vSphere Web Services SDK Programming Guide

vSphere Web Services SDK 6.0

EN-001411-02

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

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

Contents

About This Book

The *vSphere Web Services SDK Programming Guide* provides information about developing applications using theVMware® vSphere Web Services SDK 6.0.

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

Table 1. Revision History

Revision Date	Description
04Sep2015	Updated information about migrating VMs with VMotion across data centers.
12Mar2015	vSphere 6.0 - Rewrote "Exporting a Virtual Application" section in Virtual Applications chapter.
19Sep2013	vSphere 5.5 – Added a C# example of using LoginByToken; clarified limitation for HA clusters.
10Sep2012	vSphere 5.1 – Added information about using the SessionManager.LoginByToken method; added information about distributed virtual switches.
24AUG2011	vSphere 5.0 - Revised performance manager chapter. Added information about: unset properties, using vCenter to access host data, and using the QueryConfigOption to add devices; emphasized ListView instead of TaskManager; clarified limits and limitations of Linked Virtual Machines; updated samples in chapters 3,5,14, and 16; replaced information about Axis bindings with JAX-WS; and updated paths to samples supplied with SDK.
13JUL2010	Restructured manual and added chapters about host, storage, and networking. Revised property collector chapter and added appendix about HTTP access.
07MAY2009	vSphere Web Services SDK 4.0 Programming Guide.

Intended Audience

This book is intended for anyone who needs to develop applications using the vSphere Web Services SDK. Developers typically create client applications using Java 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.

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 High Availabiltiy and VMware VMotion. 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 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 ESXi hypervisor is capable of supporting multiple virtual machines and other virtual components, such as storage and networks.
- vCenter Server provides central management for all of the components of a virtualized environment, including multiple ESX/ESXi host systems, clusters, storage, and distributed virtual switches. It is distributed in two package formats:
  - Windows-based software services.
  - Linux-based VMware vCenter Server Appliance.
- vSphere Web Client is a GUI to manage vSphere. It provides the UI platform that you use to integrate your solution with vSphere. The vSphere Web Client also includes a server-side Java platform. You can develop Java plugins that use the vSphere Web Services SDK to communicate with vSphere servers. See the vSphere Web Client SDK for more information about building UI and service plugins for the vSphere Web Client.

For more information about 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:

- *vSphere Virtual Machine Administration Guide*

- *vCenter Server and Host Management 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

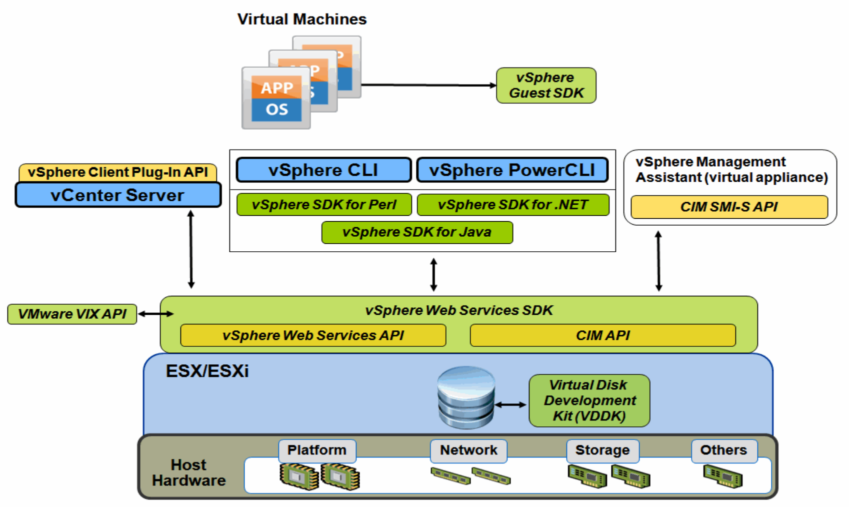
VMware vSphere PowerCLI provides a Windows PowerShell interface to the vSphere API. vSphere PowerCLI includes PowerShell Cmdlets for administering vSphere components. In addition, the vSphere PowerCLI package includes the vSphere SDK for .NET for developers who want to create their own applications.

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.

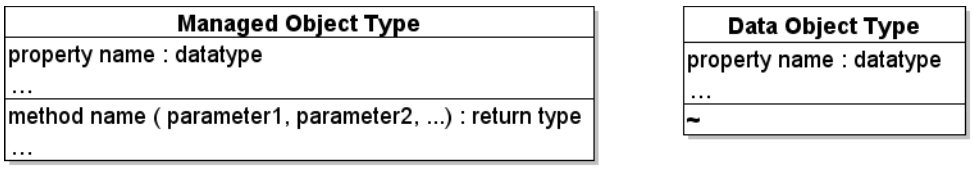
The code fragments in this guide are in part based on the Java sample applications, but present code that does not require utility applications to run.

See Appendix E, "Sample Program Overview," on page 245 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



2 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:

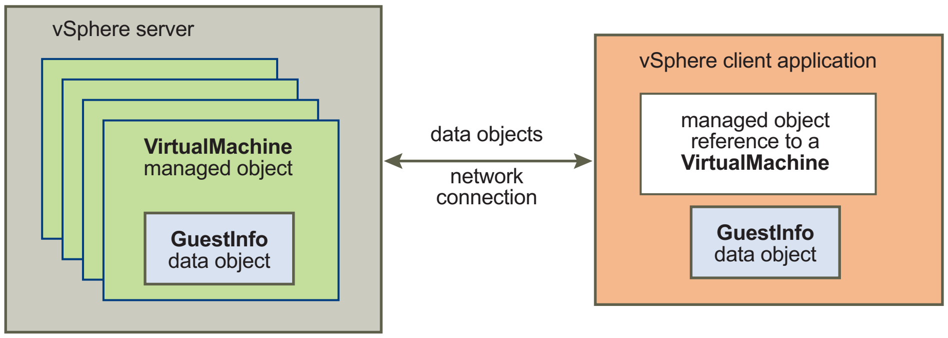
vSphere Client-Server Architecture

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


vSphere server
VirtualMachine managed object
GuestInfo data object
data objects
network connection
vSphere client application
managed object reference to a **VirtualMachine**
GuestInfo data object

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 167.

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 support 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.

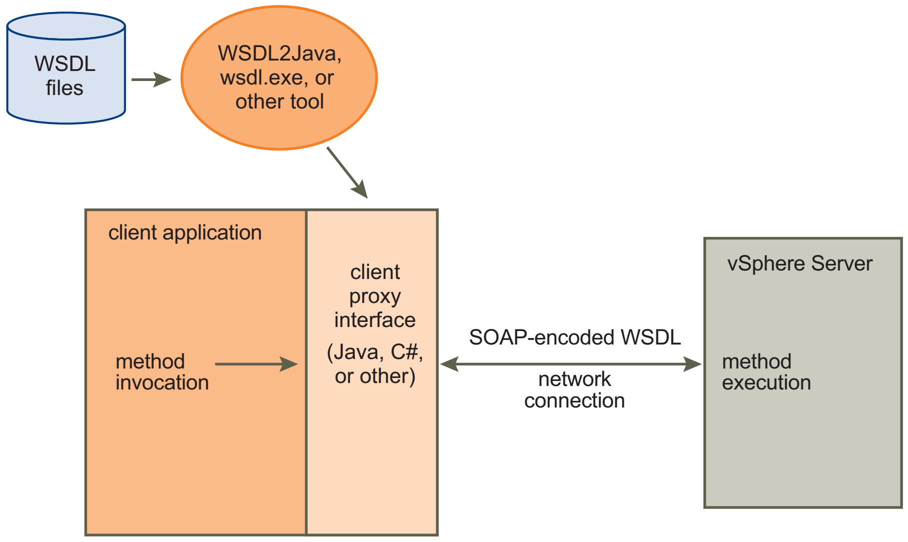
WSDL Files and the Client-Side Proxy Interface

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 JAX-WS wsimport, 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 JAX-WS toolkit. If the versions of Java and JAX-WS on your development platform are the same as those used to generate the proxy interface shipped in the SDK, you do not have to generate client-side proxy code from the WSDL. See the *Developer's Setup Guide* for information about 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	`get`*`PropertyName`* and `set`*`PropertyName`* methods defined for data objects	`get` and `set` methods defined for properties
vSphere API data objects	Data objects in the vSphere API (see the *vSphere API Reference*) 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 27.

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
`xsd:base64binary`	`byte[]`	`Byte[]`
`xsd:boolean`	`boolean`	`Boolean`
`xsd:byte`	`byte`	`SByte`
`xsd:dateTime`	`java.util.Calendar`	`DateTime`
`xsd:decimal`	`java.math.BigDecimal`	`Decimal`
`xsd:double`	`double`	`Double`
`xsd:float`	`float`	`Single`
`xsd:int`	`int`	`Int32`
`xsd:string`	`java.lang.String`	`String`

Access to Managed Objects

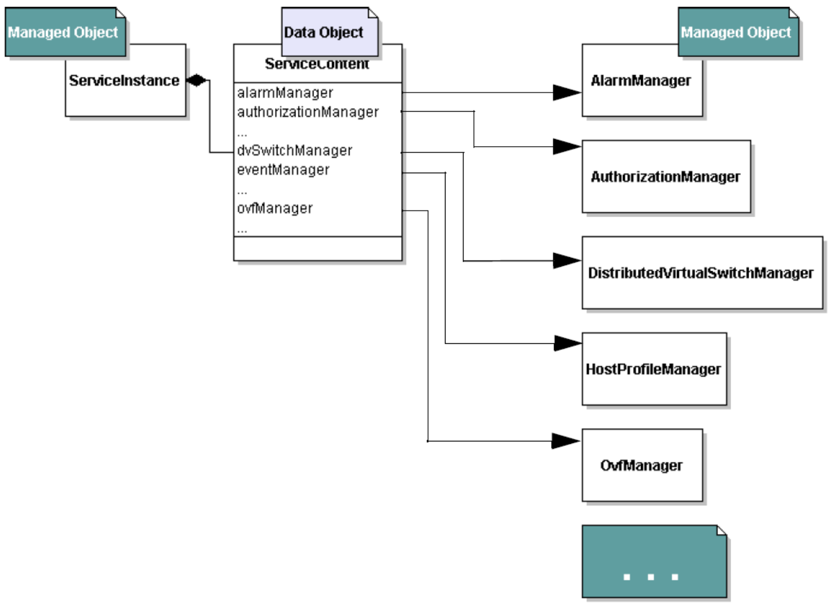
Your client application obtains access to managed objects through the `ServiceInstance` managed object and its associated `ServiceContent` data object. The `ServiceContent` data object contains managed object references to services and manager entities, and to the root folder of the inventory.

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 38 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.

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

Working with Data Structures

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

- Simple data types, such as a string, boolean, or integer (or other numeric) 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

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

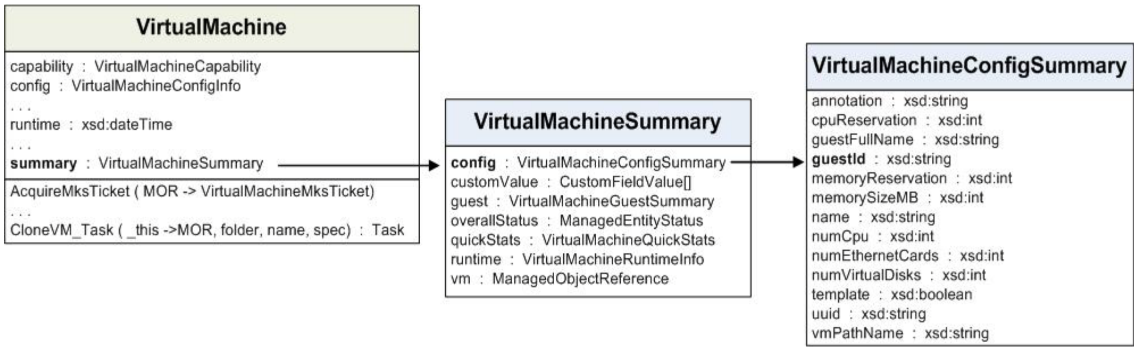
- Use dot notation to access nested properties in composite data structures.
- Cast unconstrained property values (`xsd:anyType`) to array types.
- Use keys or index values as appropriate to access array values.

Nested Properties and Property Paths in Composite Data Structures

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

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

Figure 2-4. VirtualMachine Managed Object and Nested Properties


VirtualMachine
capability : VirtualMachineCapability
config : VirtualMachineConfigInfo
. . .
runtime : xsd:dateTime
. . .
summary : VirtualMachineSummary
AcquireMksTicket (MOR -> VirtualMachineMksTicket)
. . .
CloneVM_Task (_this ->MOR, folder, name, spec) : Task
VirtualMachineSummary
config : VirtualMachineConfigSummary
customValue : CustomFieldValue[]
guest : VirtualMachineGuestSummary
overallStatus : ManagedEntityStatus
quickStats : VirtualMachineQuickStats
runtime : VirtualMachineRuntimeInfo
vm : ManagedObjectReference
VirtualMachineConfigSummary
annotation : xsd:string
cpuReservation : xsd:int
guestFullName : xsd:string
guestId : xsd:string
memoryReservation : xsd:int
memorySizeMB : xsd:int
name : xsd:string
numCpu : xsd:int
numEthernetCards : xsd:int
numVirtualDisks : xsd:int
template : xsd:boolean
uuid : xsd:string
vmPathName : xsd:string

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
`summary`	`VirtualMachineSummary` data object
`summary.config`	`VirtualMachineConfigSummary` data object
`summary.config.guestID`	string

xsd:anyType Arrays

The vSphere API uses `xsd:anyType` unconstrained type declarations. A vSphere client must map values of `xsd:anyType` to explicit data types. An `xsd:anyType` value can represent a single data value or it can represent an array. The WSDL for the vSphere API defines array types for all of the data values that a vSphere client can send or receive as arrays. The array types use the prefix "ArrayOf". An example of an array type is `ArrayOfString` for string values.

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

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

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

When the PropertyCollector returns array data, it sends it as an xsd:anyType value. The language-specific bindings contain definitions for array objects such as ArrayOfEvent, ArrayOfManagedObjectReference, and ArrayOfString, and corresponding "get" methods. To extract the actual array from a property of type xsd:anyType, cast DynamicProperty.val to the appropriate array type and use the matching get method – for example, getEvent(), getManagedObjectReference(), or getString().The following sections provide examples of how to cast returned values for a few of the array types. The code uses the JAX-WS-generated Java bindings for the VMware vSphere Web Services SDK WSDL. Each of the code fragments uses this logic:

- Use the DynamicProperty.getVal() method to retrieve the anyType property value.
- Specify the appropriate array type to cast the anyType value.
- Use the corresponding get method to assign the result of the cast operation to a list variable.

Event Array Example

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

ManagedObjectReference Array Example

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

String Array Example

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

Indexed Array and Key-Based Array Properties

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

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

Unset Optional Properties

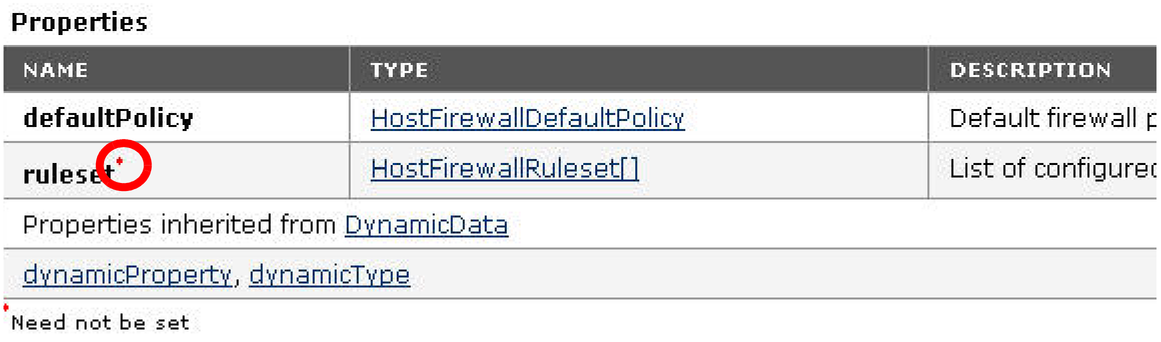
Many of the Data Objects in the vSphere Web Services SDK have optional properties that may be set by your client application or by a Server process or event. If you retrieve a data object that has a optional property that is unset, the Server will not return a value for the optional property. If you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

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

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

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

Figure 2-5. Data Object - HostFirewallInfo Properties



Properties

NAME	TYPE	DESCRIPTION
defaultPolicy	HostFirewallDefaultPolicy	Default firewall p
ruleset*	HostFirewallRuleset[]	List of configured
Properties inherited from DynamicData		
dynamicProperty, dynamicType		

*Need not be set

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

Escape Character in Name and Path Properties

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

Table 2-4. Special Characters

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

Client Applications 3

This chapter includes the following topics:

vCenter Server Connections

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

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

Table 3-1. Security Elements for Client-Server Connections

Security Element	Description
SSL certificates	vSphere Servers use standard X.509 version 3 (X.509v3) certificates to encrypt session information sent over Secure Socket Layer (SSL) protocol connections. In a production environment, client applications verify the vSphere Server certificate during the connection sequence. The examples in this chapter and the examples in the vSphere Web Services SDK accept all certificates.
HTTP tokens	A vSphere Server uses an HTTP token to identify a client session. The Server provides the HTTP token in its response to a client connection request. Subsequent messages between the client and the Server include the HTTP token in the HTTP header.
Client authentication vCenter Single Sign On token	vSphere supports vCenter Single Sign On. A vCenter client can obtain a vCenter Single Sign On token from a vCenter Single Sign On Server and use that token to login to a vCenter Server.
Client authentication username/password	Username/password authentication for client-server connections has been deprecated as of vSphere 5.1.

Establishing a Single Sign On Session with a vCenter Server

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

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

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

- "LoginByToken (C# Example)" on page 28
- "LoginByToken (Java Example)" on page 33

LoginByToken (C# Example)

The following sections describe a C# example of using the `LoginByToken` method.

vCenter Server Single Sign On Session

After you obtain a SAML token from the vCenter Single Sign On Server, you can use the vSphere API method `LoginByToken` to establish a single sign on session with a vCenter Server. To establish a vCenter Server session that is based on SAML token authentication, the client must embed the SAML token in the SOAP header of the `LoginByToken` request. The C# `LoginByToken` example uses the following .NET services to support a single sign on session.

Table 3-2. Microsoft .NET Elements for vCenter Single Sign On Sessions

.NET Element / Namespace	vCenter Single Sign On Usage
`SecurityPolicyAssertion` `Microsoft.Web.Services3.Security`	The sample creates a custom policy assertion derived from the `SecurityPolicyAssertion` class. The custom assertion contains the SAML token and X509 certificate.
`SendSecurityFilter` `Microsoft.Web.Services3.Security`	The sample defines a custom output filter derived from the `SendSecurityFilter` class. The custom filter adds the token and certificate to the outgoing SOAP message.
`ServicePointManager` `System.net`	The sample uses the `ServicePointManager` to specify SSL3 and HTTP 100-Continue behavior.
`ConfigurationManager` `System.Configuration`	The sample uses the `ConfigurationManager` to specify certificate metadata (password and certificate type).
`CookieContainer` `System.Net`	The sample uses the `CookieContainer` class to manage vCenter session cookies.

Persistent vCenter Sessions

A persistent vCenter session relies on a session cookie. When the vCenter Server receives a connection request (`SessionManager.RetrieveServiceContent`), the Server creates a session cookie and returns it in the HTTP header of the response. The client-side .NET framework embeds the cookie in HTTP messages that the client sends to the Server.

The LoginByToken request includes the SAML token and client certificate security assertions for client authentication. After successful login, the authentication overhead is no longer needed. The client resets the VimService context to eliminate the security overhead. Subsequent client requests will contain the session cookie, which is enough to support the persistent, authenticated session.

Sample Code

The code examples in the following sections show how to use the LoginByToken method with a holder-of-key security token. The code examples are based on the LoginByTokenSample project contained in the vCenter Single Sign On SDK. The project is located in the dotnet samples directory (SDK/ssoclient/dotnet/cs/samples/LoginByToken).

- Project file – LoginByToken.csproj
- Sample code – LoginByTokenSample.cs
- SOAP header manipulation code – CustomSecurityAssertionHok.cs

Using LoginByToken

The example program uses the following elements and general steps:

- LoginByTokenSample Constructor
- Token Acquisition
- Security Policies
- Connection and Login

LoginByTokenSample Constructor

The LoginByTokenSample class constructor creates the following elements to set up access to the vCenter Server.

- VimService object – Provides access to vSphere API methods and support for security policies and session cookie managment. It also stores the vCenter Server URL.
- CookieContainer – Provides local storage for the vCenter session cookie.
- ManagedObjectReference – Manually created ManagedObjectReference to retrieve a ServiceInstance at the beginning of the session.

The following code fragment shows the LoginByTokenSample constructor.

Example 3-1. LoginByTokenSample Constructor

```
// Global variables
private VimService _service;
private ManagedObjectReference _svcRef;
private ServiceContent _sic;
private string _serverUrl;


public LoginByTokenSample(string serverUrl)
{
    _service = new VimService();
    _service.Url = serverUrl;
    _serverUrl = serverUrl;
    _service.CookieContainer = new System.Net.CookieContainer();
    _svcRef = new ManagedObjectReference();
    _svcRef.type = "ServiceInstance";
    _svcRef.Value = "ServiceInstance";
}
```

Token Acquisition

The client must obtain a SAML token from a vCenter Single Sign On Server. See the *vCenter Single Sign On Programming Guide*.

Security Policies

The LoginByToken sample creates a custom policy assertion that is derived from the .NET class SecurityPolicyAssertion. The assertion class gives the .NET framework access to the SAML token and the X509 certificate.

The sample performs the following operations to set up the security policy and message handling.

- Sets the ServicePointManager properties to specify SSL3 and HTTP 100-Continue response handling. 100-Continue response handling supports more efficient communication between the client and vCenter Server. When the client-side .NET framework sends a request to the Server, it sends the request header and waits for a 100-Continue response from the Server. After it receives that response, it sends the request body to the Server.
- Creates an X509Certificate2 object, specifies the certificate file, and imports the certificate. The certificate file specification indicates a PKCS #12 format file (Public-Key Cryptography Standards) – PfxCertificateFile. The file contains the client's private key and public certificate. The PfxCertificateFile setting is defined in the app.config file in the LoginByToken project. The definition specifies the location of the file.
- Creates a custom security assertion to store the SAML token and the certificate. The token and certificate will be included in the policy data for the LoginByToken request.
- Defines a custom output filter that is derived from the .NET class SendSecurityFilter.

Custom Security Assertion The following code fragment shows the LoginByTokenSample class method GetSecurityPolicyAssertionForHokToken. The method returns a CustomSecurityAssertionHok instance which overrides the .NET class SecurityPolicyAssertion. The security assertion contains the SAML token and the X509 certificate token. This code is taken from the LoginByToken project file samples/LoginByToken/CustomSecurityAssertionHok.cs.

Example 3-2. Setting Up Security Policies

```
private SecurityPolicyAssertion GetSecurityPolicyAssertionForHokToken(XmlElement xmlToken)
{

    //When this property is set to true, client requests that use the POST method
    //expect to receive a 100-Continue response from the server to indicate that
    //the client should send the data to be posted. This mechanism allows clients
    //to avoid sending large amounts of data over the network when the server,
    //based on the request headers, intends to reject the request
    ServicePointManager.Expect100Continue = true;
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Ssl3;

    X509Certificate2 certificateToBeAdded = new X509Certificate2();
    string certificateFile = ConfigurationManager.AppSettings["PfxCertificateFile"];
    string password = ConfigurationManager.AppSettings["PfxCertificateFilePassword"];
    certificateToBeAdded.Import(certificateFile, password ?? string.Empty,
               X509KeyStorageFlags.MachineKeySet);

    var customSecurityAssertion = new CustomSecurityAssertionHok();
    customSecurityAssertion.BinaryToken = xmlToken;
    customSecurityAssertion.TokenType = strSamlV2TokenType;
    customSecurityAssertion.SecurityToken = new X509SecurityToken(certificateToBeAdded);

    return customSecurityAssertion;
}
```

Custom Output Filter The following code fragment shows the custom output filter for the custom security assertion. The custom filter provides three methods:

- CustomSecurityClientOutputFilterHok class constructor – Creates token and message signature objects for the SOAP message.
- SecureMessage – An override method for the .NET method `SendSecurityFilter.SecureMessage`. The override method adds the SAML token and message signature to the .NET Security element.
- CreateKeyInfoSignatureElement – Creates an XML document that specifies the SAML token type and ID.

Example 3-3. Output Filter for the Custom SecurityPolicyAssertion

```
internal class CustomSecurityClientOutputFilterHok : SendSecurityFilter
    {
        IssuedToken issuedToken = null;
        string samlAssertionId = null;
        MessageSignature messageSignature = null;

        /// Create a custom SOAP request filter.
        /// (Save the token and certificate.)
        public CustomSecurityClientOutputFilterHok(CustomSecurityAssertionHok parentAssertion)
            : base(parentAssertion.ServiceActor, true)
        {
            issuedToken = new IssuedToken(parentAssertion.BinaryToken,
               parentAssertion.TokenType);
            samlAssertionId = parentAssertion.BinaryToken.Attributes.GetNamedItem("ID").Value;
            messageSignature = new MessageSignature(parentAssertion.SecurityToken);
        }

        ///  Secure the SOAP message before its sent to the server.
        public override void SecureMessage(SoapEnvelope envelope, Security security)
        {
            //create KeyInfo XML element
            messageSignature.KeyInfo = new KeyInfo();
            messageSignature.KeyInfo.LoadXml(CreateKeyInfoSignatureElement());

            security.Tokens.Add(issuedToken);
            security.Elements.Add(messageSignature);
        }

        /// Helper method to create a custom key info signature element.
        /// Returns Key info XML element.
        private XmlElement CreateKeyInfoSignatureElement()
        {
            var xmlDocument = new XmlDocument();
            xmlDocument.LoadXml(@"<root><SecurityTokenReference

               xmlns=""http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
               1.0.xsd""
               xmlns:wsse=""http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd""

               wsse:TokenType=""http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#S
               AMLV2.0"">
                <KeyIdentifier
               xmlns=""http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
               1.0.xsd""

               ValueType=""http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID
               "">" + samlAssertionId +
                @"</KeyIdentifier></SecurityTokenReference></root>");
            return xmlDocument.DocumentElement;
        }
    }
```

Connection and Login

The following code fragment performs the following actions:

- Calls the `LoginByTokenSample` class method `GetSecurityPolicyAssertionForHokToken` (see "Security Policies" on page 30) and adds the security policy to the `VimService` object.

The `VimService` object contains the following data:

- vCenter Server URL.
- SAML token (stored in the security policy assertion).
- X509 certificate (stored in the security policy assertion).

- Calls the `RetrieveServiceContent` method. The method establishes the connection with the vCenter Server and it returns a session cookie. The session cookie is stored in the cookie container in the `VimService` object.
- Calls the `LoginByToken` method. The .NET framework uses the security policy assertion and the session cookie to construct the login request.
- Calls the `LoginByTokenSample` class method `resetService` to create a new `VimService` object.

Example 3-4. Connection and Login

```
// Construct the security policy assertion
SecurityPolicyAssertion securityPolicyAssertion = null;
securityPolicyAssertion = GetSecurityPolicyAssertionForHokToken(xmlToken);

// Setting up the security policy for the request
Policy policySAML = new Policy();
policySAML.Assertions.Add(securityPolicyAssertion);

// Setting policy of the service
_service.SetPolicy(policySAML);

_sic = _service.RetrieveServiceContent(_svcRef);
if (_sic.sessionManager != null)
{
    _service.LoginByToken(_sic.sessionManager, null);
}
resetService();
```

The following code fragment shows the `resetService` method. The method creates a new `VimService` object and a new cookie container. The method also obtains a new instance of the session cookie.

Example 3-5. The resetService method

```
/// Resetting the VimService without the security policies
/// as we need the policy only for the LoginByToken method
/// and not the other method calls. resetService also maintains the
/// authenticated session cookie post LoginByToken.
///
/// This method needs to be called only after successful
/// login
private void resetService()
{
    var _cookie = getCookie();
    _service = new VimService();
    _service.Url = _serverUrl;
    _service.CookieContainer = new CookieContainer();
    if (_cookie != null)
    {
        _service.CookieContainer.Add(_cookie);
    }
}


/// Method to save the session cookie
private Cookie getCookie()
{
    if (_service != null)
    {
        var container = _service.CookieContainer;
        if (container != null)
        {
            var _cookies = container.GetCookies(new Uri(_service.Url));
            if (_cookies.Count > 0)
            {
                return _cookies[0];
            }
        }
    }
    return null;
}
```

LoginByToken (Java Example)

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

- "Client Support for a vCenter Single Sign On Session with a vCenter Server" on page 33
- "Saving the vCenter Server Session Cookie" on page 34
- "Using LoginByToken" on page 36
- "Restoring the vCenter Server Session Cookie" on page 37

Client Support for a vCenter Single Sign On Session with a vCenter Server

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

- Maintain the vCenter session cookie. During the initial connection, the Server produces an HTTP session cookie to support the persistent connection. Operations during the login sequence will reset the request context so your client must save this cookie and re-introduce it at the appropriate times.
- Insert the vCenter Single Sign On token and a timestamp into the SOAP header of the `LoginByToken` message.

The example program uses these general steps:

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

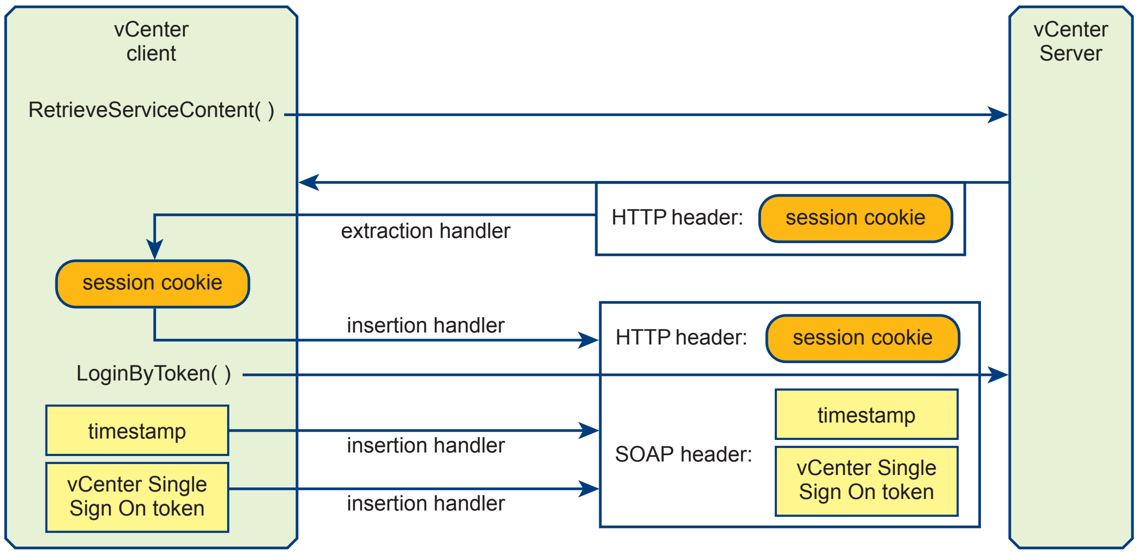
HTTP and SOAP Header Handlers

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

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

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

Figure 3-1. Starting a vCenter Session



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

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

Sample Code

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

- LoginByToken sample:

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

- Header cookie handlers:

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

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

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

Saving the vCenter Server Session Cookie

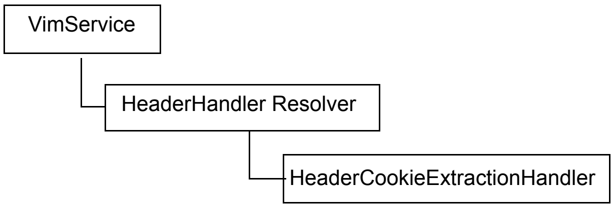
The code fragment in this section establishes an HTTP session with the vCenter Server and saves the HTTP session cookie.

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

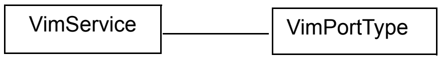
1 Use the `getHandlerResolver` method to save the default message handler. To use the HTTP and SOAP message handlers, you must first save the default message handler so that you can restore it after login. The HTTP and SOAP message handlers impose overhead that is unneccessary after login.

VimService.getHandlerResolver( )

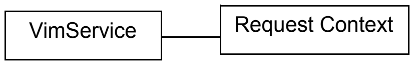
2 Set the cookie handler. The `HeaderCookieExtractionHandler` method retrieves the HTTP cookie.



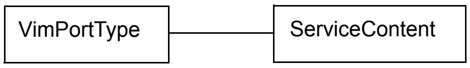
3 Get the VIM port. The VIM port provides access to the vSphere API methods, including the `LoginByToken` method.



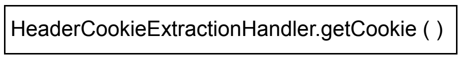
4 Set the request context endpoint address to the vCenter Server URL.



5 Retrieve the ServiceContent. This method establishes the HTTP connection and sets the session cookie.



6 Extract the cookie and save it for later use.



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

Example 3-6. Saving the vCenter Server Session Cookie

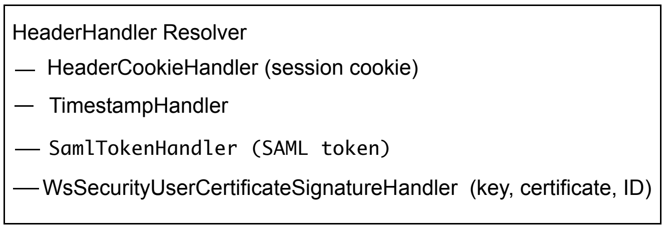
```
/*
 * The example uses a SAML token (obtained from a vCenter Single Sign On Server)
 * and the vCenter Server URL.
 * The following declarations indicate the datatypes; the token datatype (Element) corresponds
 * to the token datatype returned by the vCenter Single Sign On Server.
 *
 * Element token;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

HandlerResolver defaultHandler = vimService.getHandlerResolver();

/*
 * Create a VIM service object.
 */
vimService = new VimService();

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

```
/*
 * Create a handler resolver.
 * Create a cookie extraction handler and add it to the handler resolver.
 * Set the VIM service handler resolver.
 */
HeaderCookieExtractionHandler cookieExtractor = new HeaderCookieExtractionHandler();
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
handlerResolver.addHandler(cookieExtractor);
vimService.setHandlerResolver(handlerResolver);

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

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

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

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

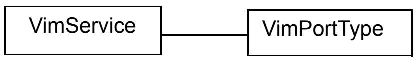
Using LoginByToken

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

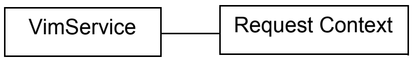
1 Create a new `HeaderHandlerResolver.` Then set the message security handlers for cookie insertion and for inserting the SAML token and credentials in the SOAP header.


HeaderHandler Resolver
— HeaderCookieHandler (session cookie)
— TimestampHandler
— SamlTokenHandler (SAML token)
— WsSecurityUserCertificateSignatureHandler (key, certificate, ID)

2 Get the VIM port.



3 Set the connection endpoint in the HTTP request context.



4 Call the LoginByToken method. The method invocation executes the handlers to insert the elements into the message headers. The method authenticates the session referenced by the session cookie.

