vSphere Web Services SDK Programming Guide

vSphere Web Services SDK 4.0

EN-000145-00

vmware

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

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

The VMware Web site also provides the latest product updates.

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

docfeedback@vmware.com

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

Contents

About This Book 9

1 Introduction to the Virtual Datacenter 11
- Virtualization and VMware vSphere Components 11
- Summary of New Features for Developers Using the SDK 12
- New Managed Entities 13
- Support for Previous APIs in vSphere 4.0 13
- Interoperability of SDK 2.5 Clients and vSphere 4.0 Servers 13
  - Client Libraries and Existing SDK 2.5 Client Applications 14
  - Modifying VI SDK 2.5-based Client Applications to use vSphere API 4.0 Features 14

2 vSphere Object Model 15
- Using the VMware vSphere Client to Explore the Object Model 15
- Using the Managed Object Browser to Explore the Object Model 16
  - Accessing the MOB 17
  - Navigating the Object Model in the MOB 17
  - Understanding the ServiceContent Data Object 18
- Understanding the Object Model as Data Structures 21
  - Understanding the Unified Modeling Language Used in this Guide 21
  - The Managed Object Type as Data Structure and Service Interface 22
    - ServiceInstance Managed Object as Starting Point to Other Service Interfaces 23
    - References to Managed Objects 24

3 Understanding the Inventory 25
- Overview of Managed Entities 25
- Understanding Container Objects and the Inventory 26
- Folder as Container 27
- Inventory Objects and Privileges 28
- Folder as Factory 30
- vCenter Server and Unmanaged ESX/ESXi Host Inventories 30

4 Overview of User Models and Server Access Control Concepts 33
- Introduction to Authentication and Authorization 33
  - Understanding System Privileges 34
  - Privileges Defined in the vSphere API Reference 34
  - HTTP-Based File Access and Privileges 34
- Overview of ESX/ESXi User Model 35
- Using HostLocalAccountManager Managed Object 35
- Overview of vCenter Server User Model 36
- Enabling Single Sign-on Support and Integration with Microsoft Active Directory 37
- Obtaining Information about Users and Groups from UserDirectory 37
- Authenticating Users through SessionManager 39
- Overview of the AuthorizationManager 39
  - Using Roles to Consolidate Sets of Privileges 40
  - Modifying Sample Roles to Create New Roles 42

Granting Privileges through Permissions 42
Obtaining Information about Permissions 43
Setting, Changing, or Deleting Permissions 43
Impact of Group Membership on Permissions 44

5 vSphere API Programming Model 45
API as a Web Service 45
API Defined in WSDL File 45
Client Application Characteristics 46
Asynchronous Client-Server Application Style 46
Distributed Object-Oriented Application Style 46
Obtaining Managed Object References 49
Do Not Store Managed Object Reference Values 49
Working with Data Structures in Client Application Code 49
Accessing Property Values 49
Accessing the API on the Web Service 50
Using the vSphere API Reference 51
Identifying Deprecated Items in the vSphere API Reference 51
Identifying Version Information in the vSphere API Reference 51
Mapping the WSDL to the Client-side Proxy Code 51
Obtaining Values from Nested Properties 52
Mapping XML Datatypes to Java and C# Datatypes 53
Escape Character Used in Name and Path Properties 53
Invoking Operations through the MOB 53
Passing Primitive Datatypes to Operations through the MOB 54
Passing Arrays of Primitives to Operations through the MOB 54
Passing Complex Structures to Operations through the MOB 54

6 Client Application Pattern 59
Minimal Client Application Pattern 59
Connecting to the Web Service 59
Obtaining a Session Token and Creating a Connection Object 59
Logging In to the Server 60
Close the Server Connection 60
Overview of a Java Sample Application 61
Saving and Reusing a Web Server Session Token or Cookie 63
Supporting Multiple API Versions 63
Identifying the API Version Supported by the Server 64
Helper Classes for Sample Applications 64
Java Helper Classes 65
C# Helper Classes 65

7 Automating Client Application Login 67
Introduction to the Credential Store 67
Overview of the Credential Store Backing File 68
Credential Store Backing File Structure 68
Using Samples to Understand the Credential Store 69
Security Best Practices and the Credential Store 69
Using the Principle of Least Privilege with the Credential Store 69
Using the CredentialStoreAdmin Tool 70
Using AuthorizationManager to Limit Access 70
Creating New User Accounts 70
Applying Permission to an Entity 71

8 Obtaining References to Objects and Property Values from the Server 73
Sample Code Reference 73
Introduction to the PropertyCollector 74
PropertyCollector Operations 74
Understanding the PropertyFilterSpec 75
TraversalSpec Data Object and Recursion 76
RetrieveProperties Operation 76
CreateFilter and Update Operations 77
Keeping Client Data in Synch with Server Objects 78
PropertyCollector Performance 79
Improving Collection Performance by Using the ViewManager and View Objects 79
Using the SearchIndex to Obtain Managed Object References 80

9 Overview of the Task Infrastructure 83
Sample Code Reference 83
Understanding TaskManager and Task Managed Objects 84
Understanding the TaskInfo Data Object 85
Monitoring TaskInfo Properties 88
Cancelling a Task 88
Using the TaskManager to Obtain Information about Tasks 88
Obtaining Information about Recent Tasks Using a PropertyCollector 89
Operations That Return Task Managed Objects 90
Introduction to the Event Data Object 92

10 Working with Virtual Machines 93
Sample Code Reference 93
Creating Virtual Machines 93
Overview of Virtual Machine Operations 94
Overview of Snapshots 97
Creating Snapshots 97
Reverting to Snapshots 97

11 Managing Storage 99
Sample Code Reference 99
Overview of Physical and Virtual Storage Concepts 99
Requirements for Storage 100
Overview of Internet SCSI Storage 101
Introduction to Managed Objects that Support Storage 101
Introduction to the HostStorageSystem Managed Object 102
Introduction to the HostDatastoreSystem Managed Object 106
Obtaining Information about Configured Storage 107
Creating New Datastores 108
Modifying Datastore Configuration 110

12 Monitoring Performance 111
Sample Code Reference 111
Introduction to PerformanceManager 112
Overview of Counter Groups and the PerfCounterInfo Data Object 112
Sampling Periods and Intervals 114
Obtaining Statistics 115

vCenter Server and PerformanceManager 117
Historical Intervals 117
Modifying Historical Intervals 117
Optimizing Query Response Time 118

13 Using vCenter Server for Virtual Machine Operations 119
Sample Code Reference 120
Overview of the VirtualMachine Managed Object Type 120
Using Templates to Deploy Standard Virtual Machines 124
Creating Templates 124
Reverting a Template to Its Active Virtual Machine Status 125
Migrating Virtual Machines Using VMotion 125
Checking the Validity of a Migration In Advance 126
Using the MigrateVM_Task Operation 126
Moving Storage Dynamically Using Storage VMotion 127

14 Event and Task Management Using vCenter Server 129
Sample Code Reference 129
Understanding EventManager and the Event Data Object 129
EventManager Managed Object 130
Event Data Objects 130
Formatting Event Message Content 131
Creating Custom Events 132
Obtaining Information about Events Using a HistoryCollector 133
Overview of HistoryCollector Managed Objects 133
New Objects Appended to a HistoryCollector 135
Deleting a HistoryCollector 135

15 Using the Alarm Infrastructure 137
Sample Code Reference 137
Overview of AlarmManager and Alarm Managed Objects 137
Obtaining a List of Alarms 138
Defining an Alarm 138
Overview of the AlarmSpec Data Object 139
Overview of the AlarmExpression Data Object 139
Overview of the AlarmAction Data Object 141
Deleting or Disabling an Alarm 142

16 Scheduling vCenter Server Operations 143
Sample Code Reference 143
ScheduledTaskManager Managed Object 143
Retrieving ScheduledTasks for a Specific Managed Entity 144
Creating a ScheduledTask Object 144
Defining the Schedule for the vCenter Server Operation 145
Defining the Actions 146
Scheduling Recurring Operations 146
Deleting a Scheduled Task 148

A Managed Object Privileges Reference 149
Privileges Required to Invoke Operations 149
Privileges Required to Read Properties 156
Privileges Defined for the Administrator Role 158

B Diagnostics and Troubleshooting 161
Best Practices for Troubleshooting 161
Overview of Configuration Files and Log Files 162
ESX Log File 163
Virtual Machine Log Files 163
vCenter Server Log Files 164
Modifying the Log Level to Obtain Detailed Information 164
Log Level Settings 165
Setting the Log Level on ESX Systems 165
Generating Logs 166
Setting the Log Level on vCenter Server System 167
Understanding and Using the DiagnosticManager 167
Using the MOB to Explore the DiagnosticManager 169
Generating Diagnostic Bundles 171

C Deploying Client Applications 173
Understanding the Client Application Deployment Model 173
Credential Store Administration Tool for Non-Interactive Clients 173
Guidelines for Using the CredentialStoreAdmin Tool 173
CredentialStoreAdmin Tool and the Backing File 174

Index 177

About This Book

The *vSphere Web Services SDK Programming Guide,* provides information about developing applications using the VMware® vSphere Web Services SDK 4.0.

VMware provides several different APIs and SDKs for various applications and goals. This book provides information about using the vSphere Web Services SDK for developers that are interested in creating client applications for managing VMware® vSphere components available on VMware ESX, VMware ESXi, and VMware vCenter Server systems.

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

Revision History

This guide is revised with each release of the product or when necessary. A revised version can contain minor or major changes. Table 1 summarizes the significant changes in each version of this guide.

Table 1. Revision History

Revision	Description
20090507	First version of the vSphere Web Services SDK 4.0 Programming Guide.

Intended Audience

This book is intended for anyone who needs to develop applications using the VMware vSphere Web Services SDK. vSphere Web Services SDK developers typically include software developers creating client applications using Java or C# (in the Microsoft .NET environment) targeting VMware vSphere.

Document Feedback

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

Technical Support and Education Resources

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

Online and Telephone Support

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

Support Offerings

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

VMware Professional Services

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

1 Introduction to the Virtual Datacenter

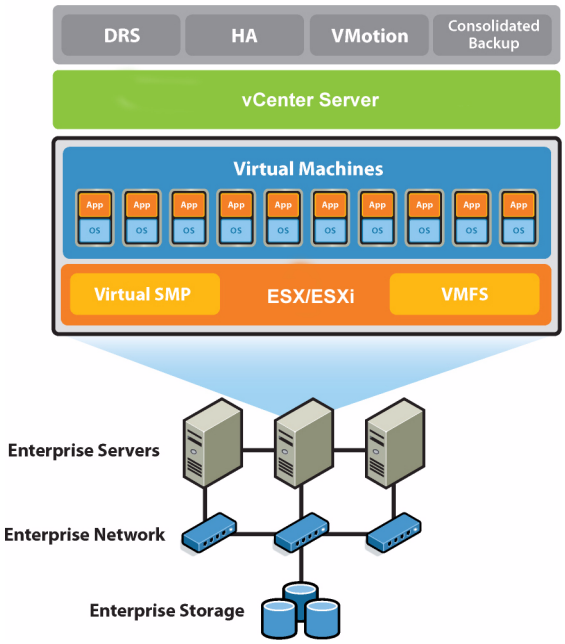
The VMware vSphere Web Services SDK gives Web services developers programmatic access to the virtual datacenter components. VMware vSphere provides a complete platform to support robust, fault-tolerant virtualized applications, networking, storage, and all other datacenter resources.

- "Virtualization and VMware vSphere Components" on page 11
- "Summary of New Features for Developers Using the SDK" on page 12
- "Support for Previous APIs in vSphere 4.0" on page 13

Virtualization and VMware vSphere Components

VMware software products virtualize computing resources, including CPU, memory, storage, and networks, for example, so that physical resources can deliver optimal value. Figure 1-1 shows some of the key VMware vSphere in the context of physical components, such as enterprise servers, networks, and storage.

Figure 1-1. VMware vSphere Components



Virtualization provides an abstraction layer between computing resources, physical storage, and networking hardware, and the applications that use all of these resources. For example, virtual memory enables software to use more memory than is physically installed in a computer by swapping data from physical memory to physical disk.

With its VMware vSphere product suite, VMware virtualizes all layers of an IT infrastructure, from networks, storage, laptop and desktop computers, to server hardware, operating systems, and applications.

VMware vSphere includes ESX/ESXi, vCenter Server, and several additional server products for distributed resource management (DRS), disaster recovery, and high availability (HA) (Figure 1-1).

- ESX/ESXi provides the foundation hypervisor capable of supporting multiple virtual machines and other virtual components, such as storage and networks.
- vCenter Server is a database-backed Windows service that provides central management for multiple ESX/ESXi systems.

If you are new to VMware vSphere or new to the VMware vSphere Web Services SDK, see one of these guides for more background information about vSphere:

- *Introduction to VMware vSphere*
- *vSphere Basic System Administration*
- *ESX Configuration Guide* and *ESXi Configuration Guide*
- *vSphere Resource Management Guide*

For more information about ESX/ESXi and vCenter Server, see the VMware vSphere documentation page on the VMware website.

Summary of New Features for Developers Using the SDK

This section lists just some of the highlights of the vSphere 4.0, with a focus on changes to the API and the object model. For all new managed object, data object, and other types, see the *vSphere API Reference*.

Rapid, Compliant Host Provisioning Using Profiles

Configure hosts quickly using profiles that comply with specific configuration policies. Several new managed objects and associated data objects support provisioning new hosts using profiles. For example, you can use a new service interface, the `HostProfileManager`, to define configurations and policies. With profiles, you configure settings for the environment once and use the profile to provision new hosts. Use profiles to configure networking, storage settings, security, and other settings across any number of hosts.

Distributed Virtual Switch

VMware Distributed Virtual Switch (DVS) provides next-generation virtual networking management that spans the entire virtual infrastructure. It also enables you to quickly increase networking capacity and reduce ongoing network maintenance tasks, by simplifying monitoring and troubleshooting activities.

In addition, DVS supports the latest in VMotion technology, so you can build complete VMotion-aware networking applications on top of vSphere. Network VMotion tracks the networking state of a virtual machine as it migrates throughout the virtual infrastructure.

See the *vSphere API Reference* for complete details about the new managed object types that support DVS, starting with the `DistributedVirtualSwitch` and `DistributedVirtualSwitchManager`.

More Granular Permissions Model for Delegated Administration

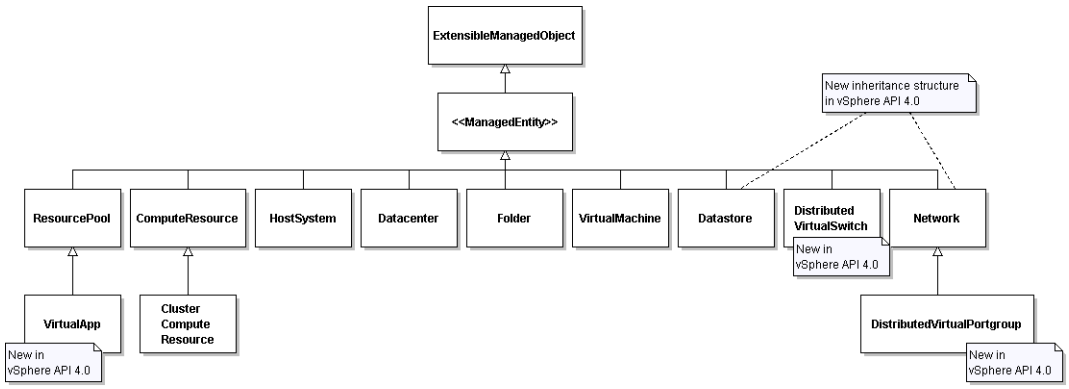
vCenter Server now supports a more discrete permissions model for datastores and networks. These entities can be granted permissions distinct from the datacenter.

Access to a datastore or network can be granted to or denied for a specific set of users or groups. This feature is supported in the object model by elevating the `Datastore` and `Network` managed objects in the hierarchy. These objects are now direct descendents of the `ManagedEntity` base class and can support permissions independently of `Datacenter` (Figure 1-2).

New Managed Entities

Many new features are implemented as one or more new managed objects. Those available through the inventory as managed entities are shown in Figure 1-2.

Figure 1-2. ManagedEntity and Its Descendents



The `VirtualApp`, `DistributedVirtualSwitch`, and `DistributedVirtualPortgroup` are new managed entities with many associated data objects, fault types, and enumerations to support them. See the *vSphere API Reference* for details.

Virtualized Application Services

A Virtualized Application Service (`VirtualApp`) is a container for one or more virtual machines. A `VirtualApp` allows an application to be managed as a unit, including power operations, clone and deploy operations, resource allocation, and monitoring. vCenter Server supports creating, running, importing, and exporting virtualized applications as files using the industry-standard Open Virtualization Format (OVF) 1.0.

Support for Previous APIs in vSphere 4.0

You can use the VMware vSphere Web Services SDK 4.0 with all vSphere server products, including ESX 4, ESXi 4, vCenter Server 4, ESX 3, ESXi 3, ESX Server 3.0.x, VirtualCenter Server 2.5, VirtualCenter Server 2.0.x. Both ESX/ESXi and vCenter Server provide a new version of the Web-services-based API, vSphere API 4.0, which is a superset of all APIs released since VI API 2.5.

Unlike prior releases of the API, the vSphere API 4.0 has no dependency on the WSDL namespace attribute for version information. Targeting multiple versions of the API in prior releases required the client application to examine the WSDL file on the server and obtain the value of the namespace annotation.

With vSphere API 4.0, all SOAP messages from the client to the server include the version information using a previously unused attribute in the SOAP header. These details are transparent to developers.

Because the versioning model has changed and no longer depends on the WSDL namespace attribute to identify the version of the API, the namespace for vSphere 4.0 API remains urn:vim25. In the vSphere Web Services SDK 4.0, the two subdirectories for the WSDL files are as follows:

- vim25, which is the WSDL for VI API 2.5, and subsequent releases, including vSphere API 4.0
- vim, which is the WSDL for VI API 2.0 and prior releases

Interoperability of SDK 2.5 Clients and vSphere 4.0 Servers

As VMware evolves the API to support new capabilities, new parameters for existing operations and new properties for existing data objects are always defined as optional so that clients that were developed with one version of the API can continue to work with new releases of the vSphere Infrastructure server products. The result of this approach is that client applications developed using the VI SDK 2.5 can work unchanged with ESX 4.0, ESXi 4.0, or vCenter Server 4.0 servers. However, modifying an existing VI SDK 2.5-based client has some implications, detailed in "Modifying VI SDK 2.5-based Client Applications to use vSphere API 4.0 Features."

Client Libraries and Existing SDK 2.5 Client Applications

Client applications that were developed using the VI SDK 2.5 can work with vSphere 4.0 servers as long the applications continue to use the client libraries from the VI SDK 2.5 package. Do not use the vSphere 4.0 client libraries with client applications that were created using VI SDK 2.5.

Use the libraries from the vSphere SDK 4.0 with VI SDK 2.5-based applications only if you are modifying such an application to use new features in the vSphere API 4.0. In addition, you might need to modify some of your existing code, as detailed in the next section.

Modifying VI SDK 2.5-based Client Applications to use vSphere API 4.0 Features

To modify an SDK 2.5 client application to incorporate new API 4.0 functionality, you must use the client libraries (`vim25.jar`) from the VMware Web Services SDK 4.0 package.

In addition, you must check your existing code for operations and data objects that might contain new parameters and properties in vSphere API 4.0. Many of the data objects available in previous versions of the API have new properties in vSphere API 4.0. For example, the `Event` data object has several new properties to support managed object references to new managed entities associated with an `Event`, such as the `DistributedVirtualSwitch`.

In addition, many operations have new parameters in this release. For example, the operations listed in Table 1-1 were available in VI API 2.5 and have new parameters in vSphere API 4.0.

Table 1-1. Operations with New Parameters in vSphere API 4.0

Operation	Operation
addHost_Task	queryVmfsDatastoreExtendOptions
addStandaloneHost_Task	relocateVM_Task
createTask	revertToCurrentSnapshot_Task
extendVirtualDisk_Task	revertToSnapshot_Task
findByUuid	updateInternetScsiAuthenticationProperties

If your existing VI SDK 2.5-based client application uses any of the operations listed in the table, you must modify the code to pass `null` for the 4.0-specific parameter, as shown in the code snippet in Table 1-2.

