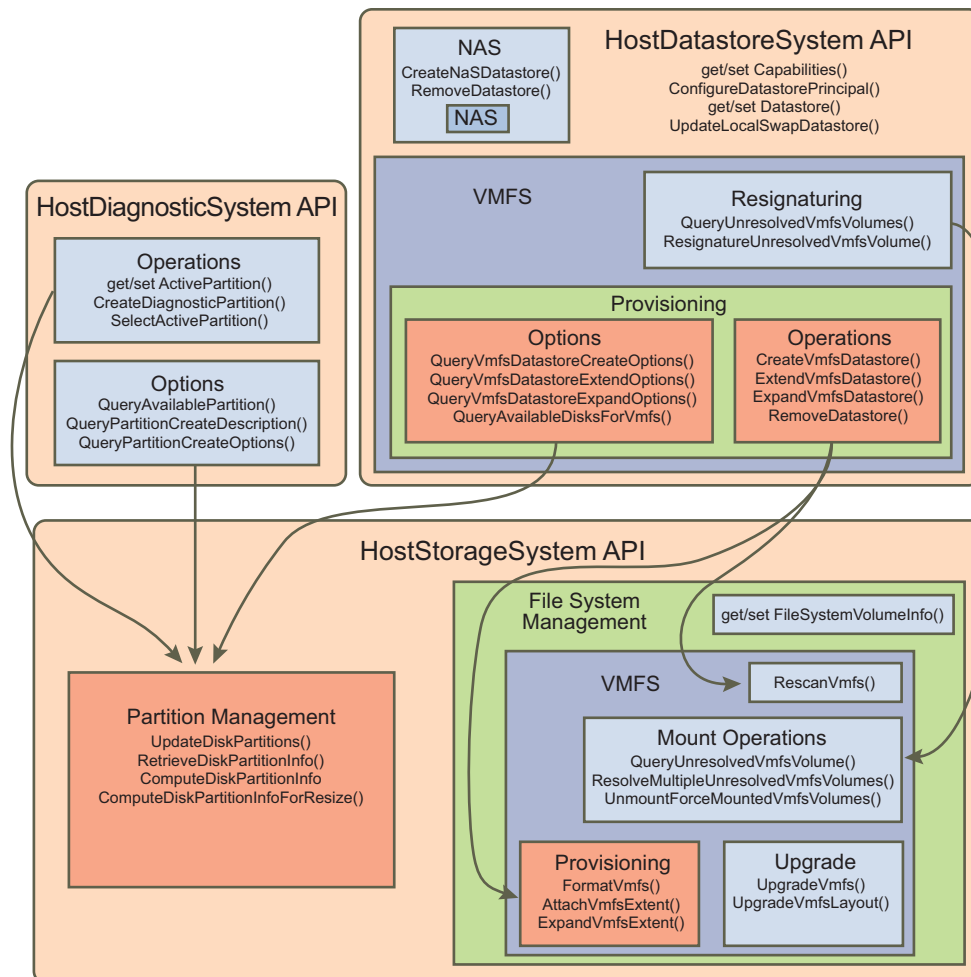


Table 8-1. Storage API Overview

Managed Object	Task	See
HostStorageSystem	Low-level operations associated with individual hosts, such as resizing or updating disk partitions.	“Configuring Disk Partitions” on page 75
HostStorageSystem	Multipath management.	“Multipath Management” on page 76
HostStorageSystem	iSCSI Storage setup and configuration.	“Configuring iSCSI Storage” on page 76
HostDatastoreSystem	Creating and managing VMFS datastores and remote datastores.	“Creating and Managing Datastores” on page 78
HostDatastoreSystem HostStorageSystem	Managing VMFS volume copies (resignature or force mount).	“Managing VMFS Volume Copies (Resignaturing)” on page 81
HostDiagnosticSystem	Creating and managing diagnostic partitions.	“Managing Diagnostic Partitions” on page 82

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. See [“Setting Up Disk Partitions”](#) on page 80.

- **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”](#) on page 76.

Multipath Management

The *ESXi Configuration Guide* and *ESX Configuration Guide* include an extensive discussion of 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 a 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 the *iSCSI Storage Configuration Guide* 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 EUJ or IQN format, as specified by the storage vendor's hardware.

Before you can set up iSCSI on a system, you must create a dedicated VMkernel network interface. See [“Adding a VMkernel Network Interface”](#) on page 89. You can then enable the VMkernel to support iSCSI and configure the initiator.

To enable the VMkernel to support software iSCSI

- 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`.

To configure iSCSI initiators

- 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 5, “Property Collector,”](#) on page 39. 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.UpdateInternetScsiName` and the alias by running `HostStorageSystem.UpdateInternetScsiAlias`.
- 6 Configure target discovery by calling `HostStorageSystem.UpdateInternetScsiHbaDiscoveryProperties`.

The method takes a `HostInternetScsiHbaDiscoveryProperties` data object that you can configure.

- 7 (Optional) Set the authentication information by calling `HostStorageSystem.UpdateInternetScsiAuthenticationProperties`.

The `HostInternetScsiHbaAuthenticationProperties` object you pass into that method includes properties for configuring CHAP and Mutual CHAP. See the *ESXi Configuration Guide* 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 objects 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.

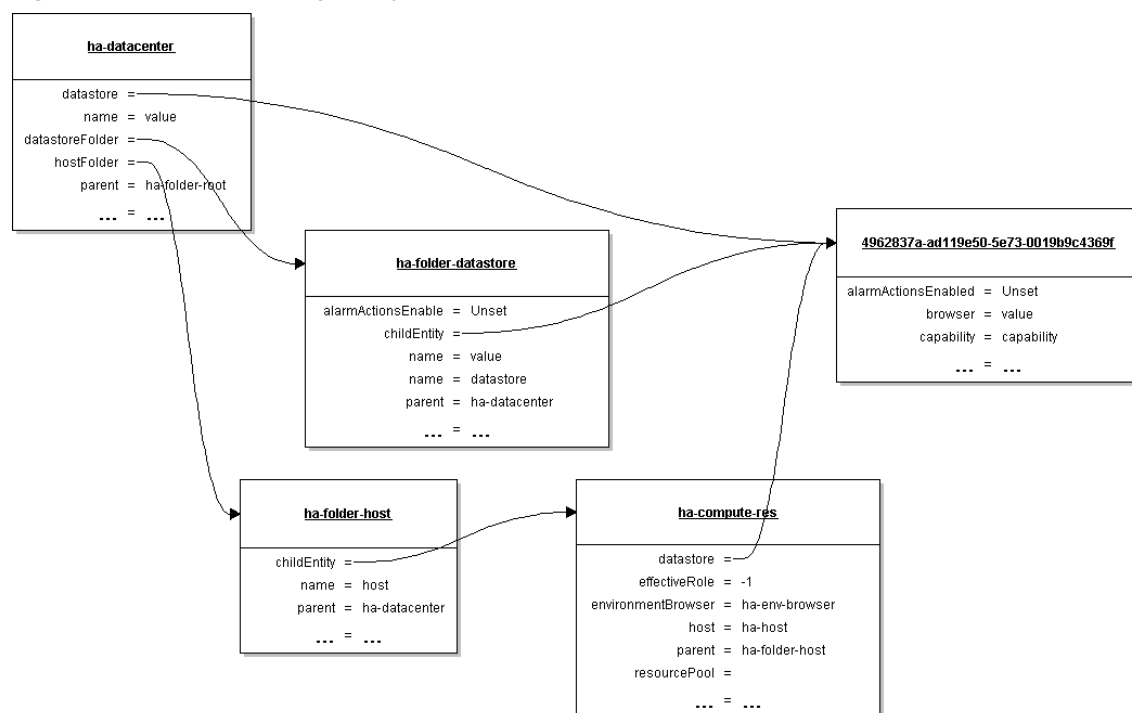
`QueryAvailableDisksForVmfs` obtains a list of disks that can be used to contain VMFS datastore extents. You can provide a datastore name to obtain the list of disks that can contain extents for the specified VMFS datastore. The operation does not return disks currently used by the VMFS datastore, nor 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

Figure 8-4 illustrates how you can access or specify datastores. See Chapter 4, “Datacenter Inventory,” on page 33 for more information about the hierarchy of managed objects.

- Each Datacenter managed object has `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 8-4. Datastore Managed Object



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. See Table 10-1, “Virtual Machine Files,” on page 98.

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.

Setting Up Disk Partitions

Setting up the disk partitions consists of these tasks:

- 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`.

Creating the VMFS Datastore

Creating the VMFS datastore consists of these tasks:

- 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 `HostScsiDisk` 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.

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

- `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

- `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 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 *ESXi Configuration Guide* for additional information.

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. See “[Generating Diagnostic Bundles](#)” on page 166. The VMware knowledge base article at <http://kb.vmware.com/kb/1004128> explains how to collect diagnostic partitions for a purple screen fault in ESXi.

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.

Creating 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

- 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.

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

[Table 8-2](#) lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 8-2. Sample Applications that Demonstrate Storage and Datastore Operations

Java (SDK\samples\Axis\java\com\vmware\samples\scsilun)	C# SDK\samples\DotNet\cs\GetVirtualDiskFiles
SCSILunName	SCSILunName.cs
GetVirtualDiskFilesV25	GetVirtualDiskFiles.cs

Networking

Before you add storage and virtual machines to an ESX/ESXi system, you should have completed networking setup. This chapter gives some background information, and then explains networking setup with vNetwork Standard Switch (vSS).

The chapter includes the following topics:

- [“Network Management Objects and Methods”](#) on page 85
- [“Introduction to Networking”](#) on page 86
- [“vNetwork Standard Switch Environment”](#) on page 86
- [“Setting Up Networking with vSS”](#) on page 88
- [“Defining the Host Network Policies”](#) on page 90
- [“NIC Teaming”](#) on page 90
- [“Setting Up IPv6 Networking”](#) on page 91
- [“Adding Networking Services”](#) on page 92
- [“Sample Code Reference”](#) on page 93

Network Management Objects and Methods

The vSphere Web Services SDK includes the following objects and methods for network configuration.

- **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”](#) on page 92.

Introduction to Networking

Network setup for your ESX/ESXi host 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. On ESX, virtual switches support service console networking. You set up virtual switches independently on each host. See [“Adding a Virtual Switch”](#) on page 88.

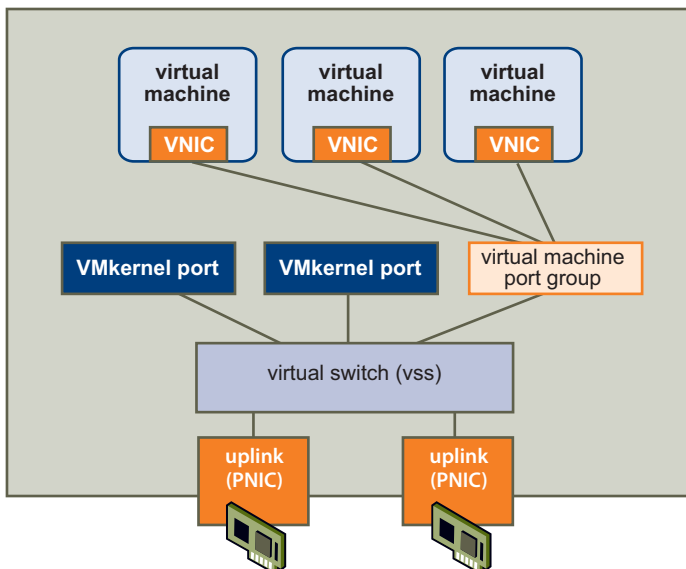
Starting with vSphere 4.0, you can use a vNetwork Standard Switch (vSS) or a vNetwork Distributed Switch (vDS). This document discusses primarily vSS.

- Adding virtual machine port groups. A virtual machine always accesses the network through a port group. See [“Adding a Virtual Port Group”](#) on page 89.
- 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 [Chapter 10, “Virtual Machine Configuration,”](#) on page 95.
- Adding VMkernel network interfaces, for example, to support iSCSI storage or VMotion. See [“Adding a VMkernel Network Interface”](#) on page 89.
- 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”](#) on page 92.

vNetwork Standard Switch Environment

A vNetwork Standard Switch (vSS) can route traffic internally between virtual machines and can link virtual machines to external networks. [Figure 9-1](#) shows the elements of a vSS environment.

Figure 9-1. 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).

The default number of logical ports for a vSwitch is 120 but it can have more ports. See the *Configuration Maximums* document available from the vSphere main documentation page. Each port can connect to a network adapter of a virtual machine, or an uplink adapter on the physical machine.

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.

Port Group

Port groups aggregate multiple ports under a common configuration. The *Configuration Maximums* document, available on the vSphere Documentation page, discusses the maximum number of ports.

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 Interface

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](#)” on page 102. 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 TCP/IP stack is completely separate from the TCP/IP stack used in the service console. Each of these TCP/IP stacks 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”](#) on page 89).

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 UI, you manipulate instead the uplink adapter. In a vSS environment, each pnict has one associated uplink adapter.

In a vDS environment, you use a DVS uplink instead of an uplink adapter.

Setting Up Networking with vSS

You can use the `HostNetworkSystem` managed object to access and manipulate the elements of an ESX/ESXi system’s 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 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 pnict for connection to the outside, and to a VMkernel port or a port group.

To add a virtual switch

- 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 90.
- 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 9-1. 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. For ESXi systems, the default port groups are `Management Network` and `VM Network`. For ESX systems, the default port groups are `Service console` and `VM Network`.

When you create a port group, you can specify a VLAN ID for it. VLANs are an important part of ESX/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

- 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 ESX/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

- 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.

[Example 9-2](#), 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 9-2. 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”](#) on page 90.
- `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.

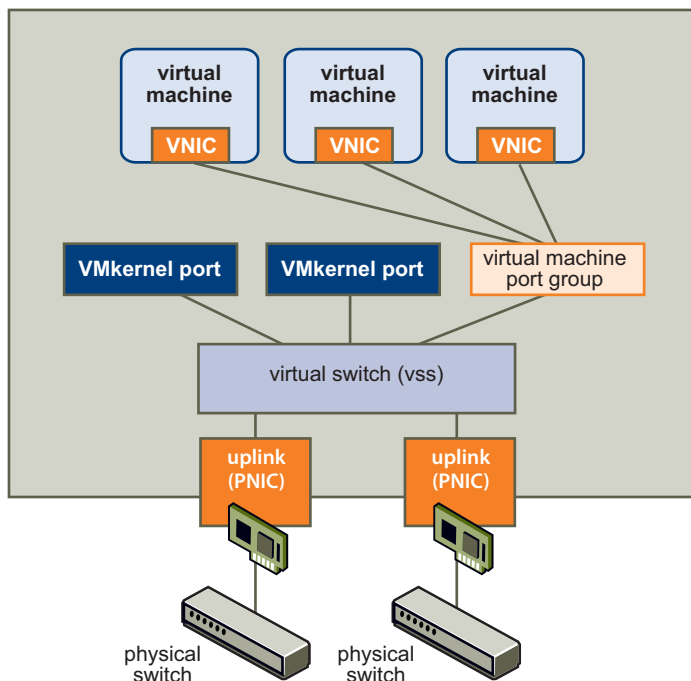
To resolve this issue, you can set up your environment so each virtual switch connects to two uplink adapters. Each uplink adapter connects to two different physical switches. The teams can then either share the load of traffic between physical and virtual networks among some or all of its members, or 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 9-2. 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 9-3. 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 ESX/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` object'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 ESX/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. The data object also allows you to specify the service console gateway device on ESX.

Setting Up SNMP

Simple Network Management Protocol (SNMP) allows management programs to monitor and control networked devices. vCenter Server and ESX/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.
- ESX/ESXi 4.0 and later includes an SNMP agent embedded in the ESX/ESXi 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 `HostSnmpSystem` managed object. This object includes two methods, `ReconfigureSnmpAgent` and `SendTestNotification`.

- `HostSnmpSystem.ReconfigureSnmpAgent` allows you to specify agent properties through a `HostSnmpConfigSpec`. That data object allows you to specify the SNMP port, read only communities, and the trap targets in an `HostSnmpDestination` object. The `HostSnmpDestination` object allows you to specify the community, and a host and port listening for notification.
- `HostSnmpSystem.SendTestNotification` allows you to test your configuration.

A `HostSnmpSystemAgentLimits` data object in the `HostSnmpSystem.limits` property specifies limits of the agent.

Sample Code Reference

Table 9-1 lists the sample applications included with the vSphere SDK that demonstrate using some of the managed objects discussed in this chapter.

Table 9-1. Networking Sample Applications

Java (SDK\samples\Axis\java\com\vmware\samples\host)	C# (SDK\samples\DotNet\cs\)
AddVirtualNic.java	AddVirtualNic
AddVirtualSwitch.java	AddVirtualSwitch
AddVirtualSwitchPortGroup.java	AddVirtualSwitchPortGroup
RemoveVirtualNic.java	RemoveVirtualNic
RemoveVirtualSwitch.java	RemoveVirtualSwitch
RemoveVirtualSwitchPortGroup.java	RemoveVirtualSwitchPortGroup

Virtual Machine Configuration

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.

The chapter includes the following topics:

- [“VirtualMachine Management Objects and Methods”](#) on page 95
- [“Creating Virtual Machines and Virtual Machine Templates”](#) on page 96
- [“Configuring a Virtual Machine”](#) on page 99
- [“Adding Devices to Virtual Machines”](#) on page 102
- [“Performing Virtual Machine Power Operations”](#) on page 104
- [“Registering and Unregistering Virtual Machines”](#) on page 104
- [“Customizing the Guest Operating System”](#) on page 105
- [“Installing VMware Tools”](#) on page 105
- [“Upgrading a Virtual Machine”](#) on page 106

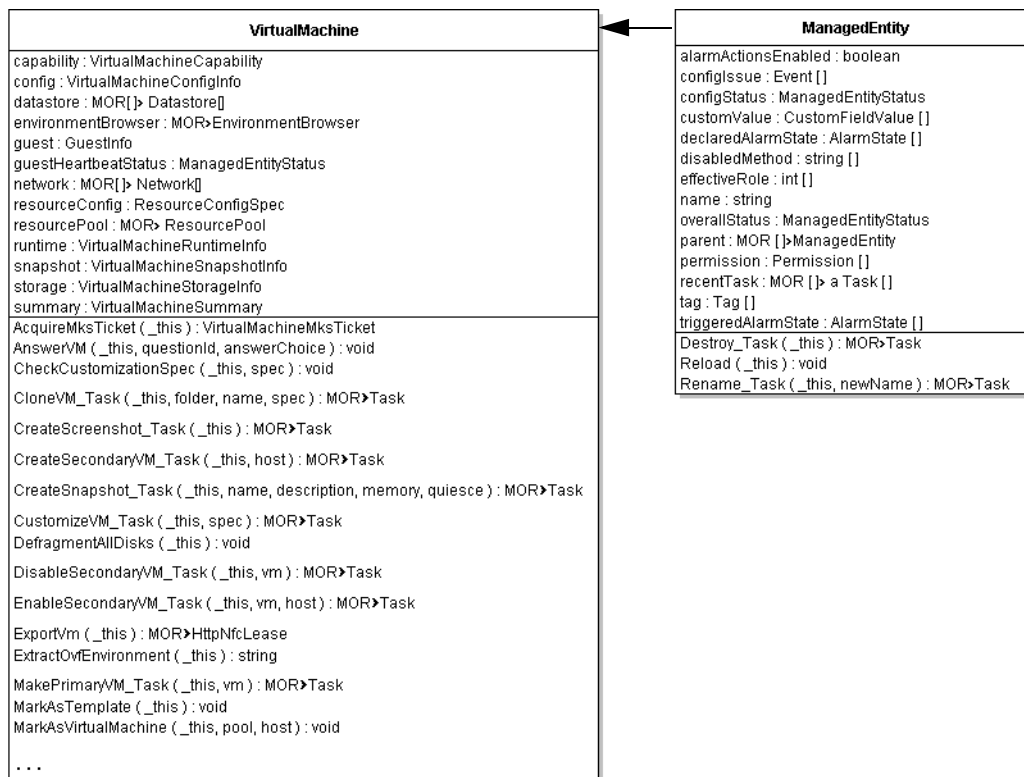
VirtualMachine Management Objects and Methods

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 10-1](#) 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 10-1. VirtualMachine Managed Object with Some Properties and Methods



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”](#) on page 97
- `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 10-1. 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”](#) on page 99 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 10-1, “Virtual Machine Files,”](#) on page 98.

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 10-2. 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 Virtual Machine

After you have created a virtual machine, you can retrieve information about the virtual machine through `VirtualMachineConfigInfo` properties. See the *API Reference* for a complete list.

After you have created a virtual machine, several files are generated and placed in the directory specified in the `VirtualMachineConfigSpec.files` property.

Table 10-1. Virtual Machine Files

File	Usage	File Description	File Format
<code>.vmx</code>	<code>vmname.vmx</code>	Virtual machine configuration file.	ASCII
<code>.vmxf</code>	<code>vmname.vmxf</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.	
<code>.log</code>	<code>vmware.log</code>	Virtual machine log file.	ASCII
<code>#.log</code>	<code>vmware-#.log</code>	Old virtual machine log files. # is a number starting with 1.	ASCII.

If you are using snapshots, the following additional files might be available. See “[Snapshots](#)” on page 108.

Table 10-2. Virtual Machine Snapshot Files

File Extension	Usage	File Description	File Format
.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 you virtual machine's maximum memory.	