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

Example 3-7. Using LoginByToken

```
/*
 * Create a handler resolver and add the handlers.
 */
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
handlerResolver.addHandler(new TimeStampHandler());
handlerResolver.addHandler(new SamlTokenHandler(token));
handlerResolver.addHandler(new HeaderCookieHandler(cookie));
handlerResolver.addHandler(new WsSecuritySignatureAssertionHandler(
                           userCert.getPrivateKey(),
                           userCert.getUserCert(),
                           Utils.getNodeProperty(token, "ID")));
vimService.setHandlerResolver(handlerResolver);

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

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

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

Restoring the vCenter Server Session Cookie

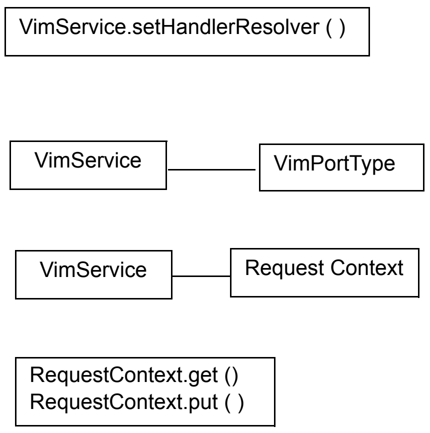
After you log in, you must restore the standard vCenter session context. The code fragment in this section restores the default message handler and the session cookie. As the cookie handler has been replaced by the default handler, the client resets the session cookie by calling request context methods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

1 Restore the default message handler. The handlers used for `LoginByToken` are not used in subsequent calls to the vSphere API.

2 Get the VIM port.

3 Set the connection endpoint in the HTTP request context.

4 Set the HTTP request header (vCenter session cookie).



The following example shows Java code that restores the vCenter session. This code requires the vCenter URL and the cookie and default handler that were retrieved before login. See “Sample Code” on page 34.

Example 3-8. Restoring the vCenter Server Session

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

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

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

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

Establishing a Session with Username and Password Credentials

You can specify username and password credentials to establish a session with a vCenter Server. This method of establishing a session has been deprecated as of vSphere 5.1. The following steps describe how a client application specifies username and password credentials for access to a vCenter Server.

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

Overview of a Java Sample Application

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

While Example 3-9, "Java Test Client Application," on page 39 is a complete client application that demonstrates the basic client capability, it uses a slightly different format than the Java sample files in the `SDK\` directory. This example, and the Java samples that are included with your vSphere Web Service SDK, have been compiled using JAX-WS bindings.

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

Example 3-9 is a stand-alone application that accepts command-line arguments for the vSphere server name (DNS name or IP address), user name, and password.

To build a simple vSphere client application

1 Import the vSphere Web Services API libraries:

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

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

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

3 Create the TestClient class:

```
public class TestClient {
```

4 Include the class variable declarations/definitions. Use a TrustManager class to accept all certificates, as shown in "Accessing the HTTP Endpoint with JAX-WS" on page 42. This is not appropriate for a production environment. Production code should implement certificate support.

5 Use the vSphere Web Services APIs to create the connection, as shown in "Accessing the vSphere Server" on page 43.

6 Retrieve data from the vSphere or vCenter server. In this example, we are just going to print out the product name, server type, and product version to prove that the client is connected and working correctly.

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

7 Use the VimPort object to close the connection, as shown in "Closing the Connection" on page 44. Always close your server connections to maintain security.

Java Client Example

Example 3-9 shows the complete sample client application code, without the explanation steps.

Example 3-9. Java Test Client Application

```
import com.vmware.vim25.*;
import java.util.*;
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;
import javax.xml.ws.soap.SOAPFaultException;

public class TestClient {

// Authentication is handled by using a TrustManager and supplying
// a host name verifier method. (The host name verifier is declared
// in the main function.)
// See Example 3-10 for more details.
    private static class TrustAllTrustManager implements javax.net.ssl.TrustManager,
        javax.net.ssl.X509TrustManager {

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

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

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

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

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

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

// Server URL and credentials.
     String serverName = args[0];
     String userName   = args[1];
     String password   = args[2];
     String url = "https://"+serverName+"/sdk/vimService";

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

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

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

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

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

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

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

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

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

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

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

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

Use the following command to compile the code for Example 3-9, "Java Test Client Application," on page 39, after you have saved it as a .java file:

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

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

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

Web Server Session Token

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

Each of the stand-alone samples in the SDK`\vsphere-ws\java\JAX-WS\samples\com\vmware\` uses the JAX-WS TrustAllTrustCertificates class, as discussed in Example 3-10 to ignore certificates, obtain a session token, and then connect to the server.

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

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

```
//cookie logic
List cookies = (List) headers.get("Set-cookie");
cookieValue = (String) cookies.get(0);
StringTokenizer tokenizer = new StringTokenizer(cookieValue, ";");
cookieValue = tokenizer.nextToken();
String path = "$" + tokenizer.nextToken();
String cookie = "$Version=\"1\"; " + cookieValue + "; " + path;

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

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

Accessing the HTTP Endpoint with JAX-WS

The steps for accessing any HTTP endpoint with JAX-WS bindings are listed at the beginning of Example 3-10, "Obtaining a Session Token - Code Fragments from VMPromoteDisks.java," on page 44. These steps include the vSphere Web Services SDK Server URL, vSphere server object, and variables.

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

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

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

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

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

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

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

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

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

- `ManagedObjectReference` for the `ServiceInstance`.

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

The following Java code fragment shows these variable declarations:

```
ManagedObjectReference SVC_INST_REF
VimService vimService;
VimPortType vimPort;
ServiceContent serviceContent;
```

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

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

5 Instantiate the trust manager object.

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

6 Create the SSL context

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

7 Create the session context

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

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

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

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

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

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

```
HttpsURLConnection.setDefaultHostnameVerifier(hv);
```

Accessing the vSphere Server

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

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

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

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

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

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

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

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

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

```
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(),
    userName,
    password,
    null);
isConnected = true;
```

Closing the Connection

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

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

Using the Java Samples as Reference

The following code fragment from the `SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\vm\VMPromoteDisks.java` sample shows another implementation of the server connection. Review the stand-alone Java samples that are shipped with your *vSphere Web Services SDK,* and use similar code to get a session token for your client application.

Example 3-10. Obtaining a Session Token - Code Fragments from VMPromoteDisks.java

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

      javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
      javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
      trustAllCerts[0] = tm;
      javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
      javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
      sslsc.setSessionTimeout(0);
      sc.init(null, trustAllCerts, null);
      javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
   }
...
private static void connect()
      throws Exception {

      HostnameVerifier hv = new HostnameVerifier() {
         public boolean verify(String urlHostName, SSLSession session) {
            return true;
         }
      };
      trustAllHttpsCertificates();
      HttpsURLConnection.setDefaultHostnameVerifier(hv);
      SVC_INST_REF.setType(SVC_INST_NAME);
      SVC_INST_REF.setValue(SVC_INST_NAME);

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

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

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

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

Multiple Versions of the vSphere API

When a client application connects to a Web service running on 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 3-11).

Example 3-11. Service-Versions File (vimServiceVersions.xml)

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

If you are developing a client application that must support multiple server versions at the same time (ESX/ESXi 5.0 and ESX/ESXi 3.x, 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.

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 older than ESX/ESXi 4.x, vCenter Server 4.x.

Java and C# Sample Applications

The vSphere Web Services SDK includes sample applications, written in Java and C#, that demonstrate features of the vSphere API and object model (see Appendix E, "Sample Program Overview," on page 245). Most of the samples do not handle exceptions, and they accept all security certificates. So use the applications as examples for extracting the types of data you want to view, but do not use the helper classes, trust store methods or exception handling techniques in your production environment.

Java Samples

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

C# Samples

The C# samples in your vSphere Web Services SDK include a .cs file and three project files in each directory (.proj, 2008.proj, and 2010.proj) so you can run them using Microsoft's Visual Studio.

Like the Java samples, the C# samples also accept command-line arguments for the vSphere server name (DNS name or IP address), user name, and password, and they accept all certificates to establish the SSL handshake.

Unlike the Java samples, the C# samples use the helper classes discussed in "Helper Classes for C# Sample Applications" on page 46.

Helper Classes for C# Sample Applications

The C# sample applications included with the vSphere Web Services SDK include C# 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.

The helper classes (listed in Table 3-3) handle command-line input such as common parameters, server name, and other details. These helper classes are located in the unpacked C# version of the SDK download, in this location:

C# Helper Classes `%SDKHOME%\vsphere-ws\dotnet\cs\samples\AppUtil`

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

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

AppUtils	Functional Description
`AppUtil.cs`	Convenient methods you can use to handle user input from command line. Catches errors (faults). Logs output to console.
`AppUtil.csproj`	Convenient methods you can use to handle user input from command line., built to run on versions of Microsoft Visual Studio that were released before 2008.
`ArgumentHandlingException.cs`	Convenient methods you can use to handle exceptions.
`CertPolicy.cs`	Convenient methods you can use to customize certificate error messages.
`ClientUtil.cs`	Convenient methods you can use to handle user input from command line. Catches errors (faults). Logs output to console.
`CustomSecurity.cs`	Convenient methods you can use to override the SOAP security filter.
`CustomSecurityAssertionBearer.cs`	Convenient methods you can use to create a custom policy assertion that applies security to a SOAP message exchange.
`Log.cs`	Convenient methods you can use to create a log file or send log output to the console.
`OptionSpec.cs`	Helper class for handling default and custom command-line arguments.
`PropertyManager.cs`	Convenient methods you can use to listen for Property Manager updates.
`ServiceUtil.cs`	Wrapper methods for the `vimService` methods (the local proxy code methods) for API 2.0 and prior releases.
`SvcConnection.cs`	Convenient methods you can use to create a web service connection handler.
`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`	Convenient methods you can use to retrieve the namespace and API version.
`VMUtils.cs`	Convenient methods you can use to create a virtual machine configuration spec.

Datacenter Inventory 4

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

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`

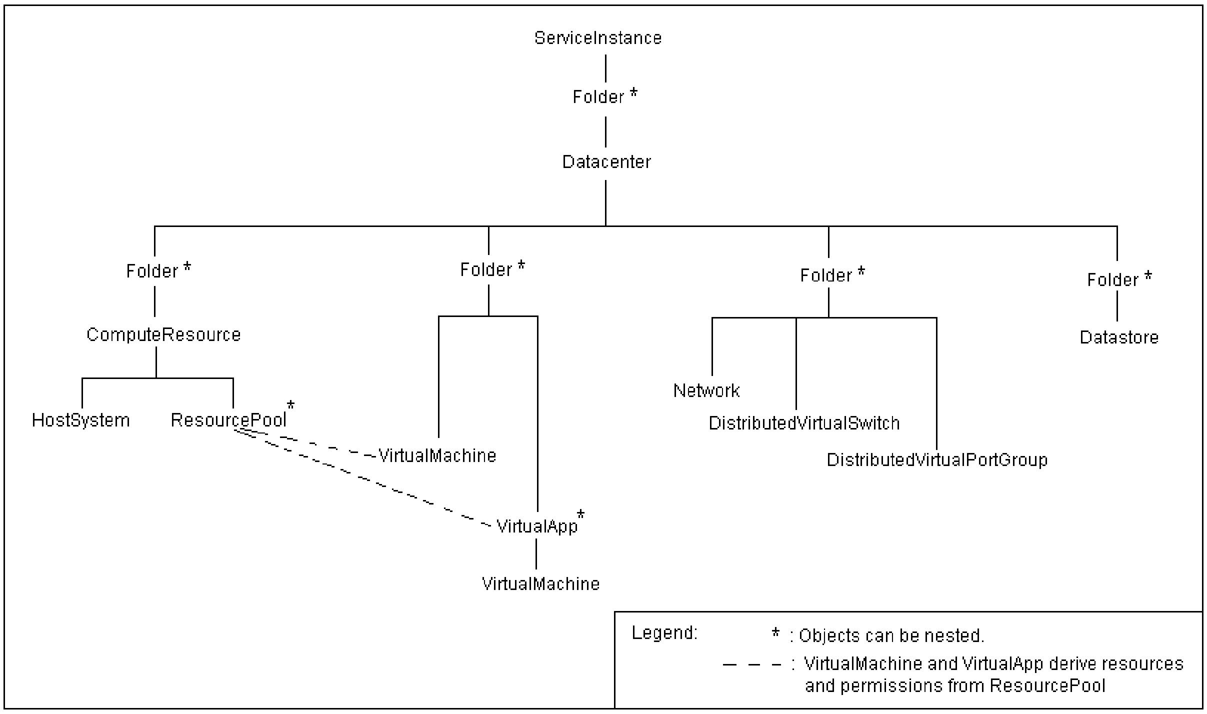
When you manage the virtual infrastructure, you access objects and their properties and methods based on their location in the inventory. Understanding the inventory structure is therefore critical for any programming task. You always start with the `ServiceInstance` associated with a session, which is the root object of the inventory, and traverse the inventory hierarchy from there. See Chapter 5, "Property Collector," on page 57. 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.

When you access a vCenter Server System, the hierarchy shown in Figure 4-1 allows you to traverse the inventory.

Figure 4-1. vCenter Server Inventory Hierarchy



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

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

Folders in the Hierarchy

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

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

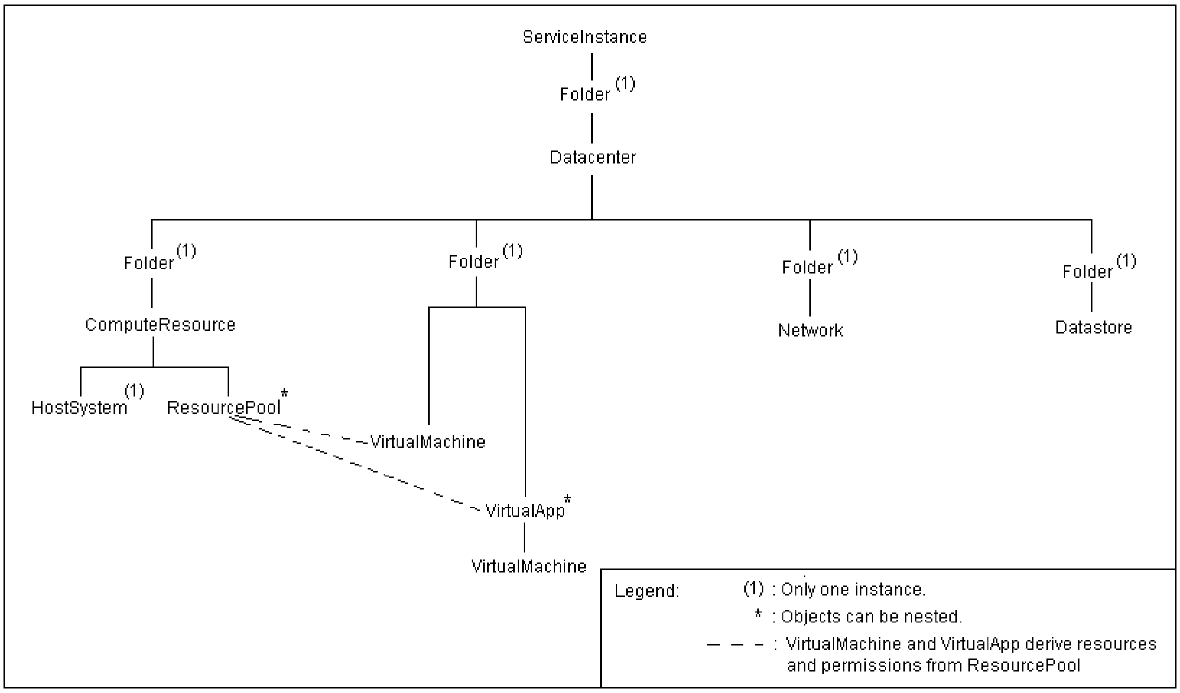
In a large deployment, the nested structure allows you to organize the objects in the datacenter into an easily manageable structure by using multiple folders and datacenters.

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

ESXi Inventory Hierarchy

When you access an ESXi host directly, rather than accessing the host through a vCenter Server system, the hierarchy shown in Figure 4-2 allows you to traverse the inventory.

Figure 4-2. ESXi Inventory Hierarchy



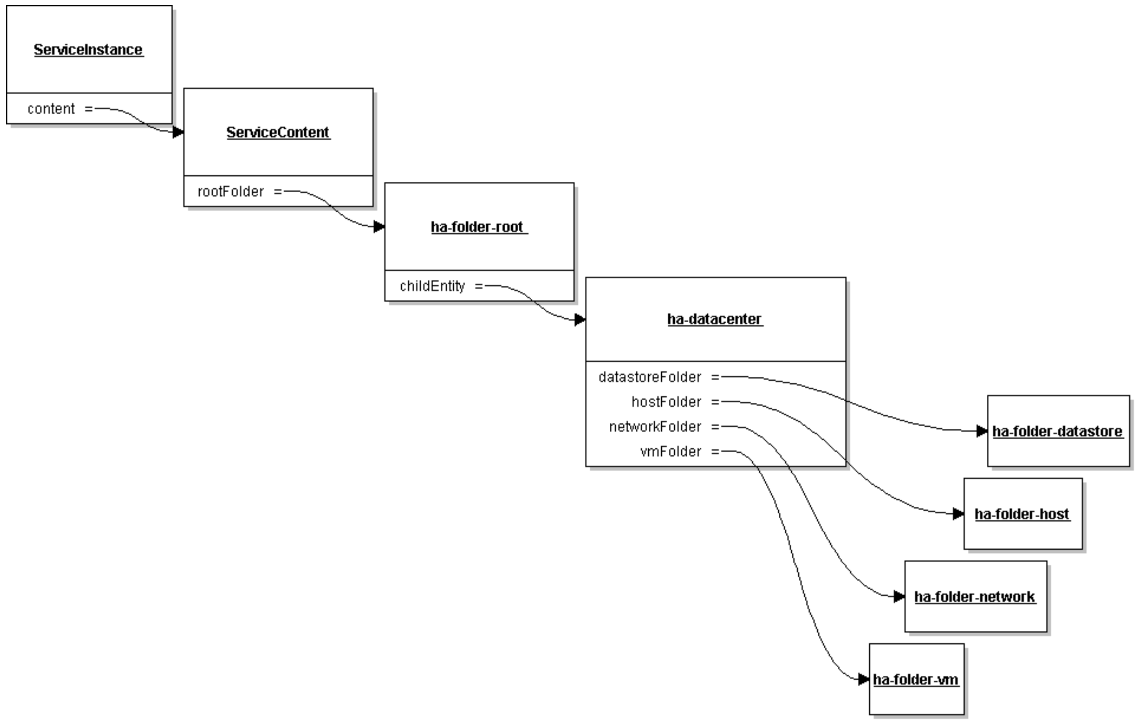
Accessing Inventory Objects

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

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

Figure 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 81 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

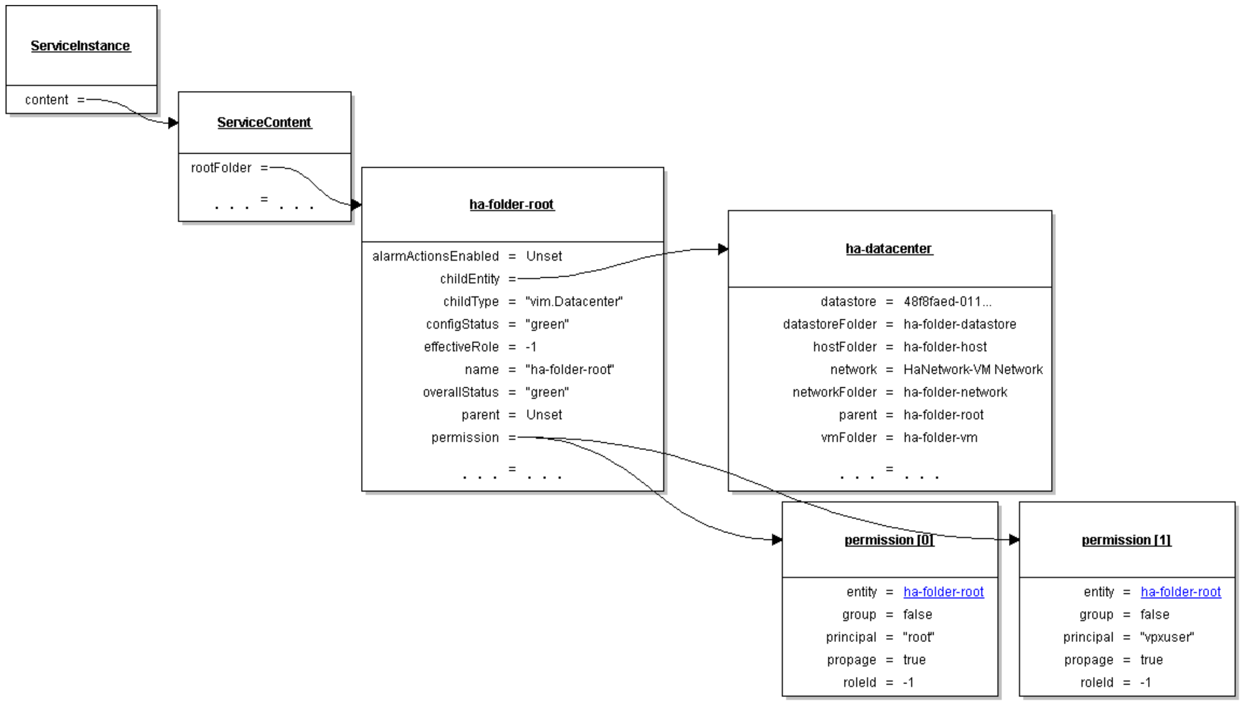
Permissions are the associations of roles with privileges on a specified managed entity. You use permissions to specify which users can access which managed entity.

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 `vpxuser` both have permissions on the `rootFolder` of the inventory. The `vpxuser` 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 `vpxuser` account is granted privileges to the `rootFolder` of each host.

IMPORTANT See Chapter 6, "Authentication and Authorization," on page 81 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 4-2, "ESXi Inventory Hierarchy," on page 51 and Figure 4-1, "vCenter Server Inventory Hierarchy," on page 50.

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 5

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

This chapter includes the following topics:

Introduction to the PropertyCollector

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

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

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

A vSphere server creates a default PropertyCollector for every session, and allows you to create multiple, additional PropertyCollector objects. Create additional PropertyCollector objects, using one per thread, to perform 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.
- Incremental property retrieval, also referred to as property monitoring, uses the `WaitForUpdatesEx` method. The initial call to this method retrieves a baseline set of managed object property values. Subsequent calls retrieve changes in property values since the last retrieval. Use `WaitForUpdatesEx` to monitor changes to the inventory or any managed object properties.

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

Inventory Traversal and Object Selection

`PropertyCollector` filter properties identify object properties 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. A view maintains a subset of inventory objects, so if there is a change in the inventory hierarchy, you do not have to recreate the view. Use a view to specify a set of objects that the `PropertyCollector` can use for data collection.

For information about the vSphere inventory, see Chapter 4, "Datacenter Inventory," on page 49.

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
`PropertyFilterSpec`	Provides access to object and property selection data. A `PropertyFilterSpec` must have at least one `ObjectSpec` and one `PropertySpec`.
`ObjectSpec`	Identifies the starting object for property collection. An `ObjectSpec` also identifies additional objects for collection.
`TraversalSpec`	Identifies the type of object for property collection. It also provides one or more paths for inventory traversal.
`SelectionSpec`	Acts as a placeholder reference to a `TraversalSpec`.
`PropertySpec`	Identifies properties for collection.
`View` objects	Identify a subset of the vSphere inventory objects.

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
Monitor properties using different filters	`CreatePropertyCollector`	Creates a new `PropertyCollector` object to monitor properties using different filters. The vSphere server handles requests for a `PropertyCollector` instance independently of any other instances of the `PropertyCollector` on the server.
	`DestroyPropertyCollector`	Destroys an instance of a `PropertyCollector` that was created by a call to `CreatePropertyCollector` from your client application.
Single collection operation	`RetrievePropertiesEx`	Retrieves property data for the specified managed objects.
	`ContinueRetrievePropertiesEx`	Retrieves additional property data for an operation started by `RetrievePropertiesEx`.
	`CancelRetrievePropertiesEx`	Cancels a `RetrievePropertiesEx` or `ContinueRetrievePropertiesEx` operation.
Incremental collection or monitoring operation	`WaitForUpdatesEx`	Retrieves changes to property data since the last `WaitForUpdatesEx` cycle. `WaitForUpdatesEx` blocks until it can satisfy the request or until the request times out. `WaitForUpdatesEx` supports chunked data transmission (see "Server Data Transmission" on page 79).
	`CancelWaitForUpdatesEx`	Cancels a `WaitForUpdatesEx` operation.
General	`CreateFilter`	Creates a new instance of a `PropertyFilter` 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 build a simple vSphere client application" on page 39.

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.

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

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

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

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

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

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

The TraversalSpec properties type and path determine path traversal. TraversalSpec.type identifies 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.getSelectSet().add(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.getPathSet().add("name");
```

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

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

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

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

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

9 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 and this token is used to retrieve the remaining properties using the `ContinueRetrievePropertiesEx` API method. For more information, see "Server Data Transmission" on page 79.

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

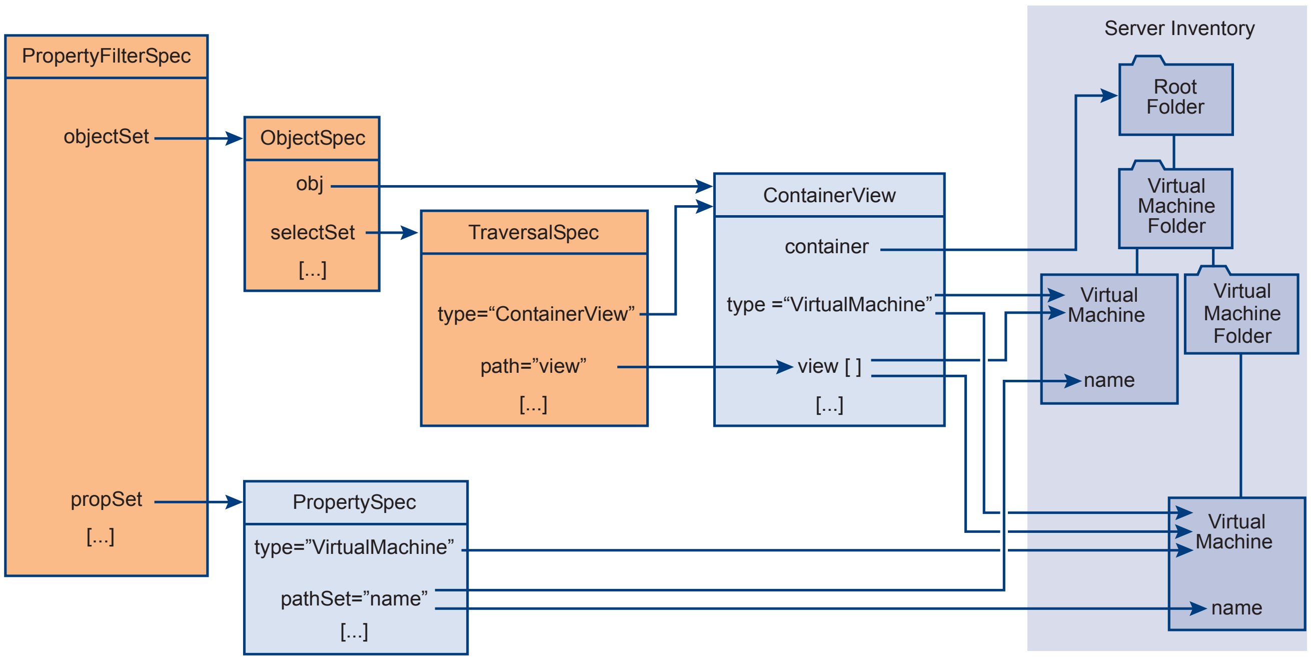
10 Print the virtual machine names.

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

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

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


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

public class PCollector {

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

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

    // use a container view for virtual machines to define the traversal
    // - invoke the VimPortType method createContainerView (corresponds
    //   to the ViewManager method) - pass the ViewManager MOR and
    //   the other parameters required for the method invocation
    // - createContainerView takes a string[] for the type parameter;
    //   declare an arraylist and add the type string to it
    List<String> vmList = new ArrayList<String>();
    vmList.add("VirtualMachine");
    ManagedObjectReference cViewRef = methods.createContainerView(viewMgrRef,
        sContent.getRootFolder(),
        vmList,
        true);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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


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

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

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

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

    // retrieve data
    collectProperties( vimPort, serviceContent );

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

    }
}
```

Inventory Traversal

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

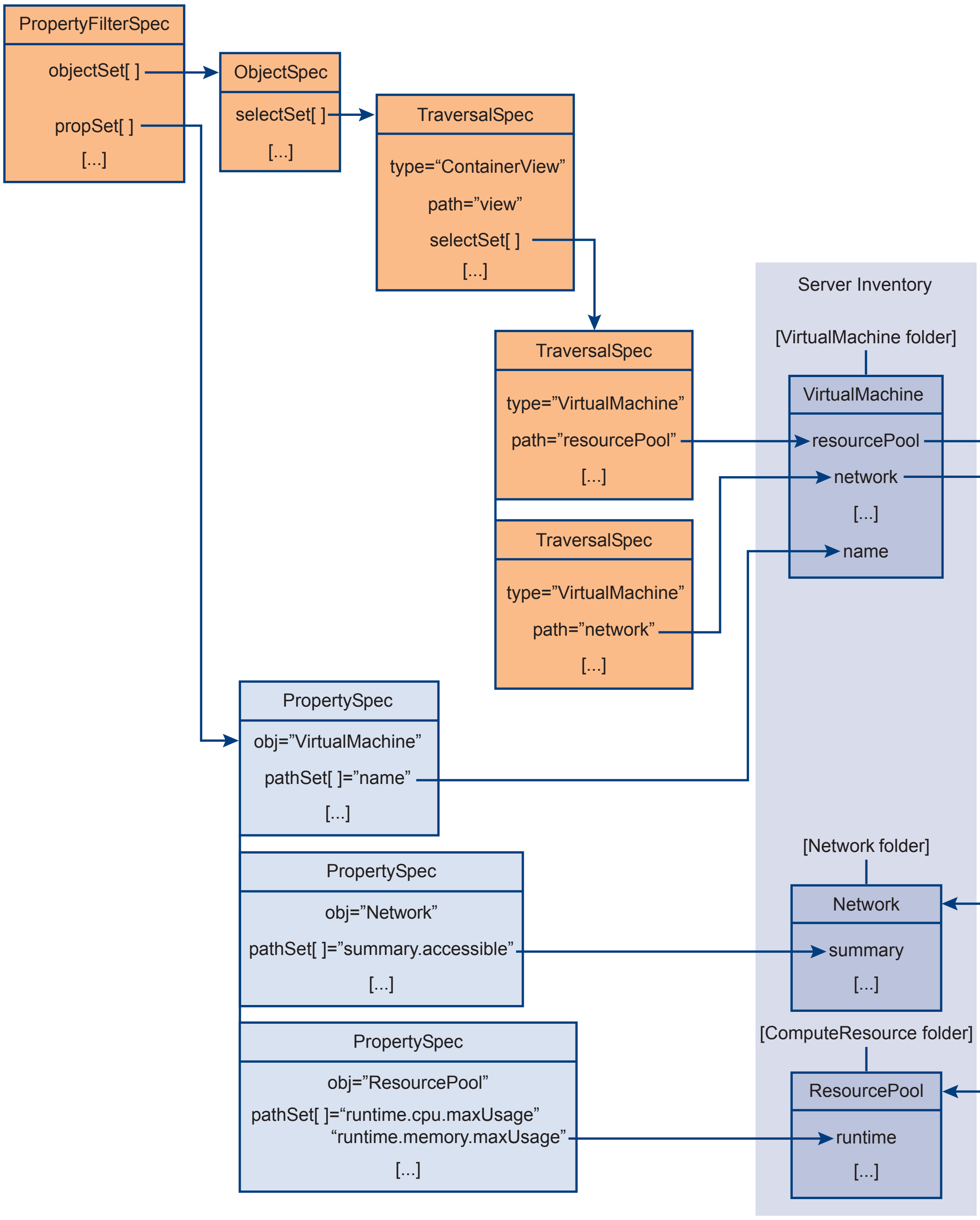
Use a `TraversalSpec` object to identify a managed object type and a traversal property in that type. `TraversalSpec` contains the following properties:

- `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 additonal 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 72.
- `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, "Simple PropertyCollector Example (Java)," on page 62, 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` 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

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

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


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

public class PCollector_traversal {

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

// retrieve data
collectProperties( vimPort, serviceContent );

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

    }
}
```

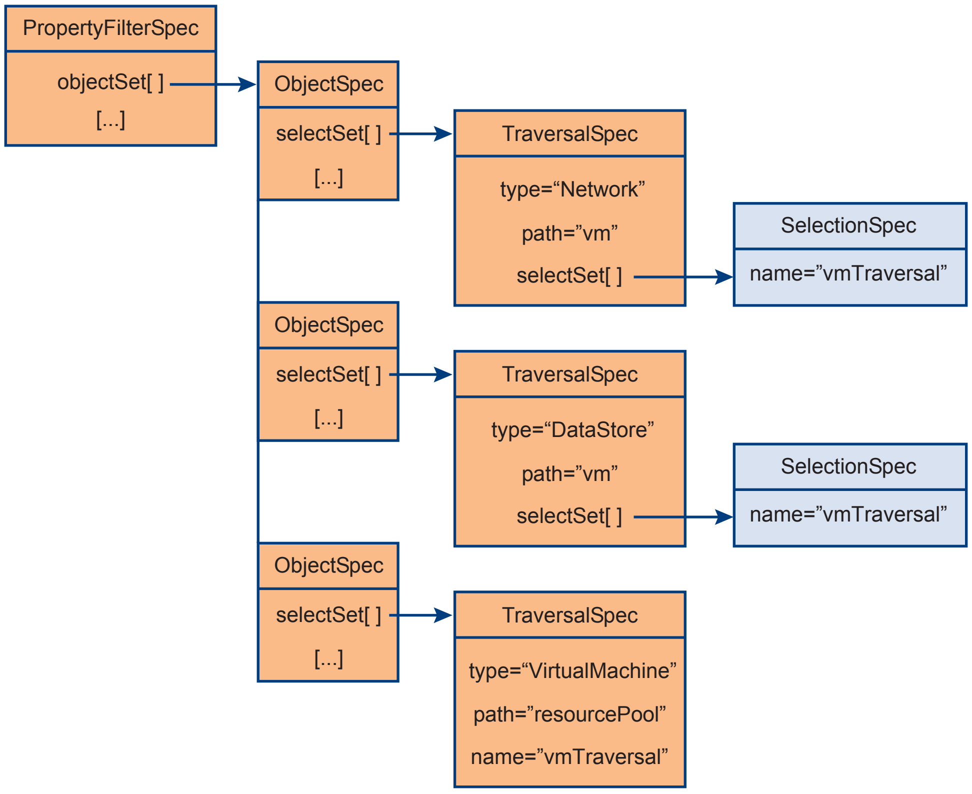
SelectionSpec Traversal

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

Simple Reference SelectionSpec

Use `SelectionSpec` references to avoid writing duplicate `TraversalSpec` declarations. The `TraversalSpec` identified in a `SelectionSpec` reference must be within the same `PropertyFilterSpec`. Figure 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



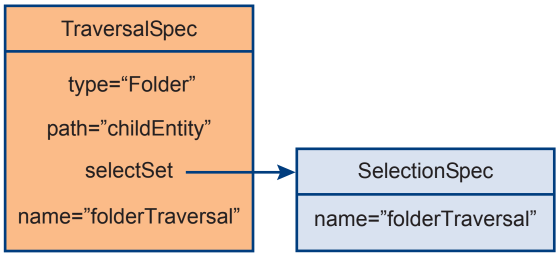
If the `ObjectSpec.selectSet` array contains a `SelectionSpec`, the referenced `TraversalSpec` must identify the same object type. `TraversalSpec.type` must match the type of the object specified in `ObjectSpec.obj`. The `PropertyCollector` applies the `TraversalSpec` to the object and use the `TraversalSpec.path` 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 50 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, "Nested Folder Traversal," on page 74 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 79.