Table 1-2. Example of Modified Code for New Parameter

Original Code	Modified Code
...revertToSnapshot_Task(snapmor,null)	...revertToSnapshot_Task(snapmor,null, null)

See the *vSphere API Reference* for complete information about these operations.

vSphere Object Model 2

As with physical IT infrastructure components, virtual infrastructure components must be provisioned, deployed, monitored, and managed. The VMware vSphere object model is a comprehensive set of robust server-side composite objects that provides the complete management capabilities.

The object model includes service interfaces for managing, monitoring, configuring, obtaining information, and controlling life-cycle operations associated with virtual infrastructure, and object types that instantiate the virtual components on the server at runtime.

External client applications access this management framework, the VMware vSphere object model, through the vSphere API, which is available through the Web service on ESX, ESXi, and vCenter Server systems. Client applications, both third-party and VMware client applications such as VMware vSphere Client, use the vSphere API to perform all life-cycle, management, and monitoring operations.

Understanding the vSphere management object model is key to using the VMware vSphere API for successful client-application development and integration. The chapter includes these topics:

- "Using the VMware vSphere Client to Explore the Object Model" on page 15
- "Using the Managed Object Browser to Explore the Object Model" on page 16
- "Understanding the Object Model as Data Structures" on page 21

Using the VMware vSphere Client to Explore the Object Model

The VMware vSphere Client is a graphical user-interface tool that connects to vCenter Server and ESX/ESXi systems for administering and managing virtual infrastructure components. Developers can use the vSphere Client as a tool for exploring the object model.

The vSphere Client renders virtual components as a hierarchical tree in the left pane of the display (Figure 2-2). When connected to a vCenter Server, the vSphere Client displays all the options available to the VMware vSphere environment, based on the configured licensing and the permissions of the user. When connected to an ESX/ESXi host, the vSphere Client displays options appropriate for single host management.

The vSphere Client layout is a single window with a menu bar, a navigation bar, a toolbar, a status bar, and a panel section. The left pane shows the inventory of objects on the server (vCenter Server, ESX, or ESXi).

See *Managing the vSphere Client Inventory—Understanding vSphere Client Objects* in the VMware vSphere online library for more information about how the vSphere Client graphically depicts server-side objects.

The VMware vSphere management framework uses the concept of an inventory to keep track of virtual infrastructure assets, including datacenters, hosts, virtual machines, storage, network switches, and so on. A VMware vSphere server inventory includes both container objects, such as folders, and non-container objects, such as virtual machines. See "Understanding the Inventory" on page 25 for more information.

The root `Folder` object and the default `Datacenter` object are not displayed in the vSphere Client (Figure 2-1), but these do exist. You can see them in the Managed Object Browser, for example. Every `ServiceInstance` has a `rootFolder` property consisting of the root `Folder` object, which has a reference to the default `Datacenter`.

Figure 2-1. vSphere Client Connected to an ESX/ESXi System

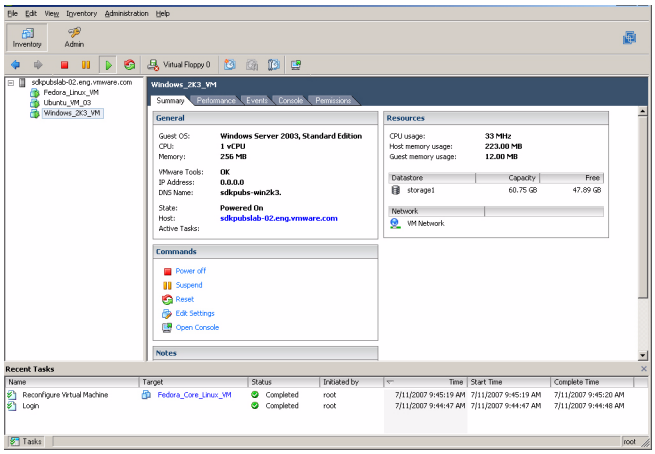
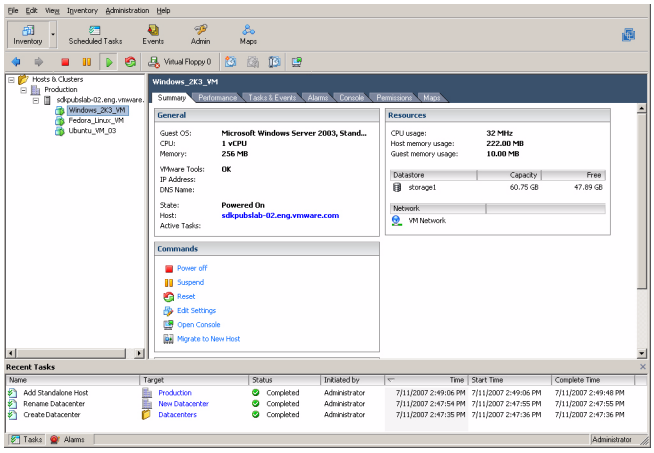


Figure 2-1 and Figure 2-2 highlight some of the differences between vCenter Server and ESX/ESXi. Figure 2-1 shows how the inventory of an ESX/ESXi system displays, in the left pane.

In terms of management services, vCenter Server provides additional service capabilities and administrative functions. For example, vCenter Server supports Scheduled Tasks, Events, and Maps (note these icons in the menu bar of Figure 2-2), while ESX or ESXi does not (see the menu bar of the ESX instance shown in Figure 2-2).

Figure 2-2. vSphere Client Connected to a vCenter Server System



The inventory as displayed in the vSphere Client is rendered from the content of the server-side objects at any given point in time. The inventory shown in Figure 2-2 and Figure 2-1 consists of a single datacenter with three different virtual machines. These same objects and services can be seen, rendered as HTML pages, in the Managed Object Browser. The Managed Object Browser provides a lower-level view of the actual server side objects that make up the components that display in the vSphere Client.

Using the Managed Object Browser to Explore the Object Model

The Managed Object Browser, or MOB, is a web-based server application available on all ESX/ESXi and vCenter Server systems. Use the MOB to look at the objects that exist on the server and navigate through the hierarchy of instantiated objects by clicking on hyperlinks that are populated with actual runtime information. You can view the names of properties, for example.

CAUTION Despite the word browser in its name, the MOB is not a read-only mechanism. The MOB is a powerful tool that can be used to make changes to the server, by clicking the **InvokeMethod** link associated with methods available on any managed object page. Whenever you explore a VMware vSphere server using the MOB, be aware of its capabilities so that you don't make unintended changes to the server.

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.

To access the managed object browser (MOB)

1 Launch a Web browser, such as Firefox or Internet Explorer.

2 Enter the fully qualified domain name (or the IP address) for the ESX/ESXi host or vCenter Server system:

```
https://<hostname.yourcompany.com>/mob
```

You'll be notified that the object browser requires a user name and password, with a prompt requiring a user name and password.

3 Enter the user account and password for the server system. Typically, this is root/<password> for the ESX/ESXi system, and Administrator/<password> for vCenter Server system.

Some preliminary warning messages regarding the authority of the SSL certificate, such as `Website Certified by an Unknown Authority` might display assuming that the default server certificate provided by VMware has not been replaced. The specific message text varies by web browser.

Disregard such warnings and continue to log in to the MOB, as long as VMware is the certificate authority.

IMPORTANT If the ESX/ESXi or vCenter Server system has been configured to support regular HTTP (non-SSL) connections, a prompt for user name and password does not display, nor do any SSL-certificate-related warnings.

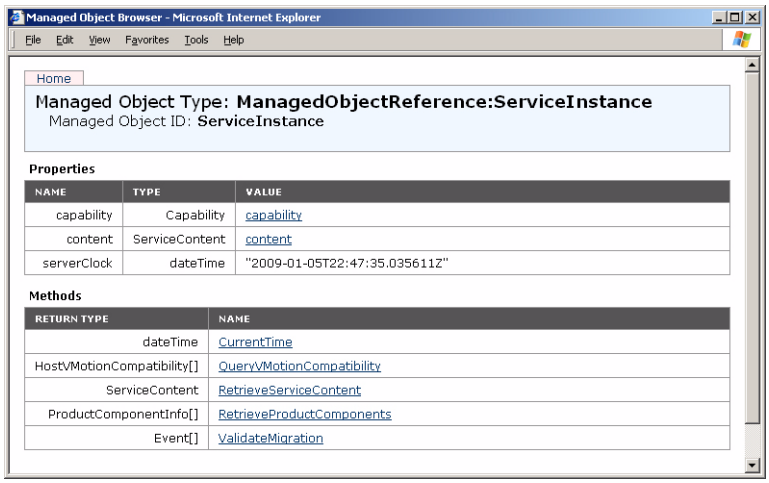
Unlike with the vSphere Client, the MOB reveals the underlying structures of the VMware vSphere management object model, which can be very useful to developers trying to understand the object model. Used in conjunction with the vSphere API Reference, the MOB can help you gain a fairly complete understanding of the model.

Navigating the Object Model in the MOB

Upon successful connection to the MOB, the browser displays the managed object reference for the service instance (Figure 2-3). Client applications do not use managed objects directly. Rather, client applications interact with server-side managed objects by reference, using instances of the `ManagedObjectReference` data object, the data structure created for this purpose. Invoking operations on the server typically requires one or more references to a managed object.

For example, the title of the page shown in Figure 2-3 identifies the object as a managed object reference to the `ServiceInstance`. The page also lists the properties and methods available through `ServiceInstance`.

Figure 2-3. Managed Object Browser Connected to a vCenter Server System



Although ServiceInstance has only a few methods and properties (Table 2-1), these ultimately provide access to the entire set of services and inventory objects available on the server.

Client applications must always obtain a reference to the server's ServiceInstance object from the appropriate VMware vSphere (ESX, ESXi, or vCenter Server) target.

Table 2-1. ServiceInstance Properties

Property	Datatype	Description
capability	Capability data object	An instance of a Capability data object. A Capability data object consists of several flags that indicate support or lack of support for features on the server instance. For example, multiHostSupported capability is true for vCenter Server but false for ESX. A provisioningSupported flag that returns "true" enables actions, such as cloning a virtual machine. Some capabilities are version-specific.
content	ServiceContent data object	An instance of a ServiceContent data object. A ServiceContent data object contains references (managed object references) to all the service interfaces available to the specific ServiceInstance, such as the session manager (sessionManager property) and the root folder of the inventory (rootFolder property).
serverClock	xsd:dateTime	Server date and time.

You can use the MOB to directly examine the relationships among objects by looking at the properties and their values and then clicking the link, if one is available, in the value column of the page. For example, to find out more about ServiceContent, click on the **content** link to display the ServiceContent data object instance (Figure 2-4).

From the ServiceContent page, you can click on the links in the Value column to display the contents of the associated objects. For example, click on the **about** link to display the current value of the about property, which displays the content of the AboutInfo data object for this server. However, most objects contain many other objects and nested objects.

Access to the inventory begins from the rootFolder property of ServiceContent. (The inventory is covered in more detail in Chapter 3, "Understanding the Inventory," on page 25.)

Understanding the ServiceContent Data Object

In simple terms, ServiceContent is a large-grained server object that provides access to the service interfaces and the inventory supported by the server. ServiceContent is a data object whose properties include many managed object references (instances of ManagedObjectReference) that point to specific instances of managed objects (Figure 2-4).

For example, the rootFolder property is a ManagedObjectReference to an instance of a Folder managed object. The perfManager is a ManagedObjectReference to a specific instance of a PerformanceManager managed object, and so on.

Figure 2-4 shows a partial `ServiceContent` data object displayed in the MOB of a vCenter Server 4.0 system.

Figure 2-4. ServiceContent Data Object Contains References to Instances of Managed Objects

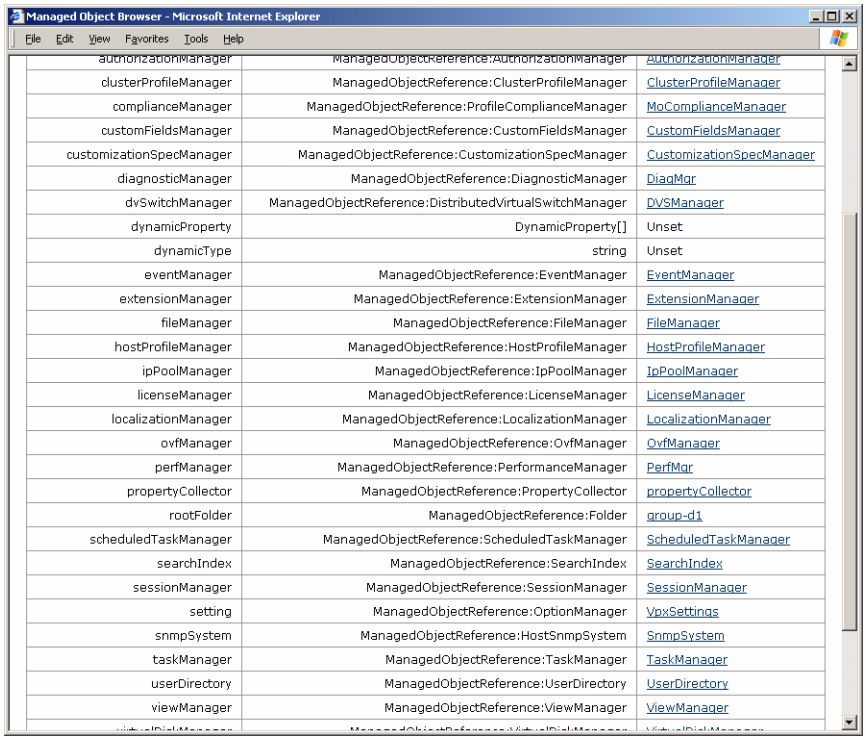


Table 2-2 summarizes the `ServiceContent` property names and values for an example ESX and vCenter Server system, side-by-side, as displayed in the MOB. The example values listed in Table 2-2 are the instance names for the objects. The properties that display an `Unset` value indicate that the managed object type is not available on the server.

For example, the property for `accountManager` which has a reference to a singleton `HostLocalAccountManager` object on ESX systems has no value on vCenter Server systems, because vCenter Server uses Windows user-account management facilities (such as Active Directory).

Conversely, the property for `alarmManager` on ESX is unset, because alarms are not supported on an unmanaged host. The `Alarm` infrastructure is managed by vCenter Server. Many other managed object types require vCenter Server, including these:

- `ClusterProfileManager`
- `DistributedVirtualSwitchManager`
- `ExtensionManager`
- `HostProfileManager`
- `HostSnmpSystem`
- `IpPoolManager`
- `LocalizationManager`
- `OvfManager`
- `ProfileComplianceManager`
- `ScheduledTaskManager`
- `VirtualMachineCompatibilityChecker`
- `VirtualMachineProvisioningChecker`

Table 2-2. ServiceContent Property Names, Return Types, and Values

		Example values	
Property name	**Type**	**ESX**	**vCenter Server**
about	AboutInfo	about	about
accountManager	ManagedObjectReference: HostLocalAccountManager	ha-localacctmgr	Unset
alarmManager	ManagedObjectReference: AlarmManager	Unset	AlarmManager
authorizationManager	ManagedObjectReference: AuthorizationManager	ha-authmgr	AuthorizationManager
clusterProfileManager	ManagedObjectReference: ClusterProfileManager	Unset	ClusterProfileManager
complianceManager	ManagedObjectReference: ProfileComplianceManager	Unset	MoComplianceManager
customFieldsManager	ManagedObjectReference: CustomFieldsManager	Unset	CustomFieldsManager
customizationSpecManager	ManagedObjectReference: CustomizationSpecManager	Unset	CustomizationSpecManager
diagnosticManager	ManagedObjectReference: DiagnosticManager	ha-diagnosticmgr	DiagMgr
dvSwitchManager	ManagedObjectReference: DistributedVirtualSwitchManager	Unset	DVSManager
eventManager	ManagedObjectReference: EventManager	ha-eventmgr	EventManager
extensionManager	ManagedObjectReference: ExtensionManager	Unset	ExtensionManager
fileManager	ManagedObjectReference: FileManager	ha-nfc-file-manager	FileManager
hostProfileManager	ManagedObjectReference: HostProfileManager	Unset	HostProfileManager
ipPoolManager	ManagedObjectReference: IpPoolManager	Unset	IpPoolManager
licenseManager	ManagedObjectReference: LicenseManager	ha-license-manager	LicenseManager
localizationManager	ManagedObjectReference: LocalizationManager	Unset	LocalizationManager
ovfManager	ManagedObjectReference: OvfManager	Unset	OvfManager
perfManager	ManagedObjectReference: PerformanceManager	ha-perfmgr	PerfMgr
propertyCollector	ManagedObjectReference: PropertyCollector	ha-property-collector	propertyCollector
rootFolder	ManagedObjectReference: Folder	ha-folder-root	group-d1
scheduledTaskManager	ManagedObjectReference: ScheduledTaskManager	Unset	ScheduledTaskManager
searchIndex	ManagedObjectReference: SearchIndex	ha-searchindex	SearchIndex
sessionManager	ManagedObjectReference: SessionManager	ha-sessionmgr	SessionManager

Table 2-2. ServiceContent Property Names, Return Types, and Values (Continued)

		Example values	
Property name	**Type**	**ESX**	**vCenter Server**
setting	ManagedObjectReference: OptionManager	HostAgentSettings	VpxSettings
snmpSystem	ManagedObjectReference: HostSnmpSystem	Unset	SnmpSystem
taskManager	ManagedObjectReference: TaskManager	ha-taskmgr	TaskManager
userDirectory	ManagedObjectReference: UserDirectory	ha-user-directory	UserDirectory
viewManager	ManagedObjectReference: ViewManager	ViewManager	ViewManager
virtualDiskManager	ManagedObjectReference: VirtualDiskManager	ha-vdiskmanager	VirtualDiskManager
vmCompatibilityChecker	ManagedObjectReference: VirtualMachineCompatibilityChecker	Unset	CompatChecker
vmProvisioningChecker	ManagedObjectReference: VirtualMachineProvisioningChecker	Unset	ProvChecker

Understanding the Object Model as Data Structures

The vSphere object model can also be viewed from a data structures perspective. The VMware vSphere management object model comprises two broad categories of data structures:

- Managed object types. A managed object type is a data structure that contains both properties and operations. The operations support specific, expected behaviors that adhere to a contract. For example, a `VirtualMachine` managed object type supports virtual machine power-off and power-on, recording all virtual machine operations, and many operations that control the guest operating system running on a virtual machine, such as rebooting the guest. These are just some examples: the `VirtualMachine` managed object type provides close to 50 operations. See Chapter 10, "Working with Virtual Machines," on page 93 and Chapter 13, "Using vCenter Server for Virtual Machine Operations," on page 119.
- Data object types. A data object is a data structure that contains properties only. Instances of data objects do not provide any operations. Data objects are analogous to structures (struct) in C, C++, and several other programming languages, or abstract data types (class definition) in Java. Java does not use the concept of a struct, but rather uses class declarations to define abstract data types.

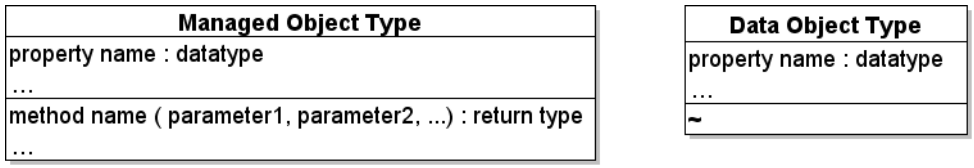
The properties of the managed object types are defined as primitive data types, such as string or int, or as specific types of data objects. Likewise, properties of data objects are defined as primitive values or as specific instances of other data objects.

NOTE Although many managed object types have words like system or manager in their names, some data object types do as well. For example, the `HostConfigManager` is a data object type. Always check the *vSphere API Reference* for details about data object and managed object types.

Understanding the Unified Modeling Language Used in this Guide

Figure 2-5 shows the unified modeling language (UML) notation used to depict the two primary types of data structures represented in this guide: managed object types, and data object types. The diagrams use a tilde to denote absence of any properties or methods (operations). Ellipses are used to denote additional content deliberately not shown.