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`. The API reference lists all properties and also includes information about required permissions. This section lists some attributes commonly specified and gives some background information.

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 13, "Resource Management,"](#) on page 121 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 ESX/ESXi host it is running on under the following conditions:

- You migrate a virtual machine created on an ESX/ESXi 3.x or earlier host to an ESX/ESXi 4.x host.
- You create a virtual machine on an ESX 4.x host using an existing virtual disk that was created on an ESX/ESXi 3.x or earlier host.
- You add a virtual disk created on an ESX/ESXi 3.x or earlier host to a virtual machine created on an ESX/ESXi 4.x host.

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

The `VirtualMachineConfigSpec` data object allows you to specify CPU and memory configuration.

CPU and Memory Resource Allocation

To allocate resources, use the `cpuAllocation` and `memoryAllocation` properties of `VirtualMachineConfigSpec`. Both properties contain `ResourceAllocationInfo` objects with the following properties:

- `reservation` – Amount of resources that is guaranteed to be available to the virtual machine. If resource utilization is less than the reservation, other running virtual machines can use the resources.
- `limit` – Upper limit for CPU or memory resources assigned to this virtual machine. The virtual machine does not exceed this limit, even if resources are available. This property is typically used to ensure consistent performance. If end users are used to work on a virtual machine that uses extra resources, and additional virtual machines are added to the host or resource pool, the virtual machine might slow down noticeably. If set to `-1`, no fixed upper limit on resource usage has been set.
- `shares` – Metric for allocating memory or processing capacity among multiple virtual machines. The `SharesInfo` data object has two properties, `level` and `shares`.
 - `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 the number of shares you want to allocate to the resource pool. The allocation is divided evenly between resource pools with the same level.

[“Resource Allocation”](#) on page 122 discusses resource allocation in the context of resource pool hierarchies. The *Resource Management Guide* includes a detailed discussion of resource allocation in the vSphere environment.

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.

Number of CPUs

You can set the number of virtual processors for the virtual machine with the `VirtualMachineConfigSpec.numCPUs` property. Legal values for this property change depending on the `guestosid` value you specify.

CPU Processors and Memory Affinity

If your virtual machine is on an ESX/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.

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. See [“Adding Devices to Virtual Machines”](#) on page 102.

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, ESX/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 ESX/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 ESX/ESXi host, and then specify a `VirtualDevice` object. Perform these tasks to add a device to a virtual machine:

- 1 Find out which devices your ESX/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 ESX/ESXi supports. For example, `VirtualMachineConfigOption.hardwareOptions` includes information about supported CPU and memory and an array of `VirtualDeviceOption` data objects.
- 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 vdcinfo = new VirtualDeviceConnectInfo();
// Allow the guest to control whether the virtual device is connected?
vdcinfo.setAllowGuestControl(false);
// Is the device currently connected?
vdcinfo.setConnected(true);
// Connect the device when the virtual machine starts?
vdcinfo.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.

Here's the complete code fragment for a CD-ROM passthrough device:

```
VirtualDevice vd = new VirtualDevice();
vd.setBacking(vcpbi);
vd.setConnectable(vdcinfo);
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”](#) on page 70.

`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 `powerOff` operation. If the power state is `poweredOff`, `ResetVM_Task` performs a `powerOn` operation.
- **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”](#) on page 105).

IMPORTANT Power operations might affect other virtual machines that are participating in a DRS cluster or VMware HA. See [Chapter 13, “Resource Management,”](#) on page 121 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 on the current host. Virtual machines managed by DRS to be placed by DRS are powered on on 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 vSphere 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. See “[Installing VMware Tools](#)” on page 105.

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

Virtual machines can act like physical computers and be configured like physical computer, as discussed in [Chapter 10, “Virtual Machine Configuration,”](#) on page 95. 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.

The chapter includes the following topics:

- [“Virtual Machine Migration”](#) on page 107
- [“Snapshots”](#) on page 108
- [“Linked Virtual Machines”](#) on page 110

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:

Table 11-1. vSphere 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. Cold migration can be used to migrate virtual machines from one datacenter to another.
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. You can migrate suspended virtual machines from one datacenter to another.
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 VMotion cannot be used to move virtual machines from one datacenter to another.
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 support the live migration of running virtual machines from one physical server to another with no downtime. You can perform this migration only between hosts in the same datacenter that are managed by a vCenter Server system. You must power off a virtual machine to move it to a different datacenter.

When you call the `VirtualMachine` object's `MigrateVM_Task` method, you can specify either a host or resource pool to migrate to. You can optionally specify the task priority and the power state of the virtual machine. The VMotion example performs the following tasks:

- Uses `QueryVMotionCompatibility_Task` to check two hosts are compatible.
- Uses `CheckMigrate_Task` to check whether migration is feasible. For example, 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.

The sample performs the migration if the hosts are compatible.

Using Storage VMotion

Storage VMotion allows you to move a running virtual machine from one VMFS volume to another. Taking the virtual machine or its associated storage offline is not required. All datastore types are supported, including local storage, VMFS, and NAS (network attached storage).

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.

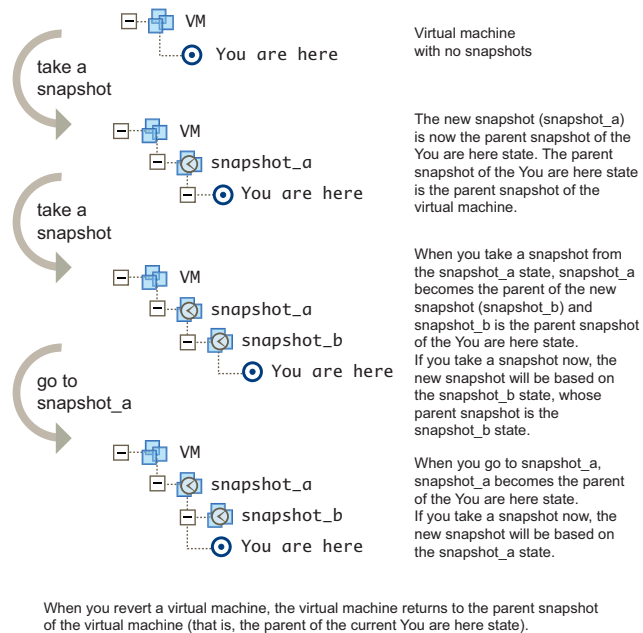
To perform storage VMotion, you use the `VirtualMachine.RelocateVM_Task` method. The `RelocateVMSpec` passed in to the method allows you to specify the target datastore and target host or resource pool.

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 11-1. Virtual Machine Snapshots

Snapshot hierarchies can become fairly complex. For example, assume that, in the example in [Figure 11-1](#), 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 `SuppressPowerOn` 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

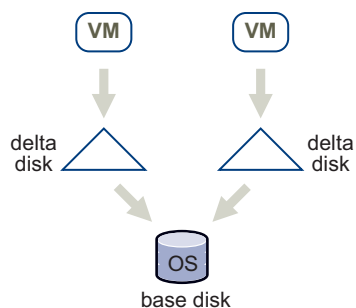
Linked virtual machines are two or more virtual machines that share storage and support efficient sharing of duplicated data.

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 them 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](#).

Figure 12. Linked Virtual Machines with Shared Base Disk Backing and Separate Delta Disk Backing



You can have up to eight virtual machines in a linked virtual machine group. The virtual machines in the group cannot be part of a VMware HA cluster.

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.

- 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 [Example 11-1](#). Creating linked virtual machines from a snapshot works with virtual machines in any power state.

Example 11-1. Creating a Linked Virtual Machine from a Snapshot

```

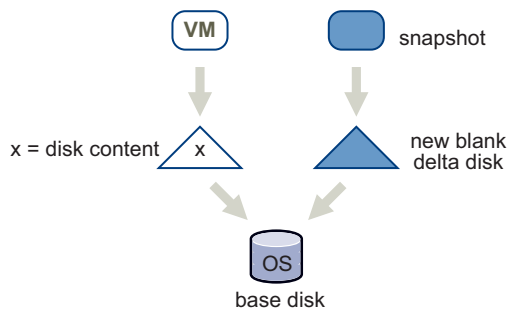
relSpec = new VirtualMachineRelocateSpec()
relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.createNewChildDiskBacking

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 13. 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 [Example 11-1](#), but use a `diskMoveType` of `moveChildMostDiskBacking`. The virtual machine can be in any power state.

Example 12. Creating a Linked Virtual Machine from the Current Running Point

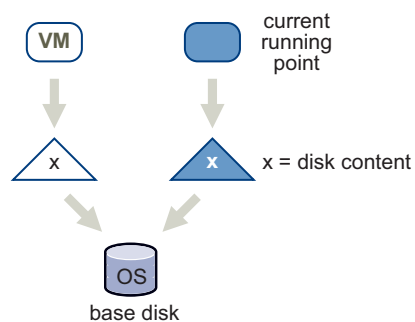
```

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 14. Creating a Linked Virtual Machine from the Current Running Point

Removing Snapshots and Deleting Linked Virtual Machine

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 ESX/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 ESX/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.

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, as shown in [Figure 13](#). 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.

Example 13. 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 [Example 14](#).

Example 14. 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`.

Both `add` and `create` allow you to create a blank delta disk on top of an existing disk. You can specify `add` or `create` in the `VirtualDeviceSpec` and pass in a virtual disk whose `parent` property is an existing disk. The methods create a new delta disk whose parent is the 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. [Example 15](#) illustrates how to add the delta disk for the first virtual disk in the virtual machine.

Example 15. Creating a Virtual Machine

```
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]
vm.Reconfigure(spec)
```

Virtual Applications

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”](#) on page 115
- [“Creating a VirtualApp”](#) on page 117
- [“Managing VirtualApp Children”](#) on page 117
- [“Exporting a Virtual Application”](#) on page 118
- [“Importing an OVF Package”](#) on page 119
- [“Virtual Application Life Cycle”](#) on page 119

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:

- 1 Call the `CreateVApp` method to create a virtual application without children. See [“Creating a VirtualApp”](#) on page 117.
- 2 Add child objects. See [“Managing VirtualApp Children”](#) on page 117.
- 3 Export the `VirtualApp` to OVF (`ExportVApp` method) See [“Exporting a Virtual Application”](#) on page 118.

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”](#) on page 117 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 `VirtualApp` 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. See [“Direct and Linked Children”](#) on page 116. 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.

Table 12-1. `VAppEntityConfigInfo` Properties

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.

Exporting a Virtual Application

You export a virtual application as an OVF template. To export the virtual application, all virtual machines in it must be powered off.

To export a virtual application, you follow steps in the following order:

- 1 “Acquiring the Export Lease” on page 118
- 2 “Downloading the Disks” on page 118
- 3 “Generating the OVF Descriptor” on page 118
- 4 “Completing the Lease” on page 118

Acquiring the Export Lease

Call the `VirtualApp.ExportVapp` method to obtain an export lease on the `VirtualApp` object. The export lease contains a list of URLs for the disks of the virtual machines in the `VirtualApp` object, as well as a ticket that gives access to these URLs.

The method returns a `HttpNfcLease` managed object reference. You can retrieve information about the state and, later, the content of the lease from this managed object reference.

NOTE Acquiring a lease requires time to complete. Check the acquisition state periodically (`HttpNfcLeaseState` type) to determine when you can continue with the process and download the disks.

While the lease is held, the virtual machines in the `VirtualApp` object are locked. Locked virtual machines cannot be powered on, deleted, or undergo any other state change that would disrupt the export process.

Downloading the Disks

When the lease is ready, the `HttpNfcLeaseInfo.deviceUrl` property is an array of `HttpNfcLeaseDeviceUrl` data object. Each object provides a mapping from logical device IDs to upload and download URLs. The URLs that are returned from the lease objects are used to directly post or get the virtual disks. The virtual disks must be stream-optimized VMDK format. This format is described in the virtual disk format specification available at (<http://www.vmware.com/technical-resources/interfaces/vmdk.html>).

While downloading each disk, you can create a filename for the resulting disk file. You should choose a name based on the corresponding key of the device URL of the lease (`HttpNfcLease.info.deviceURL.key`). OVF Descriptor generation requires the key.

The client must periodically call the `HttpNfcLeaseProgress` method on the export lease to prevent the lease from expiring.

You can call the `HttpNfcLeaseAbort` method to terminate the lease. Calling the method releases the lock on the virtual application.

Generating the OVF Descriptor

During the last step of the export process, you generate an OVF descriptor by calling the `OvfManger.createDescriptor` method. You must specify the name and size of each file in the download. Use the `CreateDescriptorParams.ovfFiles` array to describe the files. If this array is left empty, placeholders are inserted into the OVF descriptor and the client is responsible for updating the placeholders.

By creating the descriptor, you provide information about the disk files downloaded by the client, both the filenames chosen and the sizes observed, to the server. The server does have that information due to compression during download.

Completing the Lease

After generating the OVF descriptor, call `HttpNfcLease.HttpNfcLeaseComplete` to update task information and release the lock on the virtual application.

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.

- 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.

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 aborted.

- 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, aborting 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 (ESX/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

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.

The chapter includes the following topics:

- [“Resource Management Objects”](#) on page 121
- [“Introduction to Resource Management”](#) on page 122
- [“Resource Allocation”](#) on page 122
- [“Creating and Configuring Resource Pools”](#) on page 124
- [“Introduction to VMware DRS and VMware HA Clusters”](#) on page 126
- [“Creating and Configuring Clusters”](#) on page 127
- [“Managing DRS Clusters”](#) on page 128
- [“Managing HA Clusters”](#) on page 129

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.
- 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.
- The `ClusterComputeResource` data object aggregates the compute resources of multiple associated `HostSystem` objects into a single compute resource for use by virtual machines. 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 require licenses. If any clustering functionality does not work properly, check whether you have licenses for it.

Introduction to Resource Management

An ESX/ESXi host allocates each virtual machine a portion of the underlying hardware resources based on several factors:

- Total available resources for the ESX/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.
- Limits 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 [“CPU and Memory Resource Allocation”](#) on page 100.
- 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”](#) on page 122.
- 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”](#) on page 127.

Resource Allocation

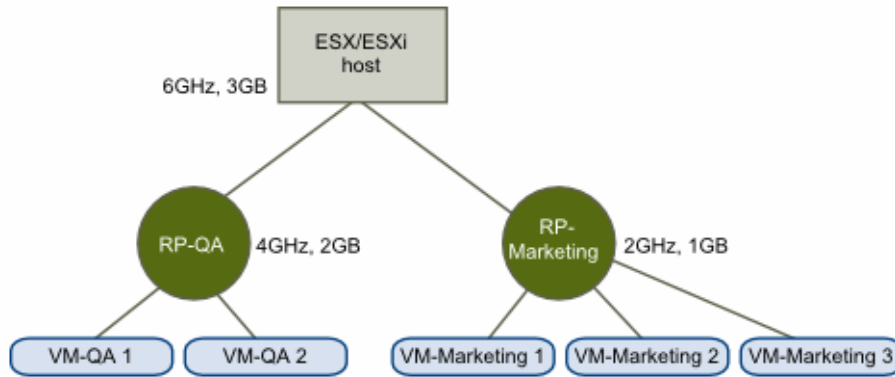
When you create a virtual machine, you always specify the resource pool that the virtual machine can draw resources from and optionally a host on which the virtual machine should run. You can access the resource pool as follows:

- **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 folder through the `ClusterComputeResource.resourcePool` property.

Resource Pool Hierarchies

Resource pool hierarchies allow detailed control over which virtual machines are allowed how many resources.

For example, assume a standalone host 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.

Figure 13-1. Allocating Resources to Resource Pools

Resource Pool Management Guidelines

The following rules govern resource pool creation.

- A root resource pool must always have at least as many resources as all its immediate children.
- Do not overcommit resource pool resources. The sum of all child pools should always be less than (not equal to or more than) the parent. For example, if four child resource pool reservations total 40 gigabytes and the parent resource pool has a reservation of 60 gigabytes, you have some room to create another resource pool. However, if the four child resource pool reservations total 60 gigabytes, you do not. For virtual machine, some overcommitment of resources is supported. See the technical white papers on the VMware web site.
- Before creating new child resource pools, check available resources in the parent pool. The `ResourcePool.runtimeInfo` property is a `ResourcePoolRuntimeInfo` data object. `ResourcePoolRuntimeInfo.cpu` and `ResourcePoolRuntimeInfo.memory` properties are `ResourcePoolResourceUsage` objects with resource usage information, including an `unreservedForPool` property. If the parent resource pool does not have enough available resources, reconfigure the reservation values of child pools before adding the new pool.
- Reconfigure child resource pools first, to ensure that the reservation properties of each child do not absorb all the resources of the parent.

Cluster Overview

vSphere supports grouping ESX/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) migrates virtual machine 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 (DTM), 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 datacenter.

See [“Creating and Configuring Clusters”](#) on page 127 and [“Managing DRS Clusters”](#) on page 128.

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 passing in a `ResourceConfig` method as an argument. The `ResourceConfig.cpuAllocation` and `ResourceConfig.memoryAllocation` properties point to a `ResourceAllocationInfo` object that allows you to specify the information.

- `reservation` – Amount of CPU or memory that is guaranteed available to 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 utilized by other resource pools or running virtual machines.
- `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 Reservation”](#) on page 124. This property is ignored for virtual machines.
- `limit` – Upper limit for CPU or memory resources assigned to this resource pool. The virtual machine or resource pool does not exceed this limit, even if resources are available. This property is typically used to ensure consistent performance. 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. 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 the number of shares you want to allocate to the resource pool. The allocation is divided evenly between resource pools with the same level.

Calling the `ResourcePool.UpdateConfig` or `ResourcePool.UpdateChildResourceConfiguration` method allows you to change the configuration.

Understanding Expandable Reservation

Expandable reservations are best illustrated with examples.

Expandable Reservation Example 1

Assume an administrator manages pool P, and defines two child resource pools, S1 and S2, for two different users (or groups).

The administrator knows that users want to power on virtual machines with reservations, but does not know how much each user will reserve. Making the reservations for S1 and S2 expandable allows the administrator to more flexibly share and inherit the common reservation for pool P.

Without expandable reservations, the administrator needs to explicitly allocate S1 and S2 a specific amount. Such specific allocations can be inflexible, especially in deep resource pool hierarchies and can complicate setting reservations in the resource pool hierarchy.

Expandable reservations cause a loss of strict isolation; that is, S1 can start using all of P's reservation, so that no memory or CPU is directly available to S2.

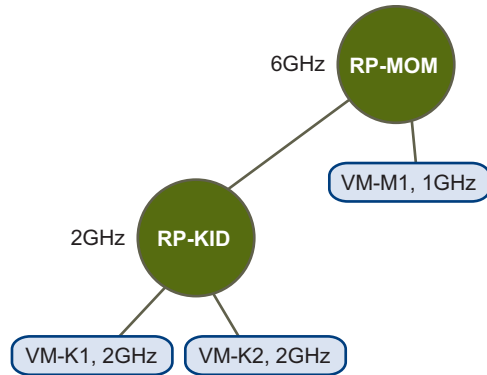
Expandable Reservation Example 2

Assume the following scenario (shown in [Figure 13-2](#)):

- Parent pool RP-MOM has a reservation of 6GHz and one running virtual machine VM-M1 that reserves 1GHz.
- You create a child resource pool RP-KID with a reservation of 2GHz and with **Expandable Reservation** selected.

- You add two virtual machines, VM-K1 and VM-K2, with reservations of 2GHz each to the child resource pool and try to power them on.
- VM-K1 can reserve the resources directly from RP-KID (which has 2GHz).
- No local resources are available for VM-K2, so it borrows resources from the parent resource pool, RP-MOM. RP-MOM has 6GHz minus 1GHz (reserved by the virtual machine) minus 2GHz (reserved by RP-KID), which leaves 3GHz unreserved. With 3GHz available, you can power on the 2GHz virtual machine.

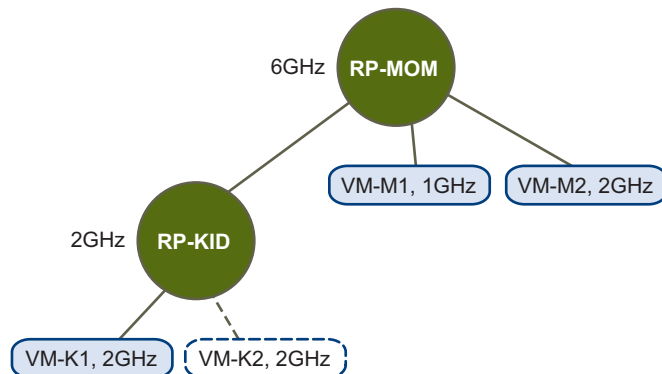
Figure 13-2. Admission Control with Expandable Resource Pools, Scenario 1



Now, consider another scenario with VM-M1 and VM-M2, shown in [Figure 13-3](#).

- Power on two virtual machines in RP-MOM with a total reservation of 3GHz.
- You can still power on VM-K1 in RP-KID because 2GHz are available locally.
- When you try to power on VM-K2, RP-KID has no unreserved CPU capacity so it checks its parent. RP-MOM has only 1GHz of unreserved capacity available (5GHz of RP-MOM are already in use—3GHz reserved by the local virtual machines and 2GHz reserved by RP-KID). As a result, you cannot power on VM-K2, which requires a 2GHz reservation.

Figure 13-3. Admission Control with Expandable Resource Pools, Scenario 2



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 VMware DRS and VMware HA Clusters

Clusters are useful primarily with VMware DRS and VMware HA. This guide only gives a brief introduction. See the following manuals for details:

- **VMware DRS.** *vSphere Resource Management Guide*
- **VMware HA.** *vSphere Availability Guide*

VMware DRS

A VMware DRS cluster is a collection of ESX/ESXi hosts and associated virtual machines with shared resources and a shared management interface. Before you can obtain the benefits of cluster-level resource management you must 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 this 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 system monitors distribution and usage 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 imbalance target. 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 making a recommendation.
- **Power Management.** When the VMware DTM (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. DTM places (or recommends placing) hosts in standby power mode if sufficient excess capacity is found. DTM 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”](#) on page 128.

VMware HA

VMware HA supports high availability for virtual machines by pooling them and the hosts they reside on into a cluster. VMware HA monitors the hosts. In the event of host failure, VMware HA 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 on a different host in case of host failure is available.

See [“Managing HA Clusters”](#) on page 129.

Creating and Configuring Clusters

The vSphere Web Services SDK includes objects and methods for all cluster management tasks. Some documentation is available in the *API Reference*. For additional background and details about the failover and load balancing behavior, see the *Resource Management Guide* and the *High Availability Guide*.

Creating a Cluster

If your environment includes a vCenter Server system and multiple ESX/ESXi hosts, you can create a cluster by calling the `Folder.CreateCluster` method. You pass in a name for the new cluster and a `ClusterConfigSpec` data object. In the data object, you can specify the following properties:

- VMware HA
 - `dasConfig` – `ClusterDasConfigInfo` data object that specifies the HA service on the cluster. Properties on 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 *API Reference*.
 - `dasVMConfigSpec` – `ClusterDasVMConfigSpec` object. `ClusterDasVMConfigSpec.info` 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.
- VMware DRS
 - `drsConfig` – `ClusterDrsConfigInfo` data object that 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` – `ClusterDrsVMConfigSpec` data object that points to a `ClusterDrsVmConfigInfo` data object which 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.

When you 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` – `ClusterDrsRuleSpec` data object that points a `ClusterRuleInfo` data object which specifies the affinity and antiaffinity rules DRS should use. See the *API Reference* entry for `ClusterRuleInfo`.

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 a cluster. The host name must be either an IP address, such as 192.168.0.1, or a DNS resolvable name. If the cluster supports nested resource pools and you specify the optional `resourcePool` argument, the host's root resource pool becomes the specified resource pool, and that resource pool and the associated hierarchy is added to the cluster.

If a cluster does not support nested resource pools and you add a host to the cluster, the standalone host resource pool hierarchy is discarded and all virtual machines on the host are added to the cluster's root resource pool.

- `ClusterComputeResource.moveHostInto_Task` moves a host that is in the same datacenter as the cluster into the cluster. If the host is already part of a different cluster, the host must be in maintenance mode.
- `ClusterComputeResource.moveTo_Task` works like `moveHostInto_Task`, but supports an array of hosts at a time. When using this method, you cannot preserve the original resource pool hierarchy of the hosts.

Reconfiguring a Cluster

You can reconfigure a cluster by calling the `ClusterComputeResource.ReconfigureCluster_Task` method. The method allows you to enable or disable VMware DRS or VMware HA and to define attributes. See “[Creating a Cluster](#)” on page 127.

To remove hosts from a cluster, you can use one of the following methods:

- `ClusterComputeResource.MoveHostInto_Task` or `MoveInto_Task`—Removes a host from a cluster and moves the host into another cluster. See “[Adding a Host to a Cluster](#)” on page 127.
- `Folder.MoveIntoFolder_Task`—Removes a host from a cluster and make it a standalone host.
- `Host.Destroy_Task`—Removes a host from inventory.

Managing DRS Clusters

The vSphere Client UI allows you to explore DRS cluster behavior, which is also described in the *Resource Management Guide*. When DRS is running, it generates recommendations and associated information that result in a well balanced cluster.

- 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 are considered or migrations with multiple priority levels.
- Checks whether DRS clusters are valid — enough resources are available to start additional virtual machines — or not valid.

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 one of the methods for moving hosts into a cluster. See [“Adding a Host to a Cluster”](#) on page 127. You might have to call `HostSystem.ReconfigureHostForDAS_Task` to reconfigure the host for HA if the automatic HA configuration fails.

Primary and Secondary Hosts

You can add a 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 and attempts to fail over the isolated host's virtual machines fail. By default, the isolated host leaves its virtual machines powered on, but you can change the host isolation response.

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 are comparatively lightly loaded. VMware HA uses the CPU and memory reservation to determine failover, while 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.

Tasks and Scheduled Tasks

VMware vSphere uses an asynchronous client-server communication model. Most methods are non-blocking, returning a reference to a `Task` managed object. You can use `Task` and `TaskManager` 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.

The chapter includes the following topics:

- [“Task Management Objects”](#) on page 131
- [“Accessing and Manipulating Tasks”](#) on page 132
- [“Examining Recent Tasks with TaskManager”](#) on page 133
- [“Using TaskInfo to Monitor Task Information”](#) on page 133
- [“Understanding ScheduledTaskManager”](#) on page 135
- [“Scheduling Tasks”](#) on page 136
- [“Using a TaskHistoryCollector”](#) on page 139
- [“Sample Code Reference”](#) on page 140

Task Management Objects

You can use `Task` and `TaskInfo` objects to monitor tasks. Each time a server runs a method, it creates a corresponding task. Some methods return tasks explicitly. You can access information about those task through the `TaskInfo` data object directly. For the majority of tasks, the `TaskManager` managed object is the interface for accessing `Task` managed objects and the associated `TaskInfo` object.

- **Scope.** A `Task` and its associated objects are session specific. A user associated with a session can only obtain information about the `Task` objects that user is authorized to view.
- **Persistence.** For standalone ESX/ESXi systems, task objects are non-persistent. All completed `Task` objects age out of the `recentTasks` array after 10 minutes.

For managed ESX/ESXi systems, the vCenter Server collects information about `Task` objects for all the hosts that it manages. You can obtain information about `Task` objects from vCenter Server using a `TaskHistoryCollector`. See [“Using a TaskHistoryCollector”](#) on page 139.

- **Asynchronous and Synchronous Operations.** Many API operations are asynchronous and return a reference to a `Task` object, but some operations are synchronous and do not. For example, operations available through the `PerformanceManager` and several other service interfaces do not return `Task` objects. Operations that include the suffix `_Task` in their names return `Task` references.

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.

Applications working with ESX/ESXi systems that are managed by a vCenter Server system can use `ScheduledTaskManager` to schedule tasks. `ScheduledTaskManager` allows you to specify the time at which the task should be happen, and the action to be performed.

Accessing and Manipulating Tasks

`TaskManager` is the service interface for accessing and manipulating Task objects. The following `TaskManager` properties are commonly used by client applications.

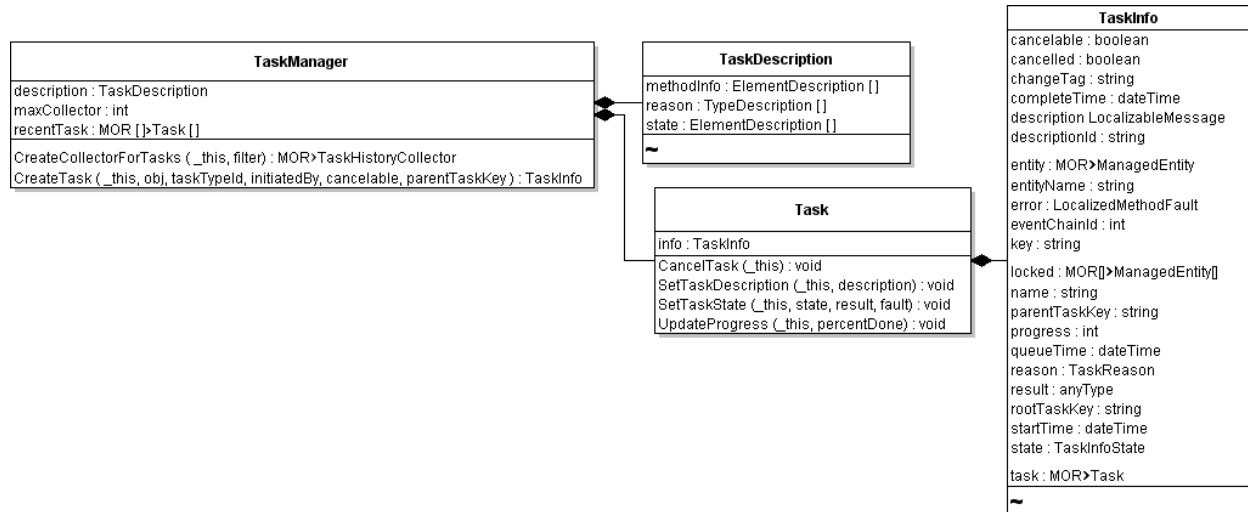
- `description` – `TaskDescription` object that includes a `methodInfo` property. `methodInfo` contains a key-based array that `TaskManager` uses to populate the value of a `TaskInfo` data object's `descriptionId` property with the name of the operation. Examples of two elements from this key-based array 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 vCenter Server systems, 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 managed object reference to a 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”](#) on page 139.

Figure 14-1 shows a UML class diagram for `TaskManager` and associated objects.

Figure 14-1. TaskManager and Task Managed Objects



Examining Recent Tasks with TaskManager

For standalone ESX/ESXi systems, a reference to a Task is available in the `recentTask` array of `TaskManager` from the moment a Task is queued until 10 minutes after it completes. You can use the `recentTask` property to obtain information about the Task objects running on the system.

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`. [Example 14-1](#) 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 5, “Property Collector,”](#) on page 39.

Example 14-1. 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 ESX/ESXi systems managed by a vCenter Server system, use a `TaskHistoryCollector`. See [“Using a TaskHistoryCollector”](#) on page 139.

Using TaskInfo to Monitor Task Information

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`.

[Table 14-1](#) 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.

Table 14-1. Sample TaskInfo Values

Property	Datatype	Start of Task Sample Values	End of Task Sample Values
cancelable	boolean	false	false
...			
completeTime	dateTime	Unset	"2009-02-19T22:53:35.015338Z"
progress	int	36	100
queueTime	dateTime	"2009-02-19T22:50:39.111604Z"	"2009-02-19T22:50:39.111604Z"
reason	TaskReason	reason	reason
result	anyType	Unset	64
....			
state	TaskInfoState	"running"	"success"

Monitoring TaskInfo Properties

To monitor the state of a Task, use the `PropertyCollector.WaitForUpdatesEx` method. See [“Client Data Synchronization \(WaitForUpdatesEx\)”](#) on page 50. 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.

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.

[Example 14-2](#) shows a code fragment that obtains values for the `info` property from each Task object in the array.

Example 14-2. 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";
        }
    }
}...