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

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

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

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

public class nestedTraversal {

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

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

    // get the top-level vm folder mor
    ManagedObjectReference sIndex = sContent.getSearchIndex();
        ManagedObjectReference rootVmFolder =
            methods.findByInventoryPath(sIndex,"datacenter1/vm");

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

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

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

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

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

    // specify the property for retrieval (folder name)
    PropertySpec pSpec = new PropertySpec();
    pSpec.setType("Folder");
    pSpec.getPathSet().add("name");

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
    // retrieve data
    collectProperties( vimPort, serviceContent );

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

    }
}
```

Client Data Synchronization (WaitForUpdatesEx)

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

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

Property Filters

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

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

WaitForUpdatesEx

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

Specify the following parameters when you call `WaitForUpdatesEx`:

- Managed object reference to a `PropertyCollector` instance.
- `version` value that identifies a sequence value. The first time you call `WaitForUpdatesEx`, specify an empty string (“”) to retrieve a complete set of results for the specified properties. Your subsequent calls should use the version value returned in the previous call. If you don’t include the version value, the server returns everything. For more information about data versions, see “Server Data Transmission” on page 79.
- `options` specifying the amount of data to transmit in a single response (the `WaitOptions.maxObjectUpdates` property) and the number of seconds the `PropertyCollector` should wait for updates (the `WaitOptions.maxWaitSeconds` property).

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

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

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

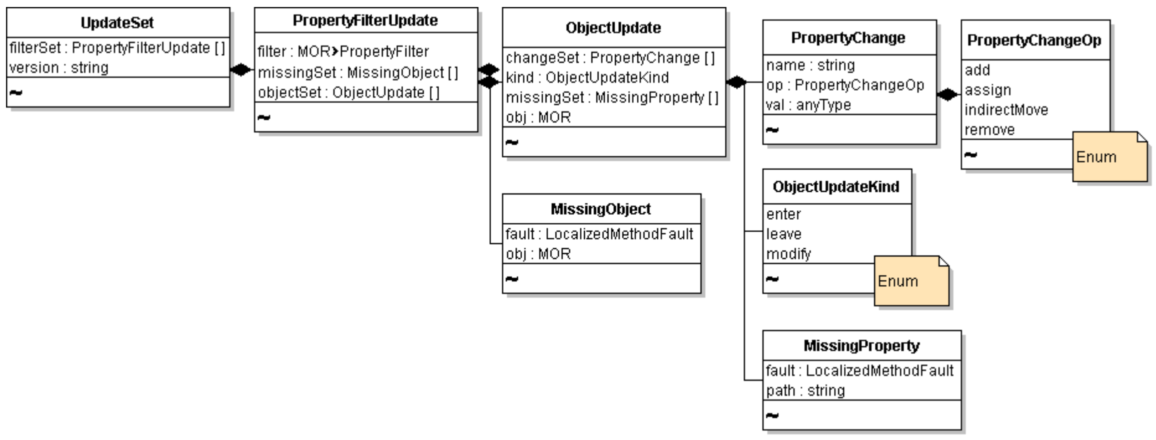
Table 5-3 lists some of the advantages and disadvantages of these two operations.

Table 5-3. WaitForUpdatesEx Operations Compared

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

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 `ContinueRetrievePropertiexEx` to retrieve the next chunk of data.

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

PropertyCollector Performance

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

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

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

To minimize `PropertyCollector` overhead and the amount of network traffic for your client application, use `View` objects with the `PropertyCollector`. 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 6

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 uses with corresponding privileges can access the entities. This chapter discusses approaches to securing the system and the related service interfaces. The chapter also discusses the user model, which is different in ESXi systems and vCenter Server systems.

The chapter includes the following topics:

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 231 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:

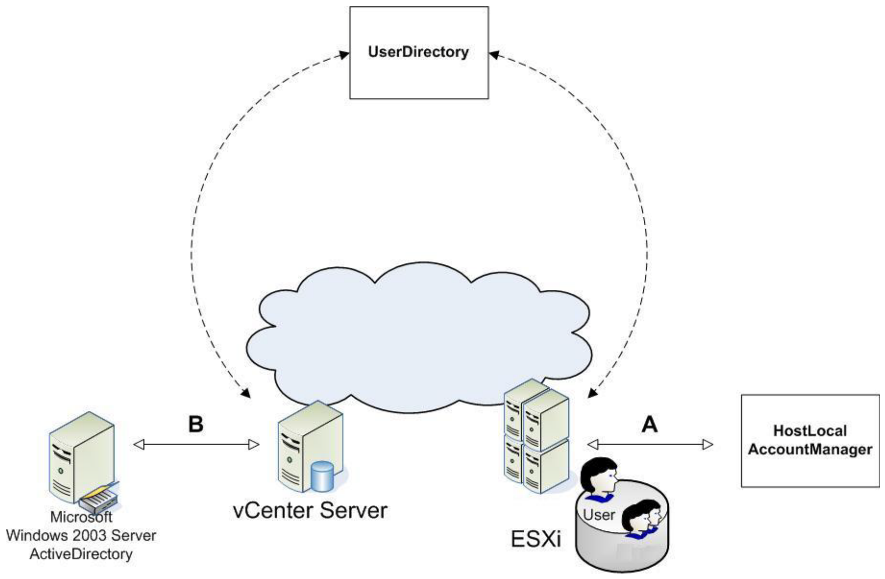
- `HostLocalAccountManager` is used to create and manage user accounts on ESXi systems. Authenticated users can view objects or invoke operations on the server depending on the permissions associated with their account. See "Managing ESXi Users with HostLocalAccountManager" on page 86.
- `AuthorizationManager` protects vSphere components from unauthorized access. Access to components is role-based: Users are assigned roles that encompass the privileges needed to view and perform operations on vSphere objects. `AuthorizationManager` has operations for creating new roles, modifying roles, setting permissions on entities, and handling the relationship between managed objects and permissions.
- `UserDirectory` provides a look-up mechanism that returns user-account information to `AuthorizationManager` or to another requestor, such as a client application. See "Obtaining User and Group Information from UserDirectory" on page 85.

- `SessionManager` provides an interface to the authentication infrastructure on the target server system (see "Authenticating Users Through SessionManager" on page 91).
    - For vCenter Server systems, `SessionManager` supports single sign-on based on SSO tokens obtained from a VMware SSO Server. See "Establishing a Single Sign On Session with a vCenter Server" on page 28.
    - For ESXi systems, `SessionManager` supports authenticating user accounts as defined on the host system, such as accounts created using vSphere Client or accounts created programmatically through the `HostLocalAccountManager` API.
- Even if a user is authorized to perform operations on a vSphere object, the operation fails if the licenses for the host or the feature have not been assigned. You use `LicenseManager` and `LicenseAssignmentManager` to manage the licenses. See "Managing Licenses with LicenseManager" on page 94.

Authentication and Authorization for ESXi and vCenter Server

Several server-side mechanisms authenticate a human user when a client application, such as the vSphere Client or a vSphere Web Services SDK application, connects to the server. Because ESXi uses Linux-based authentication, and vCenter Server 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 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 91.

ESXi User Model

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

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

vCenter Server User Model

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 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.

A vCenter Server client uses a SAML token to establish a single sign on session with the Server. See "Establishing a Single Sign On Session with a vCenter Server" on page 28.

vSphere Security Model

While the details of authentication and authorization differ between 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 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 88). 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 240 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 ESXi host. Permissions grant users the right to perform the activities specified by the role on the object to which the role is assigned.

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

Many tasks require permissions on more than one object.

Setting Up Users, Groups, and Permissions

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

1 Get information about privilege requirements and privileges associated with system and sample roles.

- Find out which operations on vSphere objects require which privileges. See the *API Reference*.
- Find out which operations the system roles and sample roles can perform. See Table 6-2, "System and Sample Roles," on page 88.

2 If necessary, create additional roles (sets of privileges). See "Modifying Sample Roles to Create New Roles" on page 89.

3 Retrieve information about existing users and groups (see "Obtaining User and Group Information from UserDirectory" on page 85) and create additional groups if needed.

4 Associate users or groups with roles using permissions. See "Managing Roles and Permissions with AuthorizationManager" on page 86.

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

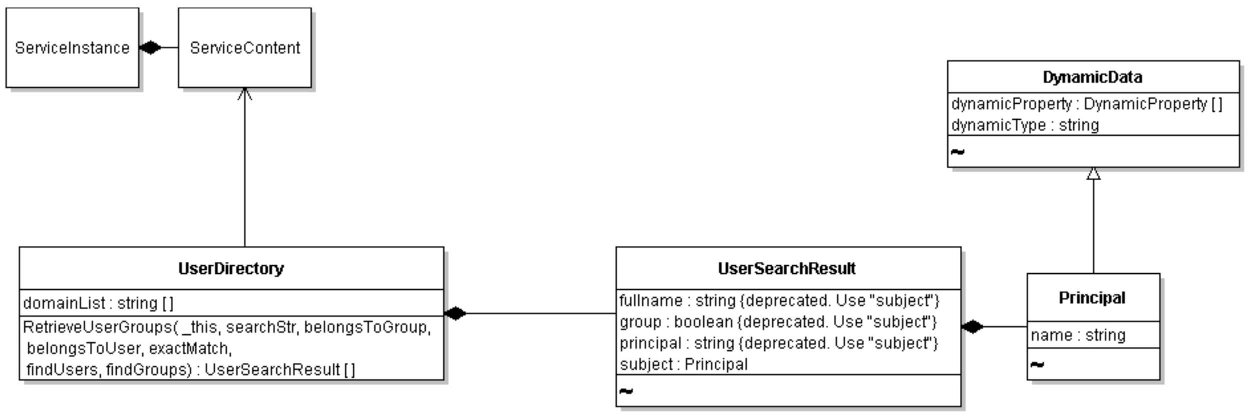
Obtaining User and Group Information from UserDirectory

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

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

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

Figure 6-2. UserDirectory Managed Object


ServiceInstance
ServiceContent
DynamicData
dynamicProperty : DynamicProperty [ ]
dynamicType : string
~
UserDirectory
domainList : string [ ]
RetrieveUserGroups(_this, searchStr, belongsToGroup, belongsToUser, exactMatch, findUsers, findGroups) : UserSearchResult [ ]
UserSearchResult
fullname : string {deprecated. Use "subject"}
group : boolean {deprecated. Use "subject"}
principal : string {deprecated. Use "subject"}
subject : Principal
~
Principal
name : string
~

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

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

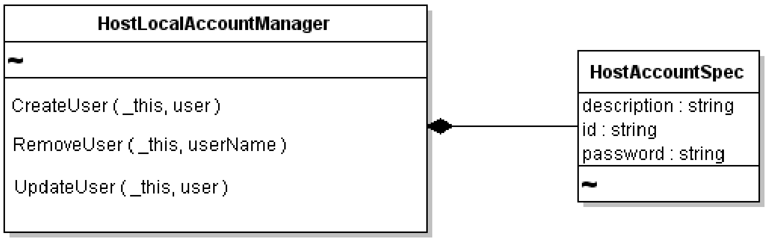
IMPORTANT Do not configure an ESXi system to use NIS or NIS+, unless it is acceptable to have NIS (or NIS+) user information available through the `UserDirectory.RetrieveUserGroups` API.

Managing ESXi Users with HostLocalAccountManager

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

IMPORTANT vCenter Server systems use the Microsoft Windows user management facilities. See "vCenter Server User Model" on page 83.

Figure 6-3. HostLocalAccountManager Managed Object



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

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

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

To create a user account on an 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 ESXi system for user account naming conventions and password requirements, such as minimum length, character set, and other requirements.
3. Call the `HostLocalAccountManager.CreateUserAccount` method, passing in the managed object reference (from step 1) and the `HostAccountSpec` data object (step 2).

After creating user accounts on the ESXi system, you can grant these users access to virtual components by using `AuthorizationManager` methods. See "Managing Roles and Permissions with AuthorizationManager" on page 86.

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.

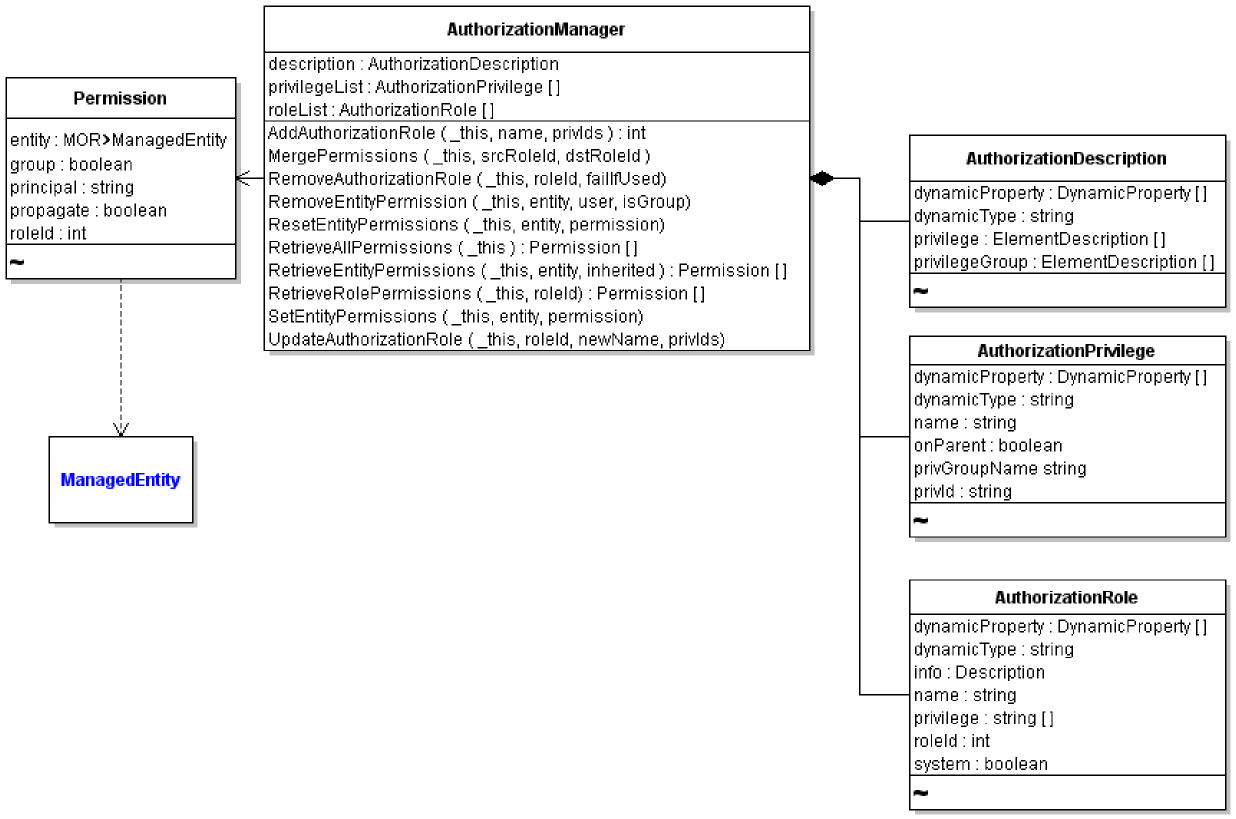
`AuthorizationManager` includes methods for managing roles and for managing permissions:

- **Roles Management**. `AddAuthorizationRole`, `RemoveAuthorizationRole`, and `UpdateAuthorizationRole`. See "Using Roles to Consolidate Sets of Privileges" on page 88 and "Modifying Sample Roles to Create New Roles" on page 89.

- **Permissions Management**. MergePermissions, RemoveEntityPermission, ResetEntityPermissions, RetrieveAllPermissions, RetrieveEntityPermissions, RetrieveRolePermissions, and SetEntityPermissions. See "Granting Privileges Through Permissions" on page 89.

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
description : AuthorizationDescription
privilegeList : AuthorizationPrivilege [ ]
roleList : AuthorizationRole [ ]
AddAuthorizationRole (_this, name, privIds) : int
MergePermissions (_this, srcRoleId, dstRoleId)
RemoveAuthorizationRole (_this, roleId, failIfUsed)
RemoveEntityPermission (_this, entity, user, isGroup)
ResetEntityPermissions (_this, entity, permission)
RetrieveAllPermissions (_this) : Permission [ ]
RetrieveEntityPermissions (_this, entity, inherited) : Permission [ ]
RetrieveRolePermissions (_this, roleId) : Permission [ ]
SetEntityPermissions (_this, entity, permission)
UpdateAuthorizationRole (_this, roleId, newName, privIds)
Permission
entity : MOR>ManagedEntity
group : boolean
principal : string
propagate : boolean
roleId : int
~
ManagedEntity
AuthorizationDescription
dynamicProperty : DynamicProperty [ ]
dynamicType : string
privilege : ElementDescription [ ]
privilegeGroup : ElementDescription [ ]
~
AuthorizationPrivilege
dynamicProperty : DynamicProperty [ ]
dynamicType : string
name : string
onParent : boolean
privGroupName string
privId : string
~
AuthorizationRole
dynamicProperty : DynamicProperty [ ]
dynamicType : string
info : Description
name : string
privilege : string [ ]
roleId : int
system : boolean
~

`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 231 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. ESXi defines system roles and user roles.

- **System roles.** Cannot be modified or deleted.
- **User roles.** Apply to different user communities or restrict access for add-on tools. Several predefined user roles are included with vCenter Server and with ESXi systems. You can create new roles using these predefined user roles as a starting point.

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 240 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 ESXi or vCenter Server system to support local sessions, which enable users with credentials on the host to log in based on those privileges.

The `SessionManager` provides these capabilities:

- **Log in and log out.** Basic operations to log in to ESXi or vCenter Server system, obtain a session, and log out. When a session terminates, all session-specific objects are destroyed.
- **Impersonation.** One user session adopts the authorization level of another user session. Impersonation is common in Web based scenarios in which a middle-tier application functions as a central account that interacts with other back-end servers or processes. Windows services impersonate a client when accessing resources on behalf of the client. `SesssionManager` supports impersonation through its `ImpersonateUser` method.
- **Delegation.** A client application that is running on behalf of a local user can call the `SessionManager.AcquireLocalTicket` method to obtain a one-time user name and password for logging in. Delegation is useful for host-based utilities that run in the local console.

If the user account associated with the session does not have the permissions required to perform an action, the `AuthorizationManager` returns a `NoPermission` fault to the client application.

Using VMware Single Sign On for vCenter Server Sessions

vSphere supports single sign on for a single point of authentication for vCenter Server clients. To use VMware Single Sign On, your vSphere Web Services SDK client connects to the VMware SSO Server to obtain an SSO token. Your client includes the token in the `SessionManager.LoginByToken` request to start a vSphere session. See "Establishing a Single Sign On Session with a vCenter Server" on page 28.

Using the Credential Store for Automated Login

To facilitate automated login for unattended applications, the vSphere Web Services SDK includes client-side credential store libraries and tools for automating the login process in a more secure manner. The libraries eliminate the need for system administrators to keep passwords in local scripts.

IMPORTANT These libraries are built on top of the vSphere Web Services SDK.

The credential store has the following components:

- A persistence file (credential store backing file) that stores authentication credentials. Currently, only passwords are supported. The persistence file maps a remote user account from an ESXi host to the password for that user on the server.
- 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 com.vmware.security.credstore (Java)	Namespace VMware.Security.CredentialStore(C#)
CredentialStore.java	CredentialStoreFactory.cs
CredentialStoreFactory.java	CredentialStore.cs

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
`addPassword(hostname, username, password)`	`AddPassword(hostname, username, password)`	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).
`removePassword(hostname, username)`	`RemovePassword(hostname, username)`	Deletes the password for the specified user from the credential store.
`clearPasswords()`	`ClearPasswords()`	Deletes all passwords from the credential store.
`getPassword(hostname, username)`	`GetPassword(hostname, username)`	Returns the password for the specified host and user from the credential store.
`getHosts()`	`GetHosts()`	Returns the set of hosts contained in the credential store.
`getUsernames(hostname)`	`GetUsernames(hostname)`	Returns the collection of all user names that have passwords stored for the specified `hostname`.
`close()`	`Close()`	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 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 88 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 ESXi hosts, vCenter Server, and special features such as VMware HA or VMware vMotion.

The *vSphere Datacenter Administration Guide* explains how to manage ESXi and vCenter Server licenses using the vSphere Client, and gives background information about license keys, license inventory, and related topics.

You can also manage licenses using the `LicenseManager` and `LicenseAssignmentManager` managed objects. You use `LicenseManager` to explicitly manage the pool of available licenses on ESXi systems released before vSphere 4.0. You use `LicenseAssignmentManager`, available through the `LicenseManager.licenseAssignmentManager` property, to manage assignment of licenses to entities in the vCenter Server inventory. You can retrieve information, add licenses, and remove licenses.

Retrieve Information

- Retrieve the `LicenseManager.evaluation` and `LicenseManager.licenses` properties to obtain information on evaluation licenses and full licenses.
- Call `LicenseManager.DecodeLicense` to decode license information. The call returns a `LicenseManagerLicenseInfo` data object, which encapsulates information about the license.
- Call `LicenseAssignmentManager.QueryAssignedLicenses` for information about assigned licenses.

Add Licenses

- Call `LicenseManager.AddLicense`, passing in a license key, to add a license to the inventory of available licenses.
- Call `LicenseAssignmentManager.UpdateAssignedLicense`, passing in a license key, to update the licenses for an entity, for example, a host system.

Remove Licenses

- Call `LicenseAssignmentManager.RemoveAssignedLicense` to remove all licenses from an entity, passing in an entity to remove licenses from. You can then assign those licenses to other entities.
- Call `LicenseManager.RemoveLicense`, passing in a license key, to remove a license from the inventory of available licenses.

Hosts 7

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 99) and networking (see Chapter 9, "vSphere Networks," on page 113), 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:

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`.

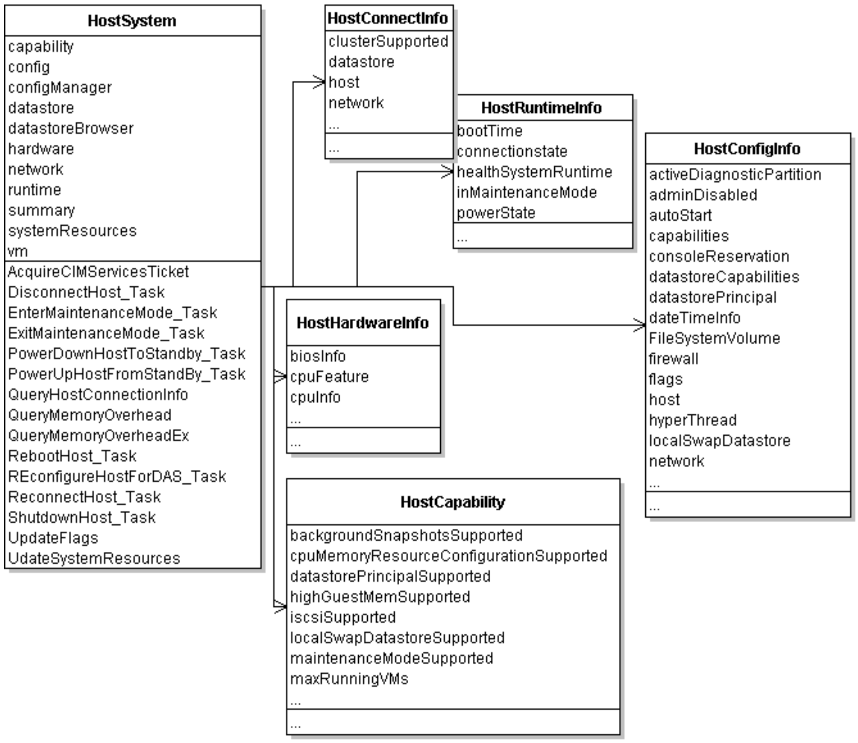
- `HostSystem.capability` is a `HostCapability` object. The `HostCapability` properties indicate the features that are supported by the host, for example, `maintenanceModeSupported` or `recursiveResourcePoolsSupported`.
- `HostSystem.runtimeInfo` is a `HostRuntimeInfo` object that contains several data 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 97.
- **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 122.

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 overheat, 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 8

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:

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, "vSphere Networks," on page 113.

Storage Management Objects

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

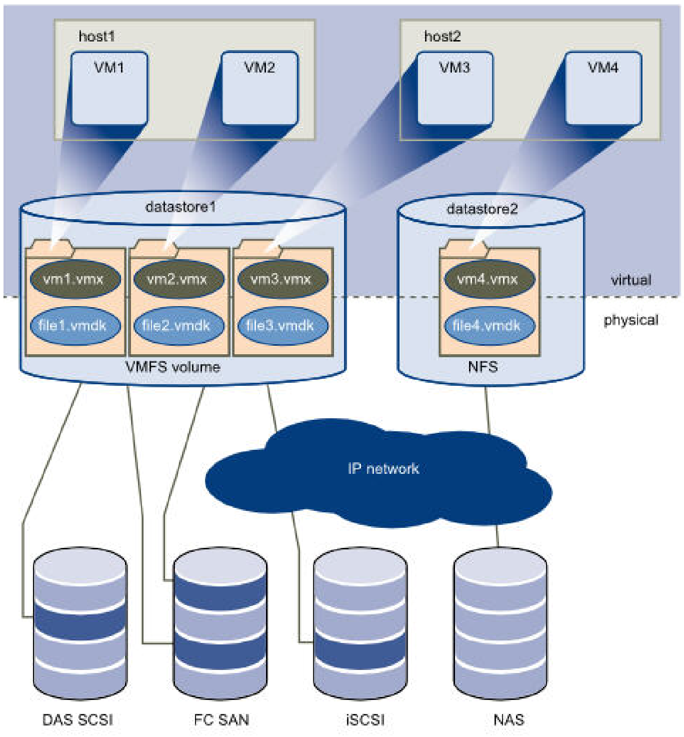
- `HostStorageSystem` – The `HostSystem.storageSystem` property is a managed object reference to the `HostStorageSystem` of the ESX/ESXi system. `HostStorageSystem` is a low-level interface that is used mainly for configuring the physical storage. See "Configuring Disk Partitions" on page 103.
- `HostDatastoreSystem` – 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 106.
- `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 106.
- `HostDiagnosticPartition` – Supports creating and querying diagnostic partitions for your ESX/ESXi host. See "Managing Diagnostic Partitions" on page 110.

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.

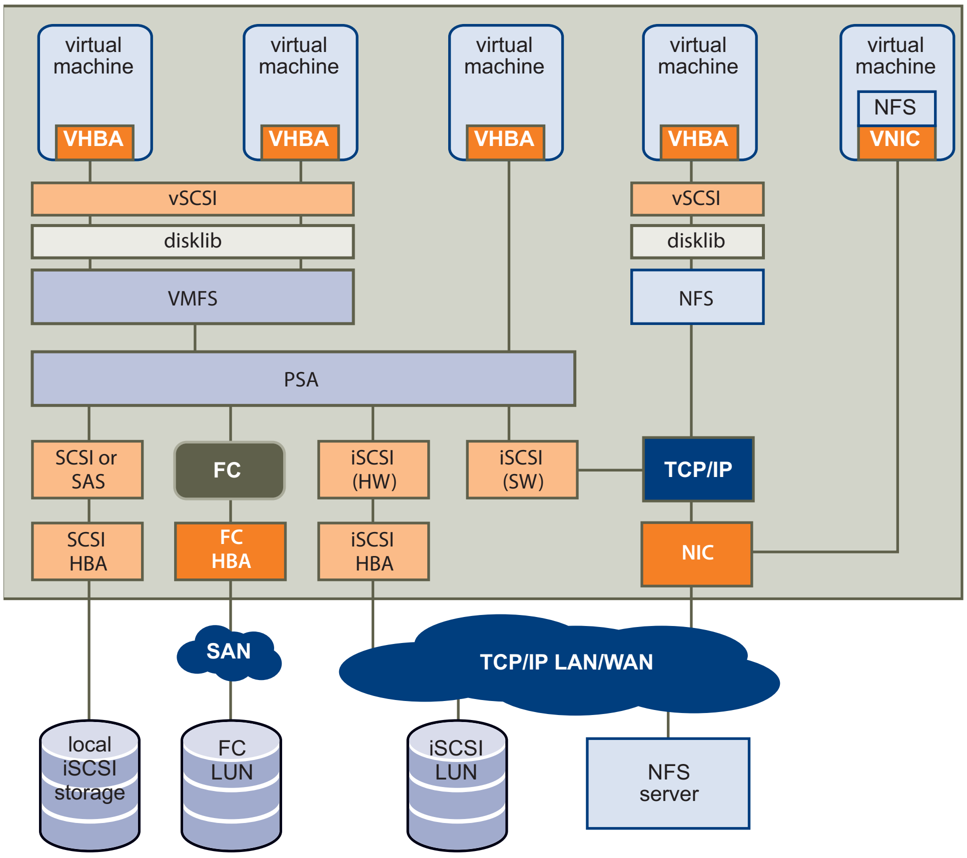
When a virtual machine communicates with its virtual disk stored on a datastore, it issues SCSI commands. Because datastores can exist on different types of physical storage, these commands are encapsulated into other forms, depending on the protocol that the ESX/ESXi host uses to connect to the physical 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



Datastores

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.

Datastores can span multiple physical storage devices. A single VMFS volume can contain one or more LUNs from a local SCSI disk array on a physical host, a Fibre Channel SAN disk farm, or iSCSI SAN disk farm. The ESX/ESXi system detects new LUNS that are added to any of the physical storage subsystems. When the user queries for a list of available devices, the newly discovered devices are included. You can extend storage capacity on an existing VMFS volume without powering down physical hosts or storage subsystems.

If any of the LUNs within a VMFS volume fails or becomes unavailable, only virtual machines with data on that LUN are affected. An exception is the LUN that has the first extent of the spanned volume (multi-extent volume). All other virtual machines with virtual disks residing on other LUNs continue to function normally.

Choosing the Storage API to Use

The `HostStorageSystem` APIs are low-level enough for performing VMFS provisioning operations. They require a knowledge of partitioning details and VMFS extent composition. They do not enforce VMFS best practices like partition alignment and optimum VMFS block sizes, and they allow you to mix extents from different datastores on the same LUN and to add extents even though expansion is preferable in most cases.

The `HostDatastoreSystem` APIs are primarily used for managing VMFS 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

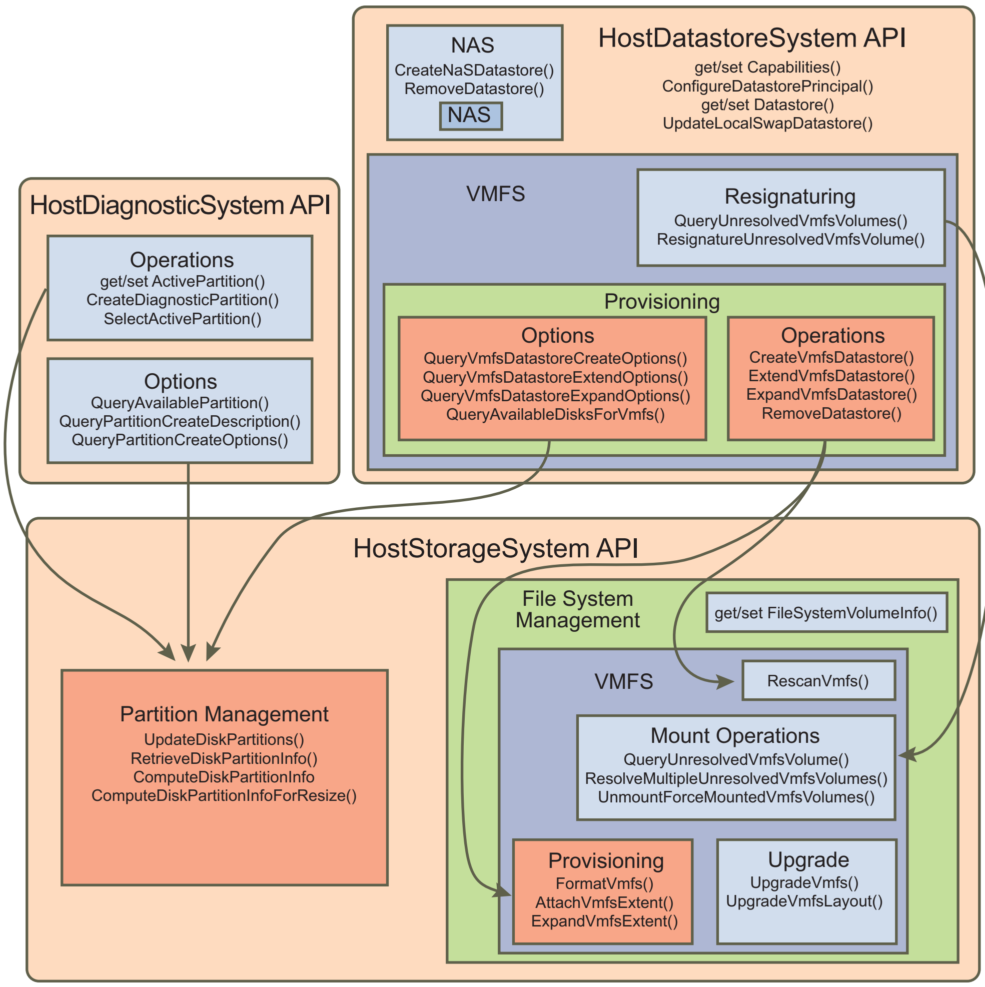

NAS
CreateNaSDatastore()
RemoveDatastore()
NAS
HostDatastoreSystem API
get/set Capabilities()
ConfigureDatastorePrincipal()
get/set Datastore()
UpdateLocalSwapDatastore()
VMFS
Resignaturing
QueryUnresolvedVmfsVolumes()
ResignatureUnresolvedVmfsVolume()
Provisioning
Options
QueryVmfsDatastoreCreateOptions()
QueryVmfsDatastoreExtendOptions()
QueryVmfsDatastoreExpandOptions()
QueryAvailableDisksForVmfs()
Operations
CreateVmfsDatastore()
ExtendVmfsDatastore()
ExpandVmfsDatastore()
RemoveDatastore()
HostDiagnosticSystem API
Operations
get/set ActivePartition()
CreateDiagnosticPartition()
SelectActivePartition()
Options
QueryAvailablePartition()
QueryPartitionCreateDescription()
QueryPartitionCreateOptions()
HostStorageSystem API
Partition Management
UpdateDiskPartitions()
RetrieveDiskPartitionInfo()
ComputeDiskPartitionInfo
ComputeDiskPartitionInfoForResize()
File System Management
get/set FileSystemVolumeInfo()
VMFS
RescanVmfs()
Mount Operations
QueryUnresolvedVmfsVolume()
ResolveMultipleUnresolvedVmfsVolumes()
UnmountForceMountedVmfsVolumes()
Provisioning
FormatVmfs()
AttachVmfsExtent()
ExpandVmfsExtent()
Upgrade
UpgradeVmfs()
UpgradeVmfsLayout()

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 103
HostStorageSystem	Multipath management.	"Multipath Management" on page 104
HostStorageSystem	iSCSI Storage setup and configuration.	"Configuring iSCSI Storage" on page 104
HostDatastoreSystem	Creating and managing VMFS datastores and remote datastores.	"Creating and Managing Datastores" on page 106
HostDatastoreSystem HostStorageSystem	Managing VMFS volume copies (resignature or force mount).	"Managing VMFS Volume Copies (Resignaturing)" on page 109
HostDiagnosticSystem	Creating an managing diagnostic partitions.	"Managing Diagnostic Partitions" on page 110

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 108.

- `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 104.

Multipath Management

The *vSphere Storage* documentation includes information about using multipathing for failover and load balancing. You can manage multipathing using the vSphere Client, the `esxcli` command, or using the following commands. Use the `HostStorageSystem.multipathStateInfo` property to access the `HostMultipathStateInfo` data object that describes runtime information about the state of 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 *Configuring iSCSI Adapters and Storage* in the *vSphere Storage* documentation for a detailed discussion.

The following `HostStorageSystem` methods are available for iSCSI storage management.

- Add a dynamic or static target.
  - `AddInternetScsiSendTarget` – Adds send target entries to the host bus adapter discovery list if the `DiscoveryProperties.sendTargetsDiscoveryEnabled` flag is set to true.
  - `AddInternetScsiStaticTargets` – Adds static target entries to the host bus adapter discovery list. The `DiscoveryProperty.staticTargetDiscoveryEnabled` flag must be set to true.
- Configure targets.
  - `UpdateInternetScsiAdvancedOptions` – Updates the advanced options that the iSCSI host bus adapter or the discovery addresses and targets associated with it.
  - `UpdateInternetScsiAlias` – Updates the alias of an iSCSI host bus adapter.
  - `UpdateInternetScsiAuthenticationProperties` – Updates the authentication properties for one or more targets or discovery addresses associated with an iSCSI host bus adapter.
  - `UpdateInternetScsiDigestProperties` – Updates the digest properties for the iSCSI host bus adapter or the discovery addresses and targets associated with it.
  - `UpdateInternetScsiDiscoveryProperties` – Updates the discovery properties for an iSCSI host bus adapter.
  - `UpdateInternetScsiIPProperties` – Updates the IP properties for an iSCSI host bus adapter.
  - `UpdateInternetScsiName` – Updates the name of an iSCSI host bus adapter.
  - `UpdateSoftwareInternetScsiEnabled` – Enables and disables software iSCSI in the VMkernel.

- Remove a dynamic or static target.
  - `RemoveInternetScsiSendTargets` – Removes send target entries from the host bus adapter discovery list. The `DiscoveryProperty.sendTargetsDiscoveryEnabled` must be set to true. If any of the targets provided as parameters are not found in the existing list, the other targets are removed and an exception is thrown.
  - `RemoveInternetScsiStaticTargets` – Remove static target entries from the host bus adapter discovery list. The `DiscoveryProperty.staticTargetDiscoveryEnabled` must be set to true. If any of the targets provided as parameters are not found in the existing list, the other targets are removed and an exception is thrown.

iSCSI initiators and targets have unique, permanent iSCSI names and addresses. An iSCSI name correctly identifies a specific iSCSI initiator or target, regardless of physical location. Names must be in EUI or IQN format, as specified by the storage vendor's hardware.

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 119. 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 57. From the `HostSystem.config` property, you can obtain the list (array) of host bus adapters by specifying this property path:

   ```
   config.storageDevice.hostBusAdapter
   ```

   The property path returns an array of host bus adapters. For example:

   ```
   hostBusAdapter["key-vim.host.BlockHba-vmhba32"]
   hostBusAdapter["key-vim.host.BlockHba-vmhba33"]
   hostBusAdapter["key-vim.host.BlockHba-vmhba34"]
   hostBusAdapter["key-vim.host.BlockHba-vmhba35"]
   hostBusAdapter["key-vim.host.BlockHba-vmhba1"]
   ...
   ```

2. From the array, select the host bus adapter (instance of `HostHostBusAdapter`) that you want to configure and obtain its `key` property, which is the device name of the host bus adapter as a string.
3. Determine the capabilities of the adapter by retrieving the properties of the `HostHostBusAdapter` object.
4. Configure the initiator.
   - For an independent hardware initiator, configure the IP address.
   - For a software initiator, enable the software initiator in the VMkernel.
5. Configure the iSCSI name by calling `HostStorageSystem.UpdateInternetScisiName` and the alias by running `HostStorageSystem.UpdateInternetScisiAlias`.
6. Configure target discovery by calling `HostStorageSystem.UpdateInternetScisiHbaDiscoveryProperties`.

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

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

   The `HostInternetScsiHbaAuthenticationProperties` object you pass into that method includes properties for configuring CHAP and Mutual CHAP. See the *vSphere Storage* documentation for information about securing your iSCSI storage array.

8 Configure access to the targets.

9 Rescan the HBAs.

   Rescan enables the HBAs to discover the new storage devices. You can either rescan a single HBA with `HostStorageSystem.RescanHba`, specifying the HBA ID as a parameter, or rescan all HBAs using `HostStorageSystem.RescanAllHba`.

Creating and Managing Datastores

Each datastore is a logical container, analogous to a file system on a logical volume, where the host places virtual disk files and other virtual machine files. Datastores hide specifics of the physical storage device and provide a uniform model for storing virtual machine files.

The `HostDatastoreSystem` managed 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.

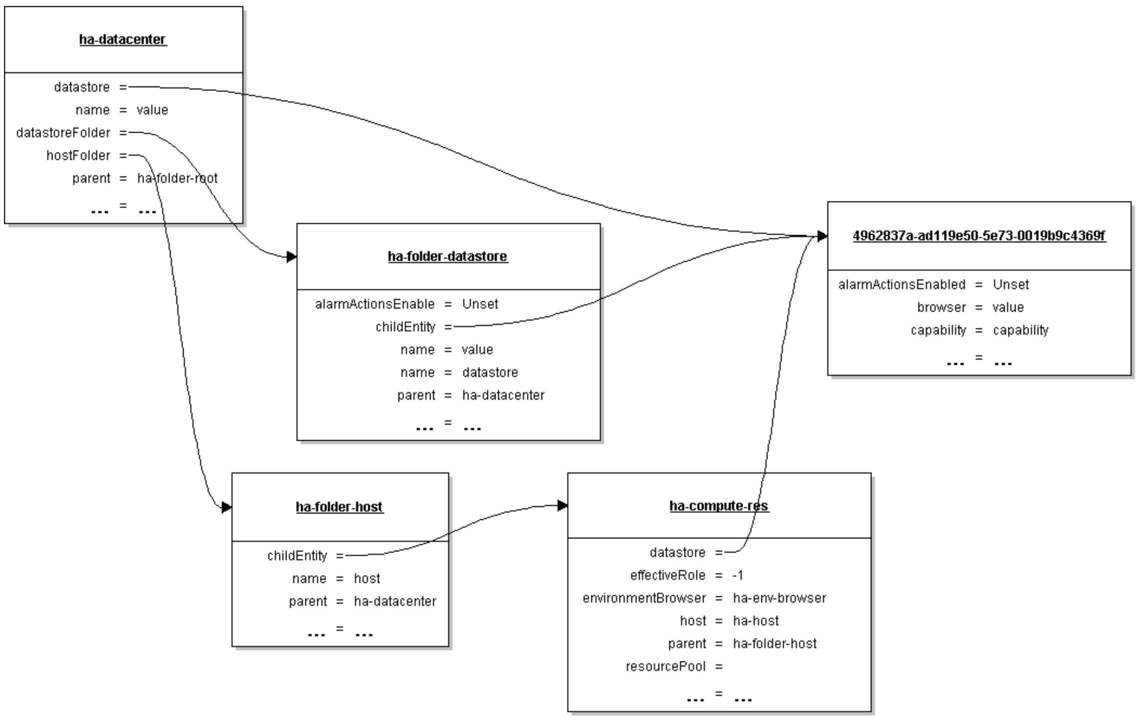
4 Call `CreateVmfsDatastore`, `ExtendVmfsDatastore`, or `ExpandVmfsDatastore` to complete the VMFS provisioning operation.

Accessing Datastores

Figure 8-4 illustrates how you can access or specify datastores. See Chapter 4, "Datacenter Inventory," on page 49 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 128.

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 `HostScisiDisk` objects.
3. Call `HostDatastoreSystem.QueryVmfsDatastoreCreateOptions` for information about options for 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 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 *vSphere Storage* documentation 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 220. 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\vsphere-ws\java\JAXWS\samples\com\vmware\)	**C# (SDK\vsphere-ws\dotnet\cs\samples\)**
scsilun\SCSILunName.java	SCSILunName\SCSILunName.cs
--	SCSILunName\SCSILunName.csproj
--	SCSILunName\SCSILunName2008.csproj
--	SCSILunName\SCSILunName2010.csproj
httpfileaccess\GetVMFiles.java	GetVirtualDiskFiles\GetVirtualDiskFiles.cs

vSphere Networks 9

Before you add storage and virtual machines to an ESXi system, you should have completed networking setup. This chapter describes how to set up virtual switches in the vSphere environment.

The chapter includes the following topics:

Virtual Switches

vSphere supports the use of virtual switches to manage network traffic to and from virtual machines.

- vCenter Server supports a distributed network model in which a distributed virtual switch manages ESXi host proxy switch configuration. In the distributed network model, a host proxy switch reflects the distributed virtual switch port settings, describes how physical network adapters are bridged to the switch, and performs network I/O.
- On a standalone ESXi host, you can use a VMware standard virtual switch to support network traffic to and from virtual machines on the host.

To configure a vSphere network you perform the following operations:

- Set up virtual switches
- Define portgroups
- Configure physical network adapters

Port Group

Port groups aggregate multiple ports under a common configuration. Each port can connect to a network adapter of a virtual machine, or an uplink adapter on the physical machine.

Each port group is identified by a network label, which is unique to the current host. Network labels make virtual machine configuration portable across hosts. All port groups in a datacenter that are physically connected to the same network (in the sense that each can receive broadcasts from the others) are given the same label. Conversely, if two port groups cannot receive broadcasts from each other, they have distinct labels.

You can use a VLAN ID to restrict port group traffic to a logical Ethernet segment within the physical network. For a port group to reach port groups located on other VLANs, the VLAN ID must be set to 4095. If you use VLAN IDs, you must change the port group labels and VLAN IDs together so that the labels properly represent connectivity.

Virtual Machine Network 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 134. A limited number of adapters is supported. KB article 1001805 (http://kb.vmware.com/kb/1001805) discusses available network adapters and which adapter is appropriate in which situation.

VMkernel Network Interfaces

The network services that the VMkernel provides (iSCSI, NFS, and VMotion) use a TCP/IP stack in the VMkernel. This stack accesses various networks by attaching to one or more port groups on one or more virtual switches.

The VMware VMkernel TCP/IP networking stack handles iSCSI, NFS, and VMotion in the following ways.

- iSCSI as a virtual machine datastore
- iSCSI for the direct mounting of .ISO files, which are presented as CD-ROMs to virtual machines
- NFS as a virtual machine datastore
- NFS for the direct mounting of .ISO files, which are presented as CD-ROMs to virtual machines
- Migration with VMotion

If you have two or more physical NICs for iSCSI, you can create multiple paths for the software iSCSI by using port binding. For more information on port binding, see the *iSCSI SAN Configuration Guide*.

A freshly installed ESX/ESXi system does not include VMkernel network interfaces. When you wish to migrate a virtual machine with VMotion, your VMkernel networking stack must be set up properly. When you want to use storage types that use TCP/IP network communications, such as iSCSI, you must provide a separate VMkernel network interface for that storage device. You must create any VMkernel ports you might need (see "Adding a VMkernel Network Interface" on page 119).

Physical Network Adapter (pnic)

The term pnic refers to the physical network adapters as seen by the primary operating system. When using the vSphere Web Services SDK, you can manipulate the adapter directly. When using the vSphere Client GUI, you manipulate instead the uplink adapter. On an ESXi host, each pnic has one associated uplink adapter.

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

Using a Distributed Virtual Switch

A `DistributedVirtualSwitch` managed object is a virtual network switch that is located on a vCenter Server. A distributed virtual switch manages configuration for proxy switches (`HostProxySwitch`). A proxy switch is located on an ESXi host that is managed by the vCenter Server and is a member of the switch. A distributed switch also provides virtual port state management so that port state is maintained when vCenter Server operations move a virtual machine from one host to another.

A proxy switch performs network I/O to support the following network traffic and operations:

- Network traffic between virtual machines on any hosts that are members of the distributed virtual switch.
- Network traffic between a virtual machine that uses a distributed virtual switch and a virtual machine that uses a VMware standard virtual switch.
- Network traffic between a virtual machine and a remote system on a physical network connected to the ESXi host.
- vSphere system operations to support capabilities such as VMotion or High Availability.

A `DistributedVirtualSwitch` is the base distributed switch implementation. It supports a VMware distributed virtual switch implementation and it supports third party distributed switch implementations. The base implementation provides the following capabilities (defined in the `DVSFeatureCapability` object):

- NIC teaming
- Network I/O control
- Network resource allocation
- Quality of service tag support
- User-defined resource pools
- I/O passthrough (VMDirectPath Gen2)

A `VmwareDistributedVirtualSwitch` supports the following additional capabilities (defined in the `DVSFeatureCapability` and `VMwareDVSFeatureCapability` objects):

- Backup, restore, and rollback for a VMware distributed virtual switch and its associated portgroups.
- Maximum Transmission Unit (MTU) configuration.
- Health check operations for NIC teaming and VLAN/MTU support.
- Monitoring switch traffic using Internet Protocol Flow Information Export (IPFIX).
- Link Layer Discovery Protocol (LLDP).
- Virtual network segmentation using a Private VLAN (PVLAN).
- VLAN-based SPAN (VSPAN) for virtual distributed port mirroring.
- Link Aggregation Control Protocol (LACP) defined for uplink portgroups.

Distributed Virtual Switch Configuration

To use a distributed virtual switch, you create a switch and portgroups on a vCenter Server, and add hosts as members of the switch.

1. Use the `Folder.CreateDVS_Task` method to create a distributed virtual switch. Use a `DVSConfigSpec` to create a switch for a third-party implementation. Use a `VMwareDVSConfigSpec` to create a VMware distributed virtual switch.
2. Use the `CreateDVPortgroup_Task` method to create portgroups for host and virtual machine network connections and for the connection between proxy switches and physical NICs. A `DistributedVirtualPortgroup` specifies how virtual ports (`DistributedVirtualPort`) will be used. When you create a distributed virtual switch, the vCenter Server automatically creates one uplink portgroup (`config.uplinkPortgroup`). Uplink portgroups are distributed virtual portgroups that support the connection between proxy switches and physical NICs.

   Port creation on a distributed switch is determined by the portgroup type (`DVPortgroupConfigSpec.type`):

   - If a portgroup is early binding (static), then `DVPortgroupConfigSpec.numPorts` determines the number of ports that get created when the portgroup is created. This number can be increased if `DVPortgroupConfigSpec.autoExpand` is true.
   - If a portgroup is ephemeral (dynamic), then numPorts is ignored and ports are created as needed.

   You can also specify standalone ports that are not associated with a port group and uplink ports that are created on ESXi hosts (`DVSConfigSpec.numStandalonePorts`).

   The `DVPortgroupConfigInfo.numPorts` property is the total number of ports for a distributed virtual switch. This total includes the ports generated by the static and dynamic portgroups and the standalone ports.

3 If you have created additional uplink portgroups, use the ReconfigureDvs_Task method to add the portgroup(s) to the `DVSConfigSpec.uplinkPortgroup` array.

4 Retrieve physical NIC device names from the host (`HostSystem.config.network.pnic[].device`).

5 Add host member(s) to the distributed virtual switch. To configure host members:

- Specify hosts (`DVSConfigSpec.host[]`).
- For each host, specify one or more physical NIC device names to identify the pNIC(s) for the host proxy connection to the network (`DistributedVirtualSwitchHostMemberConfigSpec.backing.pnicSpec[].pnicDevice`)
- Use the DistributedVirtualSwitch.ReconfigureDvs_Task method to update the switch configuration.

When you add a host to a distributed virtual switch (DistributedVirtualSwitch.config.host), the host automatically creates a proxy switch. The proxy switch is removed automatically when the host is removed from the distributed virtual switch.

6 Connect hosts and virtual machines to the distributed virtual switch.

Host connection	Specify port or portgroup connections in the host virtual NIC spec (`HostVirtualNicSpec.distributedVirtualPort` or `HostVirtualNicSpec.portgroup`).
Virtual machine connection	Specify port or portgroup connections in the distributed virtual port backing (`VirtualEthernetCardDistributedVirtualPortBackingInfo`) for the virtual Ethernet cards on the virtual machine (`VirtualEthernetCard.backing`).

Backup, Rollback, and Query Operations

If you are using a `VmwareDistributedVirtualSwitch`, you can perform backup and rollback operations on the switch and its associated distributed virtual portgroups.

When you reconfigure a VMware distributed virtual switch (`ReconfigureDvs_Task`), the Server saves the current switch configuration before applying the configuration updates. The saved switch configuration includes portgroup configuration data. The Server uses the saved switch configuration as a checkpoint for rollback operations. You can rollback the switch or portgroup configuration to the saved configuration, or you can rollback to a backup configuration (`EntityBackupConfig`).

- To backup the switch and portgroup configuration, use the `DistributedVirtualSwitchManager.DVSManagerExportEntity_Task` method. The export method produces a `EntityBackupConfig` object. The backup configuration contains the switch and/or portgroups specified in the SelectionSet parameter. To backup the complete configuration you must select the distributed virtual switch and all of its portgroups.
- To rollback the switch configuration, use the `DVSRollback_Task` method to determine if the switch configuration has changed. If it has changed, use the `ReconfigureDvs_Task` method to complete the rollback operation.
- To rollback the portgroup configuration, use the `DistributedVirtualPortgroup.DVPortgroupRollback_Task` method to determine if the portgroup configuration has changed. If it has changed, use the `ReconfigureDVPortgroup_Task` method to complete the rollback operation.

To perform query operations on a distributed virtual switch, use the `DistributedVirtualSwitchManager` methods.

VMware Standard Virtual Switch

Network setup for ESXi hosts can consist of several parts:

- Setting up one or more virtual switches. Virtual switches provide the connectivity between virtual machines on the same host or on different hosts. Virtual switches also support VMkernel network access for VMotion, iSCSI, and NFS. You set up virtual switches independently on each host. See "Adding a Standard Virtual Switch" on page 118.
- Adding virtual machine port groups. A virtual machine always accesses the network through a port group. See "Adding a Virtual Port Group" on page 119.
- Specifying the adapter for the virtual machine. This adapter is specified as a virtual device, configured as part of virtual machine setup, and discussed in "Configuring a Virtual Machine" on page 129.
- Adding VMkernel network interfaces, for example, to support iSCSI storage or VMotion. See "Adding a VMkernel Network Interface" on page 119.
- Configuring a physical adapter (pnic), the actual connection from the host to the network. You can configure the pnic through the `HostNetworkSystem.pnic` property, which is a `PhysicalNic` data object. You can specify the set of pnics associated with a virtual switch through the `VirtualSwitch.pnic` property, which takes an array of physical network adapters.
- Network configuration for the host (IP routing, DNS, SNMP). See "Adding Networking Services" on page 122.

To use a VMware standard virtual switch, you use the following elements to configure the switch on an ESXi host.

- `HostNetworkSystem` – Managed object that represents the host's networking configuration. This object's properties point to the networking data objects you can use for network management, including `HostDnsConfig` and `HostIpRouteConfig`.

  `HostNetworkSystem` properties allow you to acess `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.

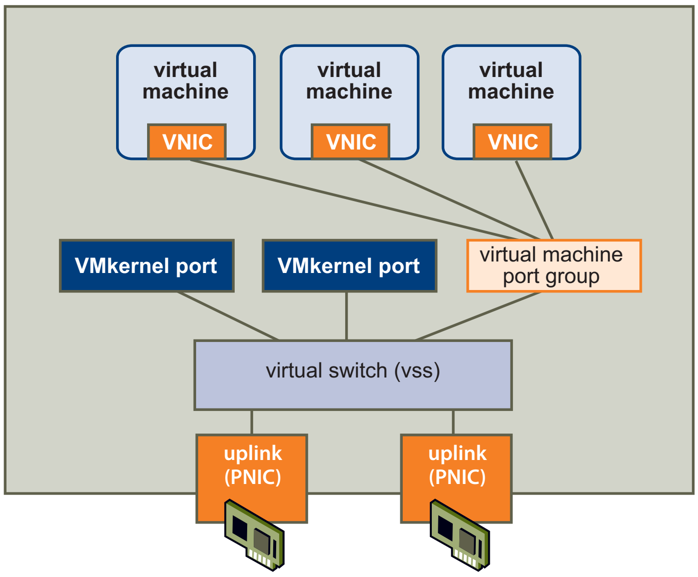
  When you add a host to a vCenter Server system, or when you add a virtual machine to an ESX/ESXi host, a `Network` is added automatically.
- `HostSystem.QueryHostConnectionInfo` and `Datacenter.QueryConnectionInfo` both return a `HostConnectInfo` data object, which describes the current network configuration.

`HostSNMPSystem` – Supports SNMP setup. See "Setting Up SNMP" on page 122.

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).

When two or more virtual machines are connected to the same vSS, network traffic between them is routed locally. If an uplink adapter is attached to the vSS, each virtual machine can access the external network that the adapter is connected to.

Setting Up Networking with vSS

You can use the `HostNetworkSystem` managed object to access and manipulate the elements of an 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 Standard Virtual Switch

You call the `HostNetworkSystem.AddVirtualSwitch` method to add one or more virtual switches. Pass in the name of the virtual switch and a `HostVirtualSwitchSpec` data object as parameters.

Inside `HostVirtualSwitchSpec` you can specify the MTU, number of ports, network policy, and bridge specification. The bridge specifies how the virtual switch connects to the physical adapter. The currently supported bond bridge provides network adapter (NIC) teaming capabilities through the use of a list of physical devices and, optionally, a beacon probe to test connectivity with physical adapters.

After you have created the virtual switch, you can connect it to a pnic for connection to the outside, and to a VMkernel port or a port group.

To add a virtual switch

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 120.

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 120.
- `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.

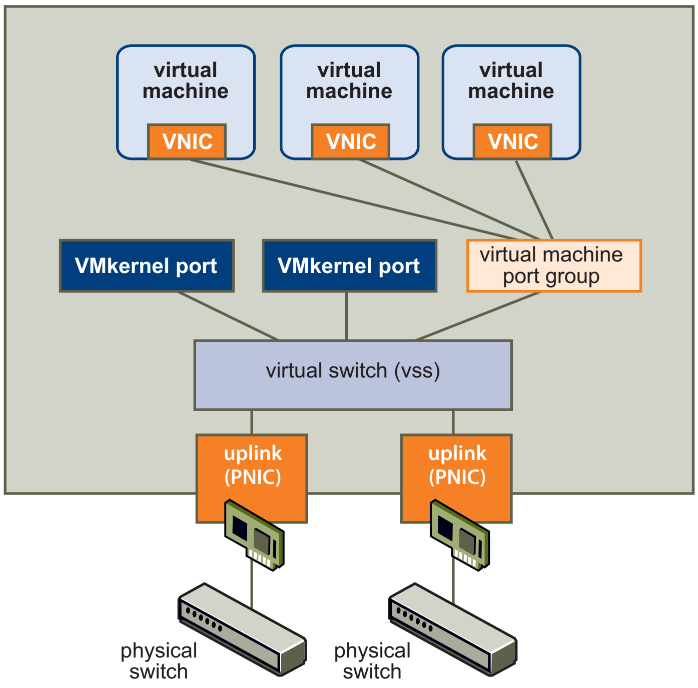
To resolve this issue, you can set up your environment so each virtual switch connects to two uplink adapters. Each uplink adapter connects to two different physical switches. The teams can then either share the load of traffic between physical and virtual networks among some or all of its members, or 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 ESXi system by using `HostConfigManager` properties and methods.

Adding an NTP Service

The `HostConfigManager.dateTimeSystem` property contains a `HostDateTimeSystem` data object. This object allows you to perform NTP and date and time related configuration.

- Query and update the date and time information by using one of the methods defined in `HostDateTimeSystem`.
- Modify the `HostDateTimeSystem.dateTimeInfo` property, which contains a `HostDateTimeInfo` object, to set up NTP. The NTP information is stored in the `HostDateTimeInfo.ntpConfig` property, which is a `HostNtpConfig` object. The `HostNtpConfig` objects's `server` property contains a list of time servers, specified by IP address or fully qualified domain name.

IMPORTANT You can start and stop the NTP daemon and retrieve information about it by using the `HostServiceSystem` object.

Setting Up the IP Route Configuration

You can use the `HostNetworkSystem.UpdateIPRouteConfig` method to specify the IP route configuration for an 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 how to use some of the managed objects discussed in this chapter.

Table 9-1. Networking Sample Applications

Java (SDK\vsphere-ws\java\JAXWS\samples\com\vmware\host)	C# (SDK\vsphere-ws\dotnet\cs\samples\)
`AddVirtualNic.java`	`AddVirtualNic\AddVirtualNic.cs`
	`AddVirtualNic\AddVirtualNic.csproj`
	`AddVirtualNic\AddVirtualNic2008.csproj`
	`AddVirtualNic\AddVirtualNic2010.csproj`
`AddVirtualSwitch.java`	`AddVirtualSwitch\AddVirtualSwitch.cs`
	`AddVirtualSwitch\AddVirtualSwitch.csproj`
	`AddVirtualSwitch\AddVirtualSwitch2008.csproj`
	`AddVirtualSwitch\AddVirtualSwitch2010.csproj`
`AddVirtualSwitchPortGroup.java`	`AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup.cs`
	`AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup.csproj`
	`AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup2008.csproj`
	`AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup2010.csproj`
`RemoveVirtualNic.java`	`RemoveVirtualNic\RemoveVirtualNic.cs`
	`RemoveVirtualNic\RemoveVirtualNic.csproj`
	`RemoveVirtualNic\RemoveVirtualNic2008.csproj`
	`RemoveVirtualNic\RemoveVirtualNic2010.csproj`
`RemoveVirtualSwitch.java`	`RemoveVirtualSwitch\RemoveVirtualSwitch.cs`
	`RemoveVirtualSwitch\RemoveVirtualSwitch.csproj`
	`RemoveVirtualSwitch\RemoveVirtualSwitch2008.csproj`
	`RemoveVirtualSwitch\RemoveVirtualSwitch2010.csproj`
`RemoveVirtualSwitchPortGroup.java`	`RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup.cs`
	`RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup.csproj`
	`RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup2008.csproj`
	`RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup2010.csproj`

Virtual Machine Configuration 10

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:

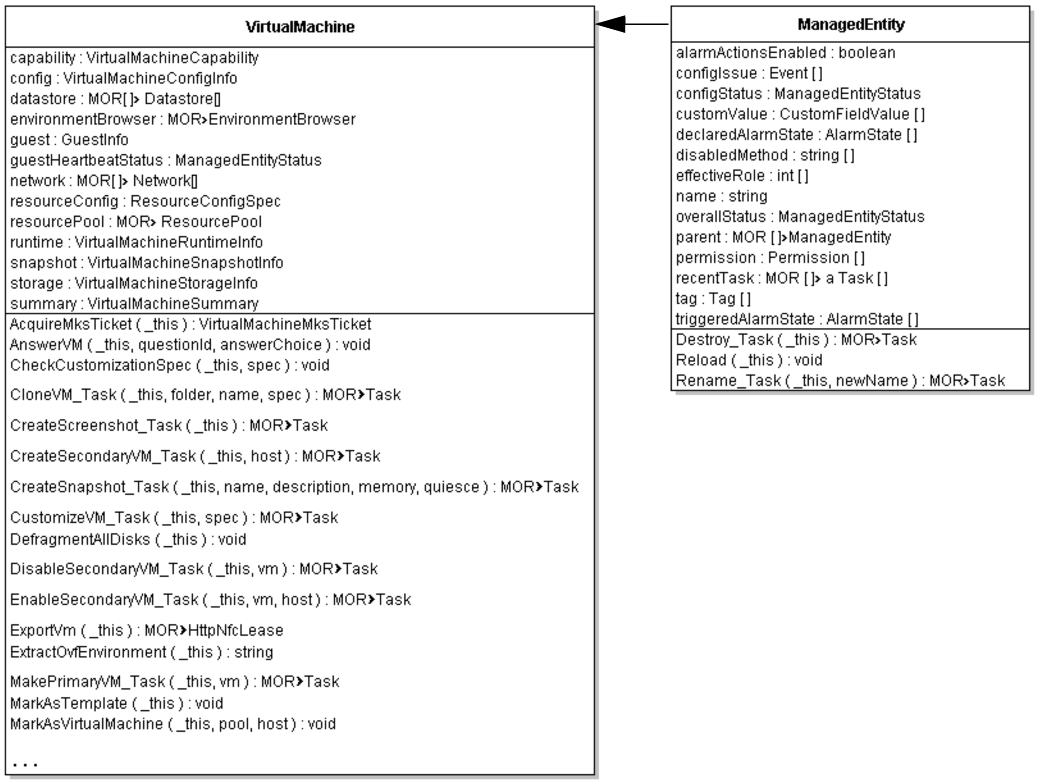
VirtualMachine Management Objects and Methods

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


VirtualMachine
capability : VirtualMachineCapability
config : VirtualMachineConfigInfo
datastore : MOR[ ]> Datastore[]
environmentBrowser : MOR>EnvironmentBrowser
guest : GuestInfo
guestHeartbeatStatus : ManagedEntityStatus
network : MOR[ ]> Network[]
resourceConfig : ResourceConfigSpec
resourcePool : MOR> ResourcePool
runtime : VirtualMachineRuntimeInfo
snapshot : VirtualMachineSnapshotInfo
storage : VirtualMachineStorageInfo
summary : VirtualMachineSummary
AcquireMksTicket (_this) : VirtualMachineMksTicket
AnswerVM (_this, questionId, answerChoice) : void
CheckCustomizationSpec (_this, spec) : void
CloneVM_Task (_this, folder, name, spec) : MOR>Task
CreateScreenshot_Task (_this) : MOR>Task
CreateSecondaryVM_Task (_this, host) : MOR>Task
CreateSnapshot_Task (_this, name, description, memory, quiesce) : MOR>Task
CustomizeVM_Task (_this, spec) : MOR>Task
DefragmentAllDisks (_this) : void
DisableSecondaryVM_Task (_this, vm) : MOR>Task
EnableSecondaryVM_Task (_this, vm, host) : MOR>Task
ExportVm (_this) : MOR>HttpNfcLease
ExtractOvfEnvironment (_this) : string
MakePrimaryVM_Task (_this, vm) : MOR>Task
MarkAsTemplate (_this) : void
MarkAsVirtualMachine (_this, pool, host) : void
. . .
ManagedEntity
alarmActionsEnabled : boolean
configIssue : Event [ ]
configStatus : ManagedEntityStatus
customValue : CustomFieldValue [ ]
declaredAlarmState : AlarmState [ ]
disabledMethod : string [ ]
effectiveRole : int [ ]
name : string
overallStatus : ManagedEntityStatus
parent : MOR [ ]>ManagedEntity
permission : Permission [ ]
recentTask : MOR [ ]> a Task [ ]
tag : Tag [ ]
triggeredAlarmState : AlarmState [ ]
Destroy_Task (_this) : MOR>Task
Reload (_this) : void
Rename_Task (_this, newName) : MOR>Task

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 127)
- `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 129 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 128.

Additional Configuration Information

The `VirtualMachineConfigInfo` and `VirtualMachineConfigSpec` objects provide the `extraConfig` property for additional configuration information. The `extraConfig` property is an array of key/value pairs that identify configuration options. The Server stores the `extraConfig` options in the `.vmx` file for the virtual machine. As the vSphere API evolves from version to version, an extraConfig option may become a standard configuration property that is part of the defined inventory data model. In this case, you must use the standard data model property for access; you cannot use the `extraConfig` property.

Creating Virtual Machine Templates

Templates allow you to create multiple virtual machines with the same characteristics, such as resources allocated to CPU and memory, or type of virtual hardware. A virtual machine template is a virtual machine that cannot be powered on and that is not associated with a resource pool.

You can convert any powered off virtual machine to a template by calling `VirtualMachine.MarkAsTemplate`. After the conversion, the original virtual machine no longer exists. You can use the template to create multiple clones of the same configuration.

Cloning a Virtual Machine

A clone is a copy of a virtual machine. The main difference between a virtual machine and a clone is that the `VirtualMachine.config.template` property is set to `true`.

You can create a clone in one of the following ways:

- If you no longer need a specific instance of a virtual machine, but you want to use the virtual machine’s configuration as a template, use the `VirtualMachine.MarkAsTemplate` method. This method sets the `config.template` property to `true`, and disables the virtual machine.
- If you want to use an existing virtual machine as a template, but keep the virtual machine, call the `VirtualMachine.CloneVM_Task` method to create a duplicate of the virtual machine.

If you use the `VirtualMachine.CloneVM_Task` method, you can customize certain attributes of the clone by specifying them in the `VirtualMachineCloneSpec` data object you pass in when you call the method.

The following code fragment from VMClone.java illustrates how you can customize a clone and specify a new location for it.

Example 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 a Virtual Machine

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

Checking Default Files

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

Table 10-1. Virtual Machine Files

File	Usage	File Description	File Format
.vmx	*.vmname*.vmx	Virtual machine configuration file.	ASCII
.vmxf	*vmname*.vmxf	Additional virtual machine configuration files, available, for example, with teamed virtual machines.	ASCII
.vmdk	*vmname*.vmdk	Virtual disk file.	ASCII
.flat.vmdk	*vmname*.flat.vmdk	Preallocated virtual disk in binary format.	Binary
.vswp	*vmname*.vswp	Swap file.	
.nvram	*vmname*.nvram or nvram	Non-volatile RAM. Stores virtual machine BIOS information.	
.vmss	*vmname*.vmss	Virtual machine suspend file.	
.log	vmware.log	Virtual machine log file.	ASCII
#.log	vmware-#.log	Old virtual machine log files. # is a number starting with 1.	ASCII
.vmtx	*vmname*.vmtx	Virtual machine template file.	ASCII

If you are using snapshots, the following additional files might be available. See "Snapshots" on page 140.

Table 10-2. Virtual Machine Snapshot Files

File Extension	Usage	File Description
`.vmsd`	*vmname*.`vmsd`	Virtual machine snapshot file.
`.vmsn`	*vmname*.`vmsn`	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.

Checking Default Devices

When you create a virtual machine, you are also creating a set of default devices, based on the hardware version associated with your SDK. You can see these devices using the `EnvironmentBrowser.QueryConfigOption` method. For example, the IDE controllers are created by default. Many of these default devices contain properties that you cannot change.

However, you can add the following optional devices to the default set: `VirtualSerialPort`, `VirtualParallelPort`, `VirtualFloppy`, `VirtualCdrom`, `VirtualUSB`, `VirtualEthernetCard`, `VirtualDisk`, and `VirtualSCSIPassthrough`. See the VirtualDevice Data Object in the *API Reference* for more information about each of these optional devices.

CAUTION Do not try to change default device properties using the `VirtualMachineConfigSpec.deviceChange` method discussed in "Adding Devices to Virtual Machines" on page 134, because the `deviceChange` method is not applicable to default device properties.

Configuring a Virtual Machine

You can configure a virtual machine during creation (`Folder.CreateVM_Task`) or cloning (`VirtualMachine.CloneVM_Task`). You can also reconfigure a virtual machine using `VirtualMachine.ReconfigVM_Task`. However, do not use the `VirtualMachine.ReconfigVM_Task` call to create or add a disk.

In vSphere 5.5 and later, the `ReconfigVM_Task` method will throw an error when it attempts to change certain virtual machine properties while the virtual machine is powered on. In previous releases, the Server would modify the properties in the configuration specification and the changes would take effect after the virtual machine resets, reboots, or performs a fast suspend and resume.

Since vSphere 5.5, a reconfigure operation modifies the virtual machine in real time, so that the virtual machine properties have been changed by the time the method returns control to the client.

You cannot reconfigure a virtual machine successfully if you specify any of the following properties in the `VirtualMachineConfigSpec` when you call the `ReconfigVM_Task` method.

Property	VirtualMachineConfigSpec Path
VirtualDevice.unitNumber	deviceChange.device.unitNumber
VirtualDevicePciBuslSlotInfo.pciSlotNumber	deviceChange.device.slotInfo.pciSlotNumber
VirtualDiskFlatVer1BackingInfo.diskMode VirtualDiskFlatVer2BackingInfo.diskMode VirtualDiskRawDiskMappingVer1BackingInfo.diskMode VirtualDiskSeSparseBackingInfo.diskMode VirtualDiskSparseVer1BackingInfo.diskMode VirtualDiskSparseVer2BackingInfo.diskMode	deviceChange.device.backing.diskMode

Property	VirtualMachineConfigSpec Path
VirtualDiskFlatVer2BackingInfo.digestEnabled VirtualDiskSeSparseBackingInfo.digestEnabled	deviceChange.device.backing.digestEnabled
VirtualMachineConfigSpec.changeTrackingEnabled	changeTrackingEnabled
VirtualEthernetCard.addressType	deviceChange.device.addressType
VirtualEthernetCard.macAddress	deviceChange.device.macAddress
VirtualEthernetCard.wakeOnLanEnabled	deviceChange.device.wakeOnLanEnabled
VirtualSCSIController.sharedBus	deviceChange.device.sharedBus
VirtualSerialPort.yieldOnPoll	deviceChange.device.yieldOnPoll
VirtualUSBController.autoConnectDevices VirtualUSBXHCIController.autoConnectDevices	deviceChange.device.autoConnectDevices
VirtualUSBController.ehciEnabled	deviceChange.device.ehciEnabled
VirtualMachineVideoCard.useAutoDetect	deviceChange.device.useAutoDetect
VirtualMachineVideoCard.videoRamSizeInKB	deviceChange.device.videoRamSizeInKB
VirtualMachineVideoCard.numDisplays	deviceChange.device.numDisplays
VirtualMachineVideoCard.use3dRendererSupported	deviceChange.device.use3dRendererSupported

The *API reference* lists all properties and includes information about required permissions for these configuration methods. The following sections describe some commonly specified attributes.

Name and Location

You can specify the display name for the virtual machine by setting the `VirtualMachineConfigSpec.name` property. Any % (percent) character used in this `name` parameter must be escaped, unless it is used to start an escape sequence. Clients can also escape any other characters in this name parameter.

Use the `annotation` field to provide a description of the virtual machine. To remove an existing description, specify the empty string as the value of `annotation`.

The location of the virtual machine is determined implicitly during creation because you call a `Folder.CreateVM_Task` method and specify resource pool and optional target host the virtual machine should belong to. See Chapter 13, "Resource Management Objects," on page 157 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 noticably. 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 158 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 134.

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.

   NOTE You cannot use the `QueryConfigOption` method to create another instance of a default device. If you attempt to add a default device, such as an IDE controller, the server ignores the operation.

2. Specify the backing information object for the device. The actual process for defining the object differs for different objects. For example, for a CD-ROM passthrough device, you use a `VirtualCdromPassthroughBackingInfo` device. The `VirtualDevice.backing` property is a `VirtualDeviceBackingInfo` object which is extended by devices.

   The following code fragment adds a CD-ROM passthrough device:

   ```
   VirtualCdromPassthroughBackingInfo vcpbi = new VirtualCdromPassthroughBackingInfo();
   // Does the virtual device have exclusive access to the CD-ROM device?
   vcpbi.setExclusive(false);
   // Specifies the device name.
   vcpbi.setDeviceName('cdrom0');
   ```

3. Specify connection information for the device.

   The `VirtualDevice.connectable` property is a `VirtualDeviceConnectInfo` data object. This object provides information about restrictions on removing the device while a virtual machine is running. This property is `null` if the device is not removable.

   ```
   VirtualDeviceConnectInfo vdci = new VirtualDeviceConnectInfo();
   ```