Figure 2-5. Legend for UML Class Diagrams



NOTE The figures shown in this guide highlight various characteristics of the object model using UML diagrams. For the sake of simplicity, these diagrams do not include complete details. For example, all data object types extend the `DynamicData` data object. However, the base properties of `DynamicData` are rarely relevant for the topic, so that object relationship is typically not shown.

In addition to class diagrams, this guide also includes several instance diagrams, especially in those sections that discuss navigating the inventory, which is best explained in the context of actual object instances. Many details are left out of the instance diagrams as well.

This guide uses these terms and abbreviations in figures and text.

- MOR. Managed object reference. An opaque data object that is used as a remote reference to managed objects. Managed object references are also called *references* in text, and `mo_ref` or `MoRef` in code samples.
- MOR>ManagedObject Type. A managed object reference identifies a specific instance of a specific type of managed object, on a VMware vSphere server. The abbreviation MOR>ManagedObject specifies the type of managed object instance to which a given MOR refers. This notation is used throughout this guide to replace the words "to a," as in "MOR to a VirtualMachine," or "MOR to a HostDatastoreSystem."

The Managed Object Type as Data Structure and Service Interface

A managed object type is a core data structure of the server-side object model. It is any one of about 60 server-side data structures having properties and operations available on the server. Different managed objects offer different operations. An operation is Web-services terminology for what might be called a method in another programming language, such as Java. The words method and operation are used interchangeably in the documentation.

Instances of managed object types are often referred to generically as "managed objects" throughout this guide. However, managed objects include the following broad categories:

- Managed objects that extend the `ManagedEntity` managed object type. These managed object types are referred to generically as "managed entities." Managed entities make-up the *inventory* of virtual components. For example, instances of host systems (`HostSystem`), virtual machines (`VirtualMachine`), and datastores (`Datastore`) are inventoried objects and extend the managed entity (`ManagedEntity`) type. Inventoried objects display in the left pane of the vSphere Client.
- Managed objects that comprise the interface to a vSphere management service. For example, `PerformanceManager` provides the interface for monitoring system performance. `DistributedVirtualSwitchManager` provides the interface for managing `DistributedVirtualSwitch` managed objects. `AlarmManager` is the service interface for setting alarms to respond to various events or state changes on specific managed entities.
- Managed objects whose life-cycle depends upon or exists solely in the context of another managed object type or service interface. For example, instances of `Task` managed objects are returned by method invocations on most all managed objects in the system. `PropertyFilter` is a managed object type associated with an instance of `PropertyCollector`. `HostNetworkSystem` managed objects are associated with specific instances of `HostSystem` managed objects. These are just a few examples.

ServiceInstance Managed Object as Starting Point to Other Service Interfaces

The chief managed object in the "service interface" category is `ServiceInstance`, which is the entry point for all other service interfaces and inventory objects. All interaction with the server begins through `ServiceInstance`. A `ServiceInstance` has only a few properties and methods. The content property is defined as an instance of `ServiceContent`, which is a data object containing references to all available service interfaces on the server instance.

Figure 2-6 highlights the relationship between `ServiceInstance` and `ServiceContent`, and the relationship between `ServiceContent` and some of the managed object references (MOR) to management services. Some services, such as `AlarmManager`, are supported on vCenter Server only, not on unmanaged hosts. Other services, such as `HostLocalAccountManager`, are supported on ESX or ESXi systems only, not on vCenter Server. Figure 2-6 is not the entire set of services available. VMware vSphere API 4.0 provides access to almost 60 management services.

Figure 2-6. Simplified UML of ServiceInstance Managed Object and ServiceContent Data Object

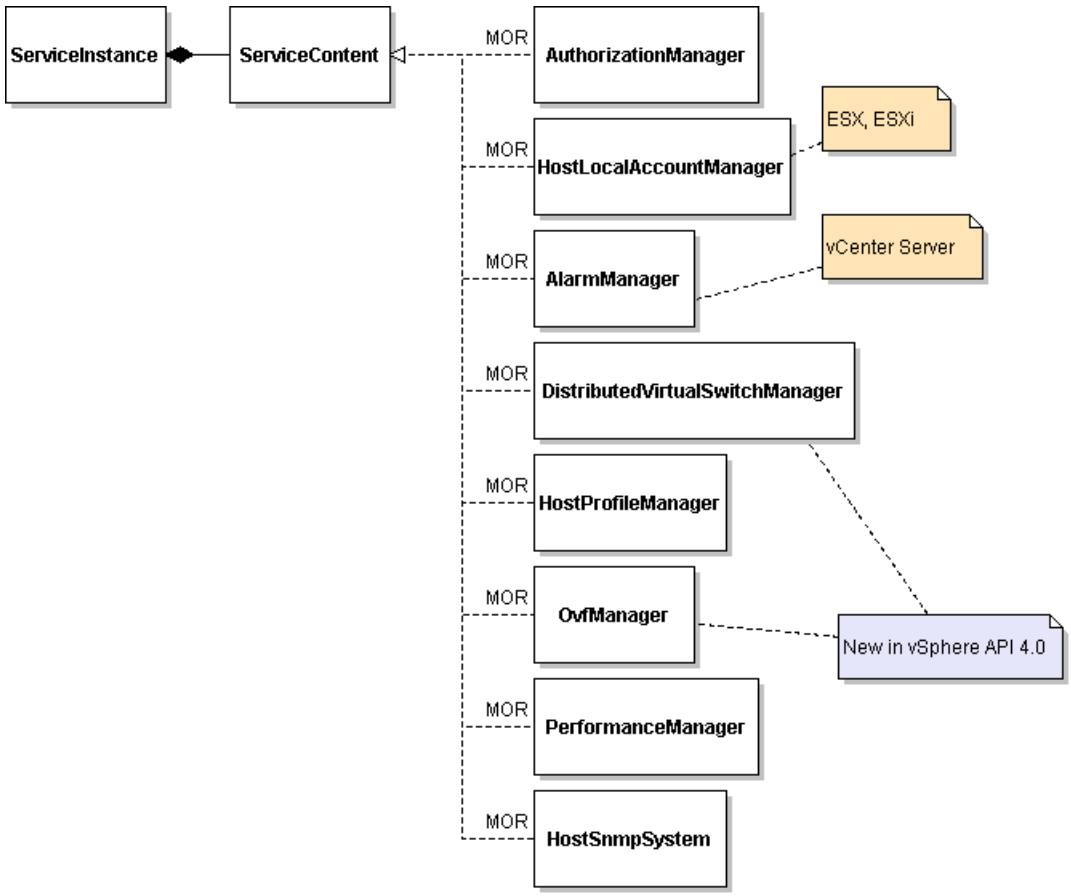
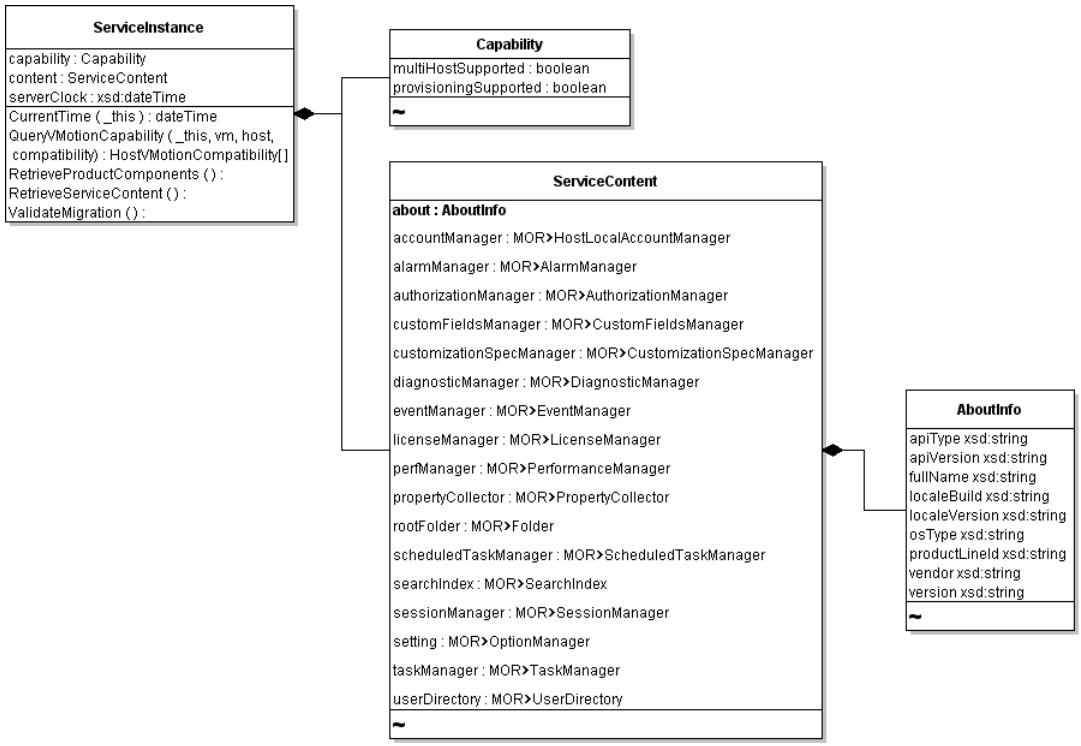


Figure 2-7 shows a more detailed UML diagram of `ServiceInstance` and `ServiceContent`. `ServiceContent`, `Capability`, and `AboutInfo` are all data objects, denoted by the tilde in section of the class definition where operations (methods) are depicted for managed objects.

Figure 2-7. The ServiceInstance Managed Object Type and Some of Its Properties



References to Managed Objects

The client application development model involves working directly with data objects only. Managed objects are never passed between server and client. Client applications use references to the managed object, or with the data objects that comprise the various properties of the managed object. Some references can be obtained through the properties of the `ServiceContent` data object.

The "reference" is an instance of a specific data object type, the `ManagedObjectReference`. Instances of this data object type facilitate remote invocation of operations on the managed objects located on the server. Instances of `ManagedObjectReference` data objects are sent between client and server.

To create an instance of a specific managed entity, you must populate one or more data objects and pass the data objects to the server with the appropriate reference (`ManagedObjectReference`) to invoke the operation on the server.

NOTE Throughout this guide, **managed object references** are referred to as *references*. Code samples might use `MoRef`, `mo_ref`, or `MOR` as a variable name for specific instances of the `ManagedObjectReference` data object. In text, a managed object reference or reference means a specific instance of the `ManagedObjectReference` data object type.

Understanding the Inventory 3

The vSphere inventory contains virtual assets, including datacenters, hosts, virtual machines, virtual applications, storage, networks, and so on. Administrators can view the inventory using the vSphere Client. The vSphere Client obtains information about all the objects in the inventory of a server and renders the information in the GUI.

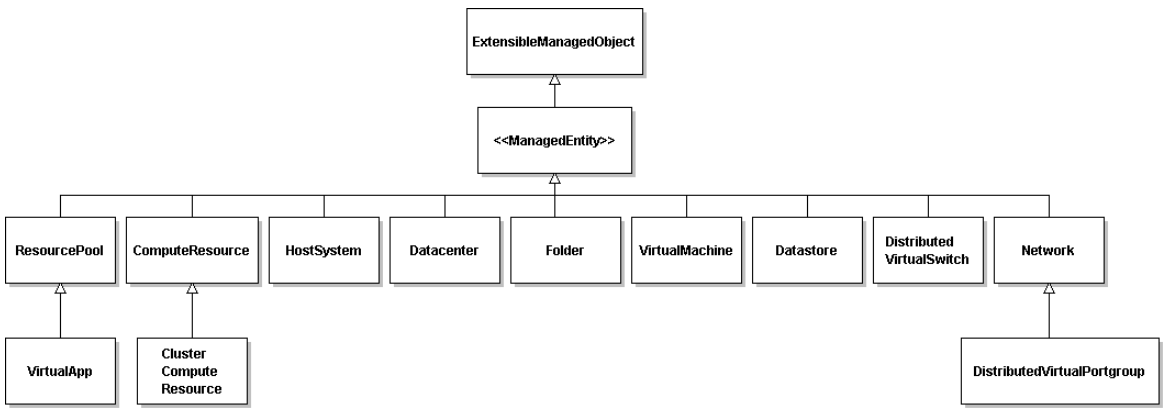
Developers who want to create client applications must understand the components that make up the inventory. You must understand the relationship of one object to another in the inventory, with respect to the the parent-child relationships among inventory objects and the permissions model, among other details. This chapter covers the basics. It includes these topics:

- "Overview of Managed Entities" on page 25
- "Understanding Container Objects and the Inventory" on page 26
- "Inventory Objects and Privileges" on page 28
- "vCenter Server and Unmanaged ESX/ESXi Host Inventories" on page 30

Overview of Managed Entities

In terms of the vSphere object model, the components that make up an inventory belong to the class of objects generally called managed entities. A managed entity is any of the dozen managed object types that extend directly or indirectly from the `ManagedEntity` managed object type. Managed entities include `VirtualMachine`, `Datacenter`, `Datastore`, `Network`, `ResourcePool`, `ComputeResource`, `HostSystem`, `VirtualApp` and the other managed object types shown in Figure 3-1.

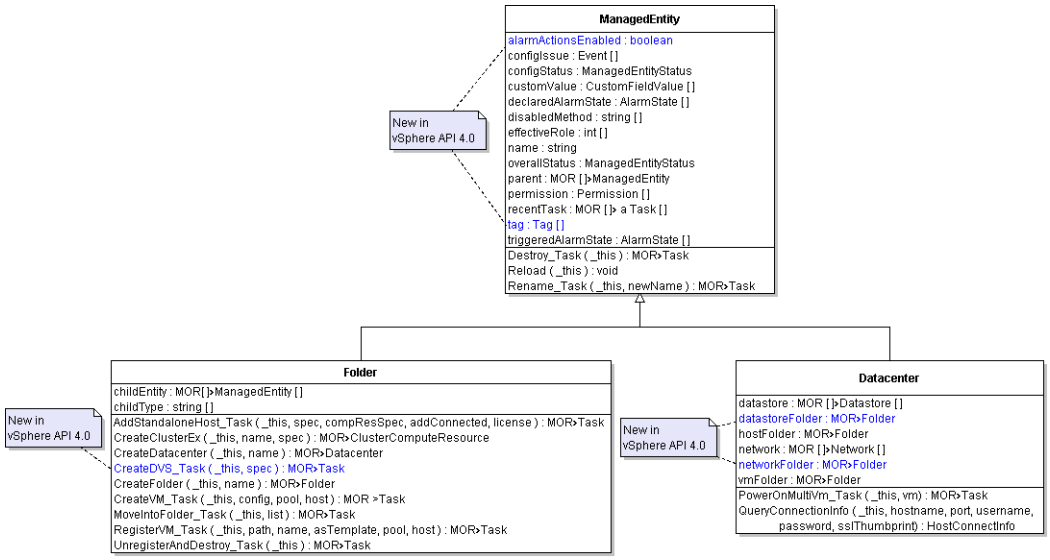
Figure 3-1. ManagedEntity and ManagedEntity Subtypes



The `ManagedEntity` managed object type is an abstract type. It includes several properties and operations that are inherited by each subtype.

For example, the `name`, `effectiveRole`, `permission`, and `parent` are some of the properties defined in the `ManagedEntity` base type that are inherited by each subtype. Lifecycle operations for the object, such as destroying an object instance, are also part of the `ManagedEntity` base type.

Figure 3-2. Datacenter and Folder Managed Object Types Extend ManagedEntity



In addition to the properties and operations inherited from `ManagedEntity`, each subtype has its own properties and special-purpose operations that model its expected behavior. For example, a `HostSystem` managed object has an `EnterMaintenanceMode_Task` operation. The other managed entities do not.

Instances of specific managed entities are created by administrators using various tools, including vSphere Client. Some managed entities are created when the server software is first installed. For example, a `HostSystem` managed entity exists in the inventory of a newly installed ESX/ESXi system. It is not explicitly created. The `HostSystem` managed entity contains the physical characteristics of an ESX/ESXi host system. The data objects that make-up the properties of a `HostSystem` managed entity contain the specific properties of the underlying hardware and its capabilities. For example, the values of the `capability` property of `HostSystem` are set during installation, based on the specifics of the hardware on which ESX/ESXi is installed.

All other entities in an inventory are explicitly created using the vSphere Client or other client tools, or by using the API. The entities that can comprise any specific inventory are constrained by the physical resources of the hardware and the rules of containment, ownership, and other properties of the managed entity types and subtypes.

Understanding Container Objects and the Inventory

The vSphere object model includes several objects that serve specific purposes, including inventory organization, application packaging, delegated administration, and resource allocation. Some objects are used to organize the inventory and establish associations among the entities in the inventory. For example, `Folder`, `Datacenter`, and `VirtualApp` are generally called container object types.

`VirtualApp` is a new managed entity type in vSphere API 4.0 designed for application packaging. `VirtualApp` extends the `ResourcePool` type with additional properties and operations to support product and vendor naming, licensing, power cycling the application, and import-export capabilities. `VirtualApp` instances are exported and imported the industry-standard OVF (open virtual format) files. A `VirtualApp` consists of the virtual machines and other `VirtualApp` containers that operate as a unit. Unmanaged ESX/ESXi host systems do not support `VirtualApp`.

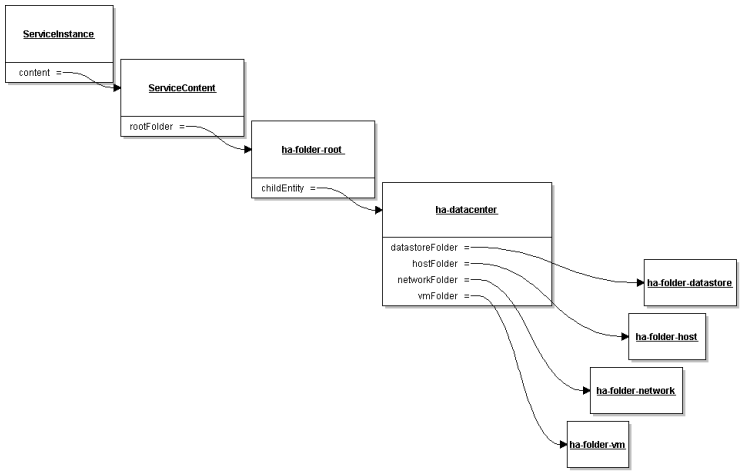
See the *vSphere API Reference* for more information about `VirtualApp`.

Folder as Container

The `Folder` managed object type is the chief container object in an inventory. Containment is conceptual, not literal, and relies on the parent-child relationship among the objects as defined in certain properties of actual instances. Specifically, every managed entity has a `parent` property that can be used to identify its relative position in the inventory hierarchy. In addition, the `Folder` managed object type has a `childEntity` property that identifies objects contained by the specific folder instance.

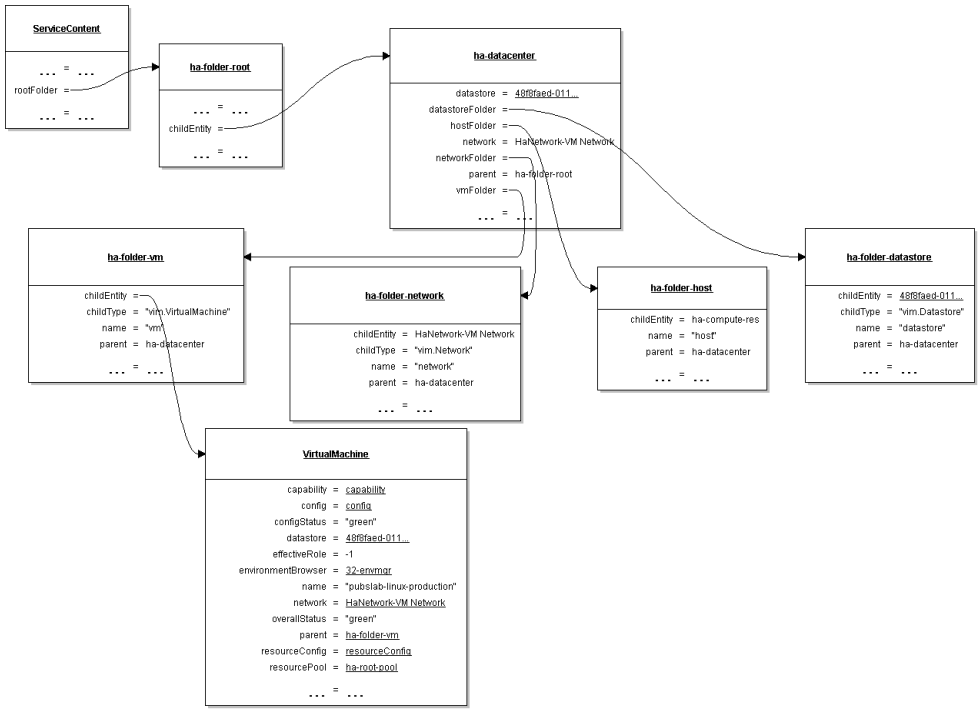
Figure 3-3 shows an instance diagram highlighting the `childEntity` and folder properties that define the default objects in the inventory of an unmanaged ESX/ESXi system. The inventory begins from the `rootFolder` property of the `ServiceContent` data object. The `rootFolder` is the root node of the inventory. It has a `childEntity` that consists of a managed object reference to a `Datacenter` managed object.