```

[Example 14-3](#) 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 14-3. 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

```

Canceling 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.

Understanding ScheduledTaskManager

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

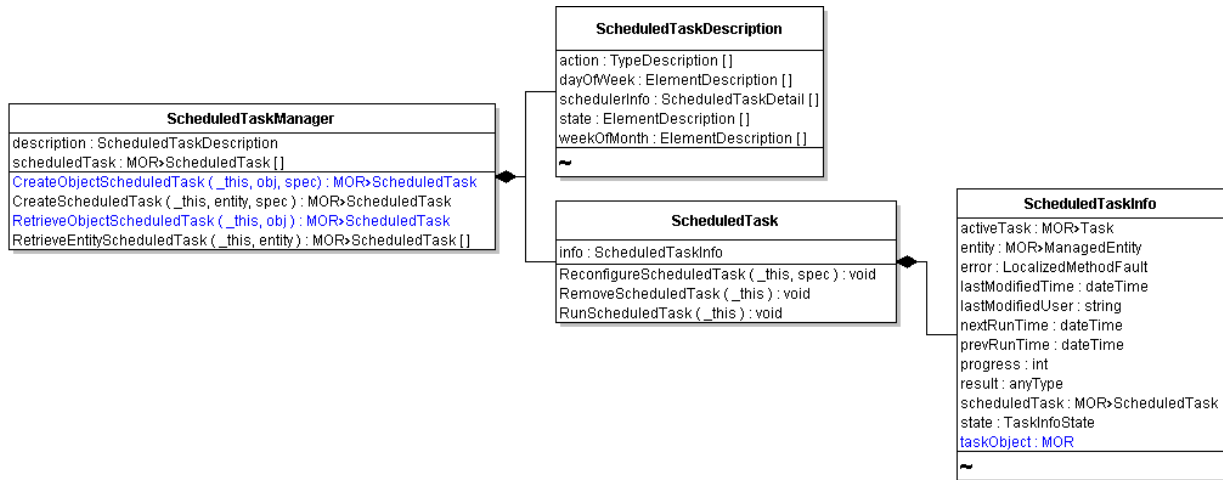
You can schedule scripts to be run or methods to be invoked on the server. You apply the action to an entity in the inventory, such as a virtual machine or a host.

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”](#) on page 136.

[Figure 14-2](#) shows the `ScheduledTaskManager` service interface and associated data objects.

Figure 14-2. 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.

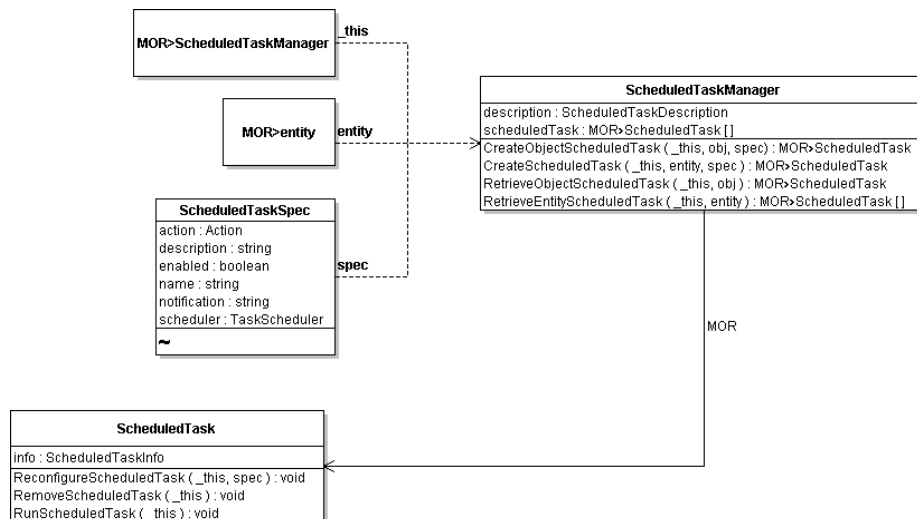
If the action specified for a `ScheduledTask` creates its own `Task` (such as with any of the asynchronous operations), the managed object reference to the `Task` populates the `activeTask` property of `ScheduledTaskInfo`.

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 descendents 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 14-3. 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](#)” on page 148.
- **notification** – Specifies the email address for sending notification messages about the `ScheduledTask`. To use notifications, the vCenter Server system must have an SNMP 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](#)” on page 137.

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.

[Table 14-2](#) provides some usage information about the `TaskScheduler` subtypes. The examples in the table are Java code fragments.

Table 14-2. TaskScheduler Data Object Subtypes

TaskScheduler subtype	Usage
<code>AfterStartupTaskScheduler</code>	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. Example: Schedule a task to run 10 minutes after vCenter Server startup. <pre>AfterStartupTaskScheduler asts = new AfterStartupTaskScheduler(); asts.setMinute(10);</pre>
<code>OnceTaskScheduler</code>	Schedule an action to run once only at the specified date and time. Example: Schedule a task to run 30 minutes after the schedule is submitted to the server. <pre>Calendar runTime = Calendar.getInstance(); runtime.add(Calendar.MINUTE, 30); OnceTaskScheduler ots = new OnceTaskScheduler (); ots.setRunAt(runTime);</pre>
<code>RecurrentTaskScheduler</code>	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.
<code>HourlyTaskScheduler</code>	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. Example: Schedule a task to run every 4 hours at half-past the hour. <pre>HourlyTaskScheduler hts = new HourlyTaskScheduler(); hts.setMinute(30); hts.setInterval(4);</pre>

Table 14-2. TaskScheduler Data Object Subtypes (Continued)

TaskScheduler subtype	Usage
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> <hr/> <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 true. 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> <hr/> <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> <hr/> <pre>MonthlyByDayTaskScheduler mbdts = new MonthlyByDayTaskScheduler(); mbdts.setDay(31); mbdts.setInterval(3); mbdts.setMinute(30); mbdts.setHour(14);</pre>
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> <hr/> <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 code fragment in [Example 14-4](#) 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 14-4. 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);
...

```

Canceling 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 executed 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 with a maximum of 1000 objects of the appropriate type (`Task`, `Event`). The array is sorted by creation date and time of the objects.

Sample Code Reference

[Table 14-3](#) lists the sample applications included with the vSphere Web Services SDK that demonstrate using tasks. In addition to the samples listed in the table, the helper classes associated with the sample applications use `Task` objects extensively.

Table 14-3. Task and ScheduledTask Sample Applications

Java	C#
<code>samples\general\TaskList.java</code>	<code>TaskList</code>
<code>samples\scheduling\DeleteOneTimeScheduledTask.java</code>	<code>DeleteOneTimeScheduledTask</code>
<code>samples\scheduling\OneTimeScheduledTask.java</code>	<code>OneTimeScheduledTask</code>
<code>samples\scheduling\WeeklyRecurrenceScheduledTask.java</code>	<code>WeeklyRecurrenceScheduledTask</code>

Events and Alarms

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.

The chapter includes the following topics:

- [“Event and Alarm Management Objects”](#) on page 141
- [“Understanding Events”](#) on page 141
- [“Using an EventHistoryCollector”](#) on page 144
- [“Using Alarms”](#) on page 145
- [“Defining Alarms Using the AlarmSpec Data Object”](#) on page 146
- [“Sample Code Reference”](#) on page 149

Event and Alarm Management Objects

`EventManager` is the service interface for working with the event infrastructure. See [“Managing Events with EventManager”](#) on page 142.

`Event` subtypes define the events that the system generates. See [“Event Data Objects”](#) on page 143 and [“Creating Custom Events”](#) on page 144.

`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”](#) on page 144.

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 ESX/ESXi system to a vCenter Server system

In the vSphere Client, information from Event objects generated on a standalone ESX/ESXi system 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 ESX/ESXi systems** – 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 ESX/ESXi host or powering off a virtual machine removes Event objects from local memory.

A standalone ESX/ESXi system 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 ESX/ESXi systems.** Event objects are persistent. Managed ESX/ESXi systems send Event data to the vCenter Server system that manages them, and the vCenter Server system stores the information in its database.

You can use the event sample applications included in the SDK package with either managed or standalone ESX/ESXi systems and with vCenter Server systems.

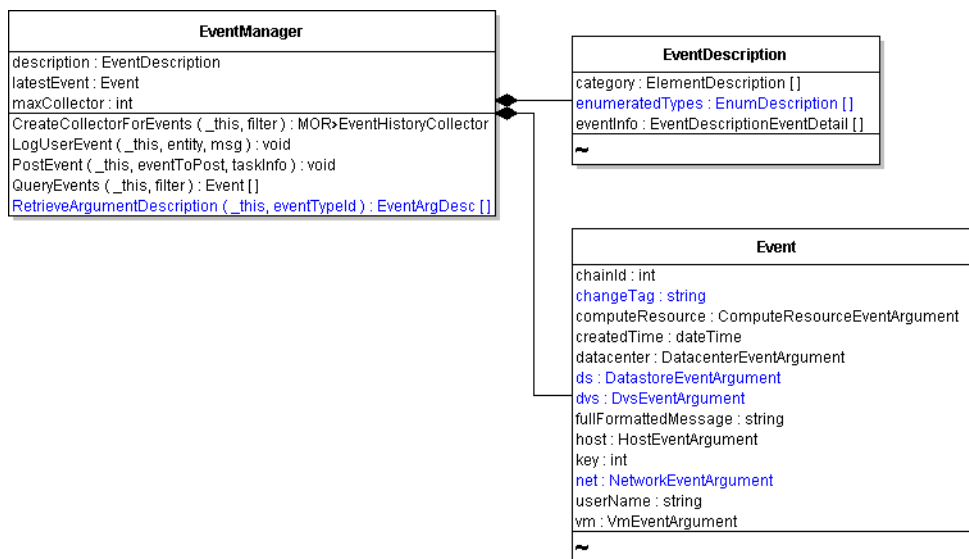
Using an `EventHistoryCollector`, you can obtain information about these objects as they are being collected on a specific ESX/ESXi system, or from a specific historical period from the database. See [“Using an EventHistoryCollector”](#) on page 144.