```
// Allow the guest to control whether the virtual device is connected?
vdci.setAllowGuestControl(false);
// Is the device currently connected?
vdci.setConnected(true);
// Connect the device when the virtual machine starts?
vdci.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(vdci);
vd.setControllerKey(257);
vd.setDeviceInfo(vddesc);
vd.setKey(2);
vd.setUnitNumber(25);
```

Performing Virtual Machine Power Operations

Just like physical machines, virtual machines have power states.

- Powered on – The virtual machine is running. If no OS has been installed, you can perform OS installation as you would for a physical machine.
- Powered off – The virtual machine is not running. You can still update the software on the virtual machine's physical disk, which is impossible for physical machines.
- Suspended – The virtual machine is paused and can be resumed; like a physical machine in standby or hibernate state.

IMPORTANT Before you power on a virtual machine, you must make sure that the host has sufficient resources. You must have enough memory for the virtual machine, and some memory overhead. See "Querying Virtual Machine Memory Overhead" on page 98.

`VirtualMachine` power operations allow you to change the power state. Each operation is sensitive to the current power state, for example, powering on a powered off virtual machine has the desired result while powering on a powered on virtual machine results in an error. You must check the current state before you run one of these tasks.

- `PowerOnVM_Task` – Powers on a virtual machine. If the virtual machine is suspended, this method resumes execution from the suspend point.
- `PowerOffVM_Task` – Powers off a virtual machine.

- ResetVM_Task – Resets power on this virtual machine. If the current state is poweredOn, ResetVM_Task first performs a hard powerOff operation. After the power state is poweredOff, ResetVM_Task performs a powerOn operation.

  Although this method functions as a powerOff followed by a powerOn, the two operations are atomic with respect to other clients, meaning that other power operations cannot be performed until the reset method completes.

- SuspendVM_Task – Suspends the virtual machine. You can later power on the suspended virtual machine to the same state.

Virtual machines are often configured to start up the guest operating system when they are started, and try to shut down the guest operating system when being shut down. However, starting and stopping a virtual machine differs from starting and stopping the guest operating system (see "Customizing the Guest Operating System" on page 137).

IMPORTANT Power operations might affect other virtual machines that are participating in a DRS cluster or VMware HA. See Chapter 13, "Resource Management Objects," on page 157 for information about DRS clusters and VMware HA.

You can use the Datacenter.PowerOnMultiVM_Task to power on multiple virtual machines in a datacenter. Pass an array of VirtualMachine managed object references and an array of option values to the method. If any of the virtual machines in the list is manually managed by VMware DRS, the system generates a DRS recommendation that the user needs to apply manually. Standalone or DRS disabled virtual machines are powered on for the current host. Virtual machines managed by DRS, to be placed by DRS, are powered on for the recommended host.

Registering and Unregistering Virtual Machines

When you create a virtual machine, it becomes part of the inventory (inside the folder from which you called the creation method by default), and it is registered. If you copy virtual machine files to relocate the virtual machine, or if you remove the files from the inventory using the vSphere Client, it becomes unregistered and unusable. You cannot power on a virtual machine that is not part of the inventory.

To restore the virtual machine to the inventory, and make it usable again, you can use the RegisterVM_Task method, defined in the Folder managed object. You can register the virtual machine to a host or to a resource pool. You can register the virtual machine as a template if you want to use it to clone other virtual machines from.

The ColdMigration.java sample illustrates both registering and reconfiguring a virtual machine. At the heart of the sample is the following call, which registers the virtual machine. Arguments include the virtual machine's current folder, datastore path, and name, whether to register as a template, and the resource pool or host to register the machine in.