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



Child entities are identified through their `childEntity` property, through a `parent` property, or through a specific folder reference in the containing object.

Figure 3-4. Parent-Child Associations Among Inventory Objects



For example, every `Datacenter` object in an inventory is guaranteed to have four specific `Folder` objects associated with it, one each for datastores, networks, hosts, and virtual machines. The folders for `Datastore` objects and `Network` objects are new as of vSphere API 4.0. These folders are associated with the `datastoreFolder` and the `networkFolder` properties of a `Datacenter` managed object (see also Figure 3-2 on page 26).

The inventory hierarchy shown in Figure 3-4, "Parent-Child Associations Among Inventory Objects," on page 27 includes the root folder object, a `Folder` object named `ha-root-folder` has a `childEntity` that is a reference to a `Datacenter`. The `VirtualMachine` instance whose ID is 32 is the `childEntity` of the `Folder` identified as `ha-folder-vm`.

Various other nodes in the hierarchy are called root nodes for their type of object. For example, the `ha-folder-datastore`, `ha-folder-host`, `ha-folder-network`, and `ha-folder-vm` are considered the root folders for the type of entities that each contains.

Inventory Objects and Privileges

Whether through the vSphere Client or by using the API, navigating the entities in 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 objects in an inventory, it does so using the privileges granted to a specific role that is associated with the principal. The server checks the permission object or objects associated with the entity before allowing a principal to view or operate on an entity.

See Chapter 4, "Overview of User Models and Server Access Control Concepts," on page 33 for more information about authentication, authorization, roles, and user identity.

For inventory navigation, it is important to understand the affect of parent-child relationships among inventory entities and how privilege requirements might apply to the object.

To view certain properties or to invoke certain operations on managed objects, a principal must have the appropriate privileges. A privilege is a system-defined requirement associated with a VMware vSphere management object. Privileges are defined by VMware. Privileges are static, and do not change for a single version of a product. Table 3-1 shows the format in which privileges are specified for VMware vSphere components.

Table 3-1. Privilege Definition Format and Examples

Format	Examples	
`<group>[.<group>].privilege`	`Datacenter.Create`	`Host.Config.Connection`
	`Datacenter.Delete`	`Host.Config.Memory`
	`Datacenter.Move`	`Host.Config.Snmp`

Some privileges are specific to objects on vCenter Server or ESX/ESXi only. For example, the `Alarm.Create` privilege is associated with `AlarmManager`, which is the service interface for creating and managing alarms that is available through vCenter Server systems only.

Complete information about required privileges for the operations of a managed entity is available in the *vSphere API Reference*.

To view any property of a managed entity typically requires the `System.View` privilege on the root folder of the inventory and the `System.Read` privilege on the specific `Folder` and `VirtualMachine` objects.

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

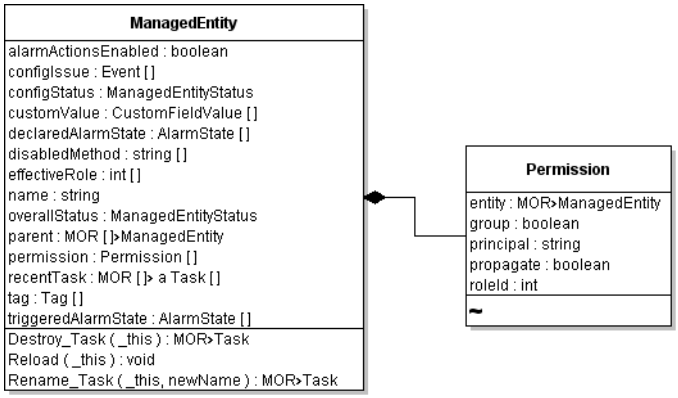
- The `VirtualMachine.Inventory.Create` privilege on the folder in which to create the virtual machine
- The `Resource.AssignVMToPool` privilege on the resource pool from which the virtual machine obtains its allocation of CPU and memory resources.

To read the `perfCounter` property of the `PerformanceManager` managed object, a user must have the `System.View` privilege on the root folder.

A child entity typically inherits the permissions of its parent. If you set the permission on a parent object and set its propagate property to true, the child entities accrue the same permission setting as the parent. However, you can set a permission directly on a child entity that is different than its parent.

The `AuthorizationManager` service interface provides operations for querying and setting permissions. See Chapter 4, "Overview of User Models and Server Access Control Concepts," on page 33 for more information about `AuthorizationManager`.

Figure 3-5. A ManagedEntity Has One or More Permission Objects



The `Permission` data object applies to the `childEntity` associated with the managed entity when the `propagate` property of Permission is set to `true`.

So for example, in Figure 3-6 the ha-folder-root object has two permission objects in the array that makes up its permission property. Both permission objects have their propagate flag set to true. The `childEntity` of `ha-folder-root` is `ha-datacenter`, so `ha-datacenter` inherits the permissions of `ha-folder-root`.

Assigning permissions to the `Datacenter` node and setting the `Permission` object's propagate flag set to `true` effectively grants permission at the `Datacenter` level and all child entities that have a specific `Datacenter` object as their parent.

Figure 3-6. Inventory and Permissions

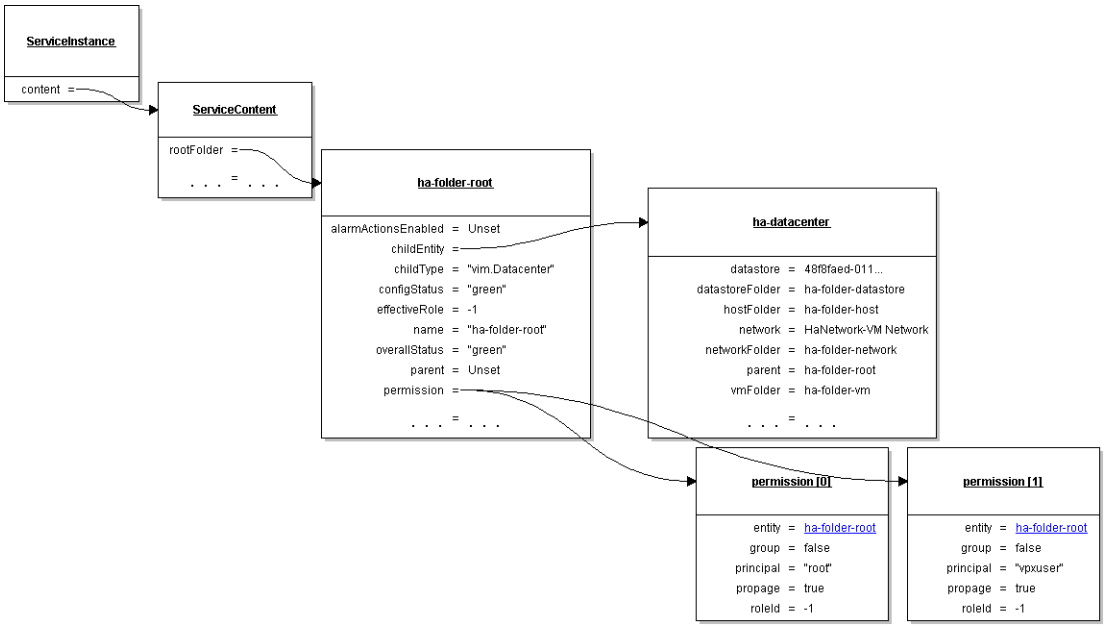


Figure 3-6 shows that users `root` and `vpxuser` both have permissions on the `rootFolder` of the inventory. The `vpxuser` is the account created on the host by the vCenter Server system when the host is added to the vCenter Server. The vCenter Server needs access to the inventory objects of the host systems that it manages, so the `vpxuser` account is granted privileges to the `rootFolder` of each host.

For example, to invoke the `Destroy_Task` operation on a specific datacenter instance requires the user to have the `Datacenter.Delete` privilege on the `datacenter` and on the parent `Folder` containing the `Datacenter`.

Folder as Factory

The `Folder` managed entity is a factory object. It provides operations for creating instances of other managed entities, including the following types:

- Datacenters
- Distributed virtual switches
- Virtual machines
- Clusters
- Folders

For an unmanaged ESX/ESXi system, the `Folder` managed entity does not support creating additional `Folder` objects or `Datacenter` objects.

vCenter Server and Unmanaged ESX/ESXi Host Inventories

The inventory of a vCenter Server system supports multiple ESX/ESXi hosts. The hosts are added to a vCenter Server using the vSphere Client, or programmatically by using the API. Adding a host to a vCenter Server system adds the host to the inventory of the vCenter Server, after which it becomes a managed host. You can configure two or more of hosts in a vCenter Server system into clusters.

The inventory of an unmanaged 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 an unmanaged host and they are visible in the MOB. The `rootFolder` is a property available in the `ServiceContent` data object, and the default datacenter is actually the `childEntity` associated with the root folder, as shown in Figure 3-3, "Instance Diagram of Root Folders in an Inventory," on page 27.

You must account for these objects when you navigate the inventory programmatically, such as when you use a `PropertyCollector` in your code. Table 3-2 summarizes the differences between the number of objects that an inventory can contain.

Table 3-2. Comparison of Unmanaged ESX/ESXi and vCenter Server Inventories

ManagedEntity Subtype	Description	ESX/ESXi Inventory	vCenter Server Inventory
`ClusterComputeResource`	Abstracts multiple HostSystem objects into a unified compute resource.	None.	Multiple instances supported.
`ComputeResource`	Abstracts the physical resources of a host system and enables them to be associated with the virtual machines that run on the host.	Exactly one only.	Multiple instances supported.
`Datacenter`	Container object for establishing associations among managed entities, including folders, virtual machines, host systems.	Exactly one only. Cannot be destroyed. Transparent.	Multiple instances supported.
`Datastore`	Logical storage volumes on which to store virtual machine files.	Multiple instances supported.	Multiple instances supported.
`DistributedVirtualSwitch`	Interface for DistributedVirtualSwitch (vNetwork Distributed Switch).	Multiple instances supported.	Multiple instances supported.

Table 3-2. Comparison of Unmanaged ESX/ESXi and vCenter Server Inventories (Continued)

ManagedEntity Subtype	Description	ESX/ESXi Inventory	vCenter Server Inventory
`Folder`	Container object used to organize virtual machines and hosts in an inventory.	Exactly one only. Cannot be destroyed. Transparent.	Multiple instances supported.
`HostSystem`	Abstraction for a single physical machine.	Exactly one only.	Multiple instances supported.
`Network`	Abstraction for a physical or a virtual network (VLAN).	Multiple instances supported.	Multiple instances supported.
`ResourcePool`	Abstraction for physical resources that can partitioned into smaller units for use by virtual machines.	Multiple instances supported.	Multiple instances supported.
`VirtualApp`	Container object for one or more virtual machines and associated objects packaged using open virtual format (OVF) 1.0.	None.	Multiple instances supported.
`VirtualMachine`	Abstraction for a virtual machine.	Multiple instances supported.	Multiple instances supported.

Overview of User Models and Server Access Control Concepts 4

VMware vSphere implements mechanisms to ensure that only valid users can access virtual infrastructure components. The user model for ESX/ESXi systems is different than the user model for vCenter Server systems. This chapter discusses these user models and the managed objects (service interfaces) that provide various mechanisms to secure the system. It includes these topics:

- "Introduction to Authentication and Authorization" on page 33
- "Overview of ESX/ESXi User Model" on page 35
- "Overview of vCenter Server User Model" on page 36
- "Obtaining Information about Users and Groups from UserDirectory" on page 37
- "Authenticating Users through SessionManager" on page 39
- "Overview of the AuthorizationManager" on page 39
- "Granting Privileges through Permissions" on page 42

Introduction to Authentication and Authorization

Several server-side mechanisms interactively authenticate a human user when a client application, such as the vSphere Client or a custom application developed using the vSphere Web Services SDK, connects to the server. The VMware vSphere interfaces for authenticating users and protecting virtual infrastructure from unauthorized access include:

- `HostLocalAccountManager` is used to create and manage user accounts and groups on ESX/ESXi systems. Depending on the permissions associated with the roles assigned to the user accounts or groups, authenticated users can view objects or invoke operations on the server. See "Overview of ESX/ESXi User Model" on page 35 for more information about the `HostLocalAccountManager`.
- `SessionManager` provides an interface to the server authentication infrastructure on the target server system.
  - For vCenter Server systems, `SessionManager` supports integration with Microsoft SSPI (Security Service Provider Interface), which enables single sign-on using Microsoft Windows local accounts or Active Directory.
  - For ESX/ESXi systems, `SessionManager` supports authenticating user accounts as defined on the host system, such as accounts created using vSphere Client or those created programmatically through the `HostLocalAccountManager` API.

- `AuthorizationManager` protects VMware vSphere components from unauthorized access. Access to components is role-based: Users and groups are assigned roles that encompass the privileges needed to perform operations on some managed entities, or to view objects. `AuthorizationManager` has operations for creating new roles, modifying user roles, setting permissions on entities, and handling all other aspects of the relationship managed objects and the permissions granted to principals (users, groups) to access them.
- `UserDirectory` provides a look-up mechanism that returns user-account information to AuthorizationManager or to another requestor, such as a client application. See "Overview of ESX/ESXi User Model" on page 35 for more information about the `UserDirectory`.

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

Accessing an object might mean reading a property value, or invoking an operation. In many cases, the privileges required to read a property value are less restrictive than the privileges required to perform operations. For example, a user might be able to see a virtual machine in the inventory, but might not be allowed to create a new virtual machine.

Understanding System 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 required on the part of a principal (user, group member) 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.

Privileges Defined in the vSphere API Reference

The *vSphere API Reference* includes information about privileges required to invoke operations and to view properties. In the documentation page of the specific managed object, look for the "Required Privileges" label to find out about specific privileges, if any, required to view the property or invoke the operation. Table 4-1 shows the format used to specify privileges.

Table 4-1. Privilege Specification Format and Examples

Format	Examples	
`<group>[.<group>].privilege`	`Datacenter.Create`	`Host.Config.Connection`
	`Datacenter.Delete`	`Host.Config.Memory`
	`Datacenter.Move`	`Host.Config.Snmp`

A privilege might be specific to vCenter Server or to ESX/ESXi systems. For example, the `Alarm.Create` privilege is defined on vCenter Server. Setting alarms requires vCenter Server, which has an `AlarmManager` service interface that integrates with many of the other management objects on the server.

Users are granted privileges to objects through roles. See "Using Roles to Consolidate Sets of Privileges" on page 40 for more information. See Table A-3, "Privileges Granted to the Administrator Role," on page 158 for a complete list of privileges encompassed by the Administrator role as defined on a vCenter Server 4.0 system.

HTTP-Based File Access and Privileges

Privilege requirements apply to system objects regardless of how a given client application attempts to access server content. For example, support for HTTP/HTTPS access to the files comprising `Datastore` objects was added in ESX/ESXi 3.5. Files comprising the objects that make-up virtual machine datastores can be accessed by specifying a URL in this format:

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

Because the objects being accessed are datastore objects in the inventory and these objects are associated with a specific folder structure, the privileges required to access these files are the same privileges required to obtain these files using any other mechanism, such as the vSphere Client.

Table 4-2. Privileges Required for Datastore Objects Apply Regardless of Access Mechanism

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`

Overview of ESX/ESXi User Model

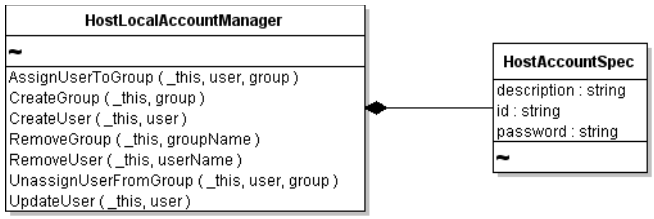
When users enter their user account and credential from a client application, the server consults the appropriate user-account store and validates the authenticity of the user account and the associated credential. Currently, the credential consists of a password, but the VMware vSphere also supports certificates, such as X.509 certificates. Authenticated users can then access objects they've been authorized to use. A user identity must exist on the target system or in a supported directory service, as a user account, to be authenticated.

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

Using HostLocalAccountManager Managed Object

The `HostLocalAccountManager` managed object provides operations for creating users, creating groups, assigning users to groups, and other user- and group-administration tasks. The `HostLocalAccountManager` service interface is available through the Web service on ESX/ESXi only. vCenter Server uses the Windows user management facilities. See "Overview of vCenter Server User Model" on page 36 for details.

Figure 4-1. HostLocalAccountManager Managed Object



You must define the `HostAccountSpec` data object using the characteristics required for the target system, as configured by the system administrator. These characteristics include password length requirements, restrict use of dictionary words, and so on.

In addition to creating user accounts and groups, `HostLocalAccountManager` provides several other operations:

- `AssignUserToGroup`, which assigns a specified user account to a group.
- `UnAssignUserFromGroup`, which removes a user from a group. Use RetrieveUserGroups first if necessary, to obtain group information for the user account.
- `UpdateUser`, which allows you to modify information, including password and description, for an existing user account.

Table 4-3 summarizes the API reference documentation for these operations. For complete information about `HostLocalAccountManager`, see the *vSphere API Reference*.

Table 4-3. HostLocalAccountManager Operations for User and Group Management

Operation	Parameter	Type	Description
`AssignUserToGroup`	_this	`MOR > HostLocalAccountManager`	Reference to the `HostLocalAccountManager` used to make the call.
	user	string	User ID (the generated number from the user store of the host system).
	group	string	Group to which the user account (user ID) is being added.
`UnAssignUserFromGroup`	_this	`MOR > HostLocalAccountManager`	Reference to the `HostLocalAccountManager` used to make the call.
	user	string	Name of user being removed from a group.
	group	string	Group from which to remove user.
`UpdateUser`	_this	`MOR > HostLocalAccountManager`	Reference to the `HostLocalAccountManager` used to make the call.
	user	`HostAccountSpec` data object	Data object that contains description, user account ID, and a password for the principal being updated.

After user accounts and groups on the host system (ESX/ESXi), you can grant these users access to virtual components by using the `AuthorizationManager` operations. See "Overview of the AuthorizationManager" on page 39 for more information.

Overview of vCenter Server User Model

vCenter Server is a Windows-based service that uses native Windows facilities and user model for identification and authentication. The service is associated with the Windows user account that was logged onto 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. Other vCenter Server users who connect to the Web service must also have a Windows account on the local Administrator group. If you use Windows domain login accounts for vCenter Server, the account must be a member of the local host system Administrator group. VMware recommends creating a Windows user account specifically for installing and managing the vCenter Server.

By default, vCenter Server and VirtualCenter 2.5 can obtain user-account information from Microsoft Active Directory, the user-account repository for Microsoft Windows domain controller.

To create an Administrator account for vCenter Server

1 Log in to the Windows server as local Administrator.

2 Create a user account for vCenter Server.

3 Add the user account to the Administrator group of the machine.

4 Log out as Administrator.

5 Log in to the Windows server again, using the new account.

6 Install vCenter Server.

For details, see the *Basic System Administration* guide in the VMware vSphere documentation set.

Enabling Single Sign-on Support and Integration with Microsoft Active Directory

Organizations that are using Microsoft Active Directory can leverage the user identities contained in a Windows 2003 Server domain controller or Active Directory service across their virtual infrastructure, for clients that run vSphere Web Services SDK client applications from Windows-based systems.