Managing Events with EventManager

`EventManager` is the service interface for working with the event infrastructure. [Figure 15-1](#) 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 15-1. EventManager Managed Object and Associated Objects



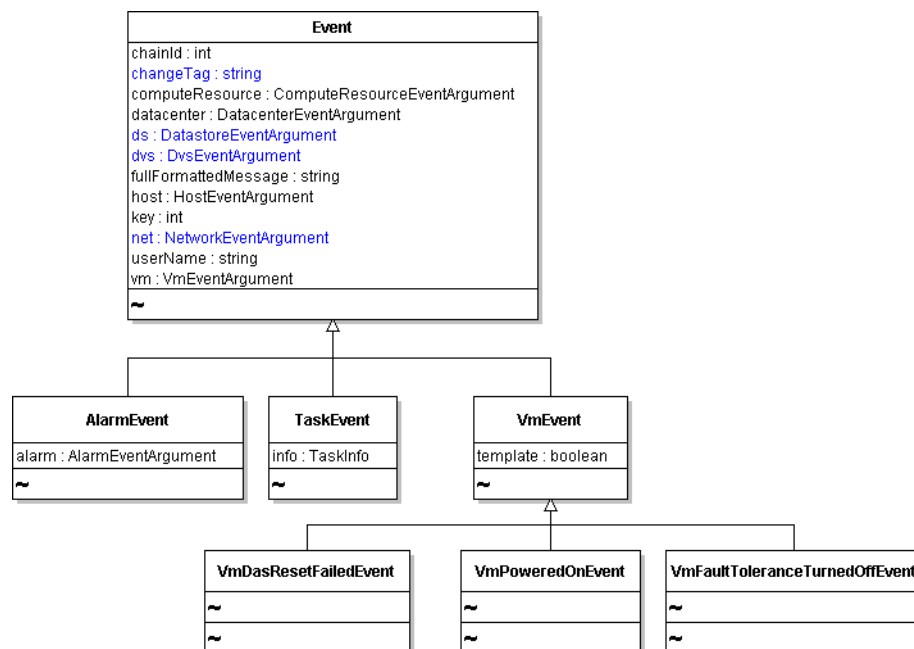
Event Data Objects

Event subtypes define the events that the system generates. [Figure 15-2](#) 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 [“Using TaskInfo to Monitor Task Information”](#) on page 133).

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 15-2. 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 `EventManager.LogUserEvent` method allows you to create custom `Event` objects. You can associate your custom `Event` with any managed entity.

To define a custom Event

- 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`.
- 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.

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.

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.

To create an EventHistoryCollector

- 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”](#) on page 144.

The `EventFilterSpec` includes an `eventTypeid` 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.

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 executed 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.

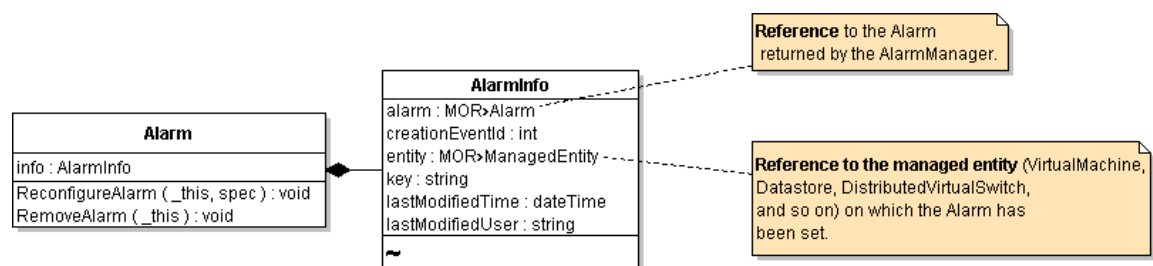
The alarm infrastructure integrates with other server components, such as events and performance counters.

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 15-3. Alarm Managed Object

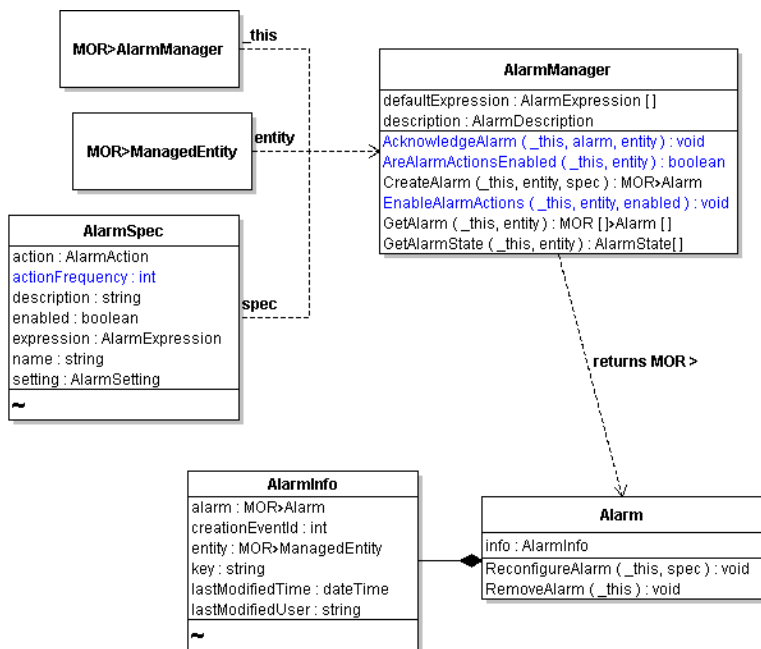


The Alarm.info property is an AlarmInfo data object. You can obtain information about active Alarms by collecting the properties of the AlarmInfo data object.

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 15-4 shows the CreateAlarm method.

Figure 15-4. CreateAlarm Method Inputs and Outputs**To create an alarm**

- 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”](#) on page 146.
- 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 15-4](#)).

The state of an alarm is contained in an `AlarmState` data object.

Defining Alarms Using the AlarmSpec Data Object

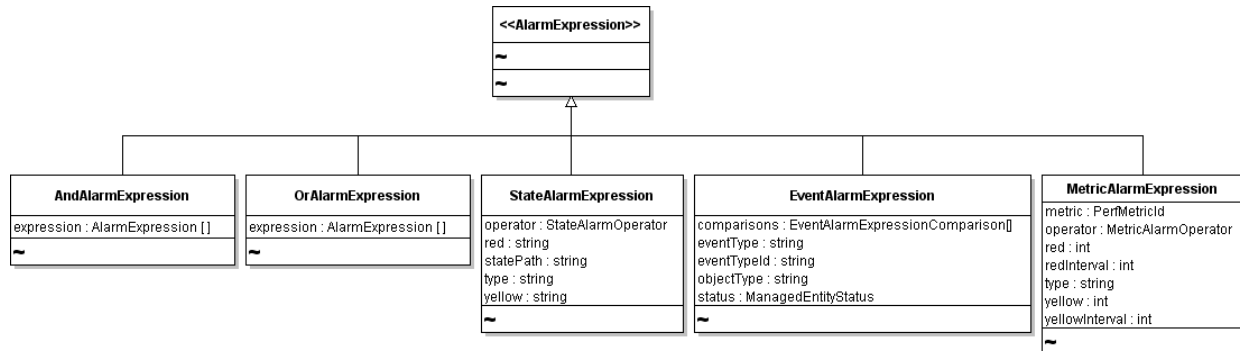
The `AlarmSpec` data object has properties for all aspects of an `Alarm`, including its expression and the action to take when the expression evaluates to true. The following properties define the alarm; see the *API Reference* for a complete list.

- `action` – Action to initiate when the `Alarm` becomes active. Specify one of the `Action` subtypes. See [“Specifying Alarm Actions”](#) on page 148.
- `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”](#) on page 147.
- `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 15-5. AlarmExpression and Its Subtypes



AlarmExpression Types

By using the appropriate type of `AlarmExpression`, you can set alarms for different conditions, states, or events.

Table 15-1. Alarm Expressions and Examples

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” on page 147.	Triggered when resource utilization metrics exceed a specified limit.
<code>EventAlarmExpression</code>	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.
<code>EventAlarmComparison</code>	Specifies the property of the Event that should trigger the alarm and the operator to use as the basis for comparison.	
<code>AndAlarmExpression</code> <code>OrAlarmExpression</code>	Combines one or more instances of the <code>AndAlarmExpression</code> and the <code>OrAlarmExpression</code> data objects into an expression that evaluates to <code>true</code> or <code>false</code> .	

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 15-6. 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

[Table 15-2](#) lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 15-2. Sample Applications that Use Alarm

Java	C#
<code>\samples\alarms\MPowerStateAlarm.java</code>	<code>VMPowerStateAlarm</code>
<code>\samples\events\EventFormat.java</code>	<code>EventFormat</code>
<code>\samples\events\EventHistoryCollectorMonitor.java</code>	<code>EventHistoryCollectorMonitor</code>
<code>\samples\events\VMEventHistoryCollectorMonitor.java</code>	<code>VMEventHistoryCollectorMonitor</code>

Performance

VMware vSphere servers are instrumented with performance counters that track resource use. At runtime, vSphere components generate performance data which you can retrieve using the `PerformanceManager` interface.

IMPORTANT Some data objects, including `HostListSummaryQuickStats`, `VirtualMachineQuickStats`, and `ResourcePoolQuickStats`, provide near real-time summary information about performance or utilization without going through the `PerformanceManager`.

This chapter includes the following topics:

- [“PerformanceManager Objects and Methods”](#) on page 151
- [“Performance Counters”](#) on page 152
- [“Sampling Periods and Intervals”](#) on page 153
- [“Obtaining Statistics”](#) on page 154
- [“vCenter Server and PerformanceManager”](#) on page 155
- [“Sample Code Reference”](#) on page 157

PerformanceManager Objects and Methods

`PerformanceManager` is the service interface for obtaining statistical data about various aspects of system performance, as generated and maintained by the system’s performance providers. A performance provider is any managed object that generates performance metrics. Performance providers include managed entities, such as hosts, virtual machines, compute resources, resource pools, datastores, and networks. Performance providers also include physical or virtual devices associated with these objects, such as virtual HBAs and network-interface controllers (NICs).

`PerformanceManager` allows you to access information about predefined performance objects on the system in its properties:

- `description` – Composite object that includes information about the types of counters (`counterType`) and statistics (`statsType`) available on the system.
- `historicalInterval` – Array of `PerfInterval` data objects that defines the performance intervals configured for the system.
 - For an ESX/ESXi system, the array contains a single object that defines the baseline sampling period for the system.
 - For vCenter Server systems, the `PerfInterval` objects control how data collected from the ESX/ESXi systems are aggregated and stored in the database.
- `perfCounter` – Array of `PerfCounterInfo` data objects that contains all the metadata associated with the performance metrics generated on the system.

The `PerformanceManager` query methods allow you to obtain metadata that defines the statistics, and to obtain performance statistics. The following methods are available.

- `QueryAvailablePerfMetric` – Obtains metrics collected for specific time frames or metrics that have a specific `intervalId`.
- `QueryPerf` – Obtains metrics for a specific list of managed entities that provide performance data.
- `QueryPerfComposite` – Obtains metrics for the specified inventory entity and its first-generation child entities, such as for a host and its virtual machines. This method accepts the `refreshRate` for current statistics or the `intervalId` of one of the historical intervals as a parameter. Supported for the `HostSystem` managed entity only.
- `QueryPerfCounter` – Obtains `PerfCounterInfo` data objects for the specified list of counter IDs.
- `QueryPerfCounterByLevel` – Obtains `PerfCounterInfo` data objects for the specified collection level.
- `QueryPerfProviderSummary` – Obtains the `PerfProviderSummary` data object for the specified managed object.

Performance Counters

Each performance provider—that is, each instrumented device or entity—has its own set of counters. The counters are modeled in the system using the `PerfCounterInfo` data object. Instances of this data object provide the metadata for the runtime statistical values. Each `PerfCounterInfo` object has a unique key, the counter ID. The actual performance metrics generated at runtime are identified by this counter ID.

Counters are organized by groups of finite system resources, such as memory, CPU, disk, and so on. 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. You can access the tables from the main entry for the `PerformanceManager` managed object.

The counter groups and specific counters used on any system depend on the configuration.

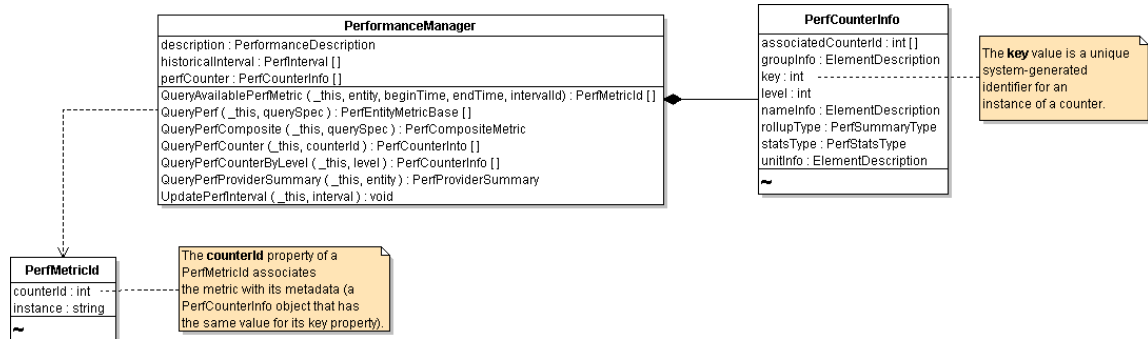
The `PerfCounterInfo` data object includes a description, the unit of measure that the counter represents, and other information about the counter.

Table 16-1. `PerfCounterInfo` Data Object Properties

Property	Description
<code>groupInfo</code>	Name of the resource group to which this counter belongs, such as <code>disk</code> , <code>cpu</code> , or <code>memory</code> .
<code>key</code>	Unique integer that identifies the counter. Also called the counter ID. The value is not static. The counter key on an ESX/ESXi system might not be the same as the counter key for the same counter on the vCenter Server system managing the ESX/ESXi system. However, the system maps the keys from ESX/ESXi to vCenter Server systems transparently.
<code>level</code>	Number from 1 to 4 that identifies the level at which this counter is aggregated with others of its kind into vCenter Server systems.
<code>nameInfo</code>	Descriptive name for the counter that consists of several nested properties.
<code>rollupType</code>	Function that controls how data is converted as it is consolidated from one interval into the next interval. 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.

Each counter's key property identifies the metric generated by the system. The value of key is unique and is not static—it might, for example, change between system reboots. At runtime, the `PerfCounterInfo.key` property is used to populate the value of the `counterId` property of a specific `PerfMetricId` instance (Figure 16-1).

Figure 16-1. PerfCounterInfo Key Matches a PerfMetricId CounterId



The instance value of `PerfMetricId` associates the instance of the metric with its source. The string for the instance value is derived from configuration names for the device, but this value can be empty. For example, for memory and aggregated metric values, this property is empty. For a CPU, this property identifies the numeric position within the CPU core, such as 0, 1, 2, 3.

Sampling Periods and Intervals

Standalone ESX/ESXi systems provide real-time statistics for the current day only. Managed hosts send statistics to the vCenter Server system that manages them. The vCenter Server consolidates the statistical performance data from all ESX/ESXi systems that it manages and stores the information in its database.

Table 16-2. Default Sampling Periods and Retention

Sampling Period	Retention Period	ESX, ESXi	vCenter Server
20 seconds	1 day	■	■
5 minutes	24 hours	■	■
1 hour	7 days	□	■
6 hour	30 days	□	■
24 hour	365 days	□	■

The `PerformanceManager.historicalInterval` property is defined as an array of `PerfInterval` data objects. For an ESX/ESXi system, the array contains a single `PerfInterval` object with the properties listed in Table 16-3.

Table 16-3. Values of `PerfInterval` Data Object from an ESX/ESXi System

Property	Value	Description
key	1	Numeric identifier for the <code>PerfInterval</code> .
name	PastDay	Name of the <code>PerfInterval</code> .
samplingPeriod	300	Number of seconds during which the system samples performance data to produce a metric.
length	129600	Number of seconds that statistics associated with the interval are kept by the system.
enabled	true	This <code>PerfInterval</code> is enabled on the system.
level	null	Statistics collection level. For an ESX/ESXi system, this property is null. The <code>PerfInterval</code> object on an ESX/ESXi system defines the baseline interval.

Obtaining Statistics

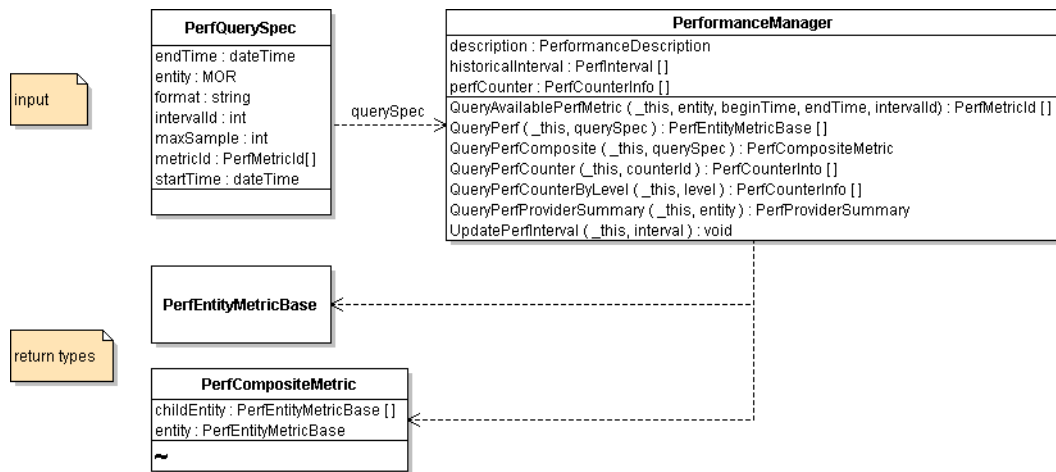
Use the `PerformanceManager.QueryPerf` and `PerformanceManager.QueryPerfComposite` methods to obtain statistical data about performance. These methods behave differently, in the following ways:

- The `QueryPerf` method retrieves performance statistics for one or more inventory entities. Without an `intervalId` as part of the query, the `QueryPerf` method summarizes across all intervals. You define one or more `PerfQuerySpec` objects for the `querySpec` parameter for this operation, and the server returns an array of one or more `PerfEntityMetricBase` objects.
- The `QueryPerfComposite` method retrieves performance statistics for an entity and its first generation child entities.

For both operations, you must define the statistics you want to retrieve using a `PerfQuerySpec` data object.

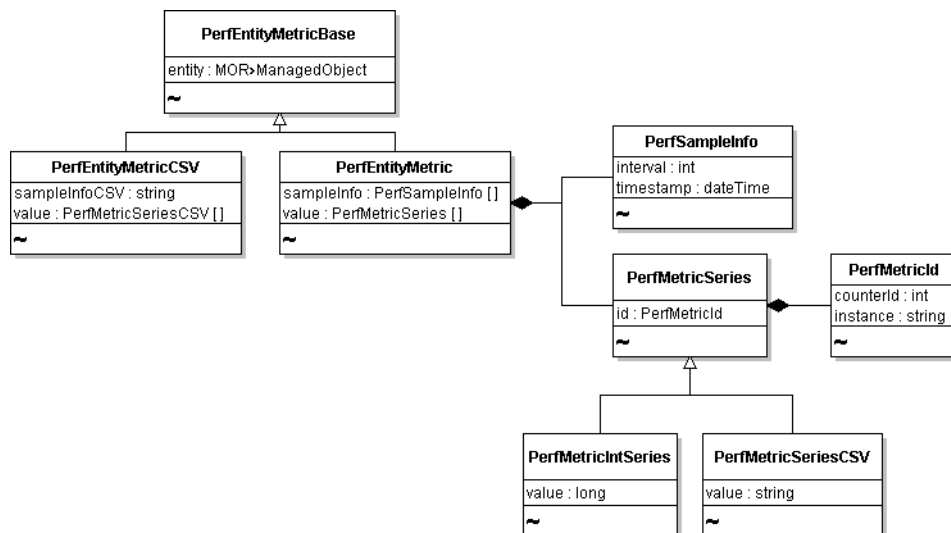
- For `QueryPerf`, you submit an array of `PerfQuerySpec` objects.
- For `QueryPerfComposite`, you submit a single `PerfQuerySpec` object.

Figure 16-2. Define a `PerfQuerySpec` to Obtain Statistics



The `QueryPerfComposite` operation works only at the host level. You can obtain statistics for a host and its associated virtual machines for the specified interval. The client can limit the returned information by specifying a list of metrics and a sample interval ID. The server accepts the `refreshRate` property or one of the historical intervals as input interval.

`QueryPerfComposite` is designed for efficient client-server communications. `QueryPerfComposite` usually generates less network traffic than `QueryPerf` because it returns a large-grained object, a `PerfCompositeMetric` data object, that contains all the data (see [Figure 16-3, “PerfEntityMetricBase Data Object,”](#) on page 155).

Figure 16-3. PerfEntityMetricBase Data Object

You can use the `PerformanceManager.QueryAvailablePerfMetric` method to retrieve information about the metrics that are available for a specified managed entity within a given time frame and use that information to specify the metrics you want to obtain. Call `QueryAvailablePerfMetric` before you use `QueryPerf` or `QueryPerfComposite`.

When you call `QueryAvailablePerfMetric`, you pass in the managed entity, begin and end time, and interval. The interval can be a refresh rate returned in the `PerfProviderSummary` returned by `PerformanceManager.QueryPerfProviderSummary` method or a `PerfInterval` data object.

For standalone ESX/ESXi systems, use the `refreshRate`, which you can obtain from the manufacturer.

vCenter Server and PerformanceManager

A vCenter Server system aggregates performance data from all ESX/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.

By default, the collection level is set to 1 for each of the four intervals. With this collection level, vCenter Server systems retain the following information:

- 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.

[Table 16-4](#) lists the default values of the `historicalInterval` property from a vCenter Server system.

Table 16-4. Values of `PerfInterval` Data Objects from 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

Table 16-5 lists some examples of the types of statistics that can be collected at various levels. The actual counters can vary from version to version of the product.