```
ManagedObjectReference taskmor = cb.getConnection().getService().registerVM_Task(
                       vmFolderMor,vmxPath,getVmName(),false,resourcePool,host);
```

After registration, the virtual machine takes its resources (CPU, memory, and so on) from the resource pool or host to which it is registered.

The RemoveManagedObject.java sample illustrates unregistering a virtual machine.

Customizing the Guest Operating System

You install the guest operating system on the virtual machine just as you would install it on a physical machine. Afterwards, you can use the 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 hybernate 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 137.

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 installallation, 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 theToolsConfigInfo 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 11

Virtual machines can perform like physical computers and can be configured like physical computers, as discussed in Chapter 10, "Configuring a Virtual Machine," on page 129. 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

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.
Migration of a suspended virtual machine	Moves a suspended virtual machine to a new host. Optionally, you can relocate configuration and disk files to new storage location.
Migration with VMotion	Moves a powered-on virtual machine to a new host. Migration with VMotion allows you to move a virtual machine to a new host without interruption in the availability of the virtual machine.
Migration with Storage VMotion	Moves the virtual disks or configuration file of a powered-on virtual machine to a new datastore. Migration with Storage VMotion allows you to move a virtual machine's storage without interruption in the availability of the virtual machine.

Migration of a suspended virtual machine and migration with VMotion are both sometimes called hot migration, because they allow migration of a virtual machine without powering it off.

You can move virtual machines manually or set up a scheduled task to perform the cold migration.

Cold Migration

If a virtual machine is shut down, you can move it to a different cluster, resource pool, or host by copying all virtual machine files to a different directory. The `ColdMigration` example illustrates this.

Migration with VMotion

VMware VMotion support the live migration of running virtual machines from one physical server to another with no downtime. The source and destination physical servers can be in the same datacenter or in different datacenters.

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.

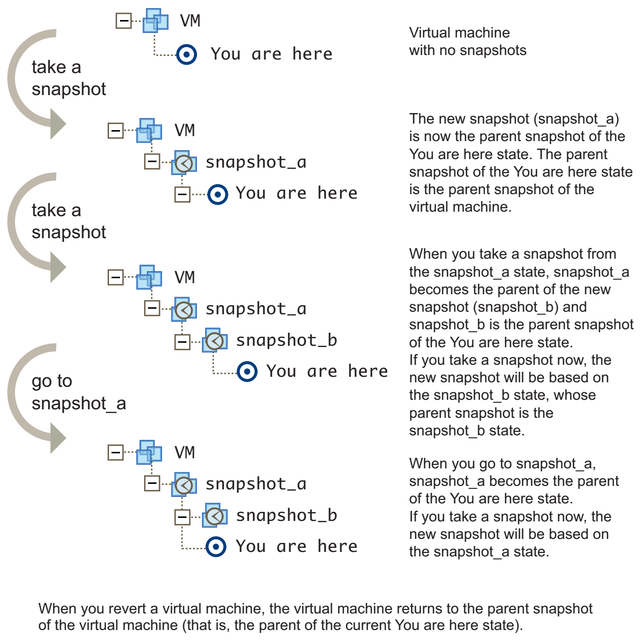
Snapshots

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


VM
You are here
Virtual machine with no snapshots
take a snapshot
VM
snapshot_a
You are here
The new snapshot (snapshot_a) is now the parent snapshot of the You are here state. The parent snapshot of the You are here state is the parent snapshot of the virtual machine.
take a snapshot
When you take a snapshot from the snapshot_a state, snapshot_a becomes the parent of the new snapshot (snapshot_b) and snapshot_b is the parent snapshot of the You are here state.
VM
snapshot_a
snapshot_b
You are here
If you take a snapshot now, the new snapshot will be based on the snapshot_b state, whose parent snapshot is the snapshot_b state.
go to snapshot_a
VM
snapshot_a
snapshot_b
You are here
When you go to snapshot_a, snapshot_a becomes the parent of the You are here state.
If you take a snapshot now, the new snapshot will be based on the snapshot_a state.
When you revert a virtual machine, the virtual machine returns to the parent snapshot of the virtual machine (that is, the parent of the current You are here state).

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 `SupressPowerOn` flag to `true`.

Deleting a Snapshot

You can delete all snapshots by calling `VirtualMachine.RemoveAllSnapshots` or by calling the `VirtualMachineSnapshot.RemoveSnapshot_Task` method. The `VirtualMachineSnapshot` object was previously returned in the task returned by the `CreateSnapshot_Task` method.

Linked Virtual Machines

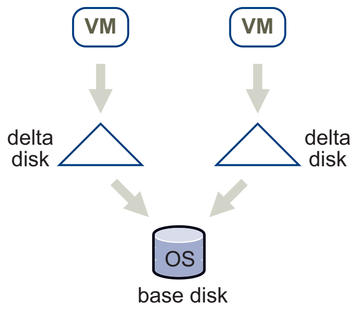
Linked virtual machines are two or more virtual machines that share storage and support efficient sharing of duplicated data.

Linked Virtual Machines and Disk Backings

In its simplest form, shared storage is achieved through the use of delta disk backings. A delta disk backing is a virtual disk file that sits on top of a standard virtual disk backing file. Each time the guest operating system on a virtual machine writes to disk, the data is written to the delta disk. Each time the guest operating system on a virtual machine reads from disk, the virtual machine first targets the disk block in the delta disk. If the data is not on the delta disk, the virtual machine looks for 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 11-2.

Figure 11-2. Linked Virtual Machines with Shared Base Disk Backing and Separate Delta Disk Backing



CAUTION We recommend a limit of up to eight host virtual machines accessing the same *base disk* in a linked virtual machine group. However, you can have an unlimited number of linked virtual machines within each host virtual machine in the group.

Limitation for HA Clusters

Virtual machines in a linked clone group can be part of a VMware HA (high availability) cluster. The number of hosts in a cluster might affect HA's ability to restart a failed virtual machine.

- **Clusters that contain ESX 5.0 or earlier hosts** – If a cluster has eight or fewer hosts, then linked virtual machines restart properly. However, if the cluster has more than eight hosts and any of the hosts are ESX 5.0 or earlier, HA might not be able to restart a virtual machine after it fails. HA is not aware that virtual machines in the linked clone group are subject to the eight host limit. In this case, when HA responds to a failure, it might try to restart the virtual machine on a host that cannot participate in the group due to the maximum host limit. HA will attempt failover five times to different hosts. Thus, in clusters with 13 or more hosts, it is possible that HA will nevdf -ker try a host that is associated with the linked clone group.
- **Clusters that contain only ESX 5.1 or later hosts** – The maximum host limit for a linked clone group is the maximum number of hosts allowed in a cluster. In this case, the number of hosts in the cluster does not affect the ability to restart failed virtual machines.

Creating a Linked Virtual Machine

You can create linked virtual machines in one of two ways:

- Clone the virtual machine from a snapshot.
- Clone the virtual machine from the current virtual machine state. This state might differ from the snapshot point.

Creating a Linked Virtual Machine From a Snapshot

You first create a snapshot, and then create the linked virtual machine from the snapshot.

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)
   ```

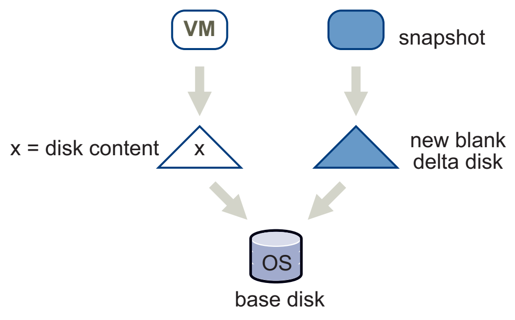
2. To create the linked virtual machine, specify the snapshot you created and use a `VirtualMachineRelocateDiskMoveOptions.diskMoveType` of `createNewDeltaDiskBacking`, as illustrated in Example 11-1. Creating linked virtual machines from a snapshot works with virtual machines in any power state.

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 11-3. Creating a Linked Virtual Machine from a Snapshot



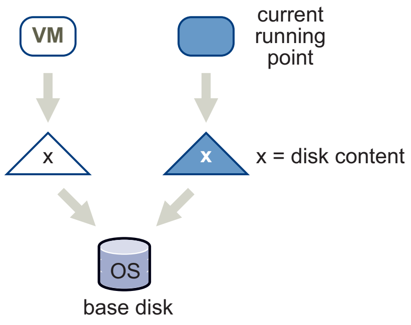
Creating a Linked Virtual Machine From the Current Running Point

To create a virtual machine from the current running point, clone the virtual machine, as in Example 11-1, but use a `diskMoveType` of `moveChildMostDiskBacking`. The virtual machine can be in any power state.

Example 11-2. 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 11-4. Creating a Linked Virtual Machine from the Current Running Point



Removing Snapshots and Deleting Linked Virtual Machines

After you have created a group of linked virtual machines, you can remove a snapshot that was the basis for a linked virtual machine, or delete a virtual machine. Those actions affect disks in the linked virtual machine group. Perform the actions when connected to a vCenter Server system for disk consolidation or deletion.

- Snapshot removal – During snapshot removal, the snapshot metadata is also removed, and the virtual machine from which the snapshot was taken is no longer shown as having snapshots. If you remove a snapshot while connected to the 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.

CAUTION Delete all linked virtual machines before deleting the master from which they were created, so that you don't have orphaned or corrupt disk files on your file system.

Relocating a Virtual Machine in a Linked Virtual Machine Group

You can move the virtual machines in a linked virtual machine group between datastores and save storage, as shown in Figure 11-3. 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 11-3. 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 11-4.

Example 11-4. 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.

CAUTION Do not use the `VirtualMachine.ReconfigVM_Task` call to create or add a delta 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 11-5 illustrates how to add the delta disk for the first virtual disk in the virtual machine.

Example 11-5. 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 12

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

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 149.
2. Add child objects. See "Managing VirtualApp Children" on page 149.
3. Export the `VirtualApp` to OVF (`ExportVApp` method) See "Exporting a Virtual Application" on page 150.

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 149 for a list of methods. Direct children share resources with the parent `VirtualApp` object. Both virtual machines and virtual application can be direct children.
- **Linked Children**. A linked child of a virtual application is a virtual machine or virtual application that you add by calling the `UpdateLinkedChildren` method. Linked children increase the flexibility of the `VirtualApp` by allowing child entities to use different resources from the parent `VirtualApp` object. Linked children can be part of a different clusters, but a virtual application and its children must be in the same `Datacenter`. Both virtual machines and virtual applications can be linked children.

Linked children gives better flexibility. In particular, you can create virtual applications that span clusters. The vSphere Client does not support adding or removing links, though it does show links.

When you add a linked child to a virtual application, the following rules apply:

- An `InvalidArgument` fault is thrown if the `UpdateLinkedChildren` method is called on a link target that is a direct child of another virtual application.
- When you add a virtual machine or virtual application that is already a linked child of another virtual application, the existing link is removed and replaced with the new link.
- The life-time of a linked child is determined by the `destroyWithParent` property on the `VAppEntityConfigInfo` data object. If set to `true`, the child is destroyed when the parent `VirtualApp` is destroyed. Otherwise, the link is removed when the `VirtualApp` is destroyed.

If you add a virtual application that consists of multiple entities, for example multiple virtual machines, the entities are moved sequentially and committed one at a time, as specified in the list. If a failure is detected, the method terminates with an exception.

OVF Packages

Open Virtualization Format (OVF) is a distribution format for virtual applications. vSphere uses the OVF package as a unit of distribution and storage for virtual applications. Because these entities are uploaded, downloaded, and stored in OVF package format, vSphere supports access to and deployment of a wide variety of virtual applications.

A virtual application typically consists of one or more virtual disk files and a configuration file.

- The virtual disk files contain the operating systems and applications that run on the virtual machines in the virtual application.
- The configuration file contains metadata that describes how the virtual application is configured and deployed.

An OVF package might also include certificate and manifest files.

The OVF package contains metadata that describes the capabilities and infrastructure requirements of the virtual application, and contains references to the virtual disks and other files that store the virtual machine state. Most of this information is stored in an XML document called the OVF envelope. When an OVF package is instantiated into either a `VirtualApp` or a `VirtualMachine` object (which depends on metadata in the envelope), then the configuration stored in the OVF envelope is applied to the `VirtualVApp` and the `VirtualMachine` objects.

Some of the information in the OVF file is used unaltered, with entire `ovf:Section_Type` elements included in the `VirtualApp` object body. Other sections are transformed or extended by instantiation. You do not need detailed knowledge of all OVF package elements, but a basic understanding of key parts of the package and how they relate to virtual applications is useful.

See the OVF specification at the DMTF Web site for additional information.

Creating a VirtualApp

You always create a `VirtualApp` without children. The `CreateVApp` method includes the following parameters:

- `resSpec` – Properties you would specify for a `ResourcePool`.
- `configSpec` – `VAppConfigSpec` data object for specifying virtual-application specific information.
- `vmFolder` – Depends on the `VirtualApp` structure:
  - When creating top-level virtual applications, that is, virtual applications with no ancestor virtual applications, you must specify a folder.
  - If the `VirtualApp` has another virtual application in the ancestry chain, the `folder` parameter must be `NULL` when you create the `VirtualApp`.

Managing VirtualApp Children

You can add virtual machines and virtual applications to your virtual application as direct or linked children. See "Direct and Linked Children" on page 148. 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
`destroyWithParent`	True if the entity should be removed when the `VirtualApp` is removed.
`key`	Key for the virtual machine or virtual application, a managed object reference to the child.
`startAction`	One of the strings in the `VAppAutoStartAction` enumeration.
`startDelay`	Delay, in seconds, before continuing with the next entity.
`startOrder`	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.
`stopAction`	Defines the stop action for the entity. Can be set to `none`, `powerOff`, `guestShutdown`, or `suspend`. If set to `none`, then the entity does not participate in auto-stop.
`stopDelay`	Delay, in seconds, before continuing with the next entity.
`tag`	Tag for the entity.
`waitingForGuest`	Determines if the virtual machine should start after receiving a heartbeat, from the guest.

Exporting a Virtual Application

To export a virtual application, you must generate an OVF package. The vSphere API supports the generation of OVF packages. It does not support the generation of OVA files. An OVA file is a tar file that contains an OVF package. The OVF package consists of one of more images and an OVF file descriptor. You can create an OVA file by creating a tar file out of the OVF package for your exported virtual application.

The following steps describe how to use the vSphere VirtualApp and OvfManager API to generate an OVF package for a virtual application. The steps assume the simplest scenario: downloading one image from one device URL. You use the same steps to download many images from many device URLs. You can also export a VirtualMachine with the same steps, but use VirtualMachine.ExportVm rather than VirtualApp.ExportVApp.

1 Call the VirtualApp.ExportVApp method, which returns HttpNfcLease. The deviceURL is stored in the info property of HttpNfcLease.

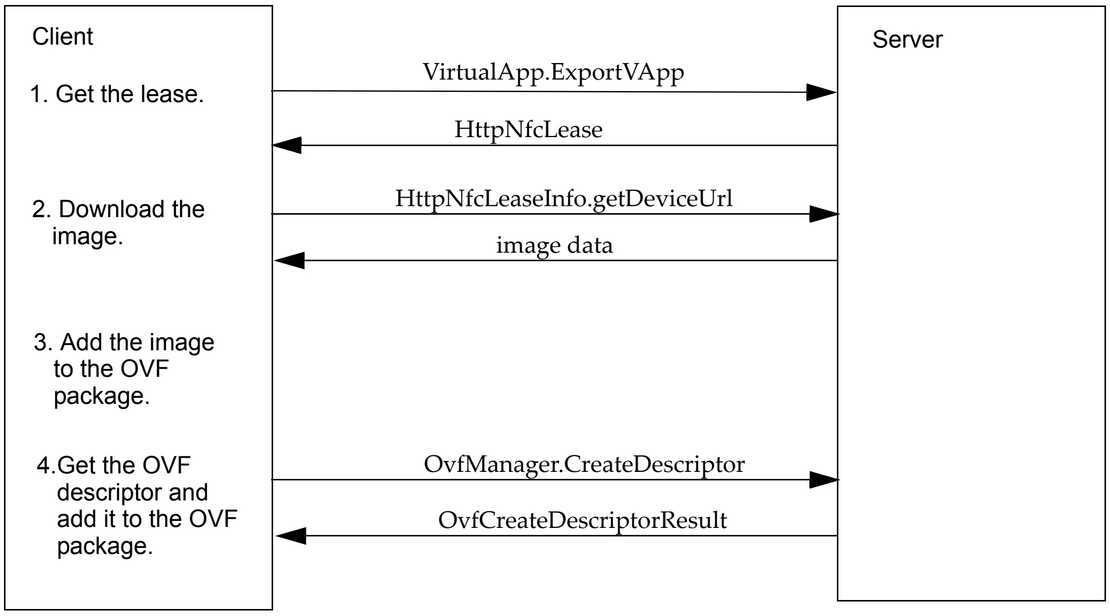
2 Call the HttpNfcLeaseInfo.getDeviceUrl method to access the device URL and download the image data from the device URL.

3 Add the image to the OVF package.

4 Call the OvfManager.CreateDescriptor method, which returns OvfCreateDescriptorResult. Write the file descriptor to a file with the file extension .ovf. Add the .ovf file to the OVF package.

Figure 12-1 shows the major steps.

Figure 12-1. Generating an OVF Package


Client
1. Get the lease.
2. Download the image.
3. Add the image to the OVF package.
4.Get the OVF descriptor and add it to the OVF package.
Server
VirtualApp.ExportVApp
HttpNfcLease
HttpNfcLeaseInfo.getDeviceUrl
image data
OvfManager.CreateDescriptor
OvfCreateDescriptorResult

VirtualApp and OvfManager Methods

Table 12-2 describes the methods used by the VirtualApp and the OvfManager API:

Table 12-2. Methods Used in Exporting a VirtualApp

Method	Description
CreateDescriptor	Creates an OVF descriptor for the specified ManagedEntity, which may be a VirtualMachine or a VirtualApp. CreateDescriptor is a method in the OvfManager managed object.

Table 12-2. Methods Used in Exporting a VirtualApp

Method	Description
ExportVApp	Obtains an export lease on the virtual application. The export lease contains a list of URLs for the disks of the virtual machines in this virtual application. ExportVApp is a method in the VirtualApp managed object.
getDeviceUrl	Retrieves the device IDs and URLs from the server. getDeviceUrl is an accessor method provided in the generated JAX-WS bindings. It does not appear in Figure 12-2..

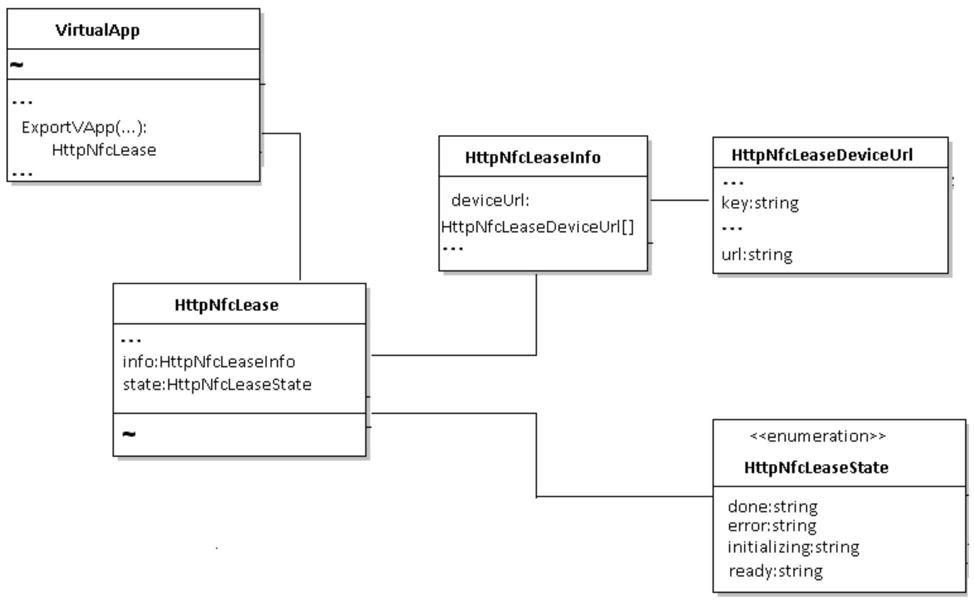
The next two sections deal with the VirtualApp and OvfManager data structures.

VirtualApp Data Structures

The VirtualApp managed object contains the ExportVApp method, which returns an HttpNfcLease. The HttpNfcLease contains the info and state properties, where info is of type HttpNfcLeaseInfo and state is of type HttpNfcLeaseState. The HttpNfcLeaseInfo data object has several properties, one of which is the deviceUrl of type HttpNfcLeaseDeviceUrl[]. The HttpNfcLeaseState has four different states—done, error, initializing and ready.

Figure 12-2 shows the UML representation of the data structures used in the VirtualApp API.

Figure 12-2. VirtualApp Class Diagram



The VirtualApp API data structures are the following:

- VirtualApp—A managed object that is a collection of virtual machines (and potentially other VirtualApp containers) that are operated and monitored as a unit.
- HttpNfcLease—A managed object returned when you call VirtualApp.ExportVApp. It represents a lease on the virtual application. While you hold the lease, you block the operations that alter the state of the virtual application.
- HttpNfcLeaseInfo—A data object that holds information about the lease, such as the virtual application covered by the lease, and the device URLs for up/downloading images.
- HttpNfcLeaseState—An enumeration that is a list of possible states of a lease.

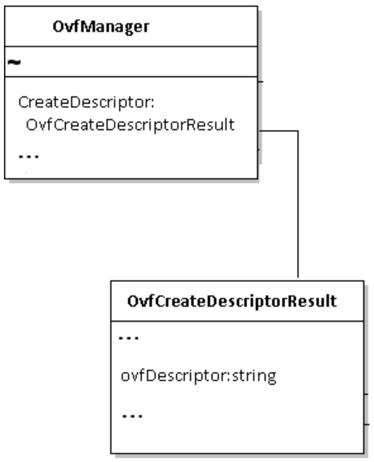
HttpNfcLeaseDeviceUrl—A data object that provides a mapping from logical device IDs to upload/download URLs.

OvfManager Data Structures

The OvfManager managed object has a CreateDescriptor method that returns an OvfCreateDescriptorResult. The OvfCreateDescriptorResult has the ovfDescriptor string.

Figure 12-3 shows the UML representation of the data structures used in the OvfManager API.

Figure 12-3. OvfManager Class Diagram



The OvfManager data structures are the following:

- OvfManager—A managed object that provides a service interface to parse and generate OVF descriptors.
- OvfCreateDescriptorResult—A data object that contains the result of creating the OVF descriptor for the virtual application.

Example of Generating an OVF Package

In summary, the steps in generating an OVF package are the following:

1. Get the managed object reference to the VirtualApp object. Call the ExportVApp method, which returns an HttpNfcLease data object. Wait for the state of the lease to turn to READY. Get the list of device URLs from the lease and store them in an array.
2. For each of the URLs in the list of device URLs, download the images from that URL to the client.
3. Save the image to the OVF package (directory/folder). Create an OvfFile object using the deviceID, absolute path of the downloaded image, and the size of the image on the local disk.
4. Call the OvfManager.CreateDescriptor method by passing the managed object reference to the VirtualApp and the OvfFile object wrapped in an OvfCreateDescriptorParams object. This method returns OvfCreateDescriptorResult, which contains the file descriptor. Write the file descriptor to a file with the file extension .ovf. Add the .ovf file to the OVF package.

The following is an example of how to generate an OVF package. The example assumes a more complex scenario: downloading more than one image from more than one device URL. The example is based on the `OVFManagerExportVAAP.java` sample, which is located in the `SDK/vsphere-ws/java/JAXWS/samples/com/vmware/vapp/` directory.

You can use the ExportVM method instead of the ExportVapp method when exporting a VirtualMachine.

```
package com.vmware.vapp;
```

```
import java.io.*;

import java.net.URL;
import java.util.*;

...
  /** 1. Get the MOR of the VirtualApp.
M       ManagedObjectReference vAppMoRef = getVAPPByName(vApp);
...
  /**    Call the ExportVApp method, which returns an HttpNfcLease data object. */
        ManagedObjectReference httpNfcLease = vimPort.exportVApp(vAppMoRef);

...
  /**    Wait for the state of the lease to turn to READY. */
           Object[] result = waitForValues.wait(httpNfcLease, new String[]{"state"},
                           new String[]{"state"},
                                new Object[][]{new Object[]{
                                        HttpNfcLeaseState.READY,
                                        HttpNfcLeaseState.ERROR}});
                if (result[0].equals(HttpNfcLeaseState.READY)) {

...
  /**    Get the list of device URLs from the lease. */
                    List<HttpNfcLeaseDeviceUrl> deviceUrlArr =
                              httpNfcLeaseInfo.getDeviceUrl();
                    if (deviceUrlArr != null) {
...
  /** 2. For each of the URLs in the list of device URLs, download the images from that URL to
               the client. */
                        for (int i = 0; i < deviceUrlArr.size(); i++) {

                            String deviceId = deviceUrlArr.get(i).getKey();
                            String deviceUrlStr = deviceUrlArr.get(i).getUrl();
                            String absoluteFile =
                                    deviceUrlStr.substring(deviceUrlStr
                                            .lastIndexOf("/") + 1);
  /** 3. Save the image to the OVF package (directory/folder). Create an OvfFile object using
   * the deviceID, absolute path of the downloaded image, and the size of the image on the
   * local disk.
   */
                            long writtenSize =
                                    writeVMDKFile(absoluteFile,
                                            deviceUrlStr.replace("*", host));
                            OvfFile ovfFile = new OvfFile();
                            ovfFile.setPath(absoluteFile);
                            ovfFile.setDeviceId(deviceId);
                            ovfFile.setSize(writtenSize);

                            ovfFiles.add(ovfFile);
}
  /** 4. Call the OvfManager.CreateDescriptor method by passing the managed object reference
   * to the VirtualApp and the OvfFile object wrapped in an OvfCreateDescriptorParams object.
   * This method returns OvfCreateDescriptorResult, which contains the file descriptor.
   * Write the file descriptor to a file with the file extension .ovf. Add the .ovf file to
   * the OVF package.
   */
                        ovfCreateDescriptorParams.getOvfFiles().addAll(ovfFiles);
                        OvfCreateDescriptorResult ovfCreateDescriptorResult =
                                vimPort.createDescriptor(
                                        serviceContent.getOvfManager(), vAppMoRef,
                                        ovfCreateDescriptorParams);

                        String outOVF = localpath + "/" + vApp + ".ovf";
                        File outFile = new File(outOVF);
                        FileWriter out = new FileWriter(outFile);

                        out.write(ovfCreateDescriptorResult.getOvfDescriptor());

                        out.close();
```

Importing an OVF Package

To import the virtual application OVF template, you follow a few basic steps. The steps are the same for an OVF package that contains a single virtual machines or an OVF package that contains a more complex virtual application.

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.

13 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

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 132.
- 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 158.
- 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 163.

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:

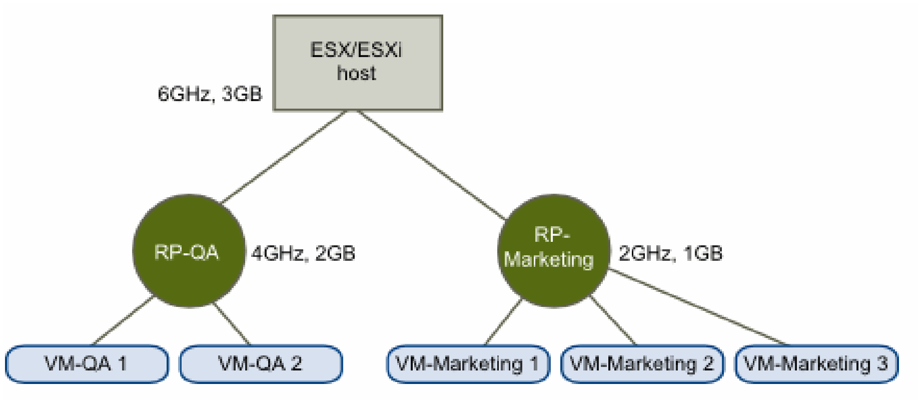
- **Standalone host** – When you call `Folder.AddStandaloneHost_Task`, the call returns a `Task` object that contains the `ComputeResource`. The `ComputeResource.resourcePool` property is the root resource pool associated with the compute resource (and with the host).
- **Cluster** – When you call `Folder.CreateClusterEx`, 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 163 and "Managing DRS Clusters" on page 164.

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 160. 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.

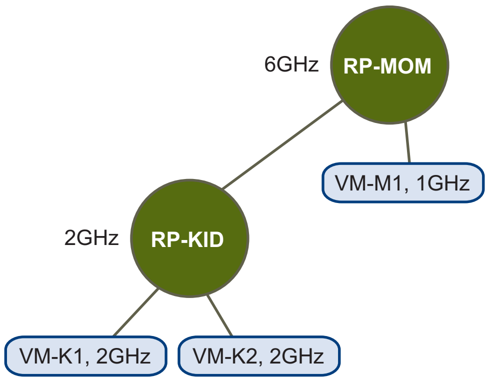
Expandable Reservation Example 2

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.

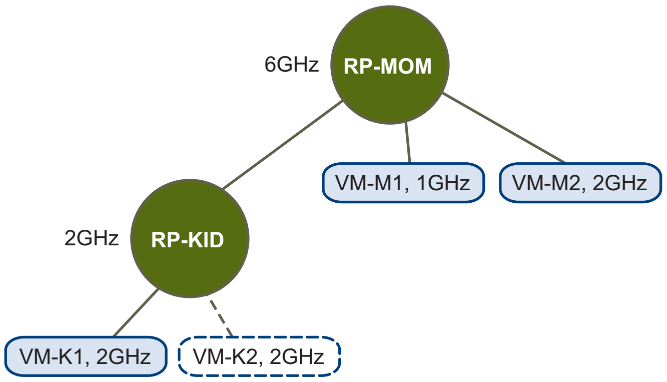
Figure 13-2. Admission Control with Expandable Resource Pools, Scenario 1



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 164.

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 165.

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.ReconfigurComputeResource_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.moveInto_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 163.

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 164.
- `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 recommentations 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 164. 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 14

VMware vSphere uses an asynchronous client-server communication model. Methods that end with _Task are non-blocking, returning a reference to a `Task` managed object. You can use `Task` and `ViewManager` managed objects to monitor tasks, cancel certain tasks, and create custom tasks.

If you are using a vCenter Server system, the `ScheduledTaskManager` allows you to schedule your own tasks for a one-time run or for repeated runs.

The chapter includes the following topics:

Creating Tasks

Each time a vSphere server runs a method, it creates a `Task` and a corresponding `TaskInfo` data object. Some methods run synchronously and return data as the `Task` completes. But methods that end with _Task run asynchronously, and return a reference to a Task that will be created and completed as a processor becomes available. They are created to perform the functions in a non-blocking manner. Therefore, you must use the reference to the `Task` to monitor the status and results of the Task. vSphere operations that include the suffix `_Task` in their names are asynchronous and return `Task` references.

The `Task` object provides information about the status of the invoked operation through its `TaskInfo` data object. An instance of `TaskInfo` populates the `info` property of the `Task` managed object at runtime. By monitoring properties of the `TaskInfo` object, a client application can take appropriate action when the `Task` completes, or can handle errors if the `Task` does not complete successfully.

When a vSphere server creates a `Task`, it also creates a `TaskEvent` object. The `TaskEvent` object contains a copy of the `TaskInfo` object (`TaskEvent.info`). The `TaskEvent` copy of the `TaskInfo` object is a snapshot of the Task state at the time of its creation. It does not change after it is created. To find the current status of the task, use the `Task.info.eventChainId` property.

Session Persistence

A `Task` and its associated objects are session specific, so they will not persist after the session is closed. When your client opens a session, you can only obtain information about the `Task` objects that your client is authorized to view.

Cancelling a Task

To cancel a `Task` that is still running, call the `Task.CancelTask` method, passing in the managed object reference to the `Task` you want to cancel, as shown in this example:

```
my_conn.cancelTask(taskMoRef);
```

You can only cancel a `Task` that has its `cancelable` property set to `true` and its `state` property set to `running`. The operation that initiates the `Task` sets the value of `cancelable` when it creates the `Task`. For example, a `CreateVM_Task` cannot be cancelled. Before attempting to cancel a running `Task`, you can check the values of the `cancelable` property and the `state` property of the `TaskInfo` data object associated with the `Task`.

Using TaskInfo to Determine Task Status

A `Task` object provides information about the status of the invoked operation through its `TaskInfo` data object. An instance of `TaskInfo` populates the `info` property of the `Task` managed object at runtime. By monitoring properties of the `TaskInfo` object, a client application can take appropriate action when the `Task` completes, or can handle errors if the `Task` does not complete successfully.

The `Task.info` property contains a `TaskInfo` data object that contains information about the `Task` the server returns to your client application.

When a `Task` is instantiated by the server, the `TaskInfo.result` property is initialized to `Unset`. Upon successful completion of an operation, the `result` property is populated with the return type specific to the operation. The `result` might be a data object, a reference to a managed object, or any other data structure as defined by the operation.

For example:

1. The `ClusterComputeResource.AddHost_Task` method returns a `Task` object whose `info` property contains a `TaskInfo` data object.
2. At the start of the operation, the `result` property is `Unset`.
3. Upon successful completion of the operation, the `result` property of `TaskInfo` contains the managed object reference of the newly added `HostSystem`.

   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 77. 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 14-1 shows a code fragment that obtains values for the `info` property from each `Task` object in the array.

Example 14-1. 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 14-2 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 14-2. Sample Run of the TaskList Java Application

```
java com.vmware.samples.general.TaskList --url https://srv/sdk --username root --password *******

Started
Task
Operation AcquireCimServicesTicket
Name srv
Type HostSystem
State -Success
Error
======================
Ended TaskList
```

Accessing and Manipulating Multiple Tasks

Use the ViewManager's ListView method to identify the set of Tasks you want to monitor.

You can specify a smaller and more efficient data set using one of the `ViewManager` views with the Property Collector. Each view represents objects you have selected on the server. Views are more efficient because you only need a single instance of a `PropertyCollector` object, instead of multiple instances with multiple filter specifications.

Gathering Data with a ViewManager Object

Use one of the `ViewManager` methods to obtain information about `Task` objects and references while the session is running. The ViewManager's `ListView method` allows you to customize your view with an input object list, the `ContainerView method` lets you view all objects in a folder, datacenter, resource pool, or other data container, and the `InventoryView method` lets you monitor the entire inventory. The smallest view you can create will be the most efficient way to retrieve task data.

The `ViewManager` has the following property:

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

See "PropertyCollector Example (RetrievePropertiesEx)" on page 59 for an example that uses the `ContainerView method` to access Inventory data.

Task Monitoring Example Using the ListView Object

Use the ViewManager's `ListView` method to specify the set of tasks that you want to monitor.

The following example uses the `ViewManager` service interface with a `ListView` method to access and manipulate `Task` objects. This example uses the property collector to monitor tasks that are created in the process of virtual machine cloning. This program creates two clones of the specified virtual machine and then monitors the tasks and prints out status and reference values when the tasks have completed.

The following steps describe the procedure for creating a program that uses the ListView managed object.

You can run the example below as a stand-alone program against your own server by copying the code sections into .java file, compiling it, and then using the following command line syntax:

```
>cloneVMTask server-name username password vm-name
```

To create a program that uses the ListView managed object

1 Import the vSphere Web Services API libraries:

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

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

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

3 Create the cloneVMTask class to create cloned virtual machine Tasks on a host, so we can demonstrate how to monitor these Tasks.

```
public class cloneVMTask {
```

4 Declare variables for the service instance objects and methods:

```
// Services and methods
static ManagedObjectReference pCollector;
static ManagedObjectReference viewMgr;
static ServiceContent serviceContent;
static VimPortType methods;

/**
* getVmRef() retrieves a reference to the specified virtual machine.
*
* vmName - name of the virtual machine specified on the command line
*
* This function retrieves references to all of the virtual machines
* in the datacenter and looks for a match to the specified name.
*/
```

5 Create a function that retrieves references to all of the virtual machines in the datacenter and looks for a match to the specified name. The function in this example uses `getVMRef(String, vmName)`, which retrieves a reference to the virtual machine that you specify on the command line (`vmName`) when you run this sample. The function also initializes the vmRef variable to null.

```
private static ManagedObjectReference getVmRef( String vmName )
throws Exception
{
ManagedObjectReference vmRef = null;
```

6 Use a container view to collect references to all virtual machines in the datacenter.

```
List<String> vmList = new ArrayList<String>();
vmList.add("VirtualMachine");
    ManagedObjectReference cViewRef = methods.createContainerView(viewMgr,
        serviceContent.getRootFolder(),
        vmList,
        true);
```

7 Create an `ObjectSpec` to define the beginning of the traversal. Use the `setObj` method to specify that the container view is the root object for this traversal. Set the `setSkip` method to true to indicate that you don't want to include the container in the results.

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

8 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");
```

9 Add the traversal spec to the object spec.

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

10 Specify the property for retrieval (virtual machine name).

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

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

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

12 Create a list for the filters and add the spec to it.

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

13 Get the data from the server.

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

14 Go through the returned list and look for a match to the specified `vmName`.

```
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
    String vmname = null;
    List<DynamicProperty> dps = oc.getPropSet();
    if (dps != null) {
        for (DynamicProperty dp : dps) {
        vmname = (String) dp.getVal();
        // If the name of this virtual machine matches
        // the specified name, save the managed object reference.
        if (vmname.equals(vmName)) {
            vmRef = oc.getObj();
            break;
        }
        }
        if (vmRef != null) { break; }
    }
    }
}
if (vmRef == null) {
    System.out.println("Specified Virtual Machine not found.");
    throw new Exception();
}
return vmRef;

}
```

15 Get the folder that contains the specified virtual machine (VirtualMachine.parent)

```
private static ManagedObjectReference getVMParent(ManagedObjectReference vmRef)
throws Exception {
```

16 Create an `Object Spec` to define the property collection. Use the `setObj` method to specify that the vmRef is the root object for this traversal. Set the `setSkip` method to true to indicate that you don't want to include the virtual machine in the results.