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

Obtaining Information about Users and Groups from UserDirectory

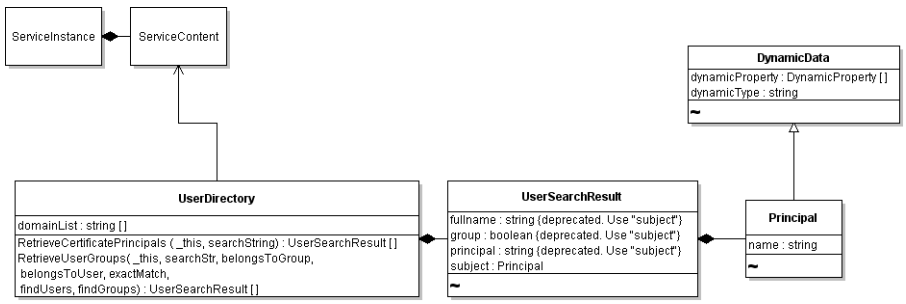
The `UserDirectory` managed object enables a client application to obtain information about users and groups on a VMware vSphere server. Properties and results vary, depending on whether the server is a vCenter Server or an ESX/ESXi system.

Table 4-4. Server Source Data for UserManager

vCenter Server	ESX, ESXi
Domain controller, Active Directory, or local Windows account repository	`/etc/passwd`

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

Figure 4-2. UserDirectory Managed Object



`UserDirectory` provides two operations for obtaining information about users and groups:

- `RetrieveCertificatePrincipals`, which returns an array of `CertificatePrincipal` data objects (principals that use X.509 certificates for authentication to the server, rather than a password).
- `RetrieveUserGroups`, which 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:
  - By user name
  - By group name
  - For an exact match
  - For for a partial string (substring)

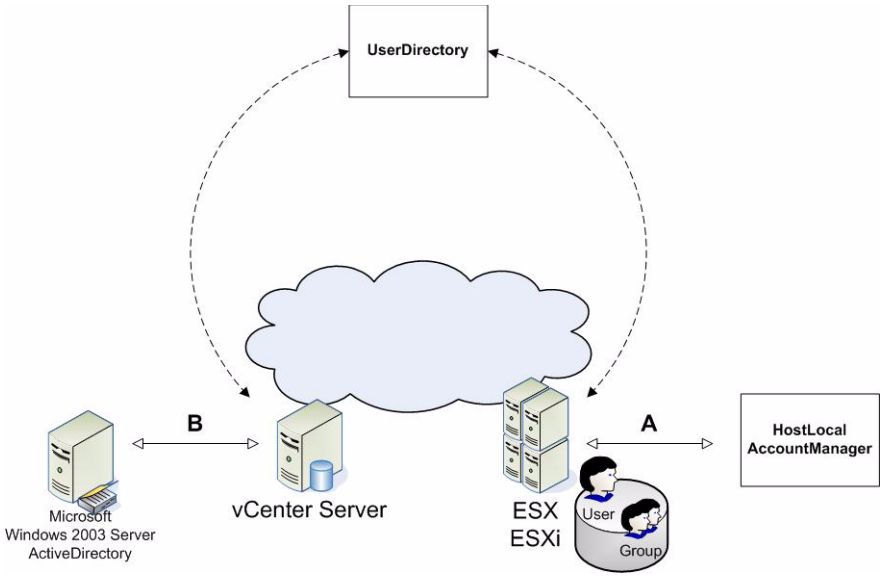
For vCenter Server, search is limited to the specified Windows domain. If the domain is omitted, the search is performed on local users and groups.

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

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

Figure 4-3 shows these two different user management mechanisms associated with the VMware vSphere server.

Figure 4-3. Managed Objects for Handling Local User Accounts



Both `RetrieveCertificatePrincipals` and `RetrieveUserGroups` require a managed object reference to the `UserDirectory`. Both operations return an array of `UserResearchResult` data objects based on the string or substring used for the filter.

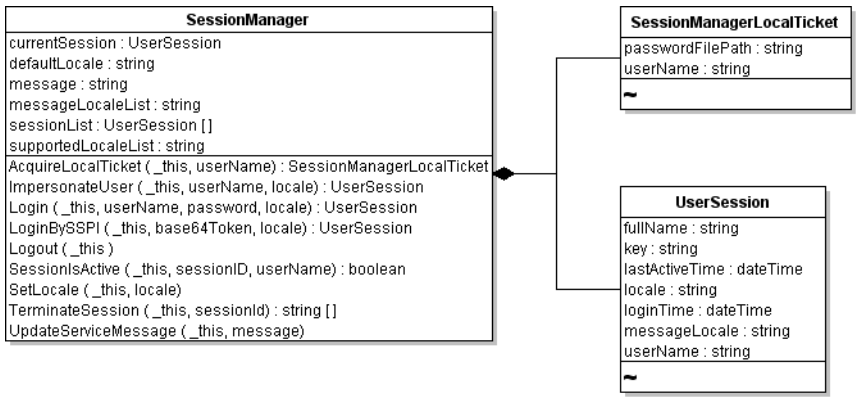
Table 4-5. UserManager Operations

Operation	Parameter	Type	Description
`RetrieveCertificate Principals`	`_this`	`MOR > UserDirectory`	Reference to the `UserDirectory`.
	`searchStr`	string	Case-insensitive string that filters results. Matches the subjectName of CertificatePrincipal data objects. Leave blank to return all CertificatePrincipal data objects.
`RetrieveUser Groups`	`_this`	`MOR > UserManager`	Reference to the `UserDirectory`.
	`domain`	string	Optional. Domain that should be searched for user account or group information. Leave unset to search local machine.
	`searchStr`	string	Required. Case-insensitive string used to filter results. Matches on login and full-name for users. Matches name and description for groups.
	`belongsToGroup`	string	Optional. ESX only. Returns users or groups that explicitly contain the specified group. Indirect members (users or groups that are members of a contained group) are not returned.
	`belongsToUser`	string	Optional. Returns groups that explicitly contain the user. Indirect users (users that are members of a contained group) are not returned.
	`exactMatch`	boolean	Required. Set to `false` to restrict results to precise match of user or group name.
	`findUsers`	boolean	Required. Set to `true` to include users in the result.
	`findGroups`	boolean	Required. Set to `true` to include group information in the result.

Authenticating Users through SessionManager

The `SessionManager` provides several operations for logging into 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.

Figure 4-4. SessionManager Managed Object



Upon successful authentication of the user account, the `SessionManager` returns a `UserSession` data object to the client application. The session is associated with that specific user account for the duration of the session. The client application can save the session locally, to a secure file, and re-use the session later to reconnect to the server. You can also configure an ESX/ESXi or vCenter Server system to support local sessions, which enable users with credentials on the host to log in based on those privileges.

The `SessionManager` provides these capabilities:

- Log in and log out. Basic operations to log in to ESX or vCenter Server, 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` operation.
- Delegation. A client application executing on behalf of a local user can invoke the `AcquireLocalTicket` operation of `SessionManager` to obtain a one-time user name and password for logging on without entering a subsequent password. Delegation is useful for host-based utilities that run in the local console.

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

Overview of the AuthorizationManager

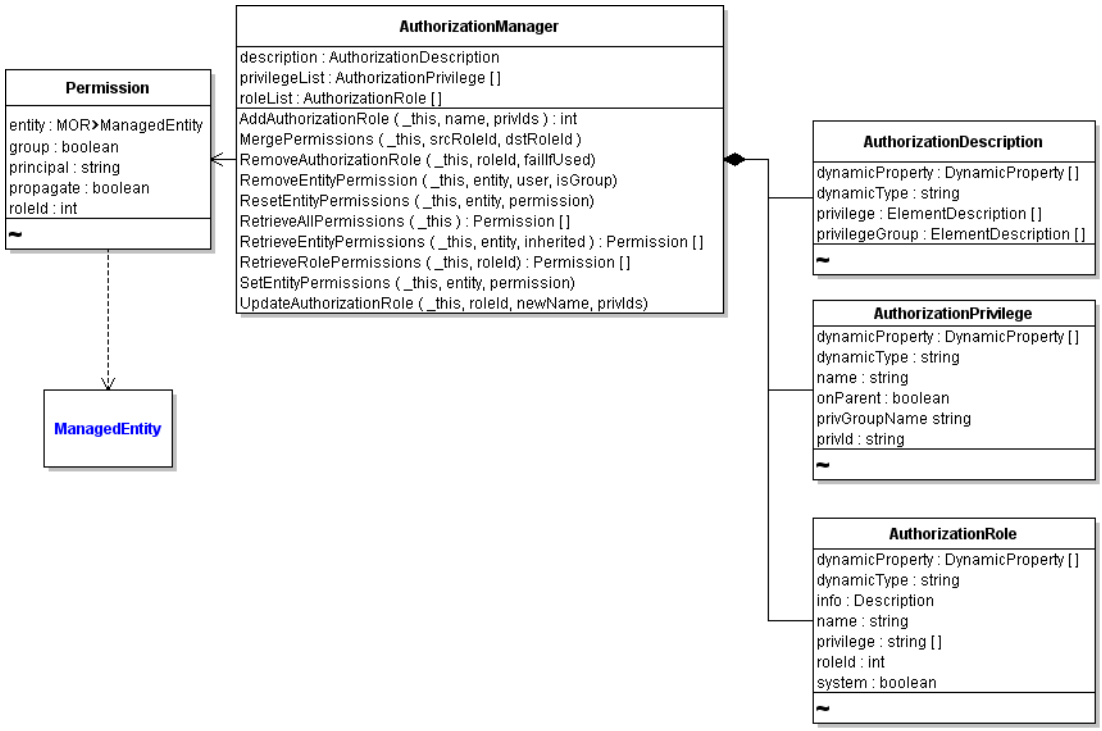
The `AuthorizationManager` is the service interface for handling permissions and roles. It includes close to a dozen operations for creating, modifying, and managing roles and permissions, and for obtaining information about the roles and permissions defined in the system. The `AuthorizationManager` enables access and prevents access to specific server objects based on the permissions associated with the object.

Operations for managing roles include `AddAuthorizationRole`, `RemoveAuthorizationRole`, and `UpdateAuthorizationRole`. See Table 4-7, "AuthorizationManager Operations for Creating, Updating, and Deleting Roles," on page 41 fore more information about these operations.

Operations for managing permissions include `MergePermissions`, `RemoveEntityPermission`, `ResetEntityPermissions`, `RetrieveAllPermissions`, `RetrieveEntityPermissions`, `RetrieveRolePermissions`, and `SetEntityPermissions`.

Figure 4-5 shows these operations in the context of a UML diagram for `AuthorizationManager` and some of its associated data objects.

Figure 4-5. AuthorizationManager Managed Object



`AuthorizationManager` also provides properties that can be used to obtain information fast. For example:

- The `privilegeList` property returns the list of all privileges defined on the system, as an array of `AuthorizationPrivilege` data objects. Privileges are defined by VMware, on the various objects and properties contained in the system. These privileges are fixed and cannot be changed by client applications. See Appendix A, "Managed Object Privileges Reference," on page 149 for lists of privileges.
- The `roleList` property returns list of all currently defined roles, including the system-defined roles, as an array of `AuthorizationRole` data objects.

You can see these property values for yourself, on your ESX or vCenter Server system, by navigating to your the `AuthorizationManager` using the MOB on your system. See "Using the Managed Object Browser to Explore the Object Model" on page 16 for more information.

Using Roles to Consolidate Sets of Privileges

A *role* is a named set of one or more privileges, grouped for convenience. Assigning a role that encompasses all the necessary privileges to perform an action is easier and faster than granting each individual privilege separately. ESX defines two types of roles:

- System roles, which cannot be modified or deleted.
- User roles, which apply to different user communities or that restrict access for add-on tools. Several pre-defined user roles are included with vCenter Server and with ESX/ESXi systems. You can create new roles using these pre-defined user roles as a starting point.

Table 4-6 describes these two types of roles in more detail.

Table 4-6. System and Pre-Defined User Roles

Type	Role name	Role ID	Description
System roles	Administrator	-1	Super-user access. Encompasses the set of all defined privileges. See Table A-3, "Privileges Granted to the Administrator Role," on page 158 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 sub-objects under a higher-level object that has propagated permissions defined.
	Read-Only	-2	Read-only access. Encompasses the set of all non-mutable 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 invoke query methods, but cannot make any changes to the system.
	View	-3	Visibility access consisting of `System.Anonymous` and `System.View` privileges. Cannot be granted.
User roles (pre-defined samples)	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.

Table 4-7 lists several `AuthorizationManager` operations available for managing roles.

Table 4-7. AuthorizationManager Operations for Creating, Updating, and Deleting Roles

Operation	Parameter	Type	Description
`AddAuthorizationRole`	`_this`	`MOR > AuthorizationManager`	Managed object reference to the server's `AuthorizationManager` obtained through `ServiceInstance`.
	`name`	string	Name for the new role.
	`privIds`	string[ ]	List of privileges to assign to the role in the format: `<group>[.<group>].privilege`

Table 4-7. AuthorizationManager Operations for Creating, Updating, and Deleting Roles (Continued)

Operation	Parameter	Type	Description
UpdateAuthorizationRole	_this	MOR > AuthorizationManager	Managed object reference to the server's AuthorizationManager obtained through ServiceInstance.
	roleId	int	Integer that identifies the role to be modified.
	newName	string	New name for the role.
	privIds	string[ ]	String array of privileges to assign to the role.
RemoveAuthorizationRole	_this	MOR > AuthorizationManager	Managed object reference to the AuthorizationManager obtained from the ServiceInstance of the server.
	roleId	int	Integer that identifies the role. The roleId is generated by AddAuthorizationRole when a new role is defined.
	failIfUsed	boolean	Set to true to ensure the role cannot be removed if it is being used.

Modifying Sample Roles to Create New Roles

The system roles listed in Table 4-6 cannot be modified or deleted. However, you can create completely new roles, or modify the sample user roles to suit your needs.

- See the *Basic System Administration* guide or the vSphere Client online help for information about creating new roles or modifying the sample roles using the vSphere Client.
- See the *vSphere Command-Line Interface Installation and Reference Guide* for information about using the `vicfg-user` vSphere CLI command to create new roles or modify the sample roles.

To create new roles using the API

1 Obtain a reference (specific instance of `ManagedObjectReference`) to the `AuthorizationManager` for the server, from the `ServiceContent` object for the `ServiceInstance`.

2 Invoke the `AddAuthorizationRole` method, passing the reference in the invocation along with a name for the role (as a string) and an array of privileges (array of strings) that has the list of privileges 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 Use the `roleID` in subsequent code, to assign the role to specific users or groups.

Granting Privileges through Permissions

A `Permission` data object 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 granularities of detail, including:

- For all users defined in the system by invoking the `RetrieveAllPermissions` operation, which returns an array of `Permission` data objects.
- For specific inventory objects, such managed entities, such as folders, datacenters, or virtual services—by invoking the `RetrieveEntityPermissions` operation.
- For a role defined in the system by invoking the `RetrieveRolePermissions` operation.

See the *vSphere API Reference* for more information. However, you can invoke any of these operations from the MOB to obtain information about your system, as you develop your code.

Setting, Changing, or Deleting Permissions

The `Permission` data object associates the privileges required to perform an action with the principals (user, group) that have been granted those privileges by virtue of the role assigned to the principals.

The `AuthorizationManager` provides several operations for managing `Permission` objects, including those listed in Table 4-8.

To set or update permissions on an entity, use the `SetEntityPermissions` operation. Setting or changing permissions on entities requires the `Authorization.ModifyPermissions` privilege on the target managed entity.

To set permissions on an entity

1 Obtain a reference to the `AuthorizationManager` for the server, from the ServiceInstance's `ServiceContent` object. For example:

```
ManagedObjectReference hostAuthorizationManager =
    cb.getConnection().getServiceContent().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, to create 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);
```

To completely replace existing permissions with a new set of permissions, use the `ResetEntityPermissions` operation.

Table 4-8. AuthorizationManager Operations for Setting, Changing, and Removing permissions

Operation	Parameter	Type	Description
`SetEntityPermissions`, `ResetEntityPermissions`	`_this`	`MOR > AuthorizationManager`	Managed object reference to the server's (`ServiceInstance`) `AuthorizationManager`.
	`entity`	`MOR > ManagedEntity [HostSystem, Folder, ComputeResource...]`	Managed entity for which you want to grant or deny access.
	`permission`	`Permission[ ]`	Array of `Permission` objects that define access rules for the entity. Passing an empty array removes all permissions from the entity.

Table 4-8. AuthorizationManager Operations for Setting, Changing, and Removing permissions (Continued)

Operation	Parameter	Type	Description
RemoveEntityPermission	_this	MOR > AuthorizationManager	Delete the permission from an entity for specified principal (user, group).
	entity	MOR > ManagedEntity [HostSystem, Folder, ComputeResource...]	Managed object reference that identifies the managed entity from which to disassociate a Permission. Use the rootFolder managed object reference if the Permission is associated with a managed object that is not a managed entity. Use the specific managed entity subclass type (Folder, Datacenter, and so on) for Permission being removed is associated with a managed entity.
	user	string	Principal (user, group) for which the permissions were defined.
	isGroup	boolean	Specifies that the user value is a group (true) or a single user (false).

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 its child entities.

Impact of Group Membership on Permissions

Users can be members of multiple groups. If a user account with membership in multiple groups has no other explicitly defined permissions, then the account is granted the combined set (union) of all group-assigned permissions.

Users also obtain the union of permissions associated with the containing resource pool and folder for virtual machines for which they have been given permission. However, user-specific permissions always take precedence over any group permissions. Table 4-9 summarizes these details.

Table 4-9. User and Group Permissions Summary

User-level permissions	Group-level permissions
■ Always take precedence over any group-level permissions.	■ Membership in multiple groups with permissions on the same object results in union of permissions. ■ Absent explicit user permissions, group level permissions apply as if granted to user directly.

Permissions are applied to the associated objects that make-up the inventory item, from the containing object to each of its child entities.

vSphere API Programming Model 5

The VMware vSphere management object model provides a server-side framework for provisioning, instrumenting, managing, and monitoring VMware vSphere components, such as virtual machines, host systems, and virtualized applications. Programmatic access to the VMware vSphere object model and management framework from a client application is through the vSphere API. This chapter includes these topics:

- "API as a Web Service" on page 45
- "Client Application Characteristics" on page 46
- "Accessing the API on the Web Service" on page 50
- "Using the vSphere API Reference" on page 51
- "Mapping the WSDL to the Client-side Proxy Code" on page 51
- "Invoking Operations through the MOB" on page 53

API as a Web Service

The vSphere API is language-neutral. It is implemented as industry-standard Web services that are hosted 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, visit the Web Services Interoperability Organization (WS-I) Web site at http://www.ws-i.org.

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

API Defined in WSDL File

The Web services API is defined in a WSDL (Web Services Description Language) file. The WSDL file is used by Web-services client-application development tools (SOAP toolkits, such as Axis or Microsoft .NET) to create client-side proxy code (stubs) that facilitate remote method invocation, marshalling and marshalling object data, and other low-level details of distributed object-oriented applications programming.

Client applications invoke operations by sending SOAP (Simple Object Access Protocol)-formatted messages. SOAP is an XML format and is programming-language neutral. One of the jobs of the client-side Web services tools is formatting the SOAP messages from the programming language that you use.

Communications between client and server occur over HTTPS (HTTP over an encrypted Secure Sockets Layer (SSL) connection). The default is HTTPS. However, the server can be configured to support HTTP. See the *Developer's Setup Guide* for details.

Client Application Characteristics

The VMware vSphere application model is asynchronous, distributed, and relies on instances of the `ManagedObjectReference` data object for remote method invocation.

Asynchronous Client-Server Application Style

The VMware vSphere application model uses an asynchronous client-server communication scheme by default. That is, most method invocations are non-blocking, returning a reference to a `Task` managed object. See Chapter 9, "Overview of the Task Infrastructure," on page 83 for a complete overview.

Distributed Object-Oriented Application Style

VMware vSphere client applications invoke operations on the server remotely, over a network. Only data objects are sent back and forth between server and client. Managed objects are accessed by reference. You use instances of the `ManagedObjectReference` data object type to invoke operations on the server objects from your client application. Each managed object reference identifies a specific managed object on the server with its type and a value (Figure 5-1).