Table 16-5. Statistics Level Settings

Level	Description
1	Counters defined with average rollup type for CPU, memory, disk, and network; plus counters for system uptime, system heartbeat, and DRS (distributed resource scheduler, tracked in the <code>clusterServices</code> group). Does not include counters for devices.
2	Counters defined with average , summation , and latest rollup types for CPU, memory, disk, and network; plus counters for system uptime, system heartbeat, and DRS (<code>clusterServices</code>). Does not include counters for devices.
3	Counters defined with average , summation , and latest rollup types for CPU, memory, disk, network, and all devices; plus counters for system uptime, system heartbeat, and DRS (<code>clusterServices</code>).
4	All counters defined for all entities and devices, for every rollup type, including minimum and maximum .

The `rollupType` identifies the function that controls how data is converted as it is consolidated from one interval to the next.

Modifying Historical Intervals

Changes to an interval are global and apply to all entities in the system. VMware recommends that you do not modify the default historical intervals. The `PerfInterval` data objects in the `historicalInterval` array are related. Modifying an object in the array can result in problems with all other objects.

If you must modify an interval, use the `UpdatePerfInterval` operation of `PerformanceManager` and follow these guidelines.

- The retention time (the `length` property of `PerfInterval`) must be a multiple of the `samplingPeriod` property value.
- The retention length of each successive period must be greater than the retention length of the previous interval in the array.
- The value of the `samplingPeriod` property for any `PerfInterval` instance cannot be changed.

Optimizing Query Response Time

The `QueryPerf` operation of `PerformanceManager` on vCenter Server has been optimized to return results faster when the query is submitted within certain parameters. If you define the `PerfQuerySpec` data object with the following parameter settings, the query circumvents vCenter Server and obtains results directly from each of the ESX/ESXi systems:

- Set the value of the `intervalId` to 300 (the default sampling interval).
- Set the value of `startTime` and `endTime` so that the time span falls within the last 30 minutes of the current time.

This optimization ensures that the vCenter Server system is not overloaded. Consider using this technique for reporting or any other client application that might subject the system to repeated queries.

The optimization is available in VirtualCenter Server 2.5 and later.

Sample Code Reference

Table 16-6 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 16-6. PerformanceManager Sample Applications

Java (SDK\samples\Axis\java\com\vmware\samples\performance)	C# SDK\samples\DotNet\cs
Basics.java	Basics
History.java	History
PrintCounters.java	PrintCounters
Realtime.java	QueryMemoryOverhead
VI Top.java	RealTime
VIUsage.java	

Diagnostics and Troubleshooting



vSphere includes several logs, which you can access and customize. You can also use the `DiagnosticManager` service interface for troubleshooting.

This appendix includes the following topics:

- [“Troubleshooting Best Practices”](#) on page 159
- [“Overview of Configuration Files and Log Files”](#) on page 160
- [“Modifying the Log Level to Obtain Detailed Information”](#) on page 162
- [“Using DiagnosticManager”](#) on page 163
- [“Using the MOB to Explore the DiagnosticManager”](#) on page 165
- [“Generating Diagnostic Bundles”](#) on page 166

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. See the `Log.cs` and `Log.java` SDK sample applications.
 - C# client logging example: `\SDK\samples\DotNet\cs\ClientUtils\Log.cs`
 - Java client logging example: `\SDK\samples\Axix\java\com\vmware\apputils\Log.java`
- 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 16, “Performance,”](#) on page 151.
 - **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 [Appendix B, “Managed Object Browser,”](#) on page 167.
 - **VMware vSphere Client GUI.** The vSphere Client allows you to examine log files for ESX/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

ESX/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. ESX/ESXi and vCenter Server services maintain separate log files. [Table A-1](#) lists log files or reports, their locations and associated configuration files.

Table A-1. Server and System Logs

Description	Log location	Filename or names	Configuration file
ESX/ESXi service log	/var/log/vmware/	hostd.log [hostd-0.log, ...hostd-9.log]	config.xml
vCenter Server agent log	/var/log/vmware/vpx/	vpxa.log	
Virtual machine kernel core file	/root/	vmkernel-core.<date> vmkernel-log.<date>	syslog.conf, logrotate.conf, various other
syslogd log	/var/log/	messages [messages.1, ... messages.4]	syslog.conf, logrotate.conf
Service console availability report	/var/log/	vmkernel [vmkernel.1, ... vmkernel.8]	syslog.conf, logrotate.conf
VMkernel messages, alerts, and availability reports	/var/log/vmkernel		syslog.conf, logrotate.conf
VMkernel warning	/var/log/	vmkwarning [vmkwarning.1 ... 4 for history]	syslog.conf, logrotate.conf
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 [“ESX/ESXi Log File”](#) on page 160. Can be used as a SOAP monitor when set to trivia log level as in [“Generating Logs”](#) on page 163.
- `vpxa.log` – Agent log file found on each managed ESX/ESXi system.
- `vmware.log` – Virtual machine log. See [“Virtual Machine Log Files”](#) on page 161.

In addition to viewing log files in real time you can also generate reports and complete diagnostic bundles. See [“Generating Diagnostic Bundles”](#) on page 166.

ESX/ESXi Log File

The ESX/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 ESX/ESXi (`hostd`) log file is shown in [Example A-1](#).

Example A-1. Sample ESX/ESXi Log (`hostd.log`) Data

```
...
[2008-05-07 09:50:04.857 'SOAP' 2260 trivia] 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 trivia] Sending soap request to
    [TCP:myservername.eng.vmware.com:443]: logout
[2008-05-07 09:50:04.857 'ProxySvc Req00588' 3136 trivia] Client HTTP stream read error
[2008-05-07 09:50:04.872 'ProxySvc Req00612' 3136 trivia] 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 trivia] 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.

- ESX/ESXi maintains six log files that rotate at each power-cycle (the default) or at a configured file size.
- ESX/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 A-2. 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.

Table A-2. Log Level Settings

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 ESX/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 ESX/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 ESX/ESXi Systems

The ESX/ESXi logs are controlled by a setting in the `config.xml` file, located in the `/etc/vmware/hostd` subdirectory of an ESX/ESXi system ([Example A-3](#)).

Example A-3. ESX/ESXi Config.xml File Excerpt Showing Default Log Level Setting

```
<config>
<vmacore>
<threadPool>
<MaxFdsPerThread>2048</MaxFdsPerThread>
</threadPool>
<ssl>
<doVersionCheck> false </doVersionCheck>
</ssl>
<vmdb>
<maxConnectionCount>8</maxConnectionCount>
</vmdb>
<loadPlugins> true </loadPlugins>
</vmacore>
<workingDir> /var/log/vmware/ </workingDir>
<log>
<directory>/var/log/vmware/</directory>
<name>hostd</name>
<outputToConsole>false</outputToConsole>
<level>info</level>
</log>
',',
</config>
```

By default, the log level setting 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.

The following procedure is meant for an ESX system. On ESXi, use `vifs` to move the `config.xml` file to a server from which you can edit the file.

To change the log level in `config.xml` for `hostd` on an ESX system

- 1 Connect to the ESX system using `putty` or an other secure shell.
- 2 Open the `config.xml` file located at `/etc/vmware/hostd`
- 3 Change `<level>info</level>` to `<level>trivia</level>`, and save and close the file.
- 4 Navigate to the `init.d` directory and restart the host agent.

```
cd /etc/init.d
./mgmt-vmware restart
```

After the service restarts, the new log level is in effect.

Generating Logs

If you are connected to ESX, 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

- 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.

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

- 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

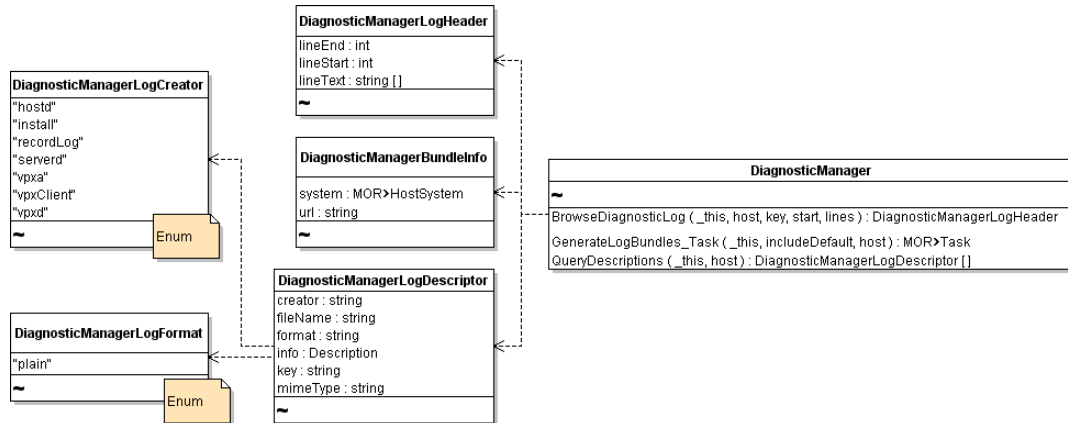
The vSphere API provides access to the `DiagnosticManager`, the service interface for obtaining information from the log files and for generating diagnostic bundles. The logs are populated based on configuration settings, such as `info`, `trivia`, and so on. See [Table A-2, "Log Level Settings,"](#) on page 162.

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 A-1 shows a UML class diagram for DiagnosticManager, which is available on ESX/ESXi and vCenter Server systems.

Figure A-1. DiagnosticManager Managed Object and Associated Data Objects



As shown in Figure A-1, 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. Table A-3 lists all string values currently available from the DiagnosticManagerLogCreator enumeration that can populate the creator property of the DiagnosticManagerLogDescriptor data object.

Table A-3. DiagnosticManagerLogCreator Enumeration

Name	Description
hostd	Host daemon
install	Installation
recordLog	System record log
serverd	Host server agent
vpxa	vCenter agent
vpxClient	vSphere Client
vpxd	vCenter service

Table A-4. ESX/ESXi Sample DiagnosticManager Log Descriptor Values

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

Table A-4. ESX/ESXi Sample DiagnosticManager Log Descriptor Values (Continued)

creator	file name	format	info.label	info.summary	key	mime type
hostd	/var/log/vmkwarning	plain	ESX Log	ESX log in plain format	vmkwarning	text/plain
vpaxa	/var/log/vmware/vpx/vpaxa.log	plain	VirtualCenter Agent Log	VirtualCenter agent log in plain format	vpaxa	text/plain

Using the MOB to Explore the DiagnosticManager

You can access the DiagnosticManager using the MOB. See [Appendix B, “Managed Object Browser,”](#) on page 167.

To explore DiagnosticManager

- 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 ESX/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 ESX/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`.

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

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
```

To export diagnostic data using the vSphere Client

- 1 Start the vSphere Client and connect to the ESX/ESXi or vCenter Server system.
- 2 Select **Administration > Export Diagnostic Data**.
 - For vCenter Server systems, you can filter for specific hosts whose logs you want to export and the location for storing the log files.
 - For ESX/ESXi host systems, you can specify the location for the bundle.
- 3 Click **OK**.

To use the command line to collect ESX log files

- 1 Use putty or an other SSH tool to connect to the service console. On ESXi systems, use the unsupported shell if VMware Technical Support requests that you use it.
- 2 Navigate to the `/usr/bin` subdirectory.
- 3 Run the `vm-support` script:

```
/usr/bin/vm-support
```

This script collects all relevant ESX system log files, configuration files, and other server details into a compressed file (the diagnostic bundle) whose name includes the date and time.

See [“Managing Diagnostic Partitions”](#) on page 82 for information about setting up your system for core dumps.

Managed Object Browser

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.

The appendix includes the following topics:

- [“Using the MOB to Explore the Object Model”](#) on page 167
- [“Using the MOB to Invoke Methods”](#) on page 168

Using the MOB to Explore the Object Model

The Managed Object Browser, or MOB, is a Web-based server application available for all ESX/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 ESX/ESXi or vCenter Server system.

To access the MOB

- 1 Start a Web browser.
- 2 Enter the fully-qualified domain name (or the IP address) for the ESX/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.

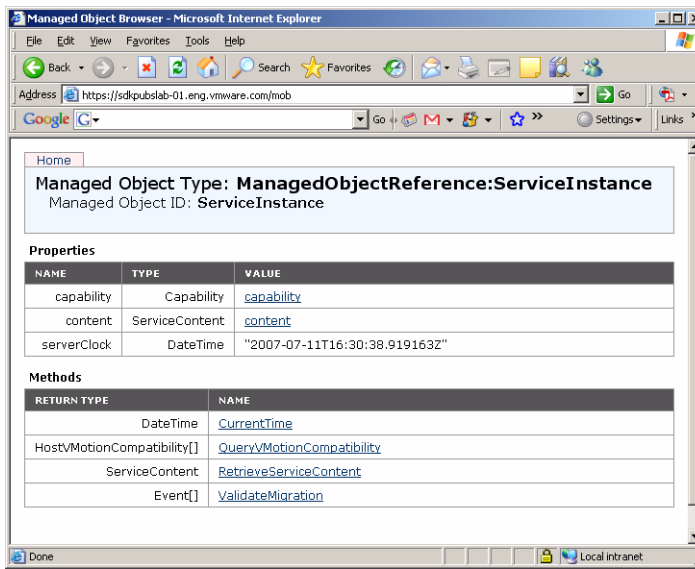
The MOB reveals the underlying structures of the object model. Seeing the structure in conjunction with the *API Reference Guide*, can help with understanding the model.

Using the MOB to Navigate the VMware Infrastructure Object Model

Upon successful connection to the MOB, the browser displays the managed object reference for `ServiceInstance` (see [Figure B-1](#)). 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 `ServiceInstance`. The `ServiceInstance` methods and properties provide access to the entire set of services and inventory objects available on the server. See [“Datacenter Inventory”](#) on page 33.

Figure B-1. Managed Object Browser Connected to a VirtualCenter Server System



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:

- 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 ESX/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 B-1. 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>
```

Example B-1 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 B-2 shows the result of using the definition listed in Example B-1 to specify the managed object reference for a target datacenter to the PerformanceManager.QueryPerfProviderSummary method.

Figure B-2. Using the MOB to Pass Complex Types to a Method

PerfProviderSummary QueryPerfProviderSummary

Parameters		
NAME	TYPE	VALUE
entity (required)	ManagedObjectReference:ManagedObject	<entity type="Datacenter">datacenter-21</entity>

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 [Example B-1](#), 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).

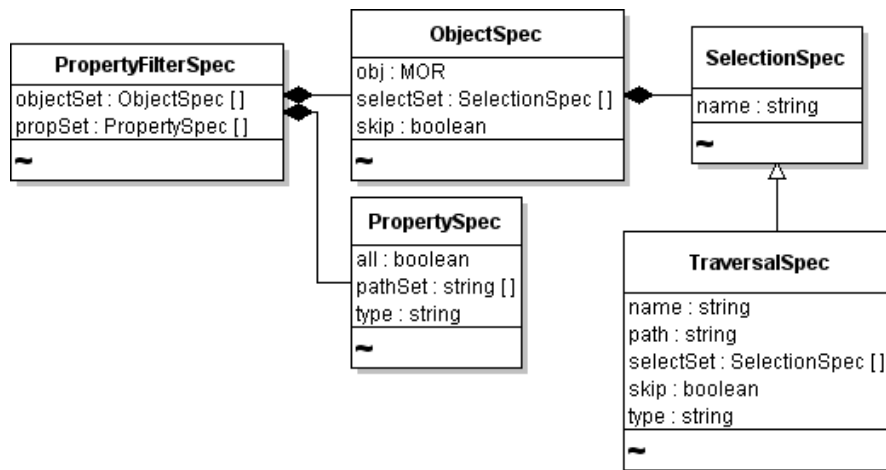
Complex Content

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 B-3](#) shows the relationships among several data objects that `PropertyFilterSpec` consists of.

Figure B-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 [Example B-2](#)).

[Example B-2](#) 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 B-2. 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*. [Example B-3](#) 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 B-3. 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. [Example B-4](#) shows the excerpt.

Example B-4. `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>
...
```

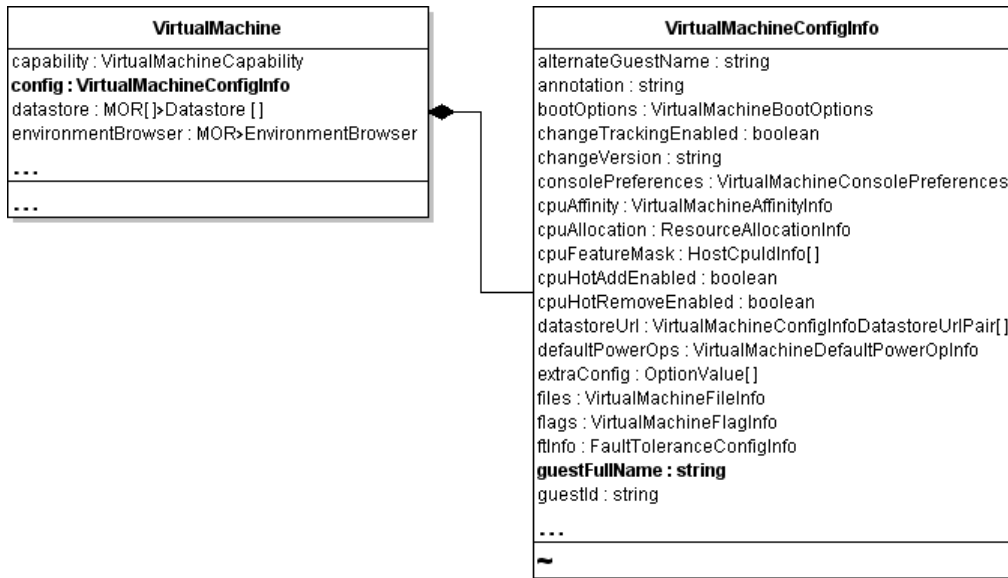
Extrapolating from the WSDL definitions shown in [Example B-2](#), [Example B-3](#), and [Example B-4](#) might produce results similar to those shown in [Example B-5](#).

Example B-5. `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 ([Example B-2](#)). The `pathSet` property defines the full path to the nested data object of interest. In [Example B-5](#), the `pathSet` property defines the path to the `guestFullName` property of the target virtual machine. [Figure B-4](#) shows the UML of these nested data objects.

Figure B-4. Nested Data Objects



All 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. [Table B-1](#) provides a brief summary of the steps involved when you use the MOB and the *vSphere API Reference* together.

Table B-1. Comparison of Datatypes for MOB Usage

Datatype	How to Input Values for Methods
Primitive	Enter the value as plain text regardless of its data type (int, string, boolean). 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

In most cases, client applications interact with vSphere servers by using the vSphere Web Services SDK, as explained in [Chapter 2, “vSphere API Programming Model,”](#) on page 17. In some cases, direct access to configuration files, log files, and other data on an ESX/ESXi or vCenter Server systems is more efficient. This appendix explains direct file access.

The appendix includes the following topics:

- [“Introduction to HTTP Access”](#) on page 173
- [“URL Syntax for HTTP Access”](#) on page 174

Introduction to HTTP Access

ESX/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 ESX/ESXi and vCenter Server systems.
- ESX/ESXi configuration and log file access on ESX/ESXi systems.
- Update bundle access on ESX/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. [Table C-1](#) shows the server access types with the corresponding URL keyword and HTTP methods.

Table C-1. HTTP Access to vSphere Servers

Server Access	URL keyword	HTTP Method or Methods
Datastore	folder	GET, HEAD, PUT, DELETE
ESX/ESXi configuration file	host	GET, HEAD, PUT (See Table C-2 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)” on page 174
- “Host File Access (/host)” on page 175
- “Update Package Access (/tmp)” on page 176

Datastore Access (/folder)

An HTTP request for datastore access uses the following syntax:

```
http-method http[s]://server/folder[[/path]?dcPath=path&&dsName=name]]
```

<i>http-method</i>	One of the methods GET, HEAD, PUT, or DELETE.
http:// or https://	Access protocol (standard access or secure access).
<i>server</i>	ESX/ESXi or vCenter Server target system. The server value can be an IP address or a DNS name.
<i>/folder</i>	Specifies datastore access on an ESX/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 <code>Datacenter</code> located in the root folder, the <i>dcPath</i> value is <code>MyDatacenter</code>. For a datacenter named <code>YourDatacenter</code> located in the folder <code>NorthAmerica</code> which is located in the root folder, the <i>cdPath</i> value is <code>NorthAmerica/YourDatacenter</code>. ■ <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 ESX/ESXi system, `dcPath=DCPATH&` is optional and defaults to `dcPath=ha-datacenter`.

Example	Description
<code>/folder</code>	Directory listing of known datacenters on this server.
<code>/folder?dcPath=<i>path</i></code>	Directory listing of all datastores available at the specified datacenter.
<code>/folder?dcPath=<i>path</i>&dsName=<i>name</i></code>	Top-level directory listing of the datastore.
<code>/folder/<i>path</i>?dcPath=<i>path</i>&dsName=<i>name</i></code>	Directory listing of all files in a datastore directory.
<code>/folder/<i>path</i>/disk-flat.vmdk?dcPath=<i>path</i>&dsName=<i>name</i></code>	Access individual files.

Host File Access (/host)

An HTTP request for access to ESX/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 Table C-2).
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 ESX/ESXi configuration file.