```
// don't include the virtual machine in the results
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(vmRef);
oSpec.setSkip(false);
```

17 Specify the property for retrieval (virtual machine parent).

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

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

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

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

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

20 Get the data from the server.

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

21 Get the parent folder reference.

```
ManagedObjectReference folderRef = null;
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
    List<DynamicProperty> dps = oc.getPropSet();
    if (dps != null) {
        for (DynamicProperty dp : dps) {
        folderRef = (ManagedObjectReference) dp.getVal();
        }
    }
    }
}

if (folderRef == null) {
    System.out.println("Folder not found.");
    throw new Exception();
}
return folderRef;
}
```

Now that we have the reference information for the virtual machine that you specified on the command line (`vmRef`) and a reference for the parent directory (`folderRef`), we are ready to create the clone virtual machines.

22 To create clones, use the `cloneVM` method and pass in the `vmRef` that we retrieved previously.

```
private static void cloneVM(ManagedObjectReference vmRef) throws Exception {
```

23 After you have created the clone managed object, create a clone specification. Use default values whenever possible.

```
VirtualMachineCloneSpec cloneSpec = new VirtualMachineCloneSpec();
VirtualMachineRelocateSpec vmrs = new VirtualMachineRelocateSpec();
cloneSpec.setLocation(vmrs);
cloneSpec.setPowerOn(true);
cloneSpec.setTemplate(false);
```

24 Get the destination folder for the clone virtual machines (VirtualMachine.parent). The clones will be created in the same folder that contains the specified virtual machine (vmName).

```
ManagedObjectReference folder = getVMParent( vmRef );
```

25 Create two clone virtual machines.

```
ManagedObjectReference cloneTask = methods.cloneVMTask( vmRef, folder, "clone__1",
               cloneSpec);
ManagedObjectReference cloneTask2 = methods.cloneVMTask( vmRef, folder, "clone__2",
               cloneSpec);
```

26 Create a list view for the clone tasks.

```
List<ManagedObjectReference> taskList = new ArrayList<ManagedObjectReference>();
taskList.add(cloneTask);
taskList.add(cloneTask2);
ManagedObjectReference cloneTaskList = methods.createListView(viewMgr, taskList);
```

Next we will set up a property filter for WaitForUpdatesEx. This includes creating an object spec, a traversal spec, a property spec, a filter spec, and finally a property filter. The next six steps will describe these procedures.

27 Create an object spec to start the traversal.

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

28 Create a traversal spec to select the list of tasks in the view.

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

29 Add the traversal spec to the object spec.

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

30 Create property spec for Task.info.state and Task.info.result.

```
PropertySpec pSpec = new PropertySpec();
pSpec.setType("Task");
pSpec.setAll(false);
pSpec.getPathSet().add("info.state");
pSpec.getPathSet().add("info.result");
```

31 Create a filter spec.

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

32 Create the filter.

```
ManagedObjectReference pFilter = methods.createFilter(pCollector, fSpec, true);
```

In the next section, we use the `waitForUpdatesEx` method to look for a change in `cloneTask.info.state` and `cloneTask.info.result`. When the state is "success", `cloneTask.info.result` is the managed object reference of the clone. Note that the order of property retrieval is not guaranteed, and it may take more than one call to `waitForUpdatesEx` to retrieve both properties for a task.

This code does not set a time-out (`WaitOptions.maxWaitSeconds` is unset), so after it has retrieved all of the property values, `waitForUpdatesEx` will block the thread, waiting for the TCP connection with the vSphere Server to time-out.

How a client application handles the session depends on the particular context. (The client can call `WaitForUpdatesEx` from its own thread, look for specific updates and then stop calling the method.)

For more information about `WaitOptions` and the `waitForUpdatesEx` method, see "Client Data Synchronization (WaitForUpdatesEx)" on page 77.

33 Initialize wait loop (?)

```
String version = "";
Boolean wait = true;
WaitOptions waitOptions = new WaitOptions();

while ( wait ) {
```

34 Call WaitForUpdatesEx.

```
UpdateSet uSet = methods.waitForUpdatesEx(pCollector, version, waitOptions);

if (uSet == null) {
wait = false;
}
else {
```

35 Get the version for subsequent calls to WaitForUpdatesEx.

```
version = uSet.getVersion();
```

36 Get the list of property updates.

```
List<PropertyFilterUpdate> pfUpdates = uSet.getFilterSet();
for (PropertyFilterUpdate pfu : pfUpdates) {
```

37 Get the list of object updates produced by the filter.

```
List<ObjectUpdate> oUpdates = pfu.getObjectSet();
for (ObjectUpdate ou : oUpdates) {
```

38 Look for ObjectUpdate.kind=MODIFY (property modified).

```
if (ou.getKind() == ObjectUpdateKind.MODIFY) {

    String name = "";
    TaskInfoState state;
    ManagedObjectReference cloneRef = new ManagedObjectReference();
```

39 Get the changed data.

```
List<PropertyChange> pChanges = ou.getChangeSet();
```

40 Retrieve the name of the property

```
for (PropertyChange pc : pChanges) {
name = pc.getName();

//The task property names are info.state or info.result;
//pc.val is an xsd:anyType:
//-- for info.state, it is the state value
//-- for info.result, it is the clone reference
if (name.equals("info.state")) {
    state = (TaskInfoState)pc.getVal();
    System.out.println("State is "+state.value());
}
else if (name.equals("info.result")) {
    cloneRef = (ManagedObjectReference)pc.getVal();
    System.out.println("Clone reference is "+cloneRef.getValue());
}
}
```

Authentication is handled using a TrustManager and supplying a host name verifier method. (The host name verifier is declared in the main function.)

For the purposes of this example, this TrustManager implementation will accept all certificates. This is only appropriate for a development environment. Production code should implement certificate support.

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

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

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

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

Now we are set to retrieve the task information, so we implement the main method.

```
// cloneVMTask( server, user, password, virtual-machine )
public static void main(String [] args) throws Exception {
```

41 We create variables to hold the values passed in from the command line.

```
String serverName = args[0];
String userName = args[1];
String password = args[2];
String vmName = args[3];
String url = "https://"+serverName+"/sdk/vimService";
```

42 Add variables for access to the API methods and services.

```
// -- ManagedObjectReference for the ServiceInstance on the Server
// -- VimService for access to the vSphere Web service
// -- VimPortType for access to methods
// -- ServiceContent for access to managed object services
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
VimService vimService;
```

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

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

44 Create the trust manager.

```
javax.net.ssl.TrustManager[] trustAllCerts = new javax.net.ssl.TrustManager[1];
javax.net.ssl.TrustManager tm = new TrustAllTrustManager();
trustAllCerts[0] = tm;
// Create the SSL context
javax.net.ssl.SSLContext sc = javax.net.ssl.SSLContext.getInstance("SSL");
// Create the session context
javax.net.ssl.SSLSessionContext sslsc = sc.getServerSessionContext();
// Initialize the contexts; the session context takes the trust manager.
sslsc.setSessionTimeout(0);
sc.init(null, trustAllCerts, null);
// Use the default socket factory to create the socket for the secure connection
javax.net.ssl.HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
```

45 Set the default host name verifier to enable the connection.

```
HttpsURLConnection.setDefaultHostnameVerifier(hv);
```

46 Set up the manufactured managed object reference for the ServiceInstance

```
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");
```

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

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

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

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

49 Retrieve the ServiceContent object and login.

```
serviceContent = methods.retrieveServiceContent(SVC_INST_REF);
    methods.login(serviceContent.getSessionManager(),
    userName,
    password,
    null);
```

50 Get references to the property collector and the view manager.

```
pCollector = serviceContent.getPropertyCollector();
viewMgr = serviceContent.getViewManager();
```

51 Get a reference to the specified virtual machine.

```
ManagedObjectReference vmRef = getVmRef( vmName );
```

52 Clone the virtual machine and wait for the result.

```
cloneVM( vmRef );
```

53 Close the connection.

```
    methods.logout(serviceContent.getSessionManager());
    }
}
```

NOTE For general task monitoring, it is a best practice to use a `ViewManager` to monitor specific tasks. See the *API Reference* for more information about using views.

Gathering Data with a TaskManager Interface

`TaskManager` is a service interface that you can also use for accessing and manipulating `Task` objects. This approach uses a `PropertyCollector` that includes the `recentTask` property of the `TaskManager` managed object that corresponds to the Recent Tasks pane at the bottom of the vSphere client User Interface.

You can use the following `TaskManager` properties in your client application.

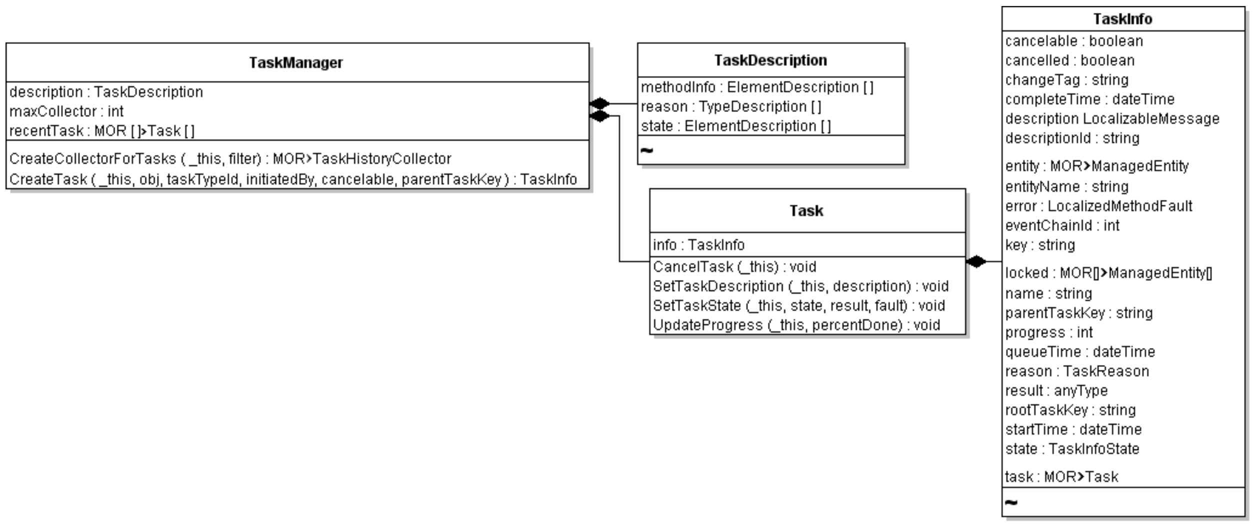
- `description` – `TaskDescription` object that includes a `methodInfo` property. `methodInfo` contains a key-based array that `TaskManager` uses to populate the value of a `TaskInfo` data object's `descriptionId` property with the name of the operation. Examples of two elements from this key-based array are `methodInfo["Folder.createVm"]` and `methodInfo["Folder.createClusterEx"]`.
- `recentTask` – Array of `Task` managed object references that are queued to run, running, or completed within the past 10 minutes. On ESX/ESXi hosts that are managed by a vCenter Server, a completed task must also be one of the 200 most recent tasks to be included in the array. A vSphere Client connected to a vSphere Server displays queued, running, and completed tasks in the Recent Tasks pane.

In addition to these properties, `TaskManager` has the following methods:

- `CreateTask` – Used by other methods to create a custom `Task` object. Developers creating extensions can use this method to create custom `Task` objects.
- `CreateCollectorForTasks` – Creates an object that contains all tasks from the vCenter Server database that meet specific criteria. You cannot run this method against an ESX/ESXi system. See "Using a TaskHistoryCollector" on page 183.

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

Figure 14-1. TaskManager and Task Managed Objects


TaskManager
description : TaskDescription
maxCollector : int
recentTask : MOR []>Task []
CreateCollectorForTasks (_this, filter) : MOR>TaskHistoryCollector
CreateTask (_this, obj, taskTypeId, initiatedBy, cancelable, parentTaskKey) : TaskInfo
TaskDescription
methodInfo : ElementDescription []
reason : TypeDescription []
state : ElementDescription []
~
Task
info : TaskInfo
CancelTask (_this) : void
SetTaskDescription (_this, description) : void
SetTaskState (_this, state, result, fault) : void
UpdateProgress (_this, percentDone) : void
TaskInfo
cancelable : boolean
cancelled : boolean
changeTag : string
completeTime : dateTime
description LocalizableMessage
descriptionId : string
entity : MOR>ManagedEntity
entityName : string
error : LocalizedMethodFault
eventChainId : int
key : string
locked : MOR[]>ManagedEntity[]
name : string
parentTaskKey : string
progress : int
queueTime : dateTime
reason : TaskReason
result : anyType
rootTaskKey : string
startTime : dateTime
state : TaskInfoState
task : MOR>Task
~

Examining Recent Tasks with TaskManager

To obtain the list of recent tasks, use a PropertyCollector to obtain references to the TaskManager and to all Task objects from the recentTask property of the TaskManager. Example 14-3 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 57.

Example 14-3. 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 183.

Understanding the ScheduledTaskManager Interface

You can use the `ScheduledTaskManager` to schedule tasks. In the vSphere Client, scheduled tasks display in the Task & Events tab.

You can define actions to occur on vCenter Server at different times:

- When a vCenter Server system starts up operations, such as after a reboot
- At a specific time and day
- At hourly, daily, weekly, or monthly intervals

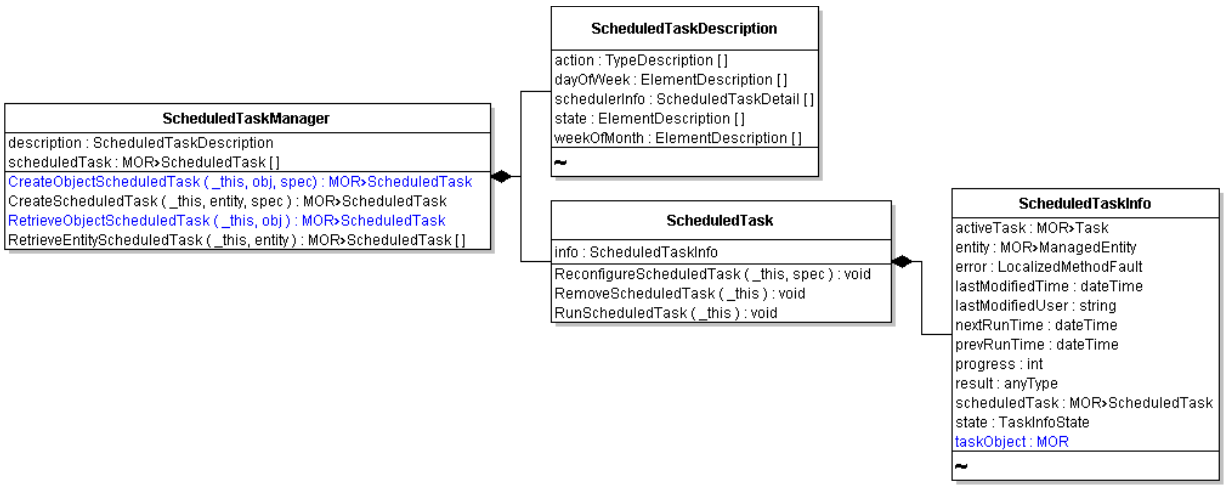
You can schedule scripts to be run or methods to be invoked on the server. You apply the action to an entity in the inventory, such as a virtual machine or a host.

You can perform the following actions with `ScheduledTaskManager`.

- Retrieve scheduled tasks for a specific managed entity by calling the `ScheduledTaskManager.RetrieveEntityScheduledTask` method.
- Create a scheduled task by calling the `ScheduledTaskManager.CreateScheduledTask` method. See "Scheduling Tasks" on page 180.

Figure 14-2 shows the `ScheduledTaskManager` service interface and associated data objects.

Figure 14-2. ScheduledTaskManager and ScheduledTask Managed Objects


ScheduledTaskDescription
action : TypeDescription [ ]
dayOfWeek : ElementDescription [ ]
schedulerInfo : ScheduledTaskDetail [ ]
state : ElementDescription [ ]
weekOfMonth : ElementDescription [ ]
~
ScheduledTaskManager
description : ScheduledTaskDescription
scheduledTask : MOR>ScheduledTask [ ]
CreateObjectScheduledTask (_this, obj, spec) : MOR>ScheduledTask
CreateScheduledTask (_this, entity, spec) : MOR>ScheduledTask
RetrieveObjectScheduledTask (_this, obj) : MOR>ScheduledTask
RetrieveEntityScheduledTask (_this, entity) : MOR>ScheduledTask [ ]
ScheduledTask
info : ScheduledTaskInfo
ReconfigureScheduledTask (_this, spec) : void
RemoveScheduledTask (_this) : void
RunScheduledTask (_this) : void
ScheduledTaskInfo
activeTask : MOR>Task
entity : MOR>ManagedEntity
error : LocalizedMethodFault
lastModifiedTime : dateTime
lastModifiedUser : string
nextRunTime : dateTime
prevRunTime : dateTime
progress : int
result : anyType
scheduledTask : MOR>ScheduledTask
state : TaskInfoState
taskObject : MOR
~

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.

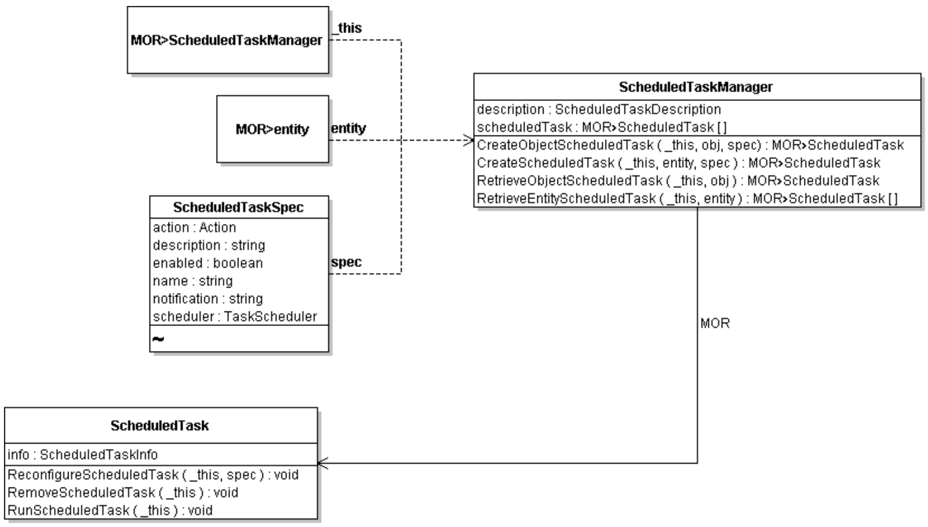
If the action specified for a `ScheduledTask` creates its own `Task` (such as with any of the asynchronous operations), the managed object reference to the `Task` populates the `activeTask` property of `ScheduledTaskInfo`.

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


MOR>ScheduledTaskManager
_this
MOR>entity
entity
ScheduledTaskSpec
action : Action
description : string
enabled : boolean
name : string
notification : string
scheduler : TaskScheduler
spec
ScheduledTaskManager
description : ScheduledTaskDescription
scheduledTask : MOR>ScheduledTask [ ]
CreateObjectScheduledTask (_this, obj, spec) : MOR>ScheduledTask
CreateScheduledTask (_this, entity, spec) : MOR>ScheduledTask
RetrieveObjectScheduledTask (_this, obj) : MOR>ScheduledTask
RetrieveEntityScheduledTask (_this, entity) : MOR>ScheduledTask [ ]
MOR
ScheduledTask
info : ScheduledTaskInfo
ReconfigureScheduledTask (_this, spec) : void
RemoveScheduledTask (_this) : void
RunScheduledTask (_this) : void

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 192.
- `notification` – Specifies the email address for sending notification messages about the `ScheduledTask`. To use notifications, the vCenter Server system must have an SMTP email gateway configured. By default, `notification` is set to an empty string.
- `scheduler` – Specifies the time, frequency, and other details of the schedule. The `TaskScheduler` data object is the base type for several specific schedule objects. See "Scheduling Recurring Operations" on page 180.

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

`AfterStartupTaskScheduler`	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.
	`AfterStartupTaskScheduler asts = new AfterStartupTaskScheduler();`
	`asts.setMinute(10);`

`OnceTaskScheduler`
.
	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.
	`Calendar runTime = Calendar.getInstance();`
	`runtime.add(Calendar.MINUTE, 30);`
	`OnceTaskScheduler ots = new OnceTaskScheduler ();`
	`ots.setRunAt(runTime);`

`RecurrentTaskScheduler`	Base type for `HourlyTaskScheduler`, `DailyTaskScheduler`, `WeeklyTaskScheduler`, and `MonthlyTaskScheduler` objects. Set the interval property to define how frequently a task should run. For example, setting the interval property of an hourly task to 4 causes the task to run every 4 hours.

`HourlyTaskScheduler`	Schedule a task to run once every hour (or every specified number of hours) at a specified time. Set the interval 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.
	`HourlyTaskScheduler hts = new HourlyTaskScheduler();`
	`hts.setMinute(30);`
	`hts.setInterval(4);`

`DailyTaskScheduler`	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.
	Example: Schedule a task to run daily at 9:30 am (EST).
	`DailyTaskScheduler dts = new DailyTaskScheduler();`
	`dts.setMinute(30);`
	`dts.setHour(14);`

`WeeklyTaskScheduler`	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.
	Example: Schedule a task to run every Tuesday and Sunday at 30 minutes past midnight.
	`WeeklyTaskScheduler wts = new WeeklyTaskScheduler();`
	`wts.setMonday(true);`
	`wts.setTuesday(true);`
	`...`
	`wts.setSaturday(false);`
	`wts.setSunday(true);`
	`dts.setMinute(30);`
	`dts.setHour(4);`

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

TaskScheduler Subtype	Usage
`MonthlyByDayTaskScheduler`	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. **Example**: Schedule a task to run every 3 months (on the last day of the month) at 12:30 p.m.
	`MonthlyByDayTaskScheduler mbdts = new MonthlyByDayTaskScheduler();`
	`mbdts.setDay(31);`
	`mbdts.setInterval(3);`
	`mbdts.setMinute(30);`
	`mbdts.setHour(14);`
`MonthlyByWeekdayTaskScheduler`	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. **Example**: Schedule a task to run on the last Wednesday of each month at 12:30 a.m.
	`MonthlyByWeekdayTaskScheduler mbwts = new MonthlyByWeekdayTaskScheduler();`
	`mbwts.setOffset(WeekOfMonth.last);`
	`mbwts.setWeekday(DayOfWeek.wednesday);`
	`mbwts.setHour(4);`
	`mbwts.setMinute(30);`

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);
...
```

Cancelling a Scheduled Task

You can cancel a scheduled task in several ways.

- To cancel the current run of a scheduled task, call `ScheduledTask.RemoveScheduledTask`. This method does not cancel subsequent runs of the `ScheduledTask`.
- To cancel an upcoming run of a `ScheduledTask`, call `ScheduledTask.ReconfigureScheduledTask` with a new `ScheduledTaskSpec` data object containing the new specifications for the schedule.
- To cancel a `ScheduledTask` that spawns a second task, create a `PropertyCollector` to obtain the reference to the `Tasks` and call its `CancelTask` method. The task must be cancellable.

Using a TaskHistoryCollector

A `TaskHistoryCollector` lets you gather information about tasks. You create a `TaskHistoryCollector` using the `TaskManager.CreateCollectorForTasks` method.

To create a TaskHistoryCollector

1 Identify the type of `Task` objects that you want to collect, and create an instance of a `TaskFilterSpec` data object that specifies your filter criteria.

  The `TaskFilterSpec` includes an `taskTypeId` 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 views 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 `Task` monitoring. In addition to the samples listed in the table, the helper classes associated with the C# sample applications use `Task` objects extensively.

Table 14-3. Task and ScheduledTask Sample Applications

Java	C#
`vsphere-ws\java\JAXWS\samples\com\vmware\general\TaskList.java`	`vsphere-ws\dotnet\cs\samples\TaskList\TaskList.cs`
	`vsphere-ws\dotnet\cs\samples\TaskList\TaskList.csproj`
	`vsphere-ws\dotnet\cs\samples\TaskList\TaskList2008.csproj`
	`vsphere-ws\dotnet\cs\samples\TaskList\TaskList2010.csproj`
`vsphere-ws\java\JAXWS\samples\com\vmware\scheduling\DeleteOneTimeScheduledTask.java`	`vsphere-ws\dotnet\cs\samples\DeleteOneTimeScheduledTask\DeleteOneTimeScheduledTask.cs`
	`vsphere-ws\dotnet\cs\samples\DeleteOneTimeScheduledTask\DeleteOneTimeScheduledTask.csproj`
	`vsphere-ws\dotnet\cs\samples\DeleteOneTimeScheduledTask\DeleteOneTimeScheduledTask2008.csproj`
	`vsphere-ws\dotnet\cs\samples\DeleteOneTimeScheduledTask\DeleteOneTimeScheduledTask2010.csproj`
`vsphere-ws\java\JAXWS\samples\com\vmware\scheduling\OneTimeScheduledTask.java`	`vsphere-ws\dotnet\cs\samples\OneTimeScheduledTask\OneTimeScheduledTask.cs`
	`vsphere-ws\dotnet\cs\samples\OneTimeScheduledTask\OneTimeScheduledTask.csproj`
	`vsphere-ws\dotnet\cs\samples\OneTimeScheduledTask\OneTimeScheduledTask2008.csproj`
	`vsphere-ws\dotnet\cs\samples\OneTimeScheduledTask\OneTimeScheduledTask2010.csproj`
`vsphere-ws\java\JAXWS\samples\com\vmware\scheduling\WeeklyRecurrenceScheduledTask.java`	`vsphere-ws\dotnet\cs\samples\WeeklyRecurrenceScheduledTask\WeeklyRecurrenceScheduledTask.cs`
	`vsphere-ws\dotnet\cs\samples\WeeklyRecurrenceScheduledTask\WeeklyRecurrenceScheduledTask.csproj`
	`vsphere-ws\dotnet\cs\samples\WeeklyRecurrenceScheduledTask\WeeklyRecurrenceScheduledTask2008.csproj`
	`vsphere-ws\dotnet\cs\samples\WeeklyRecurrenceScheduledTask\WeeklyRecurrenceScheduledTask2010.csproj`

Events and Alarms 15

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

`EventManager` is the service interface for working with the event infrastructure. See "Managing Events with EventManager" on page 186.

`Event` subtypes define the events that the system generates. See "Event Data Objects" on page 187 and "Creating Custom Events" on page 188.

`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 188.

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 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.

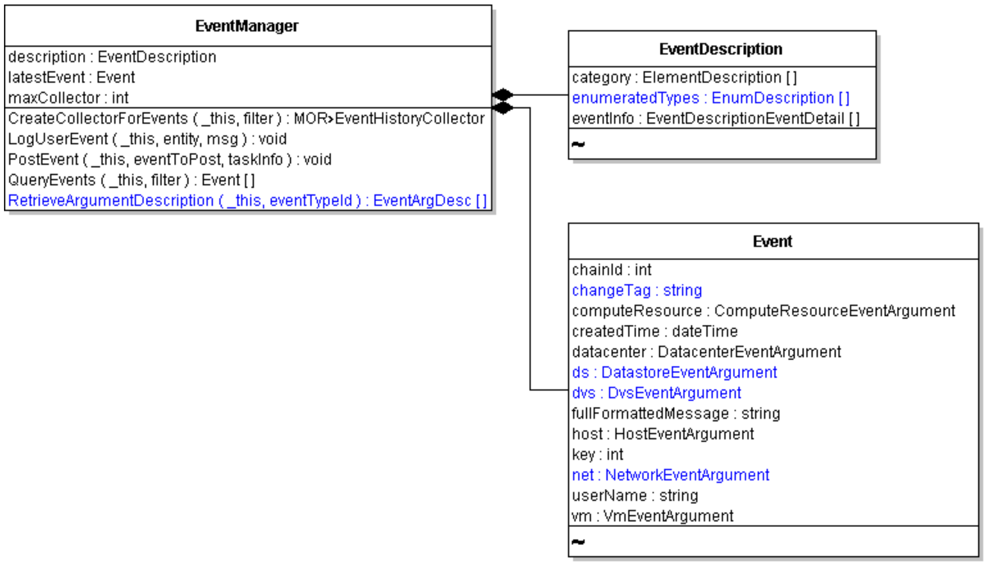
Using an `EventHistoryCollector`, you can obtain information about these objects as they are being collected on a specific ESX/ESXi system, or from a specific historical period from the database. See "Using an EventHistoryCollector" on page 188.

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


EventManager
description : EventDescription
latestEvent : Event
maxCollector : int
CreateCollectorForEvents (_this, filter) : MOR>EventHistoryCollector
LogUserEvent (_this, entity, msg) : void
PostEvent (_this, eventToPost, taskInfo) : void
QueryEvents (_this, filter) : Event [ ]
RetrieveArgumentDescription (_this, eventTypeId) : EventArgDesc [ ]
EventDescription
category : ElementDescription [ ]
enumeratedTypes : EnumDescription [ ]
eventInfo : EventDescriptionEventDetail [ ]
~
Event
chainId : int
changeTag : string
computeResource : ComputeResourceEventArgument
createdTime : dateTime
datacenter : DatacenterEventArgument
ds : DatastoreEventArgument
dvs : DvsEventArgument
fullFormattedMessage : string
host : HostEventArgument
key : int
net : NetworkEventArgument
userName : string
vm : VmEventArgument
~

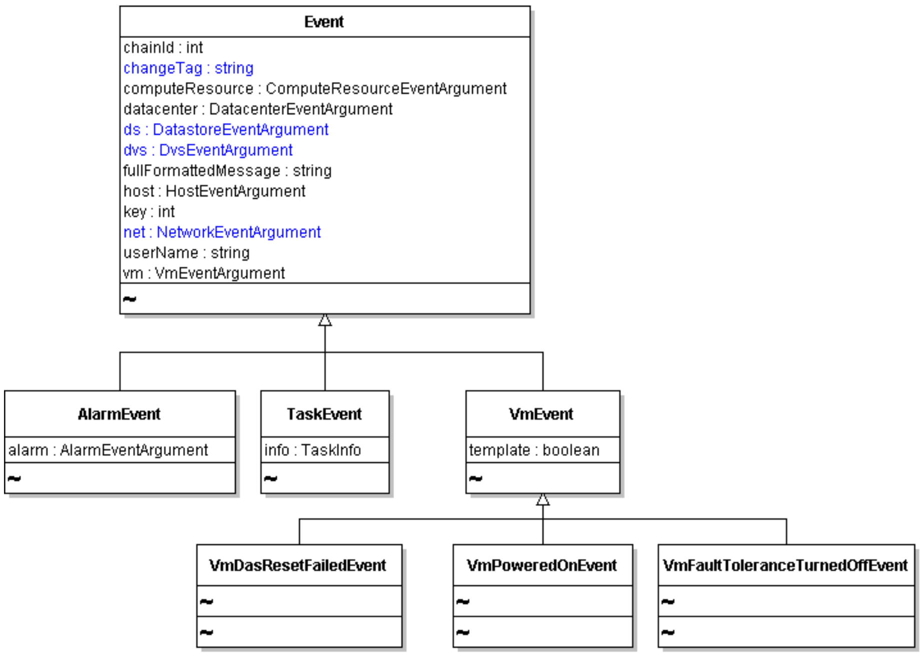
Event Data Objects

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 "Monitoring TaskInfo Properties" on page 169).

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`.

For example, suppose you have a reference to a virtual machine (myVMRef) and you want to log a message to record the fact that a virus check completed. You want to use myVMRef as a parameter to the LogUserEvent method in the next step.

3 Call the LogUserEvent method, passing in the EventManager and the Event reference and a string consisting of the Event message for the msg parameter of the operation.

```
LogUserEvent(eMgrRef, myVMRef, "Completed virus check at 1:05 AM on Sunday December 21.");
```

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 188.

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.

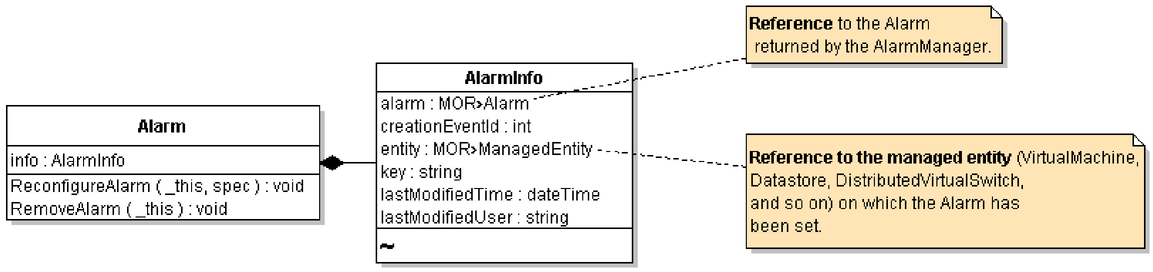
The alarm infrastructure integrates with other server components, such as events and performance counters.

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


Alarm
info : AlarmInfo
ReconfigureAlarm (_this, spec) : void
RemoveAlarm (_this) : void
AlarmInfo
alarm : MOR>Alarm
creationEventId : int
entity : MOR>ManagedEntity
key : string
lastModifiedTime : dateTime
lastModifiedUser : string
Reference to the Alarm returned by the AlarmManager.
Reference to the managed entity (VirtualMachine, Datastore, DistributedVirtualSwitch, and so on) on which the Alarm has been set.

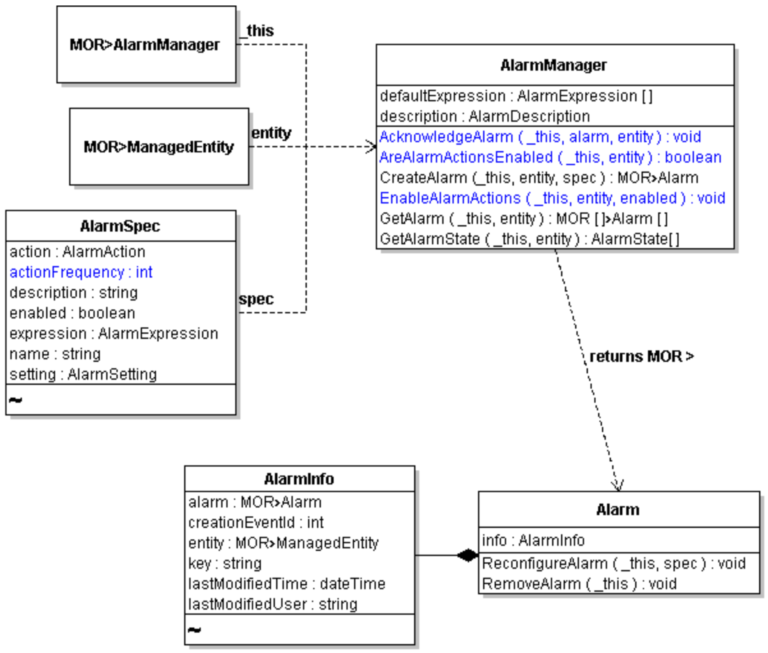
The Alarm.info property is an AlarmInfo data object. You can obtain information about active Alarms by collecting the properties of the AlarmInfo data object.

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


MOR>AlarmManager
_this
MOR>ManagedEntity
entity
AlarmSpec
action : AlarmAction
actionFrequency : int
description : string
enabled : boolean
expression : AlarmExpression
name : string
setting : AlarmSetting
spec
AlarmManager
defaultExpression : AlarmExpression []
description : AlarmDescription
AcknowledgeAlarm (_this, alarm, entity) : void
AreAlarmActionsEnabled (_this, entity) : boolean
CreateAlarm (_this, entity, spec) : MOR>Alarm
EnableAlarmActions (_this, entity, enabled) : void
GetAlarm (_this, entity) : MOR [ ]>Alarm [ ]
GetAlarmState (_this, entity) : AlarmState[ ]
returns MOR >
AlarmInfo
alarm : MOR>Alarm
creationEventId : int
entity : MOR>ManagedEntity
key : string
lastModifiedTime : dateTime
lastModifiedUser : string
Alarm
info : AlarmInfo
ReconfigureAlarm (_this, spec) : void
RemoveAlarm (_this) : void

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 190.

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.

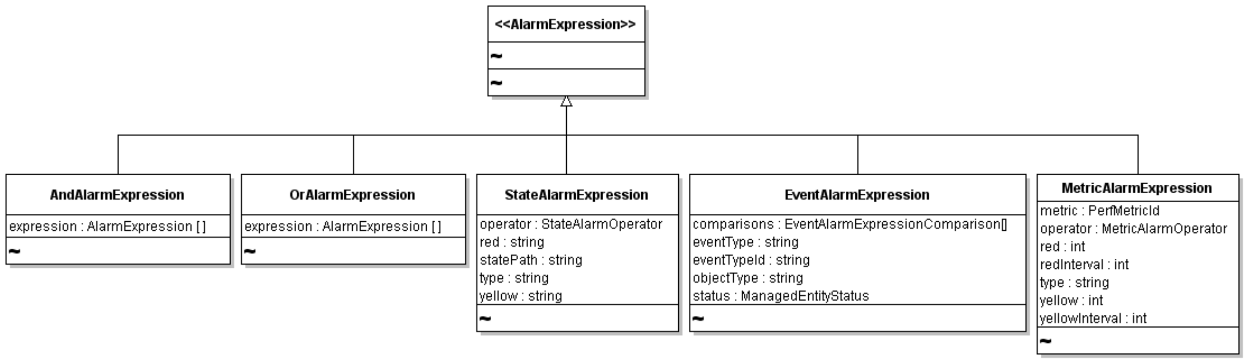
- `action` – Action to initiate when the `Alarm` becomes active. Specify one of the `Action` subtypes. See "Specifying Alarm Actions" on page 192.
- `actionFrequency` – Number of seconds that the `Alarm` remains in the state required to initiate the specified action.
- `expression` – One or more `AlarmExpression` data objects combined in a way that evaluates to a true-false expression. See "Specifying Alarm Trigger Conditions with AlarmExpression" on page 191.
- `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
`StateAlarmExpression`	Specifies thresholds that trigger the alarm.	Triggered by a power state change of a virtual machine or state change of a distributed virtual switch.
`MetricAlarmExpression`	Specifies levels at which the alarm changes state. See "Using MetricAlarmExpression" on page 191.	Triggered when resource utilization metrics exceed a specified limit.
`EventAlarmExpression`	Specifies a type of event as the basis for the alarm.	Triggered by power on or power off events of primary or secondary virtual machines in a fault-tolerant cluster.
`EventAlarmComparison`	Specifies the property of the `Event` that should trigger the alarm and the operator to use as the basis for comparison.	
`AndAlarmExpression` `OrAlarmExpression`	Combines one or more instances of the `AndAlarmExpression` and the `OrAlarmExpression` data objects into an expression that evaluates to `true` or `false`.	