Figure 5-1. ManagedObjectReference Data Object Diagram

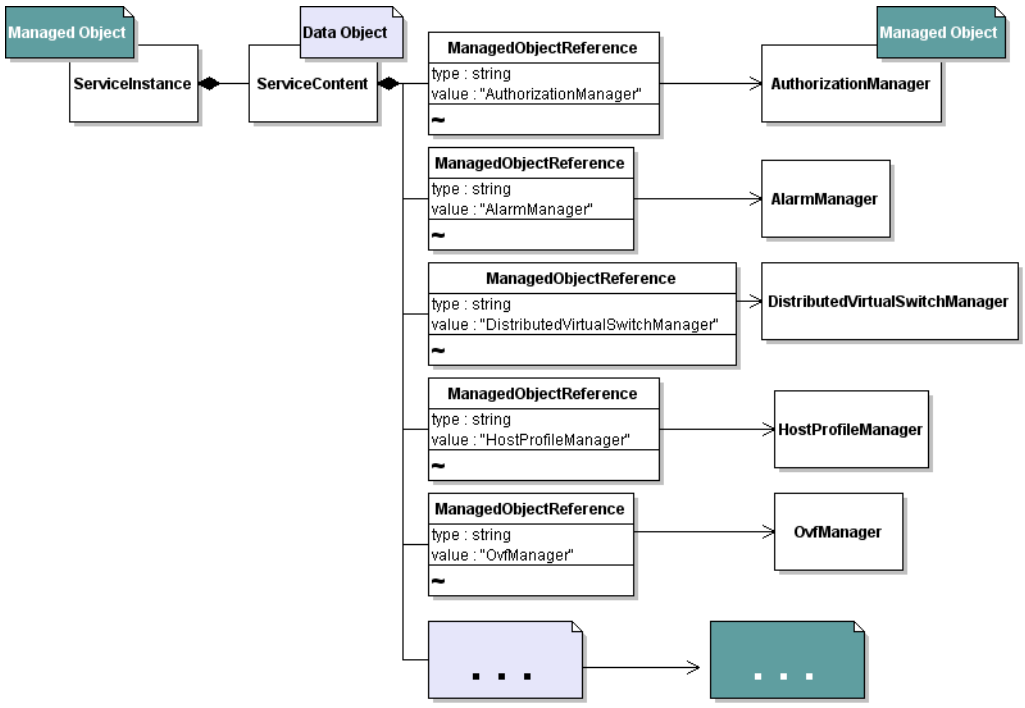


Table 5-1 lists all possible valid strings for the `value` property of `ManagedObjectReference`.

Table 5-1. Valid Values for ManagedObjectReference Type Property

Type Property	Refers to
"Alarm"	`Alarm` managed object
"AlarmManager"	`AlarmManager` managed object
"AuthorizationManager"	`AuthorizationManager` managed object
"ClusterComputeResource"	`ClusterComputeResource` managed object
"ClusterProfile"	`ClusterProfile` managed object

Table 5-1. Valid Values for ManagedObjectReference Type Property (Continued)

Type Property	Refers to
"ClusterProfileManager"	`ClusterProfileManager` managed object
"ComputeResource"	`ComputeResource` managed object
"ContainerView"	`ContainerView` managed object
"CustomFieldsManager"	`CustomFieldsManager` managed object
"CustomizationSpecManager"	`CustomizationSpecManager` managed object
"Datacenter"	`Datacenter` managed object
"Datastore"	`Datastore` managed object
"DiagnosticManager"	`DiagnosticManager` managed object
"DistributedVirtualPortgroup"	`DistributedVirtualPortgroup` managed object
"DistributedVirtualSwitch"	`DistributedVirtualSwitch` managed object
"DistributedVirtualSwitchManager"	`DistributedVirtualSwitchManager` managed object
"EnvironmentBrowser"	`EnvironmentBrowser` managed object
"EventHistoryCollector"	`EventHistoryCollector` managed object
"EventManager"	`EventManager` managed object
"ExtensibleManagedObject"	`ExtensibleManagedObject` managed object
"ExtensionManager"	`ExtensionManager` managed object
"FileManager"	`FileManager` managed object
"Folder"	`Folder` managed object
"HistoryCollector"	`HistoryCollector` managed object
"HostAutoStartManager"	`HostAutoStartManager` managed object
"HostBootDeviceSystem"	`HostBootDeviceSystem` managed object
"HostCpuSchedulerSystem"	`HostCpuSchedulerSystem` managed object
"HostDatastoreBrowser"	`HostDatastoreBrowser` managed object
"HostDatastoreSystem"	`HostDatastoreSystem` managed object
"HostDateTimeSystem"	`HostDateTimeSystem` managed object
"HostDiagnosticSystem"	`HostDiagnosticSystem` managed object
"HostFirewallSystem"	`HostFirewallSystem` managed object
"HostFirmwareSystem"	`HostFirmwareSystem` managed object
"HostHealthStatusSystem"	`HostHealthStatusSystem` managed object
"HostKernelModuleSystem"	`HostKernelModuleSystem` managed object
"HostLocalAccountManager"	`HostLocalAccountManager` managed object
"HostMemorySystem"	`HostMemorySystem` managed object
"HostNetworkSystem"	`HostNetworkSystem` managed object
"HostPatchManager"	`HostPatchManager` managed object
"HostPciPassthruSystem"	`HostPciPassthruSystem` managed object
"HostProfile"	`HostProfile` managed object
"HostProfileManager"	`HostProfileManager` managed object
"HostServiceSystem"	`HostServiceSystem` managed object
"HostSnmpSystem"	`HostSnmpSystem` managed object
"HostStorageSystem"	`HostStorageSystem` managed object
"HostSystem"	`HostSystem` managed object

Table 5-1. Valid Values for ManagedObjectReference Type Property (Continued)

Type Property	Refers to
"HostVirtualNicManager"	`HostVirtualNicManager` managed object
"HostVMotionSystem"	`HostVMotionSystem` managed object
"HttpNfcLease"	`HttpNfcLease` managed object
"InventoryView"	`InventoryView` managed object
"IpPoolManager"	`IpPoolManager` managed object
"LicenseAssignmentManager"	`LicenseAssignmentManager` managed object
"LicenseManager"	`LicenseManager` managed object
"ListView"	`ListView` managed object
"LocalizationManager"	`LocalizationManager` managed object
"ManagedEntity"	`ManagedEntity` managed object
"ManagedObjectView"	`ManagedObjectView` managed object
"Network"	`Network` managed object
"OptionManager"	`OptionManager` managed object
"OvfManager"	`OvfManager` managed object
"PerformanceManager"	`PerformanceManager` managed object
"Profile"	`Profile` managed object
"ProfileComplianceManager"	`ProfileComplianceManager` managed object
"ProfileManager"	`ProfileManager` managed object
"PropertyCollector"	`PropertyCollector` managed object
"PropertyFilter"	`PropertyFilter` managed object
"ResourcePlanningManager"	`ResourcePlanningManager` managed object
"ResourcePool"	`ResourcePool` managed object
"ScheduledTask"	`ScheduledTask` managed object
"ScheduledTaskManager"	`ScheduledTaskManager` managed object
"SearchIndex"	`SearchIndex` managed object
"ServiceInstance"	`ServiceInstance` managed object
"SessionManager"	`SessionManager` managed object
"Task"	`Task` managed object
"TaskHistoryCollector"	`TaskHistoryCollector` managed object
"TaskManager"	`TaskManager` managed object
"UserDirectory"	`UserDirectory` managed object
"View"	`View` managed object
"ViewManager"	`ViewManager` managed object
"VirtualApp"	`VirtualApp` managed object
"VirtualDiskManager"	`VirtualDiskManager` managed object
"VirtualizationManager"	`VirtualizationManager` managed object
"VirtualMachine"	`VirtualMachine` managed object
"VirtualMachineCompatibilityChecker"	`VirtualMachineCompatibilityChecker` managed object
"VirtualMachineProvisioningChecker"	`VirtualMachineProvisioningChecker` managed object
"VirtualMachineSnapshot"	`VirtualMachineSnapshot` managed object
"VmwareDistributedVirtualSwitch"	`VmwareDistributedVirtualSwitch` managed object

Obtaining Managed Object References

With a reference to a managed object, you can obtain information about the state of the server-side inventory objects and populate client-side objects based on the values. Generally, these are the different ways of obtaining information about objects from the server:

- Use an accessor method if your target has one. The client stubs provide accessor methods for each property of a data object. You can use any of these accessor methods to obtain the values of the object.
- Use a `PropertyCollector` to navigate to a selected point on the server and obtain values from specific properties. See "Obtaining References to Objects and Property Values from the Server" on page 73 for more information about PropertyCollector.
- Using 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.

Do Not Store Managed Object Reference Values

Managed object references must be unique. To ensure that they are, VMware vSphere servers each have their own namespace for managed object reference values that ensures globally unique values throughout the server instance. Managed object references to an object in an ESX/ESXi namespace is different than the managed object reference to the same object in the vCenter Server namespace (assuming the ESX/ESXi instance is being managed by the vCenter Server instance). vCenter Server must ensure that managed object references across all the systems it manages are unique. For these reasons, you must not store a managed object reference from a vCenter Server during one session and expect to use it in a subsequent session.

Working with Data Structures in Client Application Code

Properties contain information about the server-side objects at a given point in time. Each property is defined as a specific datatype. Data types include:

- Simple data types, such as a string, boolean, or integer (or other numeric) datatype. 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`.
- Complex data types. Data objects such as `AboutInfo`, `Action`, and `ServiceContent`, that have been specifically defined for the VMware vSphere object model.

Accessing Property Values

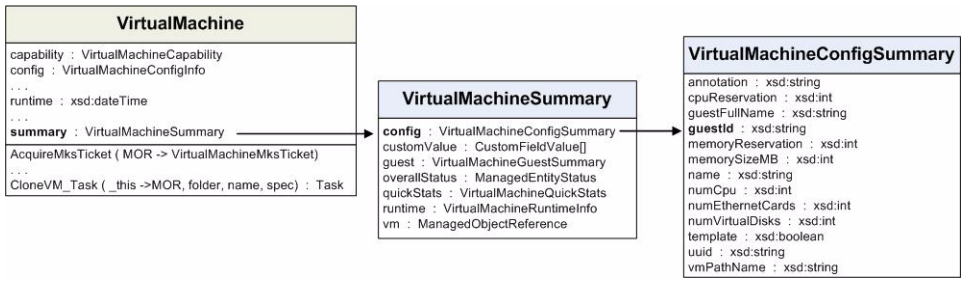
Working with composite data structures requires some understanding of nested properties and how to access the data they contain.

Nested Properties and Property Paths

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

For example, Figure 5-2 shows a UML class diagram of the `VirtualMachine` managed object type, which has a property ("runtime") that is defined as an `xsd:dateTime` data type. `VirtualMachine` also has a property named `summary` that is modeled as a `VirtualMachineSummary` data object. The `VirtualMachineSummary` data object contains a property (`config`) that is modeled as a `VirtualMachineConfigSummary` data object.

Figure 5-2. VirtualMachine Managed Object Type and Nested Properties



To refer to a nested property at any depth in your code, use dot notation between the various object names to define the "path" to the property, being aware to handle the return type correctly. The return type obtained depends on the data type at the end of the path.

For example, the property defined at the path `summary.config.guestId` is a string value, while the property defined as path `summary.config` returns a complete `VirtualMachineSummary` data object.

Table 5-2 shows several examples of how to access some of the nested properties of the `VirtualMachine` managed object type shown in Figure 5-2.

Table 5-2. Nested Properties and Return Datatypes

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

Properties Defined as Key-Based Arrays and Indexed Arrays

Nested properties can also refer to properties that are key-based arrays. For example, `a.b.c["xyz"]` refers to the property `c` that has the key value of `xyz`. An array property is any property whose type is an array. The VMware vSphere data structures include both indexed arrays and key-based arrays.

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

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

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

Accessing the API on the Web Service

The vSphere API is available on ESX/ESXi and vCenter Server systems, as a secure Web service, available over HTTPS at port 443. As is typical of other Web-services client application development projects, developing client applications using the vSphere Web Services SDK requires access to the Web Services Description Language (WSDL) file describing the operations and data types available on the Web service.

The vSphere Web Services SDK includes the WSDL that defines the API.

IMPORTANT The vSphere Web Services SDK ships with Java client-side proxy code that has been generated using the Axis toolkit. If the versions of Java and Axis on your development platform are the same as those used to generate the stubs 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.

The VMware client libraries must be imported into client applications (Table 5-3).

Table 5-3. Import Statements for Including the vSphere API Client Library

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

The API is documented in the *vSphere API Reference*, which is included in the VMware vSphere Web Services SDK package.

Using the vSphere API Reference

The *vSphere API Reference* is the HTML reference documentation generated from the VMware vSphere object model interface definitions. This documentation set is a language-neutral reference that provides complete information about all data structures that make up the VMware vSphere management framework—managed object types, data object types, fault types, enumerations, and associated properties and operations—available through the Web service.

The API reference documentation is available in the `\SDK\doc\ReferenceGuide` subdirectory of the vSphere Web Services SDK package.

The vSphere API identifies deprecated types and methods and provides cross-references to the new property, operation, or type.

Identifying Deprecated Items in the vSphere API Reference

The **deprecated** label on a type, operation, or property documented in the *vSphere API Reference* means that a new, better way to meet a specific requirement is available. For example, VI API 2.5 deprecated the `HostSystem` managed object type's `QueryMemoryOverhead` operation, in favor of a new operation, `QueryMemoryOverheadEx`. Although client applications using deprecated features continue to work, VMware recommends that you use the new type, operation, or property (rather than the deprecation) for new client application development.

Identifying Version Information in the vSphere API Reference

Version information in the *vSphere API Reference* is available starting with API 2.5 and subsequent releases. If the *vSphere API Reference* documentation does not include the version label, you can assume it is supported on API 2.0 and prior releases. New features in this release are identified as "Since vSphere API 4.0," for example. The version information is useful especially for client applications that must target multiple versions of VMware vSphere servers, some of which may support features provided in one version, but not in another.

Mapping the WSDL to the Client-side Proxy Code

Client-application developers use SOAP tools such as Axis WSDL2Java, or Microsoft .NET wsdl.exe to generate client-side proxy code from the WSDL. The generated proxy code handles the remote-method invocation, data-type mapping from server to client, and other such details.

The generated client-side proxy code includes accessor (getter) and mutator (setter) methods for each property defined in server-side data object types.

Table 5-4 shows an example of the method signatures as defined in the Java and C# client proxy code for the `AfterStartupTaskScheduler` data object's `minute` property.

Table 5-4. Example of Generated Accessor and Mutator Methods in Client Proxy Code

Data Object	Property	Type	Java	C#
`AfterStartup TaskScheduler`	`minute`	`xsd:int`	`public int getMinute() { return minute; }` `public void setMinute(int minute) { this.minute = minute; }`	`public int minute { set; get; }`

You can extrapolate from the documentation in the VMware *vSphere API Reference* to identify getter and setter methods that are available to your client application, without examining the *.java, *.cs, or other generated client-side stubs to find the actual methods.

For example, given that the `ManagedObjectReference` data object type has a `type` property, you can safely assume that you can assign a value to a managed object reference by using `setType`, passing one of the valid string values as a parameter. You can obtain the string value by using `getType`.

Table 5-5 shows some valid string values a managed object reference data object can accept or return.

Table 5-5. ManagedObjectReference Data Object Default Methods and Valid String Values

Property (Datatype)	Default Getter Method	Default Setter Method	Valid values
type (string)	`getType`	`setType`	"`Alarm`", `"AlarmManager"`, `"AuthorizationManager"`, `"ClusterComputeResource"`, `"ComputeResource"`, `"ContainerView"`, `"CustomFieldsManager"`, and so on. See complete list of supported values in Table 5-1.

As another example, the `ServiceContent` data object has an `about` property that is defined as an instance of the `AboutInfo` data object type. By examining the API Reference, you see that the `AboutInfo` data object has several other properties, including `apiVersion` and `apiType`, both of which are defined as `xsd:string` data types. These details provide the following information:

- The client stubs contain getter and setter methods for the properties (`getApiVersion`, `setApiVersion`, `getApiType`, `setApiType`).
- The datatype that can be accepted as input to and returned by these methods is a string.
- The methods can be invoked on instances of the `ServiceContent` object.

See Chapter 6, "Client Application Pattern," on page 59 for more information.

Obtaining Values from Nested Properties

Accessing nested properties requires dot-notation from the top-level object to the embedded object (see "Nested Properties and Property Paths" on page 49). The data type of the result depends on the data type at the end of the series. For example, to obtain the value of a specific virtual machine's `powerState`:

```
VirtualMachineRuntimeInfo().powerState()
```

In addition, your code must be setup to handle the specific data type returned by any method invocation or by obtaining the property value. For example, in the series `a.b.c`, if `c` is an integer, your code must be prepared to handle an integer. If `c` is a data object type, then you obtain an instance of that type. If `c` is a `ManagedObjectReference`, you get an instance of a `ManagedObjectReference`.

```
...
private ServiceContent =_sic;
private String myServerVersion;
...
myServerVersion = _sic.getAbout().getApiVersion();
...
```

Mapping XML Datatypes to Java and C# Datatypes

To simplify the presentation, the UML class and object diagrams in this guide 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 5-6 lists some of the more common XML primitive data type mappings.

Table 5-6. 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`

Escape Character Used 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 (`rootFolder` property of `ServiceContent`), denoted by the / (slash) character.

Table 5-7. 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

Invoking Operations through the MOB

This section discusses how to pass values to the server using the MOB. You can use the MOB to navigate the objects on the server and invoke operations. Any changes you make through the MOB take effect on the server. The examples in this section invoke various query operations of the `PerformanceManager` API to demonstrate how to pass primitive data types, arrays, and complex data types (data objects, including managed object references) using the MOB.

CAUTION The MOB is not a read-only mechanism. By clicking the **InvokeMethod** link in the MOB, you might change the server state (depending on the operation). On the other hand, many operations are query-only operations that only return objects from the server.

Input fields for the MOB are presented as single-entry text fields or as scrollable text areas. A text field accepts primitive data types. The scrollable text areas are for entering arrays and complex types. The remainder of this section discusses how to enter values using the MOB.

Passing Primitive Datatypes to Operations through the MOB

The 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`. The API Reference documentation provides this detail. To enter a primitive value in the MOB, enter the value as plain text, without any quotation marks or other markup. For example, to enter an integer value of 10, type `10` in the field.

Figure 5-3 shows an example. To obtain information about the available performance counters at level 4 on the server, enter a `4` in the `level` field of the `QueryPerfCounterByLevel` operation. (This operation is available only on the vCenter Server `PerformanceManager` API, not from an ESX/ESXi system.)

Figure 5-3. Using the MOB to Pass Primitive Datatypes to an Operation, as a Parameter

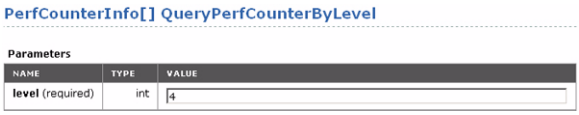
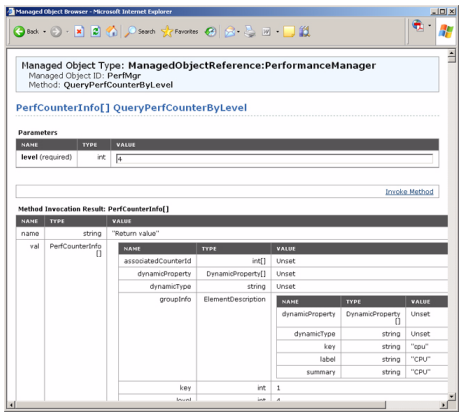


Figure 5-4 shows the results of submitting the query. The array of `PerfCounterInfo` data objects and nested objects, with populated values from the server, displays in the Web browser.