[Table C-2](#) shows ESX host configuration files and the corresponding HTTP/HTTPS methods for access. The set of files might change from version to version.

Table C-2. HTTP Host Access (/host URL)

Configuration File	HTTP Access Method(s)	Configuration File	HTTP Access Method(s)
hostAgentConfig.xml	GET, HEAD, PUT	ipmi0_sel.raw	GET, HEAD
sfcf.cfg	GET, HEAD, PUT	ipmi0_sel	GET, HEAD
openwsman.conf	GET, HEAD, PUT	ipmi0_sdr_content.raw	GET, HEAD
license.cfg	GET, HEAD, PUT	ipmi0_sdr_header.raw	GET, HEAD
vmware.lic	GET, HEAD, PUT	ipmi0_sensor_readings.raw	GET, HEAD
vmware_config	GET, HEAD, PUT	ipmi1_sel.raw	GET, HEAD
vmware_configrules	GET, HEAD, PUT	ipmi1_sel	GET, HEAD
proxy.xml	GET, HEAD, PUT	ipmi1_sdr_content.raw	GET, HEAD
snmp.xml	GET, HEAD, PUT	ipmi1_sdr_header.raw	GET, HEAD
syslog.conf	GET, HEAD, PUT	ipmi1_sensor_readings.raw	GET, HEAD
ssl_cert	GET, HEAD, PUT	ipmi2_sel.raw	GET, HEAD
ssl_key	PUT	ipmi2_sel	GET, HEAD
hosts	GET, HEAD, PUT	ipmi2_sdr_content.raw	GET, HEAD
motd	GET, HEAD, PUT	ipmi2_sdr_header.raw	GET, HEAD
vpxa.cfg	GET, HEAD, PUT	ipmi2_sensor_readings.raw	GET, HEAD
esx.conf	GET, HEAD, PUT	ipmi3_sel.raw	GET, HEAD
config.log	GET, HEAD	ipmi3_sel	GET, HEAD
messages	GET, HEAD	ipmi3_sdr_content.raw	GET, HEAD
hostd.log	GET, HEAD	ipmi3_sdr_header.raw	GET, HEAD
vpxa.log	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. [Table C-3](#) shows the required privileges.

Table C-3. Privileges Required for HTTP Access Datastore Objects

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
Host	/host	Host.Config.AdvancedConfig
	/tmp/	Host.Config.SystemManagement

Privileges Reference

D

VMware vSphere components are secured through a system of privileges, roles, and permissions (see [Chapter 6, “Authentication and Authorization,”](#) on page 53 for more information). This appendix lists privileges required to perform various operations, and privileges required to read properties.

The appendix includes the following topics:

- [“Privileges Required to Invoke Operations”](#) on page 177
- [“Privileges Required to Read Properties”](#) on page 185
- [“Privileges Defined for the Administrator Role”](#) on page 186

Privileges Required to Invoke Operations

By default, all users who are members of the Windows Administrators group on the vCenter Server system have the same access rights as a user assigned to the Administrator role on all objects. When connecting directly to an ESX/ESXi host, the root and vpxuser user accounts have the same access rights as any user assigned the Administrator role on all objects.

All other users initially have no permissions on any objects, which means they cannot view these objects or perform operations on them. A user with Administrator privileges must assign permissions to these users to allow them to perform tasks.

[Table D-1](#) lists the privileges required to perform various operations. (For privileges identified as dynamic, see the *vSphere API Reference*.) Operations can be supported by vCenter Server, ESX/ESXi, or both, as shown in [Table D-1](#).

IMPORTANT The information does not include operations new in ESX/ESXi 4.1.

Table D-1. Privileges Required for vCenter Server and ESX/ESXi Operations

Operation	Privilege	VS	ESX/ESXi
AcquireLocalTicket	System.Anonymous	X	X
AcquireMksTicket	VirtualMachine.Interact.ConsoleInteract	X	X
AddAuthorizationRole	Authorization.ModifyRoles	X	X
AddCustomFieldDef	Global.ManageCustomFields	X	
AddHost_Task	Host.Inventory.AddHostToCluster	X	
AddInternetScsiSendTargets	Host.Config.Storage	X	X
AddInternetScsiStaticTargets	Host.Config.Storage	X	X
AddPortGroup	Host.Config.Network	X	X
AddServiceConsoleVirtualNic	Host.Config.Network	X	X
AddStandaloneHost_Task	Host.Inventory.AddStandaloneHost	X	

Table D-1. Privileges Required for vCenter Server and ESX/ESXi Operations (Continued)

Operation	Privilege	VS	ESX/ESXi
AddVirtualNic	Host.Config.Network	X	X
AddVirtualSwitch	Host.Config.Network	X	X
AnswerVM	VirtualMachine.Interact.AnswerQuestion	X	X
ApplyRecommendation	Resource.ApplyRecommendation	X	
AssignUserToGroup	Host.Local.ManageUserGroups	X	X
AttachVmfsExtent	Host.Config.Storage	X	X
AutoStartPowerOff	Host.Config.AutoStart	X	X
AutoStartPowerOn	Host.Config.AutoStart	X	X
BrowseDiagnosticLog	Global.Diagnostics	X	X
CancelTask	Global.CancelTask	X	
CancelWaitForUpdates	System.View	X	X
CheckCustomizationResources	System.View	X	
CheckCustomizationSpec	VirtualMachine.Provisioning.Customize	X	
CheckForUpdates	System.View	X	X
CheckIfMasterSnmpAgentRunning	Host.Config.Snmp	X	X
CheckLicenseFeature	Global.Licenses	X	X
CloneVM_Task	NONE. Privileges are required on the virtual machine being cloned and depend on whether the virtual machine is a template. See CloneVM_Task in the <i>vSphere API Reference</i> for specific privileges. You need the VirtualMachine.Inventory.Create privilege on the folder where the new virtual machine is located.	X	
ComputeDiskPartitionInfo	Host.Config.Storage	X	X
ConfigureDatastorePrincipal	Host.Config.Maintenance	X	X
ConfigureLicenseSource	Global.Licenses	X	X
CreateAlarm	NONE. Create privilege required on the entity associated with the alarm.	X	
CreateCluster	Host.Inventory.CreateCluster	X	
CreateCollectorForEvents	System.View	X	X
CreateCollectorForTasks	System.View	X	
CreateCustomizationSpec	VirtualMachine.Provisioning.ModifyCustSpecs	X	
CreateDatacenter	Datacenter.Create	X	X
CreateDiagnosticPartition	Host.Config.Storage	X	X
CreateFilter	System.View	X	X
CreateFolder	Folder.Create	X	X
CreateGroup	Host.Local.ManageUserGroups	X	X
CreateNasDatastore	Host.Config.Storage	X	X
CreatePerfInterval	Performance.ModifyIntervals	X	X
CreateResourcePool	Resource.CreatePool	X	X

Table D-1. Privileges Required for vCenter Server and ESX/ESXi Operations (Continued)

Operation	Privilege	VS	ESX/ESXi
CreateScheduledTask	NONE. ScheduledTask.Create required on the entity associated with the scheduled task.	X	
CreateSnapshot_Task	VirtualMachine.State.CreateSnapshot	X	X
CreateUser	Host.Local.ManageUserGroups	X	X
CreateVM_Task	VirtualMachine.Inventory.Create Also, Resource.AssignVMToPool privilege required on the resource pool with which the virtual machine will be associated.	X	X
CreateVmfsDatastore	Host.Config.Storage	X	X
CurrentTime	System.View	X	X
CustomizationSpecItemToXml	System.View	X	
CustomizeVM_Task	VirtualMachine.Provisioning.Customize	X	
DeleteCustomizationSpec	VirtualMachine.Provisioning.ModifyCustSpecs	X	
DeleteFile	Datastore.DeleteFile	X	X
DeselectVnic	Host.Config.Network	X	X
Destroy_Task	See Destroy_Task in the <i>vSphere API Reference</i> .	X	X
DestroyChildren	See DestroyChildren in the <i>vSphere API Reference</i> .	X	X
DestroyCollector	NONE	X	X
DestroyDatastore	Datastore.Delete	X	X
DestroyNetwork	Network.Delete	X	X
DestroyPropertyFilter	NONE	X	X
DisableFeature	Global.Licenses	X	X
DisableHyperThreading	Host.Config.HyperThreading	X	X
DisableMultipathPath	Host.Config.Storage	X	X
DisableRuleSet	Host.Config.NetService	X	X
DisconnectHost_Task	Host.Config.Connection	X	X
DoesCustomizationSpecExist	VirtualMachine.Provisioning.ReadCustSpecs	X	
DuplicateCustomizationSpec	VirtualMachine.Provisioning.ModifyCustSpecs	X	
EnableFeature	Global.Licenses	X	X
EnableHyperThreading	Host.Config.HyperThreading	X	X
EnableMultipathPath	Host.Config.Storage	X	X
EnableRuleset	Host.Config.NetService	X	X
EnterMaintenanceMode_Task	Host.Config.Maintenance	X	X
ExitMaintenanceMode_Task	Host.Config.Maintenance	X	X
ExtendVmfsDatastore	Host.Config.Storage	X	X
FindByDatastorePath	System.View	X	X
FindByDnsName	System.View	X	X
FindByInventoryPath	System.View	X	X
FindByIp	System.View	X	X
FindByUuid	System.View	X	X
FindChild	System.View	X	X

Table D-1. Privileges Required for vCenter Server and ESX/ESXi Operations (Continued)

Operation	Privilege	VS	ESX/ESXi
FormatVmfs	Host.Config.Storage	X	X
GenerateLogBundles_Task	Global.Diagnostics	X	X
GetAlarm	System.View	X	
GetAlarmState	NONE System.Read privilege is required on the entity associated with the alarm.	X	
GetCustomizationSpec	VirtualMachine.Provisioning.ReadCustSpecs	X	
JoinDomainTask	Host.Config.AuthenticationStore		X
LeaveCurrentDomain_Task	Host.Config.AuthenticationStore		X
Login	System.Anonymous	X	X
Logout	System.View	X	X
LogUserEvent	NONE Global.LogEvent required on the entity associated with the event.	X	X
MarkAsTemplate	VirtualMachine.Provisioning.MarkAsTemplate	X	
MarkAsVirtualMachine	VirtualMachine.Provisioning.MarkAsVM Resource.AssignVMToPool required on the resource pool to associate with the virtual machine.	X	
MergePermissions	Authorization.ReassignRolePermissions	X	X
MigrateVM_Task	See MigrateVM_Task in the <i>vSphere API Reference</i> .	X	
MountToolsInstaller	VirtualMachine.Interact.ToolsInstall	X	X
MoveHostInto_Task	Host.Inventory.EditCluster Host.Inventory.MoveHost required on the host being moved.	X	
MoveInto_Task	Host.Inventory.EditCluster Host.Inventory.MoveHost required on the host being moved.	X	X
MoveIntoFolder_Task	See MoveIntoFolder_Task in the <i>vSphere API Reference</i> .	X	X
MoveIntoResourcePool	See MoveIntoFolder_Task in the <i>vSphere API Reference</i> .	X	X
OverwriteCustomizationSpec	VirtualMachine.Provisioning.ModifyCustSpecs	X	
PowerOffVM_Task	VirtualMachine.Interact.PowerOff	X	X
PowerOnVM_Task	VirtualMachine.Interact.PowerOn	X	X
QueryAvailableDisksForVmfs	Host.Config.Storage	X	X
QueryAvailablePartition	Host.Config.Storage	X	X
QueryAvailablePerfMetric	NONE System.Read is required on the entity for which available performance metrics are queried.	X	X
QueryConfigOption	System.Read	X	X
QueryConfigOptionDescriptor	System.Read	X	X
QueryConfigTarget	System.Read	X	X
QueryConnectionInfo	Host.Inventory.AddStandaloneHost	X	X
QueryDescriptions	Global.Diagnostics	X	X
QueryEvents	System.View	X	X
QueryHostConnectionInfo	System.Read	X	X

Table D-1. Privileges Required for vCenter Server and ESX/ESXi Operations (Continued)

Operation	Privilege	VS	ESX/ESXi
QueryLicenseSourceAvailability	Global.Licenses	X	X
QueryLicenseUsage	Global.Licenses	X	X
QueryMemoryOverhead	System.Read	X	X
QueryNetworkHint	Host.Config.Network	X	X
QueryOptions	System.Read	X	X
QueryPartitionCreateDesc	Host.Config.Storage	X	X
QueryPartitionCreateOptions	Host.Config.Storage	X	X
QueryPerf	NONE System.Read privilege is required on the entity whose performance statistics are being queried.	X	X
QueryPerfComposite	NONE System.Read privilege is required on the entity whose performance statistics are being queried.	X	X
QueryPerfCounter	System.View	X	X
QueryPerfProviderSummary	NONE System.Read privilege is required on the entity whose performance statistics are being queried.	X	X
QueryVmfsDatastoreCreateOptions	Host.Config.Storage	X	X
QueryVmfsDatastoreExtendOptions	Host.Config.Storage	X	X
QueryVMotionCompatibility	Resource.QueryVMotion	X	
ReadNextEvents	NONE	X	X
ReadNextTasks	NONE	X	
ReadPreviousEvents	NONE	X	X
ReadPreviousTasks	NONE	X	
RebootGuest	VirtualMachine.Interact.Reset	X	X
RebootHost_Task	Host.Config.Maintenance	X	X
RecommendHostsForVm	System.Read	X	
ReconfigureAlarm	Alarm.Edit	X	
ReconfigureAutostart	Host.Config.AutoStart	X	X
ReconfigureCluster_Task	Host.Inventory.EditCluster	X	
ReconfigureHostForDAS_Task	Host.Config.Connection	X	
ReconfigureScheduledTask	ScheduledTask.Edit	X	
ReconfigureServiceConsoleReservation	Host.Config.Memory	X	X
ReconfigVM_Task	dynamic	X	X
ReconnectHost_Task	Host.Config.Connection	X	
RefreshDatastore	System.Read	X	X
RefreshFirewall	Host.Config.NetService	X	X
RefreshNetworkSystem	Host.Config.Network	X	X
RefreshServices	Host.Config.NetService	X	X
RefreshStorageSystem	Host.Config.Storage	X	X

Table D-1. Privileges Required for vCenter Server and ESX/ESXi Operations (Continued)

Operation	Privilege	VS	ESX/ESXi
RegisterVM_Task	VirtualMachine.Inventory.Create Resource.AssignVMToPool privilege is required on the resource pool to which the virtual machine should be attached.	X	X
ReleaseLease	NONE.	X	X
Reload	System.Read	X	X
RelocateVM_Task	Resource.ColdMigrate	X	
RemoveAlarm	Alarm.Delete	X	
RemoveAllSnapshots_Task	VirtualMachine.State.RemoveSnapshot	X	X
RemoveAuthorizationRole	Authorization.ModifyRoles	X	X
RemoveCustomFieldDef	Global.ManageCustomFields	X	
RemoveDatastore	Host.Config.Storage	X	X
RemoveEntityPermission	NONE Authorization.ModifyPermissions privilege is required on the entity associated with the permission.	X	X
RemoveGroup	Host.Local.ManageUserGroups	X	X
RemoveInternetScsiSendTargets	Host.Config.Storage	X	X
RemoveInternetScsiStaticTargets	Host.Config.Storage	X	X
RemovePerfInterval	Performance.ModifyIntervals	X	X
RemovePortGroup	Host.Config.Network	X	X
RemoveScheduledTask	ScheduledTask.Delete	X	
RemoveServiceConsoleVirtualNic	Host.Config.Network	X	X
RemoveSnapshot_Task	VirtualMachine.State.RemoveSnapshot	X	X
RemoveUser	Host.Local.ManageUserGroups	X	X
RemoveVirtualNic	Host.Config.Network	X	X
RemoveVirtualSwitch	Host.Config.Network	X	X
Rename_Task	See Rename_Task in the <i>vSphere API Reference</i> .	X	X
RenameCustomFieldDef	Global.ManageCustomFields	X	
RenameCustomizationSpec	VirtualMachine.Provisioning.ModifyCustSpecs	X	
RenameDatastore	Datacenter.RenameDatastore	X	X
RenameSnapshot	VirtualMachine.State.RenameSnapshot	X	All but ESX 2.x
RenewLease	NONE	X	X
RescanAllHba	Host.Config.Storage	X	X
RescanHba	Host.Config.Storage	X	X
RescanVmfs	Host.Config.Storage	X	X
ResetCollector	NONE	X	X
ResetEntityPermissions	NONE Authorization.ModifyPermissions privilege is required on the entity associated with the permission and the entity's parent.	X	X
ResetGuestInformation	VirtualMachine.Config.ResetGuestInfo	X	X
ResetVM_Task	VirtualMachine.Interact.Reset	X	X
RestartMasterSnmpAgent	Host.Config.Snmp	X	X

Table D-1. Privileges Required for vCenter Server and ESX/ESXi Operations (Continued)

Operation	Privilege	VS	ESX/ESXi
RestartService	Host.Config.NetService	X	X
RestartServiceConsoleVirtualNic	Host.Config.Network	X	X
RetrieveAllPermissions	System.View	X	X
RetrieveDiskPartitionInfo	Host.Config.Storage	X	X
RetrieveEntityPermissions	NONE System.Read privilege is required on the entity whose performance statistics are being queried.	X	X
RetrieveEntityScheduledTask	System.View	X	
RetrieveProperties	System.View	X	X
RetrieveRolePermissions	System.View	X	X
RetrieveServiceContent	System.Anonymous	X	X
RetrieveUserGroups	System.View	X	X
RevertToCurrentSnapshot_Task	VirtualMachine.State.RevertToSnapshot	X	On all but ESX 2.x
RevertToSnapshot_Task	VirtualMachine.State.RevertToSnapshot		
RewindCollector	NONE	X	X
RunScheduledTask	ScheduledTask.Run	X	
SearchDatastore_Task	Datastore.Browse	X	X
SearchDatastoreSubFolders_Task	Datastore.Browse	X	X
SelectActivePartition	Host.Config.Storage	X	X
SelectVnic	Host.Config.Network	X	X
SetCollectorPageSize	NONE	X	X
SetEntityPermissions	NONE Authorization.ModifyPermissions required on entity associated with the permissions and its parent.	X	X
SetField	NONE Global.SetCustomField required on the entity associated with the custom field.	X	X
SetLicenseEdition	Global.Licenses	X	X
SetLocale	System.View	X	X
SetMultipathLunPolicy	Host.Config.Storage	X	X
SetScreenResolution	VirtualMachine.Interact.ConsoleInteract	X	X
ShutdownGuest	VirtualMachine.Interact.PowerOff	X	X
ShutdownHost_Task	Host.Config.Maintenance	X	X
StandbyGuest	VirtualMachine.Interact.Suspend	X	X
StartService	Host.Config.NetService	X	X
StopMasterSnmAgent	Host.Config.Snmp	X	X
StopServiceq	Host.Config.NetService	X	X
SuspendVM_Task	VirtualMachine.Interact.Suspend	X	X
TerminateSession	Sessions.TerminateSession	X	X
UnassignUserFromGroup	Host.Local.ManageUserGroups	X	X
UninstallService	Host.Config.NetService	X	X
UnmountToolsInstaller	VirtualMachine.Interact.ToolsInstall	X	X

Table D-1. Privileges Required for vCenter Server and ESX/ESXi Operations (Continued)

Operation	Privilege	VS	ESX/ESXi
UnregisterAndDestroy_Task	Folder.Delete The privilege is also required on the parent managed entity.	X	X
UnregisterVM	VirtualMachine.Inventory.Delete	X	X
UpdateAuthorizationRole	Authorization.ModifyRoles	X	X
UpdateChildResourceConfiguration	See UpdateChildResourceConfiguration in the <i>vSphere API Reference</i> .	X	X
UpdateConfig	See UpdateConfig in the <i>vSphere API Reference</i> .	X	X
UpdateConsoleIpRouteConfig	Host.Config.Network	X	X
UpdateDefaultPolicy	Host.Config.Network	X	X
UpdateDiskPartitions	Host.Config.Storage	X	X
UpdateDnsConfig	Host.Config.Network	X	X
UpdateInternetScsiAlias	Host.Config.Storage	X	X
UpdateInternetScsiAuthenticationProperties	Host.Config.Storage	X	X
UpdateInternetScsiDiscoveryProperties	Host.Config.Storage	X	X
UpdateInternetScsiIPProperties	Host.Config.Storage	X	X
UpdateInternetScsiName	Host.Config.Storage	X	X
UpdateIpConfig	Host.Config.Network	X	X
UpdateIpRouteConfig	Host.Config.Network	X	X
UpdateNetworkConfig	Host.Config.Network	X	X
UpdateOptions	See UpdateOptions in the <i>vSphere API Reference</i> .	X	
UpdatePerfInterval	Performance.ModifyIntervals	X	X
UpdatePhysicalNicLinkSpeed	Host.Config.Network	X	X
UpdatePortGroup	Host.Config.Network	X	X
UpdateServiceConsoleVirtualNic	Host.Config.Network	X	X
UpdateServiceMessage	Sessions.GlobalMessage	X	X
UpdateServicePolicy	Host.Config.NetService	X	X
UpdateSnmpConfig	Host.Config.Snmp	X	X
UpdateSoftwareInternetScsiEnabled	Host.Config.Storage	X	X
UpdateSystemResources	Host.Config.Resources	X	X
UpdateUser	Host.Local.ManageUserGroups	X	X
UpdateVirtualNic	Host.Config.Network	X	X
UpdateVirtualSwitch	Host.Config.Network	X	X
UpgradeTools_Task	VirtualMachine.Interact.ToolsInstall	X	X
UpgradeVM_Task	VirtualMachine.Config.UpgradeVirtualHardware	X	X
UpgradeVmfs	Host.Config.Storage	X	X
UpgradeVmLayout	Host.Config.Storage	X	X
ValidateMigration	See ValidateMigration in the <i>vSphere API Reference</i> . Resource.AssignVMToPool required on the target resource pool for the virtual machines.	X	
WaitForUpdates	System.View	X	X
XmlToCustomizationSpecItem	System.View	X	

Privileges Required to Read Properties

Figure D-2 lists the privileges required to read specific managed object properties.