Using MetricAlarmExpression

The `MetricAlarmExpression` data object lets you set an alarm to monitor performance metrics. The vSphere Client uses the data object to indicate when hosts or clusters do not have sufficient resources in a DAS or DRS cluster environment. See the *Resource Management Guide*.

You set the `metric` property to the `PerfMetricId` of a performance metric that you want to monitor on the system. Set the `red` or `yellow` properties to identify the level at which the metric value moves from green, to yellow, to red. You must define `red`, `yellow`, or both properties. Use each of these properties with the `isAbove` or `isBelow` `MetricAlarmOperator` enumerations to complete the definition of the threshold.

In conjunction with `red` and `yellow` properties, you can use the `redInterval` or `yellowInterval` properties. These properties enable you to set the number of seconds that the performance metric must be in `red` or `yellow` state before the expression becomes `true` and triggers the defined action.

Specifying Alarm Actions

You specify the actions that the system should take by setting the `action` property of the `AlarmSpec` data object to the `AlarmAction` data object defined for the purpose.

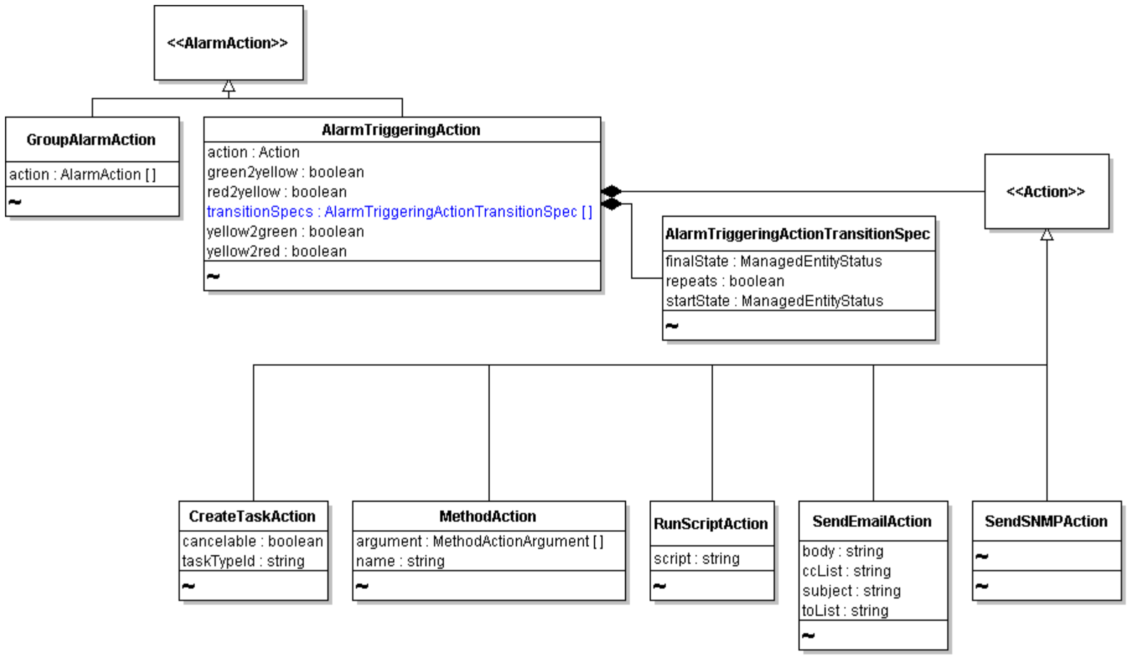
The `AlarmAction` data object is an abstract type that has two descendent objects.

- The `AlarmTriggeringAction` data object has an `action` property and a `transitionSpecs` property. `AlarmTriggeringActionTransitionSpec` allows you to define a starting state and a final state for the `Alarm`. You can limit the number of `Alarm` objects actually triggered to a single `Alarm` by specifying `false` for the `repeats` property of the `AlarmTriggeringActionTransitionSpec`.
- The `GroupAlarmAction` data object is an array version of the `AlarmAction` base type. You can create a single `AlarmAction` instance or an array of `AlarmAction` instances to take effect when the conditions specified for your alarm are met on the system.

The system can respond to an alarm in several ways:

- **Invoking an operation.** To invoke an operation, create a `MethodAction` data object.
- **Running a Script.** To run a script, create an instance of the `RunScriptAction` data object that specifies the fully qualified path to the shell script on the vCenter Server.
- **Send an email message.** To send an email message to a system administrator, use the `SendEmailAction` data object.

Figure 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#
`\samples\alarms\MPowerStateAlarm.java`	`VMPowerStateAlarm`
`\samples\events\EventFormat.java`	`EventFormat`
`\samples\events\EventHistoryCollectorMonitor.java`	`EventHistoryCollectorMonitor`
`\samples\events\VMEventHistoryCollectorMonitor.java`	`VMEventHistoryCollectorMonitor`

vSphere Performance 16

VMware vSphere servers use performance counters to track resource use. At runtime, vSphere components generate performance data which the vSphere servers store in performance counters. You can use the `PerformanceManager` interface to retrieve the data.

This chapter includes the following topics:

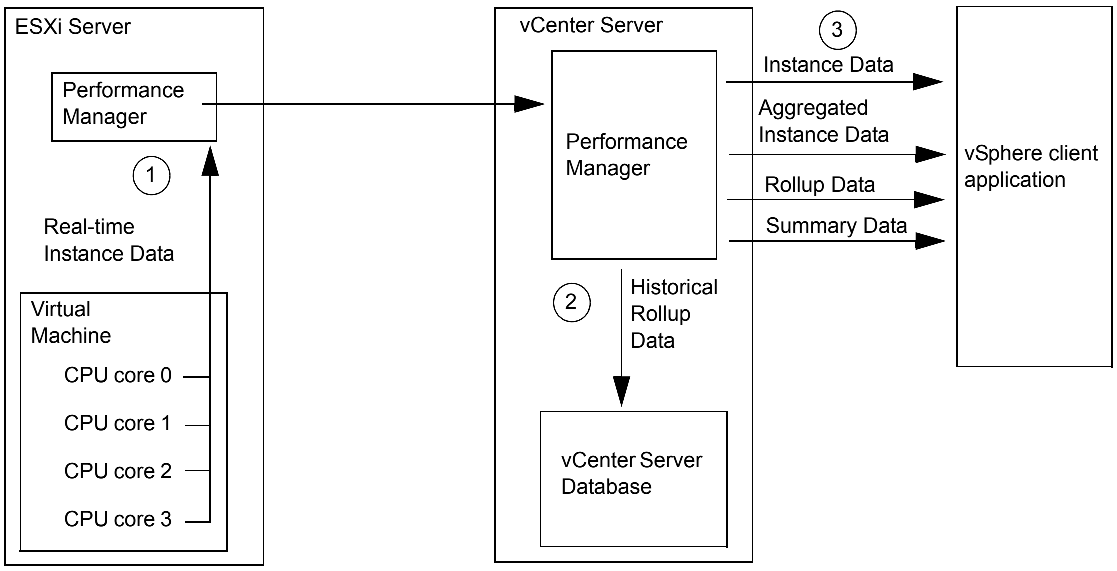
vSphere Performance Data Collection

In a vSphere environment, virtual and physical components generate performance data. To track the use of resources, ESXi Servers perform real-time data collection and vCenter Servers store the data in the vCenter database. vCenter Servers also store a historical rollup of the data according to defined performance intervals.

- Real-time data collection – An ESXi Server collects data for each performance counter every 20 seconds and maintains that data for one hour.
- Historical data rollup – A vCenter Server collects data from all of the hosts that the vCenter Server manages. The `PerformanceManager` defines performance intervals that specify time periods for performance data rollup, a methodology for combining data values. The server stores the rolled up performance counter data in the vCenter database.

The following figure represents vSphere performance data collection and retrieval.

Figure 16-1. vSphere Performance Data Collection and Retrieval



1 ESXi Servers sample performance counter instances every 20 seconds and maintain the real-time instance data for one hour. For example, the figure shows collection of CPU statistics for four CPU cores.

2 The vCenter Server retrieves and stores data from the servers that it manages. The Server produces rollup data according to the settings of the historical intervals.

3 vSphere client applications can retrieve real-time instance data, aggregated instance data, historical rollup data, and summary data.

The following table defines terms that are used to describe vSphere performance management.

Table 16-1. Performance Management Terminology

Term	Definition
performance providers	Performance providers include managed entities, such as hosts, virtual machines, compute resources, resource pools, datastores, and networks.
performance counter	Unit of statistical data collected on a vSphere server. For example, a vCenter server collects the average CPU utilization for hosts, virtual machines and clusters (the counter `cpu.usage.average`).
counter ID	System-generated identifier for a performance counter.
instance	An identifier derived from device configuration names. Examples of counter instances are the name of a virtual Ethernet adapter such as "vmnic0:", or a number that identifies a CPU core, such as 0, 1, 2, or 3. Performance data is retrieved as specific instances of performance counters.
instance data	Performance data collected at 20-second intervals.
metric ID	Combination of a counter ID and an instance. You use metric IDs – `PerfMetricId` objects – when you construct a performance query specification to identify the data to be collected. There are two system-defined instances that you can use to specify aggregate retrieval. See the description of aggregate performance data below. ■ "*" – An asterisk directs the vSphere Server to return all instances plus rollup data. This is not supported for some disk-related counters. ■ "" – A string of length zero directs the vSphere Server to return only aggregated instance data or rollup type data. The vSphere Server returns metric IDs embedded in the data objects that it returns as a response to performance queries.

Table 16-1. Performance Management Terminology

Term	Definition
performance interval	Data object (`PerfInterval`) which defines the time interval between collection events, the collection level, and the time period that the data will be stored on the Server. ■ ESXi Servers define a built-in performance interval that specifies data collection every 20 seconds for each performance counter. ESXi Servers also define a single historical interval (`PerformanceManager.historicalInterval`) that defines aggregate performance data. This system-defined performance interval specifies aggregate data collection every 300 seconds for each counter. You cannot modify the performance intervals on an ESXi Server. ■ vCenter Servers define four performance intervals that determine how collected instance data is aggregated and stored. You can modify the system-defined intervals on a vCenter Server to a limited extent.
collection level	Number between one and four that is assigned to a performance interval (`PerformanceManager.historicalInterval[].level`). The interval collection level corresponds to the level specified for individual performance counters (`PerfCounterInfo.level`). A vCenter Server uses a performance interval to perform performance data aggregation, using data for the counters with levels that match the performance interval collection level.
rollup type	Methodology for producing a single value from a set of statistical values (`PerformanceManager.perfCounter[].rollupType`). Examples of rollup types are average, latest, and summation.
aggregate performance data	A single value that represents a set of instance data values collected for a performance counter. The single value is derived using one of the rollup types.

PerformanceManager Objects and Methods

`PerformanceManager` provides methods for obtaining statistical data about various aspects of system performance, as generated and maintained by the performance providers. It also defines historical performance intervals and it identifies the set of performance counters that you can use to obtain performance data. The following table shows the `PerformanceManager` properties.

Table 16-2. PerformanceManager Properties

Property	Description
description	Composite object that includes information about the types of counters (`counterType`) and statistics (`statsType`) available on the system.
historicalInterval	Array of system-defined performance intervals (`PerfInterval` data objects). Each object defines the interval between rollup events, the collection level, and the time period that the data is stored on the system. ■ For an ESXi system, the array contains a single performance interval. You cannot modify the ESXi performance interval. ■ For vCenter Server systems, the `PerfInterval` objects control how ESXi performance data are rolled up and stored in the database. You can modify some of the `PerfInterval` properties on a vCenter Server.
perfCounter	Array of `PerfCounterInfo` data objects. The array identifies all of the performance counters known to the vCenter Server at the time a client accesses the array. The set of counters may change as ESXi hosts are added or removed from vCenter management. Each `PerfCounterInfo` object contains metadata associated with a performance counter.

The PerformanceManager methods allow you to retrieve performance statistics and to retrieve metadata that defines the statistics. The following table classifies the methods and describes their purposes.

Table 16-3. PerformanceManager Methods

Method Type	Method	Purpose
Performance data availability	QueryAvailablePerfMetric	Returns PerfMetricId objects which identify the counter data available on the specified entity. For example, a virtual machine provides the memory counter granted, which indicates the amount of physical memory that is mapped for the virtual machine. The PerfMetricId object for the mem.granted.average counter specifies the system-defined counter ID. Since this is a memory counter, the PerfMetricId.instance property is empty.
Performance data retrieval	QueryPerf	Returns statistics for a specific list of managed entities that provide performance data.
	QueryPerfComposite	Returns statistics for a host and its virtual machines. This method accepts the refreshRate for current statistics or the intervalId of one of the historical intervals as a parameter. Supported for the HostSystem managed entity only.
Performance counter metadata retrieval	QueryPerfCounter	Returns PerfCounterInfo data objects for the specified list of counter IDs.
	QueryPerfCounterByLevel	Returns PerfCounterInfo data objects for the specified collection level.
Performance provider information	QueryPerfProviderSummary	Returns the PerfProviderSummary data object for the specified managed object.
Collection parameters	ResetCounterLevelMapping	Restores a set of performance counters to their default collection levels.
	UpdateCounterLevelMapping	Changes the collection level for a set of performance counters.
	UpdatePerfInterval	Modifies the system-defined performance intervals.

Retrieving vSphere Performance Data

To retrieve collected data, your client application creates a query specification and passes the specification to a performance query method. The query specification is composed of one or more PerfQuerySpec objects. Each object identifies the following:

- Performance provider – managed entity for which the Server will return performance data (PerfQuerySpec.entity).
- Performance counters – PerfMetricId objects that identify performance counter instances (PerfQuerySpec.metricId).
- Performance interval – the sampling period that defines the data rollup (PerfQuerySpec.intervalId).
- Amount of data to be returned – start and end times (PerfQuerySpec.startTime, PerfQuerySpec.endTime) and maximum number of values (PerfQuerySpec.maxSample) to limit the amount of data to be returned.
- Output data format (PerfQuerySpec.format) – one of two kinds:
  - Normal output returned as values contained in data objects.
  - Formatted output returned as strings containing comma-separated values.

The combination of the `entity` and `metricID` properties determine the set of counters for which the server will return performance data. The combination of the `interval`, `startTime`, `endTime` properties produce instance, aggregated instance, rollup, or summarized data. The following table summarizes the different classifications of performance data that you can retrieve from a vCenter Server.

Table 16-4. Classification of Performance Data By Performance Interval

Performance Data	Description
Instance	ESXi Servers sample performance data every 20 seconds. 20-second interval data is called instance data or real-time data. To retrieve instance data, specify a value of 20 seconds for the `PerfQuerySpec.intervalId` property.
Aggregated Instance	A vSphere client can retrieve aggregated instance data. To obtain aggregated instance data, specify the following `PerfQuerySpec` properties. ■ `intervalId` – Specify 20 seconds to indicate instance data. ■ `metricId[].instance` – specify a zero-length string (“”) for aggregated instance data.
Rollup	The vCenter Server uses the historical intervals to rollup performance data from the servers that it manages. To retrieve historical performance data, specify the following `PerfQuerySpec` properties. ■ `intervalId` – Specify a value that corresponds to one of the historical intervals (`PerformanceManager.historicalInterval[].samplingPeriod`) . ■ `startTime/endTime` – If specified, use time values that are not within the last 30 minutes of the current time. If you do not specify a starting time, the Server will return values starting with the earliest data. If you do not specify an end time, the Server will return values that include the latest data.
Summary	When you call the `QueryPerf` method and specify a performance interval (`PerfQuerySpec.intervalId`) that does not match one of the historical intervals (`PerformanceManager.historicalInterval[].samplingPeriod`), the Server will attempt to summarize the stored data for the specified interval. In this case, the Server may return values that are different from the values that were stored for the historical intervals.

Performance Counter Example (QueryPerf)

The following code fragments are part of an example that uses the `PerformanceManager.QueryPerf` method to obtain performance statistics for a virtual machine. The example code in this section does not include server connection code and it does not show the code for obtaining the managed object reference for the virtual machine. See “Client Applications” on page 27 for an example of server connection code.

This example retrieves the following statistics:

- `disk.provisioned.LATEST` – virtual machine storage capacity.
- `mem.granted.AVERAGE` – amount of physical memory mapped for the virtual machine.
- `power.power.AVERAGE` – current power usage.

The example creates a query specification (`PerfQuerySpec`) to identify the data to be retrieved, calls the `QueryPerf` method, and prints out the retrieved performance data and corresponding performance counter metadata. The following sections describe the basic steps involved in retrieving performance statistics.

- Map the performance counters – “Mapping Performance Counters (Counter Ids and Metadata)” on page 199.
- Create a performance query specification and call the `QueryPerf` method – “Retrieving Statistics” on page 202.
- Process the returned data – “Handling Returned Performance Data” on page 205.

Mapping Performance Counters (Counter Ids and Metadata)

Performance counters are represented by string names, for example `disk.provisioned.LATEST` or `mem.granted.AVERAGE`. A vSphere server tracks performance counters by using system-generated counter IDs. When you create a performance query, you use counter IDs to specify the statistics to be retrieved, so it is useful to map the names to IDs.

The example must specify counter IDs in the calls to QueryPerf, and it will use performance counter metadata when it prints information about the returned data. To obtain performance counter IDs and the corresponding performance counter metadata, the example creates two hash maps. This example maps the entire set of performance counters to support retrieval of any counter.

HashMap Declarations

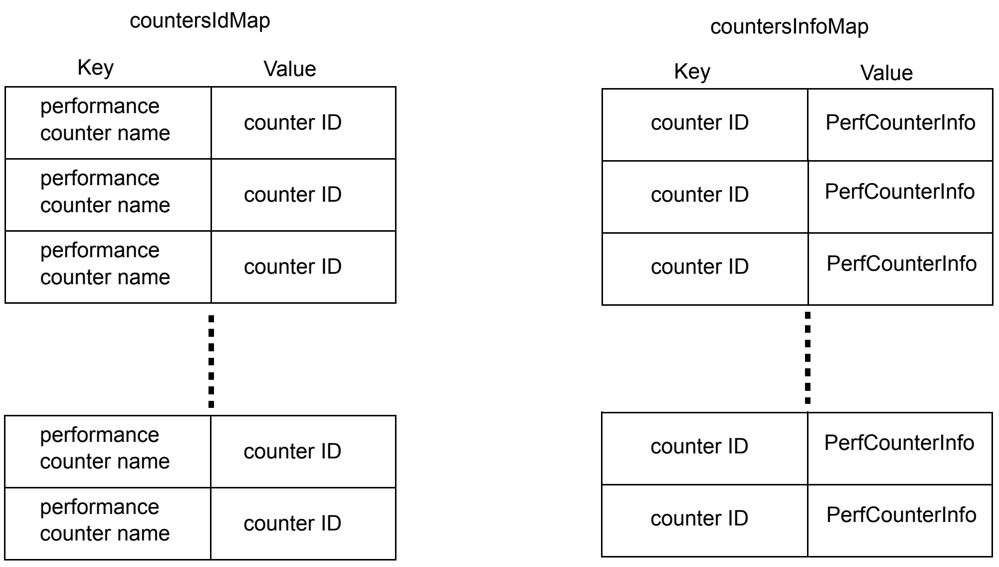
The following code fragment declares two hash maps.

- `countersIdMap` – Uses full counter names to index performance counter IDs. A full counter name is the combination of counter group, name, and rollup type. The example uses this map to obtain counter IDs when it builds the performance query specification.
- `countersInfoMap` – Uses performance counter IDs to index `PerformanceCounterInfo` data objects. The example uses this map to obtain metadata when it prints the returned performance data.

```
/*
 * Map of counter IDs indexed by counter name.
 * The full counter name is the hash key – group.name.ROLLUP-TYPE.
 */
private static HashMap<String, Integer> countersIdMap = new HashMap<String, Integer>();

/*
 * Map of performance counter data (PerfCounterInfo) indexed by counter ID
 * (PerfCounterInfo.key property).
 */
 private static HashMap<Integer, PerfCounterInfo> countersInfoMap =
               new HashMap<Integer, PerfCounterInfo>();
```

The following figure shows a representation of the hash maps.

Figure 16-2. Performance Counter Hash Maps



Creating the Map

The example uses the Property Collector to retrieve the array of performance counters (`PerfCounterInfo`) known to the vCenter Server (`PerformanceManager.perfCounter[]`). It then uses the data to create the maps. The code fragment uses the variable `apiMethods`, which is a `VimPortType` object that provides access to the vSphere API methods. For information about the `VimPortType` object, see "Overview of a Java Sample Application" on page 38.

The following code fragment performs these steps:

1. Create an `ObjectSpec` to define the property collector context. This example specifies the Performance Manager.
2. Create a `PropertySpec` to identify the property to be retrieved. This example retrieves the `perfCounter` property, which is an array of `PerfCounterInfo` objects.
3. Create a `PropertyFilterSpec` for the call to the PropertyCollector. The `PropertyFilterSpec` creates the association between the `ObjectSpec` and PropertySpec for the operation.
4. Call the `PropertyCollector.RetrievePropertiesEx` method. This method blocks until the server returns the requested property data.
5. Cast the returned `xsd:anyType` value into the array of `PerfCounterInfo` objects.
6. Cycle through the returned array and load the maps. The counter-name to counter-ID map uses a fully qualified counter name. The qualified name is a path consisting of counter group, counter name, and rollup type – *group.counter.ROLLUP-TYPE*. The rollup type must be coded in uppercase letters. Examples of qualified names are `disk.provisioned.LATEST` and `mem.granted.AVERAGE`.

```
/*
 * Create an object spec to define the context to retrieve the PerformanceManager property.
 */
ObjectSpec oSpec = new ObjectSpec();
oSpec.setObj(performanceMgrRef);

/*
 * Specify the property for retrieval
 * (PerformanceManager.perfCounter is the list of counters of which the vCenter Server is aware.)
 */
PropertySpec pSpec = new PropertySpec();
pSpec.setType("PerformanceManager");
pSpec.getPathSet().add("perfCounter");

/*
 * Create a PropertyFilterSpec and add the object and property specs to it.
 */
PropertyFilterSpec fSpec = new PropertyFilterSpec();
fSpec.getObjectSet().add(oSpec);
fSpec.getPropSet().add(pSpec);

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

/*
 * Get the performance counters from the server.
 */
RetrieveOptions ro = new RetrieveOptions();
RetrieveResult props = apiMethods.retrievePropertiesEx(pCollectorRef,fSpecList,ro);

/*
 * Turn the retrieved results into an array of PerfCounterInfo.
 */
List<PerfCounterInfo> perfCounters = new ArrayList<PerfCounterInfo>();
if (props != null) {
    for (ObjectContent oc : props.getObjects()) {
        List<DynamicProperty> dps = oc.getPropSet();
        if (dps != null) {
            for (DynamicProperty dp : dps) {
                /*
                 *  DynamicProperty.val is an xsd:anyType value to be cast
                 *  to an ArrayOfPerfCounterInfo and assigned to a List<PerfCounterInfo>.
                 */
                perfCounters = ((ArrayOfPerfCounterInfo)dp.getVal()).getPerfCounterInfo();
            }
        }
    }
}

/*
 * Cycle through the PerfCounterInfo objects and load the maps.
 */
for(PerfCounterInfo perfCounter : perfCounters) {

    Integer counterId = new Integer(perfCounter.getKey());

    /*
     * This map uses the counter ID to index performance counter metadata.
     */
    countersInfoMap.put(counterId, perfCounter);

    /*
     * Obtain the name components and construct the full counter name,
     * for example – power.power.AVERAGE.
     * This map uses the full counter name to index counter IDs.
     */
    String counterGroup = perfCounter.getGroupInfo().getKey();
    String counterName = perfCounter.getNameInfo().getKey();
    String counterRollupType = perfCounter.getRollupType().toString();
    String fullCounterName = counterGroup + "." + counterName + "." + counterRollupType;

    /*
     * Store the counter ID in a map indexed by the full counter name.
     */
    countersIdMap.put(fullCounterName, counterId);

}
```

Retrieving Statistics

The following code fragment calls the `QueryPerf` method to retrieve statistics. It performs these tasks:

1 Create a list of qualified performance counter names for retrieval. The name is a path consisting of *group-name.counter-name.ROLLUP-TYPE,* for example `mem.granted.AVERAGE`. The rollup type must be coded in uppercase letters to match the character case of the rollup type in the performance counter metadata (`PerfCounterInfo.rollupType`). See the *vSphere API Reference* for tables of available counters. The *vSphere API Reference* page for the `PerformanceManager` managed object contains links to the tables.

2 Create a list of `PerfMetricId` objects, one for each counter to be retrieved. The metric ID is a combination of the counter ID and the instance. To fill in the `PerfMetricId` properties, the example does the following:

- Use the `countersIdMap` to translate a full counter name into a counter ID.
- Specify an asterisk (*) for the `PerfMetricId.instance` property. The asterisk is the system-defined instance specification for combined instance and rollup retrieval.

3 Build a query specification for the method call. This query specifies the following:

- Virtual machine for which performance data is being retrieved (`entityMor`);
- Interval ID of 300 to collect 5-minute rollup data.
- Comma-separated value (CSV) format for the retrieved data.

4 Call the `QueryPerf` method.

```
/*
 * Use <group>.<name>.<ROLLUP-TYPE> path specification to identify counters.
 */
String[] counterNames = new String[] {"disk.provisioned.LATEST",
                                      "mem.granted.AVERAGE",
                                      "power.power.AVERAGE"};
```

```
/*
 * Create the list of PerfMetricIds, one for each counter.
 */
List<PerfMetricId> perfMetricIds = new ArrayList<PerfMetricId>();
for(int i = 0; i < counterNames.length; i++) {

    /*
     * Create the PerfMetricId object for the counterName.
     * Use an asterisk to select all metrics associated with counterId (instances and rollup).
     */
    PerfMetricId metricId = new PerfMetricId();
    /* Get the ID for this counter. */
    metricId.setCounterId(countersIdMap.get(counterNames[i]));
    metricId.setInstance("*");
    perfMetricIds.add(metricId);

}


/*
 * Create the query specification for queryPerf().
 * Specify 5 minute rollup interval and CSV output format.
 */
int intervalId = 300;
PerfQuerySpec querySpecification = new PerfQuerySpec();
querySpecification.setEntity(entityMor);
querySpecification.setIntervalId(intervalId);
querySpecification.setFormat("csv");
querySpecification.getMetricId().addAll(perfMetricIds);

List<PerfQuerySpec> pqsList = new ArrayList<PerfQuerySpec>();
pqsList.add(querySpecification);

/*
 * Call queryPerf()
 *
 * QueryPerf() returns the statistics specified by the provided
 * PerfQuerySpec objects. When specified statistics are unavailable -
 * for example, when the counter doesn't exist on the target
 * ManagedEntity - QueryPerf() returns null for that counter.
 */
List<PerfEntityMetricBase> retrievedStats = apiMethods.queryPerf(performanceMgrRef, pqsList);
```

Performance Data Returned by a vSphere Server

The query methods return sampling information and performance data. The sampling information indicates the collection interval in seconds and the time that the data was collected. When you call performance query methods, you pass in query specifications (`PerfQuerySpec`) to identify the performance data to be retrieved. To indicate the format of the output data, specify either "normal" or "csv" for the `PerfQuerySpec.format` property.

The query methods return `PerfEntityMetricBase` objects which you must cast into the appropriate type that corresponds to the `PerfQuerySpec.format` value specified in the call to the method.

- The `QueryPerf` method returns a list of `PerfEntityMetricBase` objects.
- The `QueryPerfComposite` method returns a `PerfCompositeMetric` object, which contains `PerfEntityMetricBase` objects.

Normal Output Format

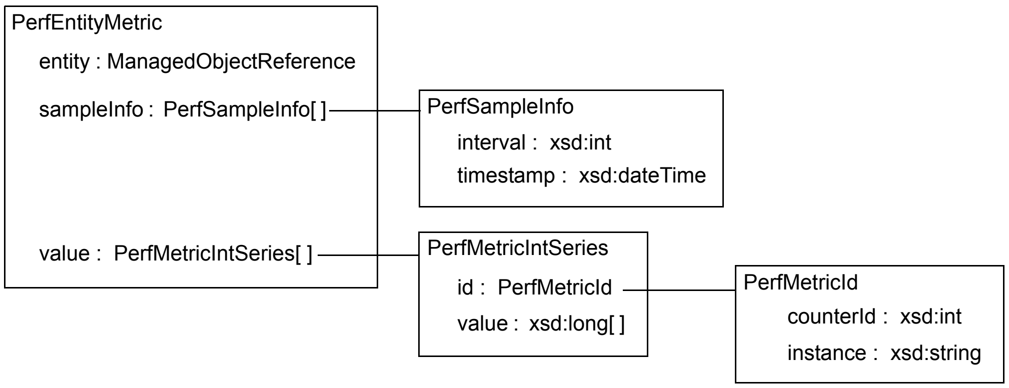
When you specify "normal" format, you must cast the returned `PerfEntityMetricBase` objects into `PerfEntityMetric` objects. Each `PerfEntityMetric` object contains the following properties:

- `entity` – Reference to the performance provider.

- `sampleInfo` – Array of sample information (`PerfSampleInfo` data objects), encoded as `xsd:int` and `xsd:dateTime` values.
- `value` – Array of data values (`PerfMetricIntSeries` data objects). Each object in the array contains the following properties:
  - `id` – Performance metric ID that identifies the counter instance.
  - `value` – Array of integers that corresponds to the array of sample information (`PerfEntityMetric.sampleInfo`).

The following figure shows a representation of the data object hierarchy returned by the query methods for normal format.

Figure 16-3. PerfEntityMetric Object Hierarchy



CSV Output Format

When you specify "csv" format, you must cast the returned `PerfEntityMetricBase` objects into `PerfEntityMetricCSV` objects. Both the sampling information and the collected data are encoded as comma-separated values suitable for display in tabular format.

The `PerfEntityMetricCSV` object contains the following properties:

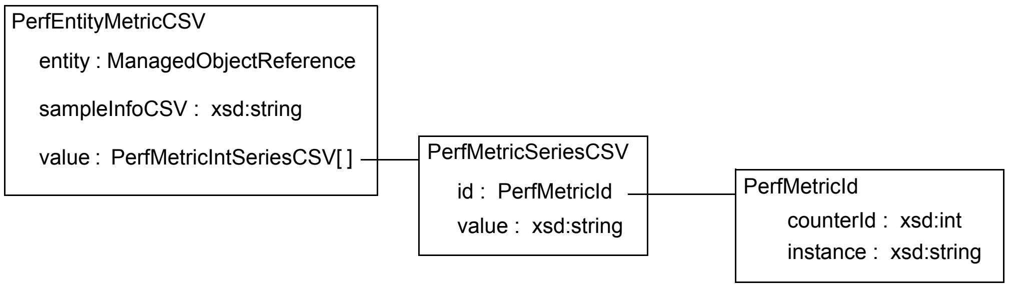
- `entity` – Reference to the performance provider.
- `sampleInfoCSV` – String containing a set of interval and date-time values. The property contains string representations of `PerfSampleInfo` `xsd:int` and `xsd:dateTime` values. The string values are encoded in the following CSV format:

  interval1, date1, interval2, date2

- value – Array of data values (`PerfMetricSeriesCSV` data objects). Each object in the array contains the following properties:
  - id – Performance metric ID that identifies the counter instance.
  - value – Set of sample values in CSV format, corresponding to the list of sample information (`PerfEntityMetricCSV.sampleInfoCSV`).

The following figure shows a representation of the data object hierarchy returned by the query methods for CSV format.

Figure 16-4. PerfEntityMetricCSV Object Hierarchy



Handling Returned Performance Data

The following code fragment prints out the returned performance data. This example uses CSV formatted data. The code fragment performs these tasks:

- Loop through the list of `PerfEntityMetricBase` objects returned by the `QueryPerf` method (`retrievedStats`).
  - Cast the `PerfEntityMetricBase` object to a `PerfEntityMetricCSV` object to handle the CSV output specified in the `PerfQuerySpec`.
  - Retrieve the sampled values.
  - Retrieve the interval information (`csvTimeInfoAboutStats`). The `sampleInfoCSV` string (`PerfEntityMetricCSV.sampleInfoCSV`) is `PerfSampleInfo` data formatted as interval,time pairs separated by commas – `interval-1,time-1,interval-2,time-2`. The list of pairs embedded in the string corresponds to the list of sampled values (`PerfEntityMetricCSV.value[]`).
  - Print the time and interval information.
  - Loop through the sampled values (`metricsValues`).
    - Use the `countersInfoMap` to translate the counter ID returned in the `PerfMetricSeriesCSV` object into the corresponding `PerfCounterInfo` object.
    - Use the counter metadata to print out identifying information about the counter along with the returned sampled value for the counter.

```
/*
 * Cycle through the PerfEntityMetricBase objects. Each object contains
 * a set of statistics for a single ManagedEntity.
 */
for(PerfEntityMetricBase singleEntityPerfStats : retrievedStats) {

    /*
     * Cast the base type (PerfEntityMetricBase) to the csv-specific sub-class.
     */
     PerfEntityMetricCSV entityStatsCsv = (PerfEntityMetricCSV)singleEntityPerfStats;

     /* Retrieve the list of sampled values. */
     List<PerfMetricSeriesCSV> metricsValues = entityStatsCsv.getValue();

     if(metricsValues.isEmpty()) {
         System.out.println("No stats retrieved. " +
                            "Check whether the virtual machine is powered on.");
         throw new Exception();
     }

    /*
     * Retrieve time interval information (PerfEntityMetricCSV.sampleInfoCSV).
     */
    String csvTimeInfoAboutStats = entityStatsCsv.getSampleInfoCSV();

    /* Print the time and interval information. */
```

```
        System.out.println("Collection: interval (seconds),time (yyyy-mm-ddThh:mm:ssZ)");
        System.out.println(csvTimeInfoAboutStats);

        /*
         * Cycle through the PerfMetricSeriesCSV objects. Each object contains
         * statistics for a single counter on the ManagedEntity.
         */
        for(PerfMetricSeriesCSV csv : metricsValues) {

            /*
             * Use the counterId to obtain the associated PerfCounterInfo object
             */
            PerfCounterInfo pci = countersInfoMap.get(csv.getId().getCounterId());

            /* Print out the metadata for the counter. */
            System.out.println("----------------------------------------");
            System.out.println(pci.getGroupInfo().getKey() + "."
                + pci.getNameInfo().getKey() + "."
                + pci.getRollupType() + " - "
                + pci.getUnitInfo().getKey());
            System.out.println("Instance: "+csv.getId().getInstance());
            System.out.println("Values: " + csv.getValue());

        }
    }
```

Large-Scale Performance Data Retrieval

The example described in the previous sections ("Performance Counter Example (QueryPerf)" on page 199) shows how to retrieve performance data for a single entity. When you design your application to retrieve performance data on a large scale, take the following information into consideration for more efficient processing.

- Use CSV formatted output. CSV format provides a more compact representation of the output data which can save on meta-data overhead.
- Create query specifications to reference a set of vSphere entities.
  - Using one `QueryPerf` method call per entity is not efficient.
  - Using a single call to `QueryPerf` to retrieve all of the performance data is not efficient.
  - As a general rule, specify between 10 and 50 entities in a single call to the `QueryPerf` method. This is a general recommendation because your system configuration may impose different constraints.
- Do not retrieve statistics more frequently than they are refreshed. For example, when you retrieve 20-second interval data, the data will not change until the next 20-second data collection event.
- Use `QueryAvailablePerfMetric` only when you intend to send a query for a specific counter using a specific performance interval. The method will return `PerfMetricId` objects that you can use for the query.

  In all other cases, create the `PerfMetricId` objects for the query.
  - For the `counterId` property, use the counter IDs from the `PerformanceManager` counter list (`PerformanceManager.perfCounter[].key`).
  - For the instance property, specify an asterisk ("*") to retrieve instance and aggregate data or a zero-length string ("") to retrieve aggregate data only.

Using the QueryPerf Method as a Raw Data Feed

The `QueryPerf` method can operate as a raw data feed that bypasses the vCenter database and instead retrieves performance data from an ESXi host. You can use a raw data feed to obtain real-time instance data associated with 20-second interval collection and aggregate data associated with the 5-minute intervals.

You can use a raw data feed on VirtualCenter Server 2.5 and later.