Figure 5-4. Example of Returned Object Displayed in the MOB



Passing Arrays of Primitives to Operations through the MOB

For an array, use the name of the parameter as the name of the property as if it were an XML element, by using opening and closing tags made up of the parameter name. For example, the `QueryPerfCounter` operation of `PerformanceManager` requires an array of integers for the `counterId` parameter. Here's an example:

```
<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 Operations through the MOB

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 `QueryPerfProviderSummary` operation in `PerformanceManager` shows that the operation 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 5-1. XML Schema Definition of ManagedObjectReference Data Object

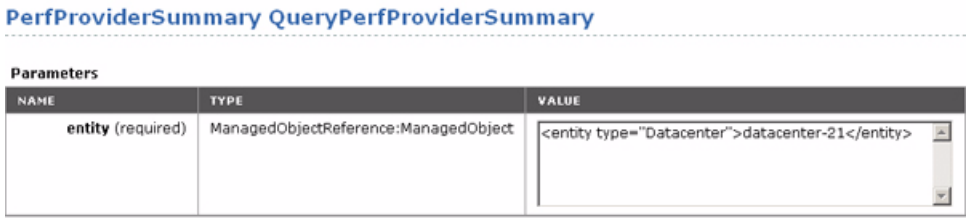
```
<complexType xmlns="http://www.w3.org/2001/XMLSchema" xmlns:vim25="urn:vim25"
               name="ManagedObjectReference">
    <simpleContent>
        <extension base="xsd:string">
            <attribute name="type" type="xsd:string"/>
        </extension>
    </simpleContent>
</complexType>
```

Example 5-1 shows that a managed object reference is defined as a `<SimpleContent>` element that consists of a string that should specify 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 5-5 shows the result of using the definition listed in Example 5-1 to specify the managed object reference for a target datacenter to the `QueryPerfProviderSummary` operation.

Figure 5-5. Using the MOB to Pass Complex Types to an Operation as a Parameter



As another example, one of the parameters required by the `CloneVM_Task` operation (of the `VirtualMachine` managed object) 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 5-1, 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 operation (`entity` type for `QueryPerfProviderSummary` operation in `PerformanceManager`, `folder` type for the `CloneVM_Task` operation in `VirtualMachine`).

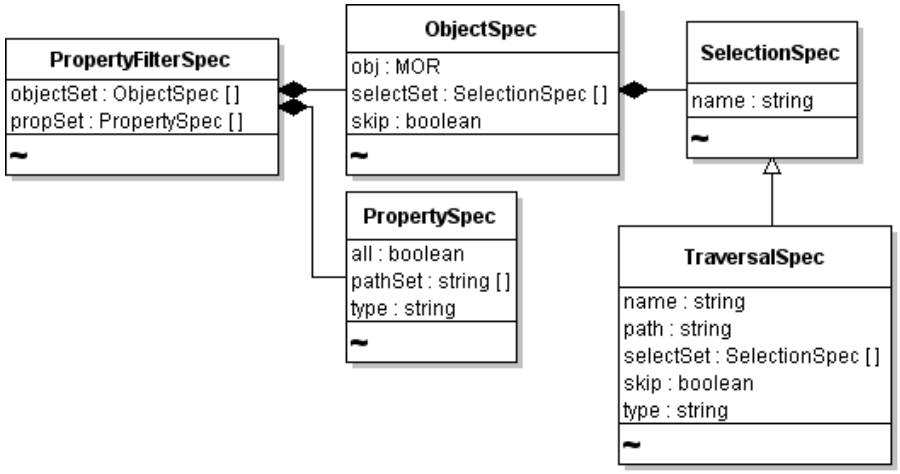
Complex Content

Unlike the `<simpleContent>` shown in Example 5-1, many of the data objects required for the MOB consist of XML Schema elements defined as `<complexContent>` that may encompass many other elements.

For example, the `CreateFilter` operation in `PropertyCollector` has a `spec` parameter that must be defined prior to invoking the operation. The `spec` parameter is defined as an instance of a `PropertyFilterSpec`.

Figure 5-6 shows the relationships among several data objects that `PropertyFilterSpec` consists of.

Figure 5-6. 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 5-2).

Example 5-2 shows that the `PropertyFilterSpec` data object is a `<complexContent>` element that extends the `DynamicData` class with a sequence of two additional properties (`propSet`, `objectSet`), each of which has its own type defined (`PropertySpec` and `ObjectSpec`, respectively).

Example 5-2. 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 5-3 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 denotes 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 5-3. 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 5-4 shows the excerpt.

Example 5-4. 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>
...
```

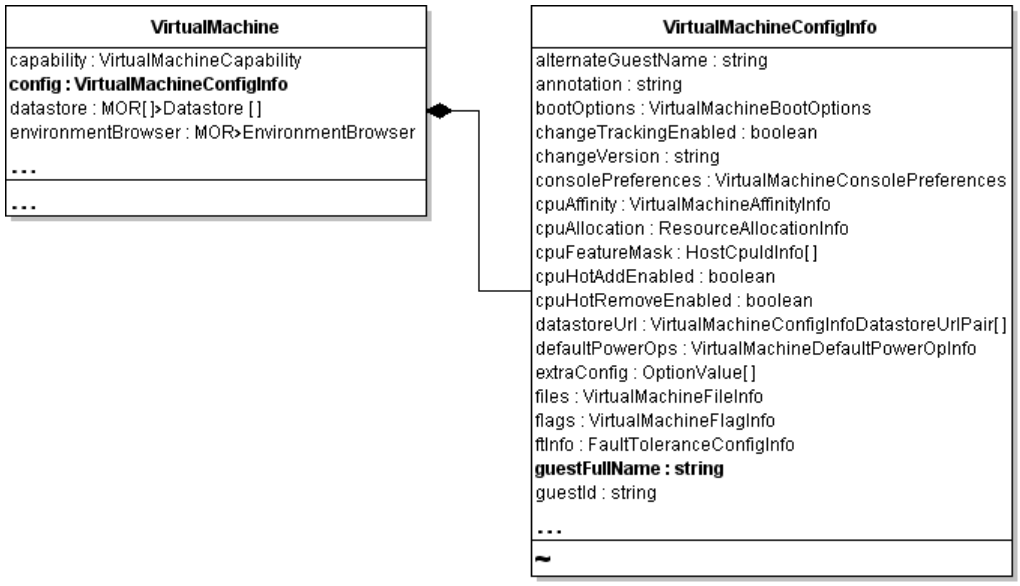
Extrapolating from the WSDL definitions shown in Example 5-2, Example 5-3, and Example 5-4 might produce results similar to those shown in Example 5-5.

Example 5-5. 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` operation. The order of the element tags is as defined in the XML Schema for the property (Example 5-2). The `pathSet` property defines the full path to the nested data object of interest. In Example 5-5, the `pathSet` property defines the path to the `guestFullName` property of the target virtual machine. Figure 5-7 shows the UML of these nested data objects.

Figure 5-7. Nested Data Objects



All these details are presented in the *vSphere API Reference*. By examining the WSDL definition, you can construct the strings needed to submit parameters through the MOB. Table 5-8 provides a brief summary of the steps involved when you use the MOB and the *vSphere API Reference* together.

Table 5-8. Comparison of Datatypes for MOB Usage

Datatype	How to Input Values for Operations
Primitive	Enter the value as plain text regardless of its data type (`int`, `string`, `boolean`). Do not use quotation marks or other mark-up.
Array	Using 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.

Client Application Pattern 6

Any client application that you create using the vSphere Web Services SDK must connect to the server, pass the appropriate user account credentials to authenticate to the server, and obtaining a session from the server. This general pattern is described in this chapter. The chapter includes these topics:

- "Minimal Client Application Pattern" on page 59
- "Overview of a Java Sample Application" on page 61
- "Saving and Reusing a Web Server Session Token or Cookie" on page 63
- "Supporting Multiple API Versions" on page 63
- "Helper Classes for Sample Applications" on page 64

Minimal Client Application Pattern

Regardless of the programming language used for development, client applications that target the vSphere API typically follow this basic pattern:

1. Obtain a session token and create a connection to the Web Service.
2. Instantiate local proxy object for reference to `ServiceInstance`.
3. Login to the server using appropriate credentials.
4. Process the objects.
5. Close the connection.

Example 6-1 provides a short client application that follows this basic pattern.

Connecting to the Web Service

To establish a connection with the Web service, you must first pass in the uniform resource locator (URL) of the Web service, user account, and password, in the connection string.

CAUTION Always go through a vCenter Server system to perform operations on managed ESX/ESXi hosts. Invoking operations on managed hosts directly (rather than through vCenter Server) can cause problems.

Obtaining a Session Token and Creating a Connection Object

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. This session token is returned to the client from the Web service and is then used transparently in the messages sent between client and server.

Table 6-1 shows two examples of session tokens returned from connections to VirtualCenter Server 2.5 and ESX/ESXi 3.5.

Table 6-1. Session Tokens as Strings

vCenter Server	`vmware_soap_session="63869259-F0FF-4DB5-9B3B-6493212AB9CD"`
ESX/ESXi	`vmware_soap_session="52b1910a-31ad-df35-95d1-210f86c55efb"`

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

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

The Web services session token can be saved to a file for later use. For example, if your client application is multithreaded, save the session and use it for each thread in the application. See "Introduction to the Credential Store," in Chapter 7, "Automating Client Application Login," on page 67 for more information.

Logging In to the Server

After obtaining the Web server session token, log in to the server by passing it a valid user account and password, and setting the locale. You can pass a null value for the locale. The result of the login process is a connection object.

NOTE See "Introduction to the Credential Store" on page 67 for information about the vSphere Web Services SDK credential store, which can be used to streamline the authentication process and mitigate the risk that administrators or other users of your client applications might store passwords in scripts or other files, such as cron or Windows Task definitions.

Setting the Locale

The `Login` operation of `SessionManager` has an optional string parameter for locale. Example 6-1, "Simple Java Client Application," on page 61 leaves the value for the locale unset:

```
_service.login(_sic.getSessionManager(), userName, password, null);
```

When this optional parameter is not provided, the operation uses the server default locale.

To set a specific locale, pass the appropriate string value that identifies the two-character ISO-639 language ID (for example, "en"). You can also include an optional two-character ISO 3166 country ID. For example, `en_US`, `de`, `fr_CA`, `zh`, `zh_CN`, or `zh_TW`.

```
_service.login(_sic.getSessionManager(), userName, password, "en");
```

Alternatively, you can use `SetLocale` operation of the `SessionManager` to set the locale in a separate step.

Close the Server Connection

For security reasons, always close your connections to the server when the application completes its work:

```
_service.logout(_sic.getSessionManager());
_service = null;
_sic = null;
```

Overview of a Java Sample Application

Example 6-1 is a complete client application that demonstrates this minimal client pattern. The sample does not use the helper classes discussed in "Helper Classes for Sample Applications," nor does it follow best practices. It does not handle exceptions, for example, and it uses the `SunFakeTrustSocketFactory` class to facilitate the secure sockets layer (SSL) handshake without using a local Java SSL certificate. This technique is recommended for development environments only. Do not use this technique for production applications.

Example 6-1 demonstrates the following common client application tasks:

- Connecting to the server
- Creating a local managed object reference to `ServiceInstance`
- Obtaining some basic information about the server available from the `ServiceContent` data object, without using a `PropertyCollector`.

Example 6-1. Simple Java Client Application

```
import com.vmware.vim25.*;
import java.net.URL;
public class MyClient {
    private ManagedObjectReference _svcRef;
    private VimServiceLocator _locator;
    private VimPortType _service;
    private ServiceContent _sic;
    private String fileName;

    private void createServiceRef() throws Exception {
        _svcRef = new ManagedObjectReference();
        _svcRef.setType("ServiceInstance");
        _svcRef.set_value("ServiceInstance");
        }
    private void connectAndLogin(String hostName, String userName, String password)
                        throws Exception {
        System.setProperty("axis.socketSecureFactory",
                        "org.apache.axis.components.net.SunFakeTrustSocketFactory");
        String url = "https://"+hostName+"/sdk/vimService";

        createServiceRef();
        _locator = new VimServiceLocator();
        _locator.setMaintainSession(true);
        _service = _locator.getVimPort(new URL(url));
        _sic = _service.retrieveServiceContent(_svcRef);
        if (_sic.getSessionManager() != null) {
            _service.login(_sic.getSessionManager(), userName, password, null);
            }
        System.out.println(_sic.getAbout().getFullName());
        System.out.println("API type is " + _sic.getAbout().getApiType());
        System.out.println("API version is " + _sic.getAbout().getApiVersion());
        }
    public static void main(String [] args) throws Exception {
    MyClient obj = new MyClient();
    String serverName = args[0];
    String userName = args[1];
    String password = args[2];
    obj.connectAndLogin(serverName, userName, password);
    }
}
```

Connecting to a server and obtaining a managed object reference requires that an application perform a number of tasks, illustrated inExample 6-1.

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

1 Define a local proxy object for the `vimService` port (`vimServiceLocator`, `VimPortType`). In Java, the variable definitions might look like this:

```
private VimServiceLocator _locator;
private VimPortType _service;
...
_locator = new VimServiceLocator();
_locator.setMaintainSession(true);
...
```

2 Connect to the server, passing the URL, user account, and password.

3 Create a managed object reference to `ServiceInstance`. The `createServiceRef()` method in Example 6-1 creates the managed object reference to `ServiceInstance` and assigns it to `_svcRef`:

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

4 To obtain the `ServiceContent` object, the `_svcRef` managed object reference is used in the invocation on the local service proxy object (`_service`):

```
_sic = _service.retrieveServiceContent(_svcRef);
```

5 After checking that an instance of `SessionManager` exists for the current `ServiceContent` object, the example code invokes the login method on the local proxy object (`_service`). The `_sic.getSessionManager()` effectively passes the managed object reference of the current `SessionManager` instance to the login method, along with the user name and password:

```
if (_sic.getSessionManager() != null) {
    _service.login(_sic.getSessionManager(), userName, password, null);
```

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

6 Obtain the root folder of the inventory, obtain a reference to one or more of the managers (`PerformanceManager`, for example), and perform other tasks defined by your application. In Example 6-1, the only real task performed by the application is obtaining the `about` property from the `ServiceContent` data object and printing out some of the values from the `AboutInfo` data object instance:

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

The compiled class is executed as follows:

```
C:\>java MyClient pubslab01.vmware.com tooluser password
```

Table 6-2 shows the output from runs of Example 6-1 targeting ESX/ESXi and vCenter Server systems.

Table 6-2. Sample Runs of Simple Client Application

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

Saving and Reusing a Web Server Session Token or Cookie

In Example 6-1, the session token is not explicitly handled by the client application. However, after your client application obtains a session token, you can save it to a local file, for re-use later. Example 6-2 shows an example of using Java and Axis client libraries to obtain the session token from the current SOAP message context, and casting it to a `String`. With the session token in `String` form, you can save it locally to a file on the client for subsequent re-use. In your connection setup code, you can check for an existing session file and use that session file when it's appropriate to do so.

Example 6-2. Obtaining Session Token from the Web Services Context

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

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

Supporting Multiple API Versions

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 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 as needed based on the client's version information, exposing to the client the API version supported by 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 6-3).

Example 6-3. Service-Versions File (vimServiceVersions.xml)

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--    Copyright 2008 VMware, Inc.  All rights reserved. -->
- <namespaces version="1.0">
    - <namespace>
          <name>urn:vim25</name>
          <version>4.0</version>
    - <priorVersions>
        <version>2.5u2</version>
        <version>2.5</version>
      </priorVersions>
    </namespace>
    - <namespace>
        <name>urn:vim2</name>
        <version>2.0</version>
      </namespace>
  </namespaces>
```

If you are developing a client application that must support multiple server versions at the same time (ESX/ESXi 4 and ESX/ESXi 3.5, for example), you must obtain information about the API versions that are supported on the server and provide logic in your code to use or not use features, based upon the version information.

The vSphere Web Services SDK package includes several sample applications demonstrating how to work with multiple server versions (Table 6-3).

Table 6-3. Sample Applications that Demonstrate Working with Multiple API Version

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

Identifying the API Version Supported by the Server

One approach to targeting multiple versions of the API from the same client application code is to check for the existence of the server-versions file on the server. If you do not find a vimServiceVersions.xml file on the server, you can assume that the server is not ESX/ESXi 4.0 or vCenter Server 4.0.

Example 6-4 is an excerpt from the `VersionUtil.java` sample that demonstrates this general approach. See the `getTargetNameSpaceAndVersion()` method of `VersionUtil.java` located in the vSphere Web Services SDK package for more information.

Example 6-4. Determining the URL from Which to Obtain the API Version Information

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

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

Helper Classes for Sample Applications

The vSphere Web Services SDK includes many sample applications, written in C# and in Java, that demonstrate features of the vSphere API and object model. The samples rely on several utility classes (or "helper" classes). The helper classes form the basis of console style, command-line applications. These classes (listed in Table 6-4 and Table 6-5) handle command-line input such as common parameters, server name, and other details. These helper classes are located in the unpacked SDK download, in these locations:

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

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

Java Helper Classes

Table 6-4 summarizes the Java versions of the client helper classes.

Table 6-4. Helper Classes for Java Sample Applications

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

C# Helper Classes

Table 6-5 lists the helper classes available for C#.

Table 6-5. Helper Classes for C# Sample Applications

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

Automating Client Application Login 7

Client applications that launch automatically for unattended operation, such as a software agent or cron job, require special treatment to facilitate the logon process. Non-interactive client applications lend themselves to poor security practices. For example, administrators may store passwords in script or batch files, to login automatically.

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 client libraries obviate the need for system administrators to keep passwords in local scripts. This chapter contains information about using credential store libraries, and about creating user accounts to use these tools. The chapter includes these topics:

- "Introduction to the Credential Store" on page 67
- "Overview of the Credential Store Backing File" on page 68
- "Security Best Practices and the Credential Store" on page 69
- "Using the CredentialStoreAdmin Tool" on page 70
- "Using AuthorizationManager to Limit Access" on page 70

Introduction to the Credential Store

The credential store included in the vSphere Web Services SDK enables more advanced client application capabilities, such as automated log in to servers. The SDK includes both C# and Java client libraries (Table 7-1) that you can use in your own applications to simplify the login and authentication process. The credential store includes:

- A persistence file (credential store backing file) that stores authentication credentials. Currently, only passwords are supported. The persistence file maps a remote user account from an ESX/ESXi host to the password for that user on the server.
- C#, Java, and Perl libraries for managing the credential store programmatically. See Table 7-2 for available methods.
- Java and Microsoft PowerShell-based command-line utilities for managing the credential store.

Table 7-1. Credential Store Client API Libraries

Namespace VMware.Security.CredentialStore(C#)	Package com.vmware.security.credstore (Java)
CredentialStoreFactory.cs	CredentialStore.java
ICredentialStore.cs	CredentialStoreFactory.java

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

Table 7-2. Credential Store Client API 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 the user if one already exists 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.

The credential store client APIs use a platform-dependent default file location when the file location is not specified explicitly.

Overview of the Credential Store Backing File

The credential store backing file is a text-based file formatted as XML that is saved locally on the client machine for access at runtime. Unless otherwise specified, the backing file is located in the default location shown in Table 7-3. The credential store persists locally on a per-user basis—each user has his or her own credential store backing file that should be protected by appropriate file access permissions.

CAUTION The credential store backing file uses file-system-level permissions to ensure that passwords remain confidential. Administrators or users who share workstations should set the appropriate file permissions on their respective credential store backing file to ensure that they alone can access the file and its contents.

Credential Store Backing File Structure

Example 7-1 shows the valid XML elements that are read and written to the file.

Example 7-1. 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>
```

Using Samples to Understand the Credential Store

The vSphere Web Services SDK `CreateUser` and `SimpleAgent` sample applications demonstrate the functionality available through the credential store client libraries (Table 7-1).

The `CreateUser` application creates a user account and password based on random-number-generation scheme for the server, and populates the local credential store backing file with this information. If the backing file does not exist, it is created in the default location (Table 7-3).

To run the `CreateUser` application, you must enter the name of the server with an ESX/ESXi administrator account and password. A generated user account name and generated password are created on the server.

```
java com.vmware.samples.simpleagent.CreateUser --server <servername> --url
https://<servername>/sdk --username <adminuser> --password <pwd> --ignorecert ignorecert
```

The `CreateUser` sample prints out as follows upon successful execution:

```
Started
Successfully created user and populated the credential store
Ended CreateUser
```

The `SimpleAgent` sample application demonstrates how the credential store libraries can be used to extract the user account and password at runtime to authenticate a user account, non-interactively. To run the `SimpleAgent` sample:

```
java com.vmware.samples.simpleagent.SimpleAgent <servername>
```

The console should display a message similar to the following:

```
Connected Successfully VMware ESX Server 3.5.0 build-78591
```

CAUTION The `CreateUser` sample application `CreateUser` is for demonstration purposes only and should not be used as a model for production code. `CreateUser` breaks one of the cardinal rules of security best practices: Never give user accounts more privileges than needed do the tasks required of them.