Table D-2. Privileges Required for Reading Object Properties

Object	Property	Privilege
AlarmManager	defaultExpression	System.View
	description	System.View
AuthorizationManager	privilegeList	System.View
	roleList	System.View
	description	System.View
ComputeResource	resourcePool	System.View
	host	System.View
CustomFieldsManager	field	System.View
CustomizationSpecManager	info	VirtualMachine.Provisioning.ReadCustSpecs
	encryptionKey	System.View
Datacenter	vmFolder	System.View
	hostFolder	System.View
EventManager	description	System.View
	latestEvent	System.View
	maxCollector	System.View
Folder	childType	System.View
	childEntity	System.View
HostCpuSchedulerSystem	hyperThread	Host.Config.HyperThreading
HostDiagnosticSystem	activePartition	Host.Config.Storage
HostFirewallSystem	firewallInfo	Host.Config.NetService
HostMemoryManagerSystem	consoleReservationInfo	Host.Config.Memory
HostNetworkSystem	capabilities	Host.Config.Network
	networkConfig	Host.Config.Network
	networkInfo	Host.Config.Network
	offloadCapabilities	Host.Config.Network
HostPowerSystem	capability	Host.Config.Power
	info	Host.Config.Power
HostServiceSystem	serviceInfo	Host.Config.NetService
HostSnmpSystem	snmpConfig	Host.Config.Snmp
HostStorageSystem	fileSystemVolumeInfo	Host.Config.Storage
	storageDeviceInfo	Host.Config.Storage
HostVMotionSystem	netConfig	Host.Config.Network
	ipConfig	Host.Config.Network
LicenseManager	source	Global.Licenses
	sourceAvailable	Global.Licenses
	featureInfo	Global.Licenses

Table D-2. Privileges Required for Reading Object Properties (Continued)

Object	Property	Privilege
ManagedEntity	parent	System.View
	effectiveRole	System.View
	name	System.View
PerformanceManager	description	System.View
	historicalInterval	System.View
	perfCounter	System.View
PropertyCollector	filter	System.View
ResourcePool	owner	System.View
ScheduledTaskManager	scheduledTask	System.View
	description	System.View
ServiceInstance	serverClock	System.View
	capability	System.View
SessionManager	sessions	Sessions.TerminateSession
	currentSession	System.Anonymous
	message	System.View
	messageLocaleList	System.Anonymous
	supportedLocaleList	System.Anonymous
	defaultLocale	System.Anonymous
TaskManager	recentTask	System.View
	description	System.View
	maxCollector	System.View
UserDirectory	domainList	System.View

Privileges Defined for the Administrator Role

The Administrator role as defined on a vCenter Server 4.0 system contains all the privileges listed in [Table D-3](#).

Table D-3. Privileges Granted to the Administrator Role

Privilege	Privilege
Alarm.Acknowledge	Resource.AssignVAppToPool
Alarm.Create	Resource.AssignVMToPool
Alarm.Delete	Resource.ColdMigrate
Alarm.DisableActions	Resource.CreatePool
Alarm.Edit	Resource.DeletePool
Alarm.SetStatus	Resource.EditPool
Authorization.ModifyPermissions	Resource.HotMigrate
Authorization.ModifyRoles	Resource.MovePool
Authorization.ReassignRolePermissions	Resource.QueryVMotion
Datacenter.Create	Resource.RenamePool
Datacenter.Delete	ScheduledTask.Create
Datacenter.IpPoolConfig	ScheduledTask.Delete
Datacenter.Move	ScheduledTask.Edit

Table D-3. Privileges Granted to the Administrator Role (Continued)

Privilege	Privilege
Datacenter.Rename	ScheduledTask.Run
Datastore.AllocateSpace	Sessions.GlobalMessage
Datastore.Browse	Sessions.ImpersonateUser
Datastore.Delete	Sessions.TerminateSession
Datastore.DeleteFile	Sessions.ValidateSession
Datastore.FileManagement	StorageViews.ConfigureService
Datastore.Move	StorageViews.View
Datastore.Rename	System.Anonymous
DVPortgroup.Create	System.Read
DVPortgroup.Delete	System.View
DVPortgroup.Modify	Task.Create
DVPortgroup.PolicyOp	VApp.ApplicationConfig
DVPortgroup.ScopeOp	VApp.AssignResourcePool
DVSwitch.Create	VApp.AssignVApp
DVSwitch.Delete	VApp.AssignVM
DVSwitch.HostOp	VApp.Clone
DVSwitch.Modify	VApp.Create
DVSwitch.Move	VApp.Delete
DVSwitch.PolicyOp	VApp.Export
DVSwitch.PortConfig	VApp.ExtractOvfEnvironment
DVSwitch.PortSetting	VApp.Import
DVSwitch.Vspan	VApp.InstanceConfig
Extension.Register	VApp.Move
Extension.Unregister	VApp.PowerOff
Extension.Update	VApp.PowerOn
Folder.Create	VApp.Rename
Folder.Delete	VApp.ResourceConfig
Folder.Move	VirtualMachine.Config.AddExistingDisk
Folder.Rename	VirtualMachine.Config.AddNewDisk
Global.CancelTask	VirtualMachine.Config.AddRemoveDevice
Global.CapacityPlanning	VirtualMachine.Config.AdvancedConfig
Global.Diagnostics	VirtualMachine.Config.ChangeTracking
Global.DisableMethods	VirtualMachine.Config.CPUCount
Global.EnableMethods	VirtualMachine.Config.DiskExtend
Global.GlobalTag	VirtualMachine.Config.DiskLease
Global.Health	VirtualMachine.Config.EditDevice
Global.Licenses	VirtualMachine.Config.HostUSBDevice
Global.LogEvent	VirtualMachine.Config.Memory
Global.ManageCustomFields	VirtualMachine.Config.QueryUnownedFiles
Global.Proxy	VirtualMachine.Config.RawDevice
Global.ScriptAction	VirtualMachine.Config.RemoveDisk

Table D-3. Privileges Granted to the Administrator Role (Continued)

Privilege	Privilege
Global.ServiceManagers	VirtualMachine.Config.Rename
Global.SetCustomField	VirtualMachine.Config.ResetGuestInfo
Global.Settings	VirtualMachine.Config.Resource
Global.SystemTag	VirtualMachine.Config.Settings
Global.VCServer	VirtualMachine.Config.SwapPlacement
Host.Cim.CimInteraction	VirtualMachine.Config.UpgradeVirtualHardware
Host.Config.AdvancedConfig	VirtualMachine.Interact.AnswerQuestion
Host.Config.AutoStart	VirtualMachine.Interact.Backup
Host.Config.Connection	VirtualMachine.Interact.ConsoleInteract
Host.Config.DateTime	VirtualMachine.Interact.CreateScreenshot
Host.Config.Firmware	VirtualMachine.Interact.CreateSecondary
Host.Config.HyperThreading	VirtualMachine.Interact.DefragmentAllDisks
Host.Config.Maintenance	VirtualMachine.Interact.DeviceConnection
Host.Config.Memory	VirtualMachine.Interact.DisableSecondary
Host.Config.NetService	VirtualMachine.Interact.EnableSecondary
Host.Config.Network	VirtualMachine.Interact.MakePrimary
Host.Config.Patch	VirtualMachine.Interact.PowerOff
Host.Config.PciPassthru	VirtualMachine.Interact.PowerOn
Host.Config.Resources	VirtualMachine.Interact.Record
Host.Config.Settings	VirtualMachine.Interact.Replay
Host.Config.Snmp	VirtualMachine.Interact.Reset
Host.Config.Storage	VirtualMachine.Interact.SetCDMedia
Host.Config.SystemManagement	VirtualMachine.Interact.SetFloppyMedia
Host.Inventory.AddHostToCluster	VirtualMachine.Interact.Suspend
Host.Inventory.AddStandaloneHost	VirtualMachine.Interact.TerminateFaultTolerantVM
Host.Inventory.CreateCluster	VirtualMachine.Interact.ToolsInstall
Host.Inventory.DeleteCluster	VirtualMachine.Interact.TurnOffFaultTolerance
Host.Inventory.EditCluster	VirtualMachine.Inventory.Create
Host.Inventory.MoveCluster	VirtualMachine.Inventory.Delete
Host.Inventory.MoveHost	VirtualMachine.Inventory.Move
Host.Inventory.RemoveHostFromCluster	VirtualMachine.Provisioning.Clone
Host.Inventory.RenameCluster	VirtualMachine.Provisioning.CloneTemplate
Host.Local.CreateVM	VirtualMachine.Provisioning.CreateTemplateFromVM
Host.Local.DeleteVM	VirtualMachine.Provisioning.Customize
Host.Local.InstallAgent	VirtualMachine.Provisioning.DeployTemplate
Host.Local.ManageUserGroups	VirtualMachine.Provisioning.DiskRandomAccess
Host.Local.ReconfigVM	VirtualMachine.Provisioning.DiskRandomRead
Network.Assign	VirtualMachine.Provisioning.GetVmFiles
Network.Config	VirtualMachine.Provisioning.MarkAsTemplate
Network.Delete	VirtualMachine.Provisioning.MarkAsVM
Network.Move	VirtualMachine.Provisioning.ModifyCustSpecs

Table D-3. Privileges Granted to the Administrator Role (Continued)

Privilege	Privilege
Performance.ModifyIntervals	VirtualMachine.Provisioning.PromoteDisks
Profile.Clear	VirtualMachine.Provisioning.PutVmFiles
Profile.Create	VirtualMachine.Provisioning.ReadCustSpecs
Profile.Delete	VirtualMachine.State.CreateSnapshot
Profile.Edit	VirtualMachine.State.RemoveSnapshot
Profile.View	VirtualMachine.State.RenameSnapshot
Resource.ApplyRecommendation	VirtualMachine.State.RevertToSnapshot

Sample Program Overview

This appendix lists the available sample programs and provides some information about each program.

The appendix includes the following topics:

- “Java Sample Programs” on page 191
- “C# DotNet Sample Programs” on page 194

Java Sample Programs

When you download the SDK, you can find the java sample programs and related files in the following directories.

- SDK\samples\Axis\java\com\vmware – Top-level directory for Java samples
- SDK\samples\Axis\java\com\vmware\apputils – Helper applications.
- SDK\samples\Axis\java\com\vmware\samples – Samples. Details listed in [Table E-1](#).
- SDK\samples\Axis\java\com\vmware\security – Credential store utility
- SDK\samples\Axis\java\com\vmware\vim and SDK\samples\Axis\java\com\vmware\vim25 – Stub directories. The vim directory contains stubs for servers that precede VirtualCenter 2.5, the vim25 directory contains stubs for VirtualCenter 2.5 and later, including vSphere 4.0 and later.

Table E-1. Java Sample Programs

Directory	Example	Description
alarm	VMPowerStateAlarm	Creates an alarm to monitor a virtual machine's power state.
ciminfo	CIMInfo	Includes functions for information retrieval, such as functions that create an association traversal filter, retrieve an instance of a specified CIM class, get a single instance of the class provided, and run the specified operation for that instance of the class.
	CIMUtil	CIM utility functions, including a wrapper that provides an enumerator over all the instances of the specified class and some simpler wrappers.
events	EventFormat	Retrieves and formats the last event from the host daemon or vpxd. Includes a function that formats the event message.
	EventHistoryCollectorMonitor	Demonstrates how to create and monitor an EventHistoryCollector. Uses the latestPage property of EventHistoryCollector to filter the events.

Table E-1. Java Sample Programs (Continued)

Directory	Example	Description
	VMEventHistoryCollectorMonitor	Standalone client that demonstrates how to perform the following tasks: <ul style="list-style-type: none"> ■ Logging into the web service. ■ Creating EventHistoryCollector filtered for a single virtual machine. ■ Monitoring events using the latestPage property of the EventHistoryCollector.
general	Browser	Prints 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.
	Connect	Connects to an ESX/ESXi system or a vCenter Server system.
	Create	Creates a managed entity such as a folder, datacenter, or cluster.
	Delete	Deletes a managed entity from the inventory tree. The managed entity can be a virtual machine, a ClusterComputeResource, or a folder.
	GetUpdates	Demonstrates how to use the PropertyCollector 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.
	LicenseManager	Demonstrates uses of the licensing API using License managed object reference.
	Move	Moves a managed entity from its current location in the inventory to a new location in a specified folder
	PropertyCollector	Illustrates the use of the PropertyCollector API.
	RemoveManagedObject	Destroys or unregisters a managed inventory object like a Host, VirtualMachine, Folder, and so on.
	Rename	Renames a managed entity object.
	SearchIndex	Illustrates the use of the SearchIndex API.
	SimpleClient	Connects to the server, logs in, lists the inventory contents (managed entities) at the console, and logs out.
	TaskList	Displays a list of tasks performed on a specified managed object.
host	AddVirtualNic RemoveVirtualNic	Adds a virtual NIC to a port group on a virtual switch and removes a virtual NIC from a port group.
	AddVirtualSwitch RemoveVirtualSwitch	Adds a virtual switch to a host and removes a virtual switch from the host.
	AddVirtualSwitchPortGroup RemoveVirtualSwitchPortGroup	Adds a port group to a virtual switch.
	ColdMigration	Puts virtual machine files into a specified datacenter and datastore and registers and reconfigures the corresponding virtual machine.
httpfileaccess	GetVMFiles	Retrieves configuration files, snapshots files, log files, and virtual disk files of a virtual machine and places them on the system on which the program is run.
performance	Basics	Displays available performance counters or other metadata for an ESX/ESX host.
	History	Reads performance measurements from the current time, or from a specified start time, for a specified duration.
	PrintCounters	Writes 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.

Table E-1. Java Sample Programs (Continued)

Directory	Example	Description
	Realtime	Displays performance measurements from the current time at the console.
	VItop	An <code>esxtop</code> -like sample application that lets administrators specify the CPU and memory counters by name to obtain metrics for a specified host.
	VIUsage	Creates a GUI for graphical representation of the counters.
scheduling	DeleteOneTimeScheduledTask	Demonstrates deleting a <code>ScheduledTask</code> .
	OneTimeScheduledTask	Demonstrates creating a <code>ScheduledTask</code> using the <code>ScheduledTaskManager</code> .
	WeeklyRecurrenceScheduledTasks	Demonstrates creating a weekly recurrent scheduled task.
scsilun	SCSILunName	Displays the <code>CanonicalName</code> , <code>Vendor</code> , <code>Model</code> , <code>Data</code> , <code>Namespace</code> and <code>NamespaceId</code> of the host's SCSI LUN.
simpleagent	SimpleAgent	Accesses the local credential store to obtain a single user account to log in to the specified server.
	CreateUsers	Creates a user account and password and stores them in the local credential store.
vm	VMClone	Locates an existing virtual machine on the vCenter Server system, makes a template from this virtual machine, and deploys instances of the template onto a datacenter.
	VMCreate	Creates a virtual machine. Different command-line input creates the virtual machine in different ways.
	VMDeltaDisk	Creates a delta disk on top of an existing virtual disk in a virtual machine, and simultaneously removes the original disk using the <code>reconfigure</code> API. Use delta disks in conjunction with linked virtual machines.
	VMLinkedClone	Creates a linked virtual machine from an existing snapshot.
	VMotion	Checks whether migration with <code>VMotion</code> is feasible between two hosts. Performs the migration if the hosts are compatible.
	VMpowerOps	Performs power operations on a virtual machine.
	VMPromoteDisks	Consolidates a linked virtual machine by using the <code>VirtualMachine.PromoteDisks</code> method.
	VMReconfig	Reconfigures a virtual machine. Includes reconfiguring the disk size, disk mode and so on.
	VMRelocate	Relocate a linked virtual machine using disk move type.
	VMSnapshot	Performs virtual machine snapshot operations
version	Samples in the version directory illustrate running against hosts that have different versions of vSphere or VMware Infrastructure installed. The <code>HostPowerOps</code> sample includes running against Virtual Infrastructure 2.0. The other samples apply to Virtual Infrastructure 2.5 and later versions including vSphere 4.0.	
	<code>displaynewpropertieshost/</code> <code>DisplayNewPropertiesHost</code> <code>DisplayNewPropertiesHostV25</code>	Displays properties of the host based on the API version supported by ESX/ESXi or vCenter Server.
	<code>displaynewpropertiesvm/</code> <code>DisplayNewPropertiesVM</code> <code>DisplayNewPropertiesVMV25</code>	Displays properties of a virtual machine based on the API version supported by the host to which you connect.
	<code>getvirtualdiskfiles/</code> <code>GetVirtualDiskFiles</code> <code>GetVirtualDiskFilesV25</code>	Searches the virtual disk files in all the datastores available in a specified host using the properties <code>ControllerType</code> and <code>ThinProperty</code> .

Table E-1. Java Sample Programs (Continued)

Directory	Example	Description
	hostpowerops/ HostPowerOps HostPowerOpsV25	Powers down a host. This samples illustrates how to run a sample against hosts of multiple versions, including Virtual Infrastructure 2.0.
	installhostpatch/ InstallHostPatch InstallHostPatchV25	Uses the <code>HostPatchManager</code> managed object to upgrade the components in an ESXi system.
	querymemoryoverhead/ QueryMemoryOverhead QueryMemoryOverheadV25	Determines the amount of memory overhead necessary to power on a virtual machine with the specified characteristics.
	recordsession/ RecordSession RecordSessionV25	Demonstrates recording a session.

C# DotNet Sample Programs

The C# (DotNet) sample programs are in `SDK\samples\DotNet`

The samples include a `GeneratingStubs.txt` file and a `readme_dotnet.html` file at top level. The `readme` file explains how to build the examples using Visual Studio 2005 or Visual C# 2005 Express.

Table E-2. C# (DotNet) Sample Programs

Example	Description
AddVirtualNic	Adds a virtual NIC to the ESX/ESXi system. First specifies a <code>HostVirtualNicSpec</code> , and then adds the NIC to the host.
AddVirtualSwitch	Adds a virtual switch to the ESX/ESXi system.
AddVirtualSwitchPortGroup	Adds a virtual port group to the ESX/ESXi system.
AppUtil	Contains the following utility applications: <ul style="list-style-type: none"> ■ <code>AppUtil</code> – Utility application that drives the user input mechanism for other samples and includes some other utility functions. ■ <code>ArgumentHandlingException</code> – Handles command-line exceptions. Used by <code>AppUtil</code>. ■ <code>CertPolicy</code> – Handles certification problems by displaying informational messages. ■ <code>ClientUtil</code> – Client utilities related to prompting the user and logging. Used by <code>AppUtil</code>. ■ <code>Log</code> – Logger to file or console. ■ <code>OptionSpec</code> – Option parsing utility. ■ <code>ServiceUtils</code>, <code>ServiceUtilsV25</code> – Utilities for connecting to the server. ■ <code>VersionUtil</code> – Utility that determines the name space and supported versions. ■ <code>VMUtils</code> – Utility that sets values for a basic virtual machine. Some of the setup, such as adding a floppy disk drive, might not always be needed.
Basics	Uses the <code>PerformanceManager</code> for basic monitoring.
Browser	Retrieves the contents of the <code>ServiceInstance</code> starting at the root folder, and prints a listing of <code>ManagedEntity</code> objects. Optionally, obtains properties for a specific type, or by default, for <code>ManagedEntity</code> itself.
CIMInfo	CIMInfo versioning sample that retrieves the details of <code>CIM_Fan</code> like <code>Activecooling</code> , <code>Caption</code> , <code>CommunicationStatus</code> and so on.
Coldmigration	Migrates a powered off virtual machine from one host to another.
Connect	Simple example that logs in and logs out.
Create	Creates a <code>Folder</code> , <code>Cluster</code> , <code>Datacenter</code> or standalone host. Prompts the user for the item to create and where to put the item, for example, in a folder.