Table 16-5. Raw Data Feed

Performance Interval	Description
20-second	ESXi servers collect data for each performance counter every 20 seconds and maintain that data for an hour. When you specify a 20-second interval in the query specification for the `QueryPerf` method (`PerfQuerySpec.intervalId`), the method operates as a raw data feed. The Server ignores the historical interval collection levels and retrieves data for all of the requested counters from the ESXi servers. When you send a query for 20-second instance data, the server returns the most recent data collected for the 20-second interval. The server does not perform additional, unscheduled data collection to satisfy the query.
5-minute	ESXi servers aggregate performance data according to the system-defined performance interval which specifies data collection every 300 seconds. To use a raw data feed for this data, specify the following `PerfQuerySpec` properties in the call to the `QueryPerf` method. ■ `intervalId` – Specify 300 seconds to match the system-defined performance interval. ■ `startTime`/`endTime` – Specify time values within the last 30 minutes of the current time. The `QueryPerf` method checks the performance interval collection level on the vCenter Server. The method returns aggregated statistics for performance counters that specify a collection level (`PerfCounterInfo.level`) at or below the vCenter Server performance interval for the 300 second sampling period (`PerfInterval.level`). For example, if the vCenter Server performance interval is set to level one, and your query specification requests only performance counters that specify level four, the `QueryPerf` method will not return any data.

Comparison of Query Methods

The following table presents a comparison of performance query methods.

Table 16-6. Performance Query Methods

Method	Notes
QueryPerf	■ Specify an array of `PerfQuerySpec` objects. ■ An unset `PerfQuerySpec.metricId` property produces results for all counters defined for `PerfQuerySpec.entity`. ■ `PerfQuerySpec.maxSample` is ignored for historical statistics. You can use this method to retrieve historical statistics; you can also use it as a raw data feed. For information about retrieving the raw data collected on ESXi servers, see "Using the QueryPerf Method as a Raw Data Feed" on page 206.
QueryPerfComposite	■ Method works only at the host level. You can use a single call to the `QueryPerfComposite` method to retrieve performance data for a host and its virtual machines. ■ Specify a single `PerfQuerySpec` object. ■ You must specify a list of performance metrics to identify the data to be retrieved (`PerfQuerySpec.metricId`). ■ You cannot specify `PerfQuerySpec.maxSample`. This method 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.

Retrieving Summary Performance Data

You can obtain near real-time summary information about performance or utilization without using the PerformanceManager methods. vSphere servers maintain "quick stats" data objects for hosts (`HostListSummaryQuickStats`), virtual machines (`VirtualMachineQuickStats`), and resource pools (`ResourcePoolQuickStats`). For more information about these objects, see the *vSphere API Reference*.

Performance Counter Metadata

Performance counters are organized by groups of system resources. Examples of performance counter groups are memory, CPU, and disk. The counter groups and specific counters used on any vSphere server depend on the server configuration. The *vSphere API Reference* contains a table for each counter group. The table includes the counter name, type of statistics being collected, unit of measurement, level, and so on. The *vSphere API Reference* page for the `PerformanceManager` managed object contains links to the tables.

The `PerformanceManager.perfCounter` property is an array of `PerfCounterInfo` data objects. Each object provides metadata for the collected data. A `PerfCounterInfo` object has a unique key, the counter ID. The actual performance data collected at runtime are identified by this counter ID. The following table lists the `PerfCounterInfo` properties.

Table 16-7. PerfCounterInfo Data Object Properties

Property	Description
`groupInfo`	Name of the resource group to which this counter belongs, such as disk, cpu, or memory.
`key`	Unique integer that identifies the counter. Also called the counter ID. The value is unique and it is not static—it might, for example, change between system reboots. The counter key on an ESXi system might not be the same as the counter key for the same counter on the vCenter Server system managing the ESXi system. However, the system maps the keys from ESXi to vCenter Server systems automatically.
`level`	Number from 1 to 4 that identifies the level at which data values for this counter are aggregated.
`nameInfo`	Descriptive name for the counter. The name component of a fully qualified counter name, for example "granted" is the `nameInfo` property for the `mem.granted.AVERAGE` counter.
`rollupType`	Indicates how multiple samples of a counter are transformed into a single statistical value. Examples of rollup types are average, summation, and minimum. No conversion of values occurs for counters that specify absolute values, such as the total number of seconds that the system has been running continuously since startup. The `PerfSummaryType` is an enumeration containing valid constants for this property.
`statsType`	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 `PerfStatsType` is an enumeration containing valid constants for this property.
`unitInfo`	Unit of measure, such as megahertz, kilobytes, kilobytes per second, and so on. The `ElementDescription`'s `key` property is populated using one of the constants available in the `PerformanceManagerUnit` enumeration.

Performance Intervals

The `PerformanceManager` defines performance intervals which specify the period of time between collection events, how much data will be collected, and how long the collected data will be saved.

- An ESXi server has a built-in performance interval that produces discrete data values from counter instances sampled every 20 seconds. The server will maintain this instance data for one hour.
- Additional data collection is specified by historical performance intervals which produce data aggregated from counter instances according to the individual intervals.

The `PerformanceManager.historicalInterval` property is an array of `PerfInterval` objects. The following table lists the `PerfInterval` properties.

Table 16-8. PerfInterval Properties

Property	Description
samplingPeriod	Number of seconds for the interval. You can modify this property on a vCenter Server only.
length	Period of time for which the server will save the data that it collects. You can modify this property on a vCenter Server only.

Table 16-8. PerfInterval Properties (Continued)

Property	Description
level	Level at which the Server collects data. The interval level corresponds to the performance counter level (`PerfCounterInfo.level`). The Server will collect data for all counters with levels that match `PerfInterval.level`, and for all counters with levels lower than `PerfInterval.level`. You can modify this property on a vCenter Server only.
enable	Enable/disable performance data collection. You can modify this property on a vCenter Server only.
key	Unique identifier for the interval. You cannot modify this property.
name	Label for the historical interval; one of the following strings: ■ "Past Day" ■ "Past Week" ■ "Past Month" ■ "Past Year" The `PerformanceManager` uses the `samplingPeriod`, `level`, and `length` properties to determine its collection behavior. It does not interpret the name string. You cannot modify this property.

ESXi Server Performance Intervals

An ESXi server collects performance data for each performance counter every 20 seconds. The `PerformanceManager.historicalInterval` array for an ESXi Server contains a single, readonly `PerfInterval` object that specifies rollup data collection every 5 minutes. You cannot retrieve 5-minute rollup data from an ESXi Server directly. You can use a vCenter Server connection to obtain 5-minute rollup data for an ESXi Server. The following table shows the historical interval property values on an ESXi server. You cannot modify this performance interval.

Table 16-9. Values of PerfInterval Data Object from an ESXi System

Property	Value	Description
`key`	`1`	Numeric identifier for the `PerfInterval`.
`name`	`PastDay`	Name of the `PerfInterval`.
`samplingPeriod`	`300`	Time interval between data sampling events.
`length`	`129600`	Number of seconds that statistics associated with the interval are kept by the vCenter Server.
`enabled`	`true`	This `PerfInterval` is enabled on the system.
`level`	`null`	Statistics collection level. For an ESXi system, this property is `null`. The `PerfInterval` object on an ESXi system defines the baseline interval.

vCenter Server Performance Intervals

A vCenter Server system aggregates performance data from all ESXi systems that it manages. The amount of data aggregated depends on the level setting configured for the vCenter Server. The level settings are reflected in the `PerformanceManager.historicalInterval` property for the vCenter Server system. `historicalInterval` is an array of `PerfInterval` data objects that define four different level settings, 1 through 4.

Table 16-10 lists the default values for the performance intervals on a vCenter Server system.

Table 16-10. 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

By default, the collection level is set to 1 for each of the four intervals. Using the default level, a vCenter Server will collect data for all performance counters that specify collection level 1. Using the default length value, a vCenter Server will save collection data for the following time periods:

- 5-minute samples for the past day
- 30-minute samples for the past week
- 2-hour samples for the past month
- 1-day samples for the past year

Data older than a year is purged from the vCenter Server database.

vSphere Performance and Data Storage

The following sections provide information about modifying the operation of the `PerformanceManager` and vSphere Server performance data collection and storage.

- "Modifying Historical Intervals" on page 210
- "Modifying Performance Counter Collection Levels" on page 210

Modifying Historical Intervals

Changes to a vCenter performance interval are global and apply to all entities in the system. VMware recommends that you do not modify the historical intervals. The `PerfInterval` data objects in the `PerformanceManager.historicalInterval` array are related. Modifications to a performance interval affects the entire system and may cause problems.

If you must modify a performance interval, use the `PerformanceManager.UpdatePerfInterval` method and follow these guidelines.

- Performance data retention time (`PerfInterval.length`) must be a multiple of the collection interval (`PerfInterval.samplingPeriod`).
- Performance data retention length must increase in each interval compared to its predecessor. The `PerfInterval.length` value for each successive performance interval must be greater than the `length` property for the previous interval in the historical interval array.
- You cannot modify the value of the PerfInterval.`samplingPeriod` property on ESXi systems.

Modifying Performance Counter Collection Levels

The `PerformanceManager` provides the `UpdateCounterLevelMapping` method to change the collection level for individual performance counters (`PerfCounterInfo.level`). Consider carefully the performance and storage consequences of using the `UpdateCounterLevelMapping` method. If you use this method, you may cause a significant increase in data collection and storage, along with a corresponding decrease in performance. vCenter Server performance and database storage requirements depend on the collection levels defined for the performance intervals (`PerformanceManager.historicalInterval`) and the collection levels specified for individual performance counters (PerfCounterInfo.level).

Performance Counter Data Collection

vSphere defines four levels of data collection for performance counters. Each performance counter specifies a level for collection. The historical performance intervals (`PerformanceManger.historicalInterval`) define the sampling period and length for a particular collection level.

The amount of data collected for a performance counter depends on the performance interval and on the type of entity for which the counter is defined. For example, a datastore counter such as datastoreIops (the aggregate number of IO operations on the datastore) will generate a data set that corresponds to the number of datastores on a host. If a vCenter Server manages a large number of hosts with a large number of datastores, the Server will collect a large amount of data.

There are other counters for which the vCenter Server collects a relatively smaller amount of data. For example, memory counters are collected as a single counter per virtual machine and a single counter per host.

Performance Counter Data Storage

The performance interval collection level (`PerfInterval.level`) defines the set of counters for which the vCenter Server stores performance data. The Server will store data for counters at the specified level and for counters at all lower levels.

By default, all the performance intervals specify collection level one. Using these defaults, the vCenter Server stores performance counter data in the vCenter database for all counters that specify collection level one. It does not store data for counters that specify collection levels two through four.

Performance Manager Method Interaction

You can use the `UpdateCounterLevelMapping` method to change the collection level for individual counters. You can also use the `UpdatePerfLevel` method to change the collection level for the system-defined performance intervals. These methods can cause a significant increase in the amount of data collected and stored in the vCenter database.

- By default the system-defined performance intervals use collection level one, storing data for all counters that specify collection level one. If you use the `UpdateCounterLevelMapping` method to change the collection level of performance counters to level one, you will increase the amount of stored performance data.
- If you use the `UpdatePerfLevel` method to increase the collection level for the system-defined performance intervals, you will increase the amount of stored performance data.

To restore counter levels to default settings use the `ResetCounterLevelMapping` method.

vSphere Client Management of Performance Statistics

The vSphere Client displays the Performance Manager historical interval collection levels in the vCenter management statistics display. The vSphere Client also displays an estimate of the amount of storage that is required for data collection at the displayed levels. If individual counter levels are modified through the vSphere API (the `UpdateCounterLevelMapping` method), the vSphere Client will show a modified estimate. However, the vSphere Client cannot detect that the method has been called and it cannot display the current levels for individual counters. If you see a significantly increased estimate for storage, be aware that someone may have used the vSphere API to modify data collection.

Sample Code Reference

Table 16-11 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 16-11. PerformanceManager Sample Applications

Java (SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\performance)	**C# (SDK\vsphere-ws\dotnet\cs\samples\)**
`Basics.java`	`Basics\Basics.cs`
	`Basics\Basics.csproj`
	`Basics\Basics2008.csproj`
	`Basics\Basics2010.csproj`
`History.java`	`History\History.cs`
	`History\History.csproj`
	`History\History2008.csproj`
	`History\History2010.csproj`
`PrintCounters.java`	`PrintCounters\PrintCounters.cs`

Table 16-11. PerformanceManager Sample Applications (Continued)

Java (SDK\vsphere-ws\java\JAX-WS\samples\com\vmware\performance)	**C# (SDK\vsphere-ws\dotnet\cs\samples\)**
	`PrintCounters\PrintCounters.csproj`
	`PrintCounters\PrintCounters2008.csproj`
	`PrintCounters\PrintCounters2010.csproj`
	`QueryMemoryOverhead\QueryMemoryOverhead.cs`
	`QueryMemoryOverhead\QueryMemoryOverhead.csproj`
	`QueryMemoryOverhead\QueryMemoryOverhead2008.csproj`
	`QueryMemoryOverhead\QueryMemoryOverhead2010.csproj`
`RealTime.java`	`RealTime\RealTime.cs`
	`RealTime\RealTime.csproj`
	`RealTime\RealTime2008.csproj`
	`RealTime\RealTime2010.csproj`
`VITop.java`	
`VIUsage.java`	

Diagnostics and Troubleshooting

A

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

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` sample application as an example.
  - C# client logging example: `\SDK\vsphere-ws\dotnet\cs\samples\AppUtil\Log.cs`
- 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, "vSphere Performance," on page 195.
  - **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 221.
  - **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 214. Can be used as a SOAP monitor when set to trivia log level as in"Generating Logs" on page 217.
- `vpxa.log` – Agent log file found on each managed ESX/ESXi system.
- `vmware.log` – Virtual machine log. See "Virtual Machine Log Files" on page 215.

In addition to viewing log files in real time you can also generate reports and complete diagnostic bundles. See "Generating Diagnostic Bundles" on page 220.

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

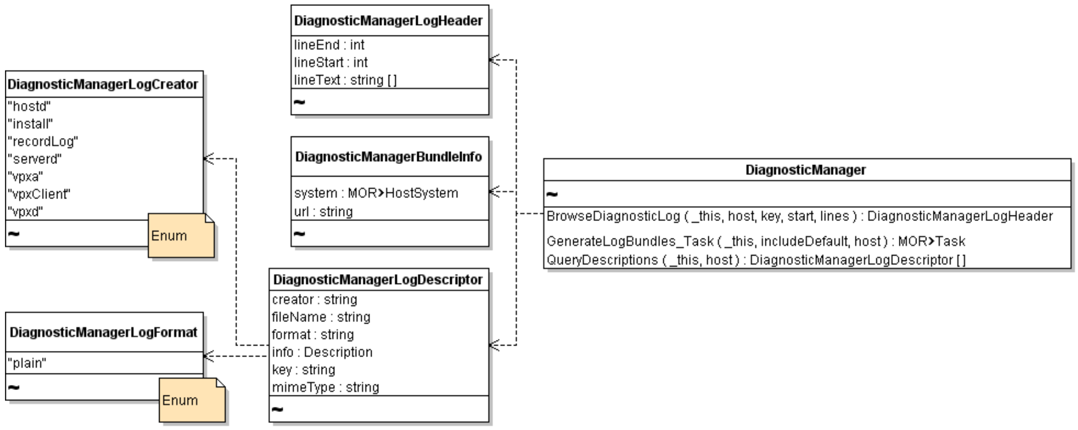
The vSphere API provides access to the `DiagnosticManager`, the service interface for obtaining information from the log files and for generating diagnostic bundles. The logs are populated based on configuration settings, such as `info`, `trivia`, and so on. See Table A-2, "Log Level Settings," on page 216.

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


DiagnosticManagerLogHeader
lineEnd : int
lineStart : int
lineText : string []
DiagnosticManagerLogCreator
"hostd"
"install"
"recordLog"
"serverd"
"vpxa"
"vpxClient"
"vpxd"
Enum
DiagnosticManagerBundleInfo
system : MOR>HostSystem
url : string
DiagnosticManager
BrowseDiagnosticLog (_this, host, key, start, lines) : DiagnosticManagerLogHeader
GenerateLogBundles_Task (_this, includeDefault, host) : MOR>Task
QueryDescriptions (_this, host) : DiagnosticManagerLogDescriptor []
DiagnosticManagerLogDescriptor
creator : string
fileName : string
format : string
info : Description
key : string
mimeType : string
DiagnosticManagerLogFormat
"plain"
Enum

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
`vpxa`	`/var/log/vmware/vpx/vpxa.log`	plain	VirtualCenter Agent Log	VirtualCenter agent log in plain format	`vpxa`	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 221.

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 110 for information about setting up your system for core dumps.

Managed Object Browser

B

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 221
- "Using the MOB to Invoke Methods" on page 222

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.

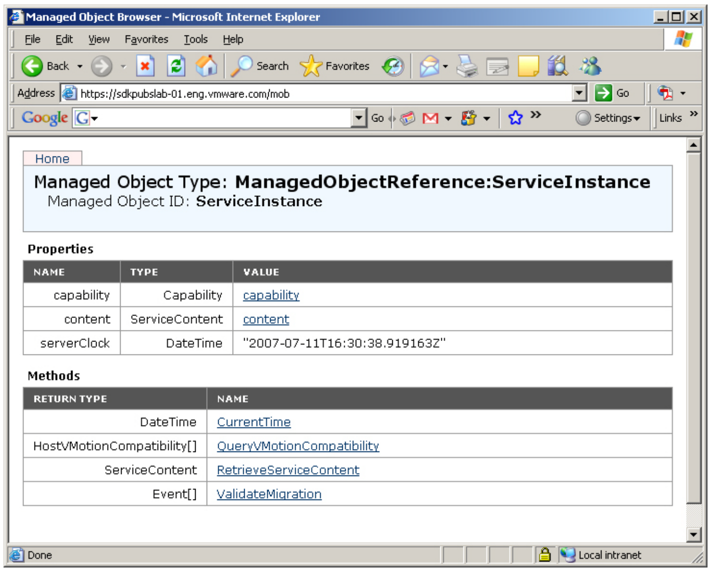
The MOB reveals the underlying structures of the object model. Seeing the structure in conjunction with the *API Reference Guide*, can help with understanding the model.

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 a ServiceInstance object. The `ServiceInstance` methods and properties provide access to the entire set of services and inventory objects available on the server. See Chapter 4, “Datacenter Inventory,” on page 49.

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


Managed Object Browser - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address https://sdkpubslab-01.eng.vmware.com/mob
Go
Google Go Settings Links
Done
Local intranet

Home

Managed Object Type: **ManagedObjectReference:ServiceInstance**
Managed Object ID: **ServiceInstance**

Properties

NAME	TYPE	VALUE
capability	Capability	capability
content	ServiceContent	content
serverClock	DateTime	"2007-07-11T16:30:38.919163Z"

Methods

RETURN TYPE	NAME
DateTime	CurrentTime
HostVMotionCompatibility[]	QueryVMotionCompatibility
ServiceContent	RetrieveServiceContent
Event[]	ValidateMigration

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

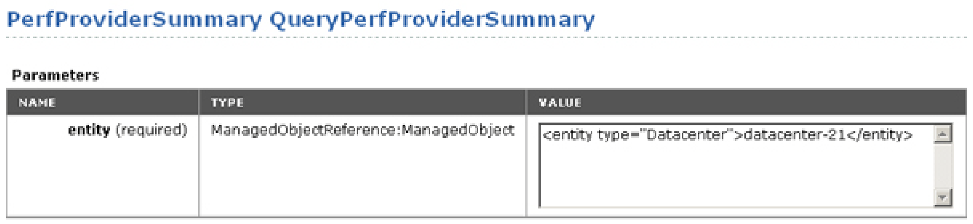
```
<complexType xmlns="http://www.w3.org/2001/XMLSchema" xmlns:vim25="urn:vim25"
               name="ManagedObjectReference">
    <simpleContent>
        <extension base="xsd:string">
            <attribute name="type" type="xsd:string"/>
        </extension>
    </simpleContent>
</complexType>
```

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).

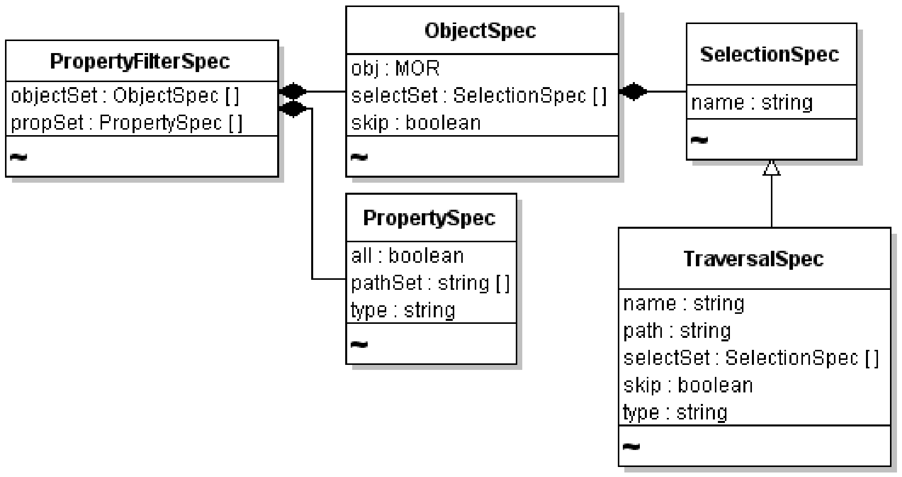
Complex Content

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>
...
```

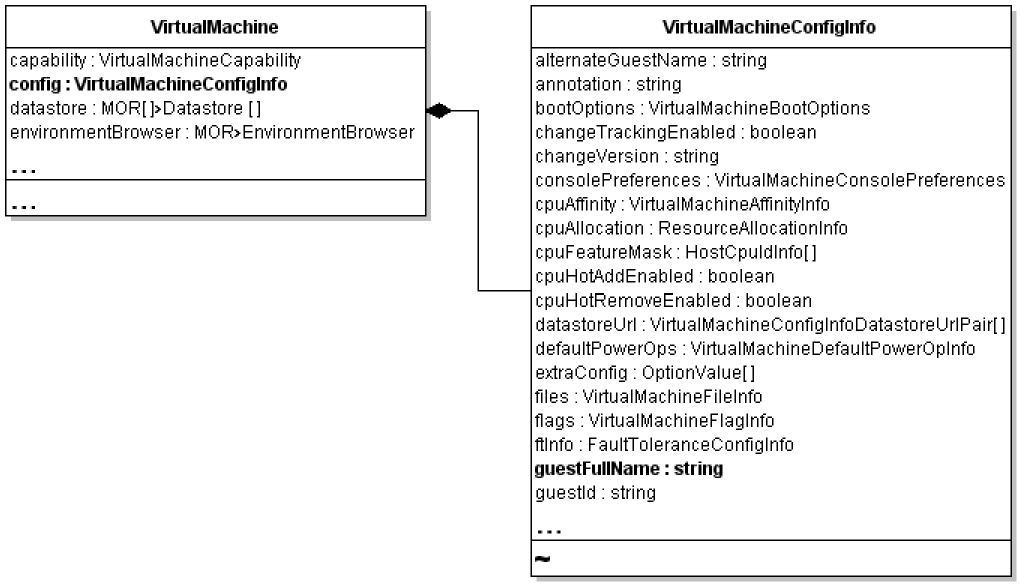
Extrapolating from the WSDL definitions shown in Example B-2, Example B-3, and Example B-4 might produce results similar to those shown in Example B-5.

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


VirtualMachine
capability : VirtualMachineCapability
config : VirtualMachineConfigInfo
datastore : MOR[ ]>Datastore [ ]
environmentBrowser : MOR>EnvironmentBrowser
...
...
VirtualMachineConfigInfo
alternateGuestName : string
annotation : string
bootOptions : VirtualMachineBootOptions
changeTrackingEnabled : boolean
changeVersion : string
consolePreferences : VirtualMachineConsolePreferences
cpuAffinity : VirtualMachineAffinityInfo
cpuAllocation : ResourceAllocationInfo
cpuFeatureMask : HostCpuIdInfo[ ]
cpuHotAddEnabled : boolean
cpuHotRemoveEnabled : boolean
datastoreUrl : VirtualMachineConfigInfoDatastoreUrlPair[ ]
defaultPowerOps : VirtualMachineDefaultPowerOpInfo
extraConfig : OptionValue[ ]
files : VirtualMachineFileInfo
flags : VirtualMachineFlagInfo
ftInfo : FaultToleranceConfigInfo
guestFullName : string
guestId : string
...
~

All of these details are available in the *vSphere API Reference*. By examining the WSDL definition, you can construct the strings needed to submit parameters through the MOB. 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 *vSphere API Reference* 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

C

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

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 228
- "Host File Access (/host)" on page 229
- "Update Package Access (/tmp)" on page 230

Datastore Access (/folder)

An HTTP request for datastore access uses the following syntax:

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

http-method	One of the methods `GET`, `HEAD`, `PUT`, or `DELETE`.
`http://` or `https://`	Access protocol (standard access or secure access).
server	ESX/ESXi or vCenter Server target system. The server value can be an IP address or a DNS name.
`/folder`	Specifies datastore access on an ESX/ESXi or vCenter Server system. The datastore URL can include the following optional elements: ■ *path* – Path to a file or directory in the datastore, relative to the root of the datastore. ■ `dcPath` – Inventory path to a datacenter. Specify the datacenter path as a name-value pair in the request. For a datacenter named `Datacenter` located in the root folder, the `dcPath` value is `MyDatacenter`. For a datacenter named `YourDatacenter` located in the folder `NorthAmerica` which is located in the root folder, the `cdPath` value is `NorthAmerica/YourDatacenter`. ■ `dsName` – 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**
`/folder`	Directory listing of known datacenters on this server.
`/folder?dcPath=`*`path`*	Directory listing of all datastores available at the specified datacenter.
`/folder?dcPath=`*`path`*`&dsName=`*`name`*	Top-level directory listing of the datastore.
`/folder/`*`path`*`?dcPath=`*`path`*`&dsName=`*`name`*	Directory listing of all files in a datastore directory.
`/folder/`*`path`*`/disk-flat.vmdk?dcPath=`*`path`*`&dsName=`*`name`*	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
sfcb.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 81 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

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 *vSphere API Reference* 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 *vSphere API Reference*.	X	X
DestroyChildren	See DestroyChildren in the *vSphere API Reference*.	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 *vSphere API Reference*.	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 *vSphere API Reference*.	X	X
MoveIntoResourcePool	See MoveIntoFolder_Task in the *vSphere API Reference*.	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 *vSphere API Reference*.	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
`StopMasterSnmpAgent`	`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 *vSphere API Reference*.	X	X
`UpdateConfig`	See `UpdateConfig` in the *vSphere API Reference*.	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 *vSphere API Reference*.	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 *vSphere API Reference*. `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

Table 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

E

VMware vSphere Web Services SDK includes samples for both the Java and C# platforms. This appendix lists the available sample programs and provides some information about each program. Both the Java and C# samples have been re-compiled with JAX-WS bindings for this release, and they use JAX-WS credential store classes that allow you to ignore certificates when you connect to a server with the samples.

The information is presented in the following topics:

- "Java Sample Programs (JAXWS Bindings)" on page 245
- "C# Sample Programs" on page 249

Java Sample Programs (JAXWS Bindings)

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

- `SDK\vsphere-ws\java\JAXWS\samples\com\vmware` – Top-level directory for Java samples. Details listed in Table E-1.
- `SDK\vsphere-ws\java\JAXWS\samples\com\vmware\security` – Credential store utilities
- `SDK\vsphere-ws\java\JAXWS\samples\com\vmware\vim25` – Stub directories. The `vim25` directory contains stubs for VirtualCenter 2.5 and later, including vSphere 4.0 and later.
- `SDK\vsphere-ws\java\JAXWS\samples\com\vmware\vm` – samples written for a single vm

Table E-1. Java Sample Programs

Directory	**Example**	Description
`alarm`	`VMPowerStateAlarm`	Creates an alarm to monitor a virtual machine's power state.
`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.
	`VMEventHistoryCollectorMonitor`	Standalone client that demonstrates how to perform the following tasks: Logging into the web service. Creating `EventHistoryCollector` filtered for a single virtual machine. Monitoring events using the latestPage property of the `EventHistoryCollector`.

Table E-1. Java Sample Programs (Continued)

Directory	**Example**	Description
`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.
	`GetCurrentTime`	Retrieves the current time from the vSphere Server.
	`GetHostName`	Retrieves the hostname of the ESX Server.
	`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, or Folder.
	`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.
`guest`	`CreateTemporaryFile`	Creates a temporary file inside a virtual machine.
	`DownloadGuestFile`	Downloads a file from the guest to a specified path on the host where the client is running.
	`RunProgram`	Runs a specified program inside a virtual machine. RunProgram re-directs output to a temporary file inside the guest and downloads the output file.
	`UploadGuestFile`	Uploads a file from the client machine to a specified location inside the guest operating system.

Table E-1. Java Sample Programs (Continued)

Directory	Example	Description
host	AcquireSessionInfo	Acquires a session with a vCenter Server or ESX host and prints a cim service ticket and related session information to a file.
	AddVirtualNic RemoveVirtualNic	Adds a virtual NIC to a port group on a virtual switch. Removes a virtual NIC from a port group.
	AddVirtualSwitch RemoveVirtualSwitch	Adds a virtual switch to a host. Removes a virtual switch from the host.
	AddVirtualSwitchPortGroup RemoveVirtualSwitchPortGroup	Adds a port group to a virtual switch. Removes a port group from a virtual switch.
	DVSCreate	Creates a distributed virtual switch.
	HostProfileManager	Demonstrates the use of HostProfileManager and ProfileComplainceManager.
	NIOCForDVS	Adds a network resource pool to a distributed virtual switch.
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.
	ColdMigration	Puts virtual machine files into a specified datacenter and datastore and registers and reconfigures the corresponding virtual machine.
performance	Basics	Displays available performance counters or other metadata for an ESX/ESXi 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.
	Realtime	Displays performance measurements from the current time at the console.
	VItop	An ESXtop-like sample application that lets administrators specify the CPU and memory counters by name to obtain metrics for a specified host.
	VIUsage	Creates a GUI for graphical representation of the counters.
scheduling	DeleteOneTimeScheduledTask	Demonstrates deleting a ScheduledTask.
	OneTimeScheduledTask	Demonstrates creating a ScheduledTask using the ScheduledTaskManager.
	WeeklyRecurrenceScheduledTaks	Demonstrates creating a weekly recurrent scheduled task.
scsilun	SCSILunName	Displays the CanonicalName,Vendor, Model, Data, Namespace and NamespaceId of the host's SCSI LUN.

Table E-1. Java Sample Programs (Continued)

Directory	Example		Description
`security`	`credstore`	`Base64`	A fast, memory efficient class that encodes and decodes to and from BASE64 in full accordance with RFC 2045.
		`CredentialStore`	Creates an example credential store.
		`CredentialStoreAdmin`	A command-line tool that provides completenaccess to the credential store backing file on the local machine.
		`CredentialStoreFactory`	Factory class providing instances of a credential store.
		`CredentialStoreImpl`	Implementation class for `CredentialStoreAdmin`.
		`CredentialStoreObfuscate`	Converts the hostname string to lowercase, so that a uniform representation of the hostname is used to obfuscate and de-obfuscate the password.
		`CredentialStoreStorage`	This class provides the same functionality as `FileInputStream`, except that the close() method is overridden so that `FileInputStream` class' close() method does not get called.
`simpleagent`	`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.
`storage`	`CreateStorageDRS`		Creates storage DRS.
	`SDRSRecommendation`		Runs storage DRS on an SDRS cluster to obtain SDRS recommendations.
	`SDRSRules`		Configures rules for an SDRS cluster.
`vApp`	`OVFManagerExportVAPP`		Demonstrates the `OvfManager` by exporting VMDKs and OVF Descriptors of all VM's in the vApps.
	`OVFManagerExportVMDK`		Demonstrates how the `OvfManager` exports VMDKs from a VM to the localSystem.
	`OVFManagerImportLocal`		Use this class to import or deploy an OVF Appliance from a local drive.
	`OVFManagerImportVAppFromUrl`		Use this class to import or deploy an OVF Appliance from a specified URL.
`vim25`	This directory contains the Java classes that define the JAXWS bindings to the vSphere API.		

Table E-1. Java Sample Programs (Continued)

Directory	**Example**	Description
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 reconfigure API. Use delta disks in conjunction with linked virtual machines.
	`VMDiskCreate`	Creates a virtual disk.
	`VMLinkedClone`	Creates a linked virtual machine from an existing snapshot.
	`VMManageCD`	Configures a CDROM for a virtual machine. Also lists information about the CDROMs associated with a virtual machine.
	`VMManageFloppy`	Configures a floppy drive for a virtual machine. Also lists information about the floppy drives associated with a virtual machine.
	`VMotion`	Checks whether migration with `VMotion` 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 `VirtualMachine.PromoteDisks` method.
	`VMReconfig`	Reconfigures a virtual machine. Includes reconfiguring the disk size and disk mode.
	`VMRelocate`	Relocate a linked virtual machine using disk move type.
	`VMSnapshot`	Performs virtual machine snapshot operations

C# Sample Programs

The C# (.NET) sample programs are located in the `SDK\vsphere-ws\dotnet\cs\samples\` directory. Details listed in Table E-2. Each of the examples listed in the table is actually a directory that contains a .cs file, a .csproj file, a *filename*2008.csproj file, and a *filename*2010.csproj file.

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# (.Net) Sample Programs

Example	**Description**
`AddVirtualNic`	Adds a virtual NIC to the ESX/ESXi system. First specifies a `HostVirtualNicSpec`, 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.

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

Example	Description
AppUtil	Contains the following utility applications: ■ AppUtil – Utility application that drives the user input mechanism for other samples and includes some other utility functions. ■ ArgumentHandlingException – Handles command-line exceptions. Used by AppUtil. ■ CertPolicy – Handles certification problems by displaying informational messages. ■ ClientUtil – Client utilities related to prompting the user and logging. Used by AppUtil. ■ Log – Logger to file or console. ■ OptionSpec – Option parsing utility. ■ ServiceUtils, ServiceUtilsV25 – Utilities for connecting to the server. ■ VMUtils – 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 PerformanceManager for basic monitoring.
Browser	Retrieves the contents of the ServiceInstance starting at the root folder, and prints a listing of ManagedEntity objects. Optionally, obtains properties for a specific type, or by default, for ManagedEntity itself.
CIMInfo	CIMInfo versioning sample that retrieves the details of CIM_Fan like Activecooling, Caption, CommunicationStatus 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 Folder, Cluster, Datacenter or standalone host. Prompts the user for the item to create and where to put the item, for example, in a folder.
CreateUser	Creates a user. Specifies permissions for the user using the AuthorizationManager.
CredentialStorePSCmdLets	Multiple commandlets for managing the credential store.
Delete	Deletes a managed entity.
DeleteOneTimeScheduledTask	Extracts a task from scheduledTaskManager and deletes it. This sample is well commented and illustrates using the PropertyCollector. You can create the task using the OneTimeScheduledTask 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 EventHistoryCollector 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,

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

Example	Description
`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 `ScheduledTask` that powers off a virtual machine and schedules the task using a `OnceTaskScheduler`. You can delete the task using the `DeleteOneTimeScheduledTask` commandlet.
`PrintCounters`	Defines a `printEntityCounters` function that prints counters for a virtual machine, host, or resource pool.
`PropertyCollector`	Illustrates use of the `PropertyCollector`.
`QueryMemoryOverhead`	Illustrates use of the `QueryMemoryOverhead` method. The folder includes two examples, `QueryMemoryOverheadV25` uses the currently valid `QueryMemoryOverheadEx` method, `QueryMemoryOverhead` uses the deprecated `QueryMemoryOverhead` 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.
`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 command 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.

Axis 4.1

- 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-3.
- 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-3. 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.
	VMEventHistoryCollectorMonitor	Standalone client that demonstrates how to perform the following tasks: ■ 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.

Table E-3. Java Sample Programs (Continued)

Directory	Example	Description
	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 an 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/ESXi 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.
	Realtime	Displays performance measurements from the current time at the console.
	VItop	An esxtop-like sample application that lets administrators specify the CPU and memory counters by name to obtain metrics for a specified host.
	VIUsage	Creates a GUI for graphical representation of the counters.
scheduling	DeleteOneTimeScheduledTask	Demonstrates deleting a ScheduledTask.
	OneTimeScheduledTask	Demonstrates creating a ScheduledTask using the ScheduledTaskManager.
	WeeklyRecurrenceScheduledTask	Demonstrates creating a weekly recurrent scheduled task.
scsilun	SCSILunName	Displays the CanonicalName,Vendor, Model, Data, Namespace and NamespaceId 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 reconfigure 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 VMotion is feasible between two hosts. Performs the migration if the hosts are compatible.

Table E-3. Java Sample Programs (Continued)

Directory	Example	Description
	VMpowerOps	Performs power operations on a virtual machine.
	VMPromoteDisks	Consolidates a linked virtual machine by using the VirtualMachine.PromoteDisks 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 HostPowerOps sample includes running against Virtual Infrastructure 2.0. The other samples apply to Virtual Infrastructure 2.5 and later versions including vSphere 4.0.	
	displaynewpropertieshost/ DisplayNewPropertiesHost DisplayNewPropertiesHostV25	Displays properties of the host based on the API version supported by ESX/ESXi or vCenter Server.
	displaynewpropertiesvm/ DisplayNewPropertiesVM DisplayNewPropertiesVMV25	Displays properties of a virtual machine based on the API version supported by the host to which you connect.
	getvirtualdiskfiles/ GetVirtualDiskFiles GetVirtualDiskFilesV25	Searches the virtual disk files in all the datastores available in a specified host using the properties ControllerType and ThinProperty.
	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 HostPatchManager 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.

Index