Table E-2. C# (DotNet) Sample Programs (Continued)

Example	Description
CreateUser	Creates a user. Specifies permissions for the user using the <code>AuthorizationManager</code> .
CredentialStorePSCmdLets	Multiple commandlets for managing the credential store.
Delete	Deletes a managed entity.
DeleteOneTimeScheduledTask	Extracts a task from <code>scheduledTaskManager</code> and deletes it. This sample is well commented and illustrates using the <code>PropertyCollector</code> . You can create the task using the <code>OneTimeScheduledTask</code> example.
DisplayNewProperties	Retrieves the specified set of properties for the given managed object reference into an array of result objects (returned in the same order as the property list)
DisplayNewPropertiesHost	Displays properties of an ESX/ESXi host. The properties displayed depend on the version of the host.
DisplayNewPropertiesVM	Displays a set of properties for a virtual machine. The properties displayed depend on the version of the software on the host.
EventFormat	Retrieves and formats the last event on the ESX/ESXi or vCenter Server system. Demonstrates event formatting.
EventHistoryCollectorMonitor	Creates an <code>EventHistoryCollector</code> and monitors the corresponding events.
GetUpdates	Retrieves updates for a virtual machine or an ESX/ESXi host.
GetVirtualDiskFiles	Retrieves the virtual disk files from a host's datastores.
GetVMFiles	Downloads the files in the virtual machine configuration directory as well as the files in the virtual machine snapshot, suspend, and log directories. Writes progress to the console.
History	Displays the performance measurements of a specified counter of a specified ESX/ESXi for a specified duration, or 20 minutes (default) at the console.
HostPowerOps	Performs reboot, shutdown, or suspend (power off to standby) operations on an ESX/ESXi system.
LicenseManager	Displays licensing information. The user can specify a license server,
MobStartPage	Includes a program, image files, and HTML files for displaying a Managed Object browser.
Move	Moves a managed entity from one folder to another.
OneTimeScheduledTask	Creates a <code>ScheduledTask</code> that powers off a virtual machine and schedules the task using a <code>OnceTaskScheduler</code> . You can delete the task using the <code>DeleteOneTimeScheduledTask</code> commandlet.
PrintCounters	Defines a <code>printEntityCounters</code> function that prints counters for a virtual machine, host, or resource pool.
PropertyCollector	Illustrates use of the <code>PropertyCollector</code> .
QueryMemoryOverhead	Illustrates use of the <code>QueryMemoryOverhead</code> method. The folder includes two examples, <code>QueryMemoryOverheadV25</code> uses the currently valid <code>QueryMemoryOverheadEx</code> method, <code>QueryMemoryOverhead</code> uses the deprecated <code>QueryMemoryOverhead</code> method.
RealTime	Displays the current performance measurements of selected CPU counters of any specified virtual machine at the console.
RecordSession	Records a session and allows you to retrieve a specified set of properties for a specified managed object reference into an array of result objects.
RemoveManagedObject	Removes a host from a cluster or a virtual machine from a host. Handles errors, for example, if the host is not in a cluster, by printing that information to the command line.
RemoveVirtualNic	Removes a virtual NIC from the ESX/ESXi system.
RemoveVirtualSwitch	Removes a virtual switch from the ESX/ESXi system.
RemoveVirtualSwitchPortGroup	Removes a virtual port group from the ESX/ESXi system.
Rename	Renames a managed entity.

Table E-2. C# (DotNet) Sample Programs (Continued)

Example	Description
SCSILunName	Prints the virtual machine file system volumes on a specified SCSI LUN.
SearchIndex	Illustrates the use of the SearchIndex API.
SimpleClient	Demonstrates connecting to a service, logging on to service, obtaining service content, and logging out from the service.
SSPI	Illustrates how to use an SDK application with Microsoft SSPI.
SSPICIMClient	Illustrates how to use an CIM client application with Microsoft SSPI.
TaskList	Displays currently running tasks and their state.
VMClone	Clones a virtual machine.
VMCreate	Creates a virtual machine.
VMEventHistoryCollectorMonitor	Returns all events on the latest page on the EventHistoryCollector.
VMotion	Validates that migration with VMotion is feasible between two hosts, and performs the migration if the hosts are compatible.
VMPowerOps	Retrieves a reference to a virtual machine and invokes power operations specified on the command line on that virtual machine.
VMPowerStateAlarm	Creates an alarm that monitors virtual machine state and sends email if the virtual machine power is off. Includes error handling, for example, when the commandlet is attempted with an ESX/ESXi host as a target.
VMReconfig	Reconfigures a virtual machine by changing its memory, cpu, disk, nic, or cd.
VMSnapshot	Performs snapshot operations such as create, revert to, remove, remove all, and so on.
VMware.Security.CredentialStore	Illustrates use of the VMware credential store.
WatchVM	Monitors updates on a virtual machine using the PropertyCollector.
WeeklyRecurrenceScheduledTask	Creates a task that reboots a virtual machine once a week.

Index

A

- accessing property values **22**
- accessor (getter) method **19, 21**
- Action object **148**
- Active Directory **57**
- AddHost_Task **127**
- administrator role **186**
- admission control with expandable resource pools **125**
- affinity
 - CPU **101**
 - DRS **126**
 - memory **101**
- Alarm object **145, 146**
- AlarmAction object **148**
- AlarmExpression object **147**
- AlarmManager object **145**
- alarms
 - creating **145**
 - deleting **149**
 - disabling **149**
 - email message **148**
 - introduction **141**
 - listing **145**
 - red or yellow **147**
 - running script **148**
 - samples **149**
- AlarmSpec object **146**
- anti-affinity, DRS **126**
- API versions **29, 30**
- applying permissions **62**
- AppUtil (.NET connection object) **19**
- apputils **30, 31**
- architecture, client-server **17**
- array properties **23**
- asynchronous tasks **131**
- authenticating users **63**
- authentication **53**
- authorization **53**
- AuthorizationManager **53, 58, 59**
- automated login **63**
- Axis **26**
 - vSphere web service connection **19**
 - WSDL files **18**

B

- bond bridge **90**
- boot options **100**

C

- C#
 - credential store library **63**
 - datatypes, map to XML **20**
 - helper classes **31**
 - sample programs **194**
- certificates **54**
- changing permissions **61**
- CHAP **78**
- childEntity property **35**
- chunked data **52**
- CIM APIs **14**
- client applications
 - deploying **167**
 - patterns **25, 53**
- client data synchronization **50**
- client-server architecture **17**
- client-side proxy interface **18**
- ClusterComputeResource **121, 127**
- clusters
 - adding hosts **127**
 - configuring **127**
 - creating **127**
 - overview **123**
 - reconfiguring **128**
 - VMware DRS **126**
 - VMware HA **126**
- cold migration **107**
- collecting properties **39**
- complex content **170**
- concurrent sessions **63**
- config.xml file **162**
- configuration files **160**
- ContainerView **45**
- core dumps **82**

CPU

- affinity **101**
- allocation **100**
- expandable reservation **124**
- limit **124**
- number of CPUs **101**
- resource pools **122**
- shares **122, 124**
- CreateAlarm operation **146**
- CreateCluster **127**
- CreateFilter **171**
- CreateUser sample **65**
- CreateVM_Task **96**
- creating custom events **144**
- creating datastores **78**
- creating ScheduledTask object **136**
- creating UserAccount objects **62**
- creating virtual machines **96**
- credential store
 - automated login **63**
 - backing file **64**
 - client API libraries **64**
 - CreateUser sample **65**
 - file format **65**
 - roles **65**
 - SimpleAgent sample **65**

D

- data chunk **52**
- data objects **17**
- data structures **22**
- Datastore object **72, 79**
- datastores
 - access **79**
 - creating **78**
 - definition **73**
 - managing **78**
 - removing **81**
 - updating **81**
 - VMFS **79, 80**
- datatypes for MOB usage **172**
- default log level setting **162**
- default port groups **89**
- deleting alarms **149**
- deleting permissions **61**
- deleting scheduled tasks **139**
- delta disks **113**
- deploying client applications **167**
- DestroyChildren method **125**
- destroying resource pools **125**
- devices for virtual machines **102**
- diagnostic bundles **166**
- diagnostic partitions **82, 83**

DiagnosticManager

- introduction **163**
- log descriptor values **164**
- managed object **164**
- DiagnosticManagerLogCreator enumeration **164**
- diagnostics **159**
- disabling alarms **149**
- disk backings for linked VMs **110**
- disk partitions
 - configuration **75**
 - configuring **75**
 - HostStorageSystem use **80**
- Distributed Power Management **126**
- DRS **123, 126, 128**
- DTM **126**
- dumps **82**

E

- embedded SMNP agent **92**
- enumerated types **22**
- escape character **23**
- ESX/ESXi
 - config.xml file **162**
 - embedded SNMP agent **92**
 - hypervisor **13**
 - inventory **37**
 - log file **160**
 - log level **162**
 - managed hosts **37**
 - PerfInterval **153**
 - sample log **160**
 - standalone hosts **37**
 - user model **54**
- Event objects **141, 143**
- EventManager object **142**
- events
 - creating **144**
 - formatting messages **143**
 - message content **143**
 - persistance **142**
- expandable reservations **124**
- expiration of sessions **29**

F

- failure detection **129**
- Fibre Channel NPIV settings **102**
- file format, credential store **65**
- file system volumes **75**
- files of virtual machines **98**
- filters **39**
- folders **35**
- force mount **82**

G

generating logs **163**
 getting data from a server **21**
 granting privileges **61**
 groups **57**
 assigning users **58**
 membership **62**
 permissions **62**
 guest OS, customizing **105**

H

HA Clusters, managing **129**
 hardware version **99**
 helper classes **31**
 C# **31**
 java **31**
 sample applications **30**
 hierarchies **34, 35**
 historical intervals **156**
 HistoryCollector overview **140, 145**
 host network isolation **129**
 hostd.log **160**
 HostDatastoreBrowser **71**
 HostDatastoreSystem **71**
 HostDiagnosticPartition **72, 82**
 HostDiskPartitionInfo **75**
 HostLocalAccountManager object **53, 57, 58**
 HostNetworkConfig **85**
 HostNetworkSecurityPolicy **90**
 HostNetworkSystem **85**
 HostNetworkTrafficShapingPolicy **90**
 HostNicTeamingPolicy **90**
 hosts
 adding to cluster **127**
 clusters, creating **127**
 configuring **68**
 disconnecting **69**
 health status **67**
 lifecycle **69**
 management objects **67**
 network policies **90**
 NUMA information **68**
 reboot **69**
 reconnecting **69**
 retrieve information **67**
 time **70**
 HostStorageSystem **81**
 HostStorageSystem object **71, 75, 81**
 HostSystem **67**
 HostVirtualSwitchBondBridge **90**
 HTTP access **173, 174, 175, 176**
 HTTPS server access **19**

hypervisor **13, 106**

I

import client libraries **19**
 incremental property retrieval **40**
 indexed arrays **23**
 interface **18**
 inventory
 folders **35**
 introduction **38**
 managed and standalone hosts **37**
 permissions **37**
 privileges **36**
 root folders **35**
 inventory management privileges **36**
 inventory traversal **45**
 IP gateway **92**
 IP route configuration **92**
 IPv6 **91**
 iSCSI
 ISO file mounting **87**
 storage **76**
 virtual machine datastore **87**
 VMkernel **77**

J

java
 credential store library **63**
 datatypes, map to XML **20**
 helper classes **31**
 sample programs **191**
 TaskList application **135**

K

key-based arrays **23**

L

language-specific classes and methods **19**
 LicenseAssignmentManager **54, 66**
 LicenseManager **54, 66**
 licenses
 adding **66**
 introduction **66**
 removing **66**
 limit **124**
 linked virtual machines
 creating **110**
 disk backings **110**
 from running point **111**
 from snapshot **110**
 relocating **112**
 Linux pluggable authentication module **55**
 local user accounts **54**

- log descriptor values **164**
- log files
 - overview **160**
 - vCenter Server **162**
 - virtual machines **161**
- log in **57**
- log level
 - default settings **162**
 - ESX system **162**
 - modifying **162**
 - settings **162**
 - vCenter Server system **163**
- logs
 - generating **163**
 - hostd **160**
 - server **160**
 - system **160**

M

- MAC address **101**
- manage VMFS **81**
- managed hosts **37**
- Managed Object Browser **167, 172**
- managed object references **17**
- managed objects
 - definition **17**
 - handling local user accounts **54**
- ManagedObjectReference XML schema **169**
- managing storage **71**
- mapping XML datatypes to java and C# datatypes **20**
- maxObjects **52**
- maxObjectsUpdates **52**
- memory
 - affinity **101**
 - allocation **100**
 - expandable reservation **124**
 - limit **124**
 - overhead **70**
 - shares **124**
- messages (event), formatting **143**
- Methods object **19**
- Microsoft .NET **18, 19**
- migration **107**
- MOB **167, 172**
- monitoring performance **151**
- multipath management **76**
- multiple API versions **29**
- mutator (setter) method **19**
- Mutual CHAP **78**

N

- nested data objects **172**

- nested properties **22, 23**
- network access **19**
- Network Information System (NIS) **57**
- Network object **85**
- network policies **90**
- networking
 - adding services **92**
 - configuration **88**
 - introduction **86**
 - IPv6 **91**
 - objects and methods **85**
 - virtual machines **101**
- NFS **87**
- NIC teaming **90**
- NIS **57**
- NPIV **102**
- NTP service
 - adding **92**
 - querying **70**

O

- ObjectSpec definition as XML Schema **171**
- obtaining statistics **154**
- obtaining values from nested properties **23**
- operations
 - passing complex datatypes **169**
 - privileges required **177**
 - required privileges for **177**
- optimizing query response time **156**

P

- PAM **55**
- parent property **35**
- partitions **75, 80**
- passing complex datatypes to operations **169**
- PerfCounterInfo
 - matching PerfMetricId CounterId **153**
 - properties **152**
- PerfEntityMetricBase data object **155**
- PerfInterval data objects
 - ESX/ESXi system **153**
 - vCenter Server system **155**
- PerfMetricId CounterId **153**
- performance **151**
- performance data, default collection **153**
- PerformanceManager
 - overview **155**
 - samples **157**
- PerfQuerySpec **154**
- Perl
 - credential store library **63**
 - vSphere SDK for Perl **14**

- permissions **61**
 - AuthorizationManager **58**
 - changing **61**
 - definition **56**
 - deleting **61**
 - group membership **62**
 - information **61**
 - inheritance **37**
 - inventory management **37**
 - setting **61**
 - users and groups **62**
 - physical network adapter **88**
 - pnic **88**
 - port 443 (server access) **19**
 - port groups **87, 89**
 - power management **126**
 - PowerCLI **14**
 - powering on VM **139**
 - PowerOnMultiVM_Task **104**
 - PowerShell **14**
 - Privileges **55**
 - privileges
 - administrator role **186**
 - ESX/ESXi operations **177**
 - for inventory management **36**
 - format **36, 55**
 - inventory management **36**
 - invoking operations **177**
 - operations, required for **177**
 - properties, required for **185**
 - reading object properties **185**
 - roles **60**
 - vCenter Server operations **177**
 - properties
 - arrays **22**
 - complex **22**
 - enumerated types **22**
 - filters **50**
 - incremental retrieval **40**
 - retrieving **39**
 - simple data types **22**
 - property paths **22**
 - property values **22**
 - PropertyCollector
 - example **43**
 - filters **50**
 - introduction **39**
 - methods **41**
 - performance **52**
 - SearchIndex **52**
 - view objects **40**
 - WaitForUpdatesEx **50**
 - PropertyFilter **50**
 - PropertyFilterSpec **133, 170**
 - PropertySpec XML schema **171**
 - proxy interface **18**
- ## Q
- QuickStats object **151**
- ## R
- recentTask array **134**
 - recurring operations **137**
 - recursive traversal **48**
 - red alarms **147**
 - rescan **75**
 - reservations **124**
 - resignaturing **81**
 - resource allocation **122**
 - resource management **122**
 - resource management objects **121**
 - resource pools
 - adding children **126**
 - allocating resources **123**
 - creating **124**
 - deleting **125**
 - expandable reservation **124**
 - hierarchy **122**
 - management guidelines **123**
 - moving into resource pools **126**
 - root resource pool **124**
 - RetrieveOptions **52**
 - RetrievePropertiesEx method **40, 49**
 - retrieving data from a server **21**
 - return datatypes **23**
 - role ID **60**
 - roles
 - and security management **60**
 - creating **61**
 - creating from sample roles **61**
 - credential store **65**
 - definition **56**
 - modifying **61**
 - sample roles **60**
 - system roles **60**
 - user roles **60**
 - root folders **35**
 - root resource pool **124**
 - route configuration **92**
 - RunScriptAction **148**
- ## S
- sample applications **30, 31**
 - sample code reference **93**
 - sample ESX log **160**

- sample programs
 - C# **194**
 - Java **191**
 - sample roles **60, 61**
 - samples
 - alarm **149**
 - datastore operations **83**
 - overview **140**
 - PerformanceManager **157**
 - storage operations **83**
 - sampling periods **153**
 - ScheduledTask **136**
 - deleting **139**
 - introduction **136**
 - power on VM **139**
 - ScheduledTaskManager **136**
 - scheduling recurring operations **137**
 - scheduling vCenter Server operations **137**
 - SearchIndex **52**
 - security **55, 65, 90**
 - segmented data **52**
 - SelectionSpec **48**
 - SendEmailAction **148**
 - server logs **160**
 - ServiceContent **26**
 - ServiceInstance **21, 26, 35**
 - session tokens **28**
 - SessionManager **54, 63**
 - SessionManager.Login method **57**
 - sessions
 - concurrent **63**
 - default expiration **29**
 - state **28**
 - token **28**
 - setting permissions **61**
 - shares **124**
 - simple client application **27**
 - SimpleAgent sample **65**
 - Snapshots **108**
 - snapshots
 - creating **109**
 - removing **112**
 - reverting **109**
 - virtual machine **98**
 - SNMP **92**
 - SOAP **18, 26, 28**
 - SOAP tools **19**
 - software iSCSI **77**
 - special characters **23**
 - SSL certificate **25**
 - standalone hosts **37**
 - standby mode **69**
 - statistics
 - levels **156**
 - obtaining **154**
 - storage
 - architecture **72**
 - introduction **72**
 - iSCSI **76**
 - management objects **71**
 - multipath management **76**
 - overview **72**
 - rescan **75**
 - virtual machines **72**
 - storage API, choosing **74**
 - storage VMotion **108**
 - SunFakeTrustSocketFactory **25**
 - supported version **30**
 - sVMotion **108**
 - synchronous tasks **131**
 - system logs **160**
 - system roles **60**
- ## T
- task infrastructure **131**
 - Task object **131, 132**
 - TaskInfo
 - object **133**
 - properties **134**
 - TaskInfoState values **134**
 - TaskList java application **135**
 - TaskManager **133**
 - TaskManager object **131, 132**
 - tasks
 - asynchronous **131**
 - availability **133**
 - synchronous **131**
 - TaskScheduler subtypes **137**
 - templates **97**
 - templates, converting to virtual machine **98**
 - threads **28**
 - time zone **70**
 - tokens **28, 52**
 - traffic shaping **90**
 - traversal, recursive **48**
 - TraversalSpec **42, 45**
 - troubleshooting **159**
- ## U
- UML diagrams **15**
 - Unified Modeling Language **15**
 - URL syntax for HTTP access **174**
 - user accounts **54, 62**
 - creating **58**
 - on ESX/ESXi and vCenter **55**

- user roles **60**
 - UserAccount object **62**
 - UserDirectory **53, 57**
 - users **57**
 - assign to group **58**
 - credential store **65**
 - overview **57**
 - permissions **62**
 - utility classes **30, 31**
- V**
- vCenter Server
 - definition **13**
 - inventory **37**
 - log files **162**
 - scheduling operations **137**
 - user model, overview **55**
 - versions **29, 30**
 - view objects **40**
 - VimPortType (Axis methods object) **19, 26, 42**
 - VimService (Microsoft .NET methods object) **19**
 - VimServiceLocator **26**
 - VimServiceLocator (Axis connection object) **19**
 - vimServiceVersions.xml **29**
 - Virtual **95, 107**
 - virtual machines
 - access **98**
 - adding devices **102**
 - and deleted resource pools **125**
 - attributes **97**
 - cloning **97, 98**
 - configuring **99**
 - CPU allocation **100**
 - creating **96**
 - creating from scratch **96**
 - files **98**
 - hardware version **99, 106**
 - log files **161**
 - MAC address **101**
 - management objects **95**
 - managing **115**
 - memory allocation **100**
 - migrating **107**
 - moving into resource pools **126**
 - network interface **87**
 - overhead **70**
 - power operations **104**
 - powering on **139**
 - registering **104**
 - snapshot files **98**
 - snapshots **109**
 - storage access **72**
 - templates **97, 98**
 - upgrading **106**
 - VirtualMachineConfigSpec **96**
 - VMware DRS **104**
 - virtual port groups, adding **89**
 - virtual switches **87**
 - adding **89**
 - overview **87**
 - VirtualMachine
 - nested properties **22**
 - overview **22**
 - VirtualMachineConfigSpec **96, 97**
 - VIX API **14**
 - VMFS
 - force mount **82**
 - managing **81**
 - provisioning **78, 79**
 - resignaturing **81**
 - VMFS datastores **73, 79, 80, 81**
 - VMkernel availability report **161**
 - VMkernel network interface **89**
 - adding **89, 90**
 - VMotion **87, 108**
 - VMware DRS **126**
 - and clusters **123**
 - introduction **126**
 - powering on VMs **104**
 - standby mode **69**
 - VMware DTM **126**
 - VMware HA
 - overview **123**
 - primary and secondary hosts **129**
 - VMware Tools **105**
 - vNetwork Standard Switch **85, 86, 88**
 - volumes **75**
 - vSphere
 - object model **173**
 - security model **55**
 - vSphere SDK for Perl **14**
 - vSS environment **86, 88**
- W**
- WaitForUpdatesEx method **40, 50**
 - WaitOptions **52**
 - web service network access **19**
 - Windows PowerShell **14**
 - WSDL **18, 19**
- X**
- X.509 certificates **54**
 - XML schema
 - ManagedObjectReference **169**
 - PropertyFilterSpec **170**
 - PropertySpec **171**

XML Schema Primitives to .NET mapping **20**

XML Schema Primitives to java mapping **20**

xsd datatypes **22**

Y

yellow alarms **147**