Security Best Practices and the Credential Store

The principle of least privilege should be applied to any agent-like software or automated application that takes advantage of the credential store. This common security tenet dictates that user accounts be given the minimal number of privileges on the system that they require to do their jobs.

Using the Principle of Least Privilege with the Credential Store

To follow this common-sense best practice, VMware recommends that the credential store be used only as described here in a production environment. In development and test environments, security might not be as much of a concern.

To follow security best practices while using the credential store

1. Never grant administrator privileges to a user account associated with an automated script or software agent, especially one that uses the credential store.
2. Create specific roles solely for purposes of using any SDK-based application that uses the credential store. Or, select a pre-defined sample user role if one exists that meets your needs.

   For example, if you are developing an agent-like application to automatically launch the VMware Consolidated Backup utility, you might use the "VMware Consolidated Backup Utility" role (roleID 7).

   If a pre-defined user role that meets the needs of your application does not exist, create a role with only those privileges needed for the application. See Table 4-6, "System and Pre-Defined User Roles," on page 41 for more information about roles.
3. Create a user account expressly for use with the agent or application.

4 Apply the specific role created in step 2 to the user account created in step 3.

5 Store the special user account and password in the credential store, using the `CredentialStoreAdministration` tool as detailed in"Credential Store Administration Tool for Non-Interactive Clients" on page 173.

CAUTION The `CreateUser` sample breaks the principle of least privilege, by granting the user account the Administrator role (-1). Never do this in a production environment.

Using the CredentialStoreAdmin Tool

In addition to the files listed inTable 7-1, the vSphere Web Services SDK also includes an administration tool for creating and managing the credential store. You can use the `CredentialStoreAdmin` tool to examine the contents of the credential store. For example, you can use the `CredentialStoreAdmin` tool to see the generated user accounts and passwords contained in the file.

Table 7-3. Default Credential Store Locations

Client	Default Location
Linux	`$HOME/.vmware/credstore/vicredentials.xml`
Windows Vista	`C:\Users\[User Name]\AppData\Roaming\VMware\credstore\vicredentials.xml`
Windows XP	`C:\Documents and Settings\[User Name]\Application Data\VMware\credstore\vicredentials.xml`
Windows 2000	

If you use the client libraries for the credential store in an application that you create, you must also manage the credential store on the client machines that might run your application. See "Credential Store Administration Tool for Non-Interactive Clients," in Appendix C, "Deploying Client Applications," on page 173 for more information.

Using AuthorizationManager to Limit Access

`AuthorizationManager` provides operations for adding roles and associating permissions with managed entities in a vSphere inventory. You might want to create new roles if the existing roles defined for ESX/ESXi or vCenter Server do not meet your needs.

For example, agent-style client applications that log on and authenticate automatically using the credential store, should be assigned roles that encompass only the privileges needed. If a pre-defined role does not meet this need, you should define a new one that contains only the minimum set of required privileges.

It is also best practice to create specific user accounts for such agent-style client applications and many other client applications that perform administrative tasks. Rather than use the `root` or other account from the administrators group, create a new user account with only the necessary privileges on the server.

Creating New User Accounts

New user accounts for vSphere servers (ESX/ESXi) can be created through the API by using the `HostLocalAccountManager` service interface.

To create a user account on ESX/ESXi through the API

1 Obtain a managed object reference to the `HostLocalAccountManager` of the target system.

2 Create a `HostAccountSpec` data object that defines the properties of the user account, including description and password.

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

3 Invoke the `CreateUserAccount` operation, passing the managed object reference (from step 1) and the `HostAccountSpec` data object (step 2) in the invocation.

   At this point, you can create a group (if it does not yet exist).

4 Create a `HostAccountSpec` data object that defines the properties of the group (description, id). No password is needed.

5 Invoke the `CreateGroup` operation, passing the managed object reference (from step 1) and the `HostAccountSpec` data object (from step 4) in the invocation.

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

6 Invoke the `AssignUserToGroup` operation, passing the managed object reference to the `HostLocalAccountManager`, the `userid`, and the `groupid` in the invocation.

Applying Permission to an Entity

Example 7-2 shows some of the code required to create a new user account and apply a permission to an entity that grants access to the user account based on a role. The role (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`.

Example 7-2. Creating a User Account

```
...
anagedObjectReference _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});
...
```

Obtaining References to Objects and Property Values from the Server

8

For any client application you develop using the vSphere API, you must obtain one or more managed object references to the service interfaces that you want to use, and to specific objects in the inventory that you want to manage or use in your application. The `PropertyCollector` the service interface that you use to obtain references to managed objects, to obtain values of the properties from managed objects, and to monitor changes and update your client-side with new values for server-side objects as needed in the application. This chapter includes these topics:

- "Sample Code Reference" on page 73
- "Introduction to the PropertyCollector" on page 74
- "Understanding the PropertyFilterSpec" on page 75
- "RetrieveProperties Operation" on page 76
- "CreateFilter and Update Operations" on page 77
- "PropertyCollector Performance" on page 79
- "Using the SearchIndex to Obtain Managed Object References" on page 80

Sample Code Reference

Table 8-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 8-1. Sample Applications that Demonstrate PropertyCollector and SearchIndex Operations

Java	C#
SDK\samples\Axis\java\com\vmware\samples\general	SDK\samples\DotNet\cs\
`GetUpdates.java`	`GetUpdates`
`PropertyCollector.java`	`PropertyCollector`
`SearchIndex.java`	`SearchIndex`
`SimpleClient.java`	`SimpleClient`
`TaskList.java`	`TaskList`
	`WatchVM`

The samples listed in Table 8-1 depend on several helper classes that use the `PropertyCollector` extensively to navigate through the inventory, to obtain references and values of properties, and to keep the client-side object state synchronized with the state of server objects. See the classes located in the `AppUtil` subdirectory of the Java and C# samples, respectively, for additional examples of using `PropertyCollector`.

Introduction to the PropertyCollector

Beyond the basic client application pattern introduced in Chapter 6, "Client Application Pattern," on page 59, any client application typically performs many different operations to obtain information about the server objects at any given time. The `PropertyCollector` is a service interface that supports these kinds of tasks from a client application:

- Navigate the inventory and obtain information about specific managed objects from the inventory, such as whether a virtual machine is powered on or not, whether a host in a cluster is off-line or on, and so on.
- Keep data about server state in the client application synchronized with the state of server objects as much as required by the application.

Using a `PropertyCollector` requires some knowledge of the relationships among managed objects. You must understand the general hierarchical arrangement of entities in the inventory before you can create a filter that starts collecting objects or properties at any point in the inventory hierarchy.

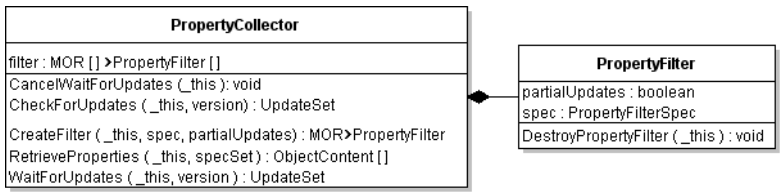
For example, an inventory always starts from a `rootFolder` whose `parent` property value is always `Unset`. Objects in the inventory descend from this `rootFolder` in the order determined by the relationship of parent properties and `childEntity` properties, following the constraints imposed by the underlying managed entity types. Step through some of the samples listed in the "Sample Code Reference" on page 73, including the helper classes, for examples of using `PropertyCollector` to navigate all objects available in the hierarchy.

See Chapter 3, "Understanding the Inventory," on page 25 for more information about inventory entities and their relationships.

PropertyCollector Operations

The `ServiceInstance` of a vSphere server provides each client session with its own instance of the `PropertyCollector`. A `PropertyCollector` has a single property, `filter`, that is defined as an array of references to `PropertyFilter` managed objects.

Figure 8-1. PropertyCollector and PropertyFilter Managed Object Types



The `filter` property is session-specific. It does not exist beyond the session, nor can it be shared with another session. When a session ends, the server destroys any filters associated with the session.

The `PropertyCollector` provides operations for creating filters to obtain specific properties and objects, check for updates on specified objects, and monitor changes to properties on the server. Table 8-2 provides summary information about all the operations available in a `PropertyCollector`.

Table 8-2. PropertyCollector Operations

Operation	Parameter	Type	Description
`CancelWaitForUpdates`	`_this`	`MOR >PropertyCollector`	Cancels an in-progress `WaitForUpdates` operation. Upon successful cancellation, this operation throws a `Cancelled` fault.
`CheckForUpdates`	`_this`	`MOR > PropertyCollector`	Returns an `UpdateSet` data object that contains all requested properties for the specified version.
	`version`	`string`	
`CreateFilter`	`_this`	`MOR > PropertyCollector`	Returns a reference to a `PropertyFilter` managed object that meets the specifications defined in the `spec` parameter.
	`spec`	`PropertyFilterSpec`	
	`partialUpdates`	`boolean`	

Table 8-2. PropertyCollector Operations (Continued)

Operation	Parameter	Type	Description
`RetrieveProperties`	`_this`	`MOR > PropertyCollector`	Returns an array of `ObjectContent` data objects containing managed object references and properties for managed objects.
	`specSet`	`PropertyFilterSpec[]`	
`WaitForUpdates`	`_this`	`MOR > PropertyCollector`	Returns an `UpdateSet` data object containing the changes between the previous version and the current values of the properties.
	`version`	`string`	

Through the operations listed in Table 8-2, `PropertyCollector` supports two general approaches to obtaining objects and properties from the server:

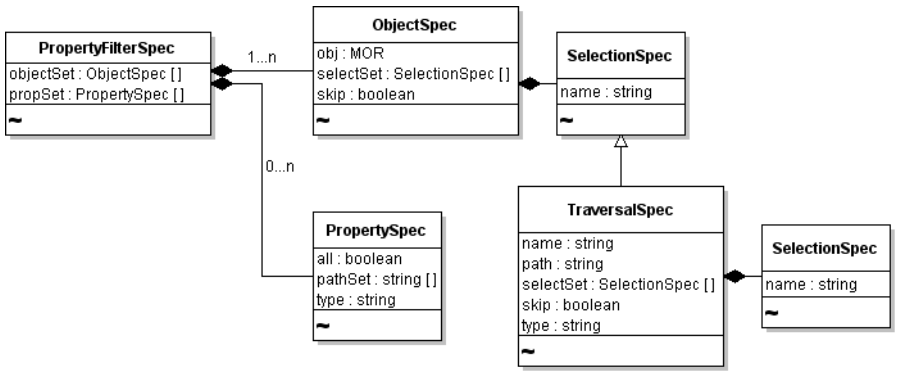
- If you do not need your client application to keep client state synchronized with server state, you use the `RetrieveProperties` operation. The server instantiates a filter, collects your specified objects and properties as a `ObjectContent` data object and returns it to the client application. See "RetrieveProperties Operation" on page 76 for more information.
- If your application must keep client state synchronized with the server state, you use the `CreateFilter` operation in conjunction with `CheckForUpdates` or `WaitForUpdates` operations. See "Keeping Client Data in Synch with Server Objects" on page 78 for more information.

Both approaches require you to create a complex data structure called a `PropertyFilterSpec` that defines the properties and objects you want to retrieve from the server.

Understanding the PropertyFilterSpec

As shown in Figure 8-2, `PropertyFilterSpec` data object has two properties, `objectSet` and `propSet`, defined as instances of an array of `ObjectSpec` data objects and an array of `PropertySpec` data objects, respectively.

Figure 8-2. PropertyFilterSpec and Associated Data Objects



`PropertyFilterSpec` requires at least one instance of an `ObjectSpec`, but the array of `PropertySpec` objects can be empty. You use the `ObjectSpec` to specify the starting point in the inventory that you want the filter to use for the collection. The `obj` property of `ObjectSpec` specifies the type of object (the managed object type name, as a string) to collect from the server.

For the `obj` property, you use any of the valid strings defined for the `type` property of the `ManagedObjectReference` data object. Valid strings are the nearly 60 managed object types, from `Alarm` through `VmwareDistributedVirtualSwitch`." See Table 5-1, "Valid Values for ManagedObjectReference Type Property," on page 46 for the list of valid type strings.

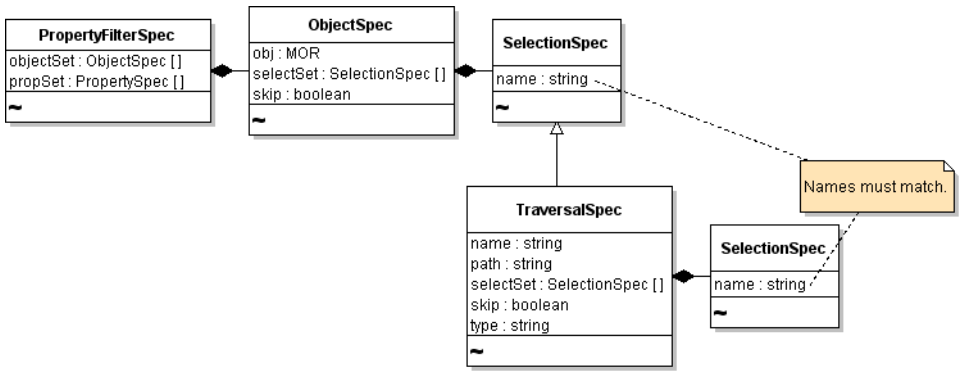
The `obj` property of `ObjectSpec` is defined as an instance of the `ManagedObjectReference` for which you want to start the collection. Using `ObjectSpec` without a `PropertySpec` returns the complete object from the specified location. To return specific properties from the object, you must use `ObjectSpec` in conjunction with one or more `PropertySpec` objects that identifies the properties.

See the *vSphere API Reference* for complete information.

TraversalSpec Data Object and Recursion

The `TraversalSpec` data object type extends the `SelectionSpec` data object with several properties that enable your application to retrieve properties and objects recursively. You can create extensive navigational specifications using the `TraversalSpec` in conjunction with `SelectionSpec` data objects.

Figure 8-3. Nested TraversalSpec



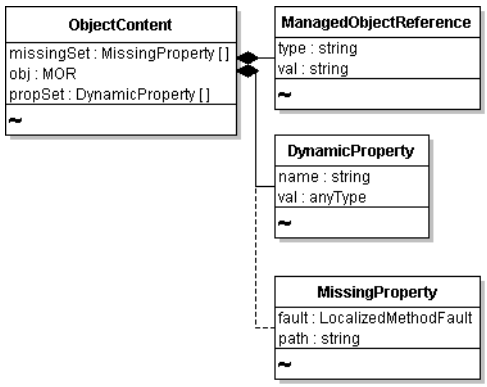
In a series of multiple `TraversalSpec` data objects associated with a `SelectionSpec`, the `name` of the last `TraversalSpec` data object in the series must match the `name` of the `SelectionSpec` (Figure 8-3).

RetrieveProperties Operation

The `RetrieveProperties` operation is designed for one-time or ad hoc use. It collects all the objects and properties you specify in one or more `PropertyFilterSpec` objects. The filter is not added to the array that makes up the `filter` property of `PropertyCollector`, but is instead used immediately by the server to collect the objects and properties specified in your `PropertyFilterSpec` objects, and then destroyed after the server returns the results to your client.

The `RetrieveProperties` operation accepts an array of `PropertyFilterSpec` objects for its `specSet` parameter. The operation returns an `ObjectContent` data object (Figure 8-4).

Figure 8-4. ObjectContent Data Object Returned by RetrieveProperties Operation



In your client code, you obtain the values from this composite data structure and process further in your client application as needed.

Example 8-1 shows an excerpt from the `ServiceUtil.java` helper class contained in the vSphere Web Services SDK samples package.

Example 8-1. Loop that Obtains References from an ObjectContent Data Object

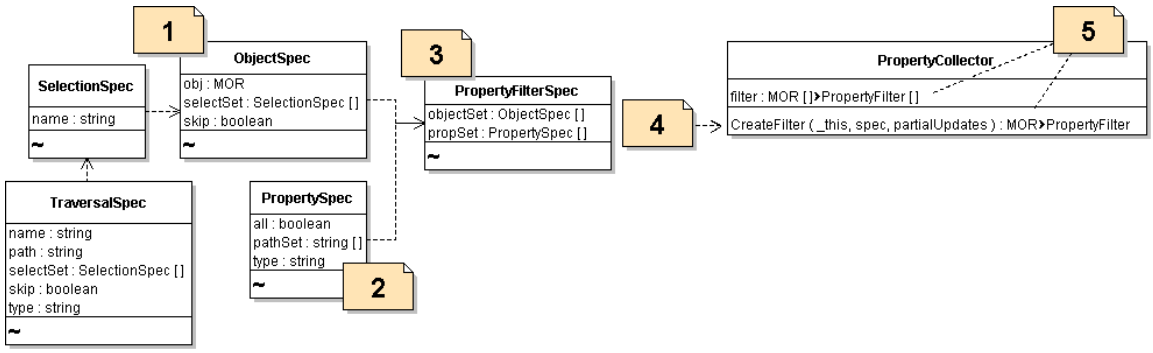
```
...
ObjectContent oc = null;
ManagedObjectReference mor = null;
DynamicProperty[] propary = null;
String propval = null;
boolean found = false;

for (int oci = 0; oci < ocary.length && !found; oci++) {
    oc = ocary[oci];
    mor = oc.getObj();
    propary = oc.getPropSet();
    propval = null;

    if (type == null || typeIsA(type, mor.getType())) {
        if (propary.length > 0) {
            propval = (String)propary[0].getVal();
            }
        found = propval != null && name.equals(propval);
        }
    }

    if (!found) {
        mor = null;
    }
    return mor;
} // close for loop
...
```

CreateFilter and Update Operations

To keep client side objects synchronized with the server objects, you start by using `CreateFilter` to define the filter. You obtain server content using `CheckForUpdates` or `WaitForUpdates`, after your filters exist in your `PropertyCollector`.

The `CreateFilter` operation has only three parameters. However, the `spec` property, defined as an instance of a `PropertyFilterSpec`, consists of several other data objects, most notably, an `ObjectSpec` and a `PropertySpec`. See "Understanding the PropertyFilterSpec" on page 75 for additional background information about these data objects. Figure 8-5 shows the general relationship of these objects as input to `PropertyCollector` `CreateFilter` operation.

Figure 8-5. Using the PropertyCollector



To define the data objects needed to create a filter

1 Identify the starting object for the collection by creating an `ObjectSpec` that specifies the type of object in in the `obj` property of `ObjectSpec`. If you want the collection to include the starting object, set the `skip` property of `ObjectSpec` to `false`.

2 Define the specification for the `PropertySpec`, or pass `null` for the `selectSet` parameter to retrieve complete objects.

3 Instantiate the `PropertyFilterSpec` using the arrays of `ObjectSpec` and `PropertySpec` objects that specify the details you want to gather in the collection.

4 Use the `CreateFilter` operation of `PropertyCollector`, passing the `PropertyFilterSpec` with the reference to the `PropertyCollector`. When you invoke `CreateFilter`, you also pass a value for the `partialUpdate` property:

- Set `partialUpdate` to `false` to obtain complete property values.
- Set `partialUpdate` to `true` to obtain nested property values only.

5 The `PropertyCollector` instantiates the `PropertyFilter` and adds it to the array of filters that make up the filter property on that `PropertyCollector` instance. The operation returns the managed object reference to the filter to your client application.

Keeping Client Data in Synch with Server Objects

When you use the `CreateFilter` operation to specify the objects or properties that you want to work with in your application, the server creates filter objects that meet your specification, populating the filter property of the `PropertyCollector` with the array of references to `PropertyFilter` objects. `CreateFilter` does not return any objects or property values to the client.

To obtain data on the client side, you must use `CheckForUpdates` or `WaitForUpdates` to obtain `UpdateSet` objects from the server, using the filter or filters defined during the session with the `CreateFilter` operation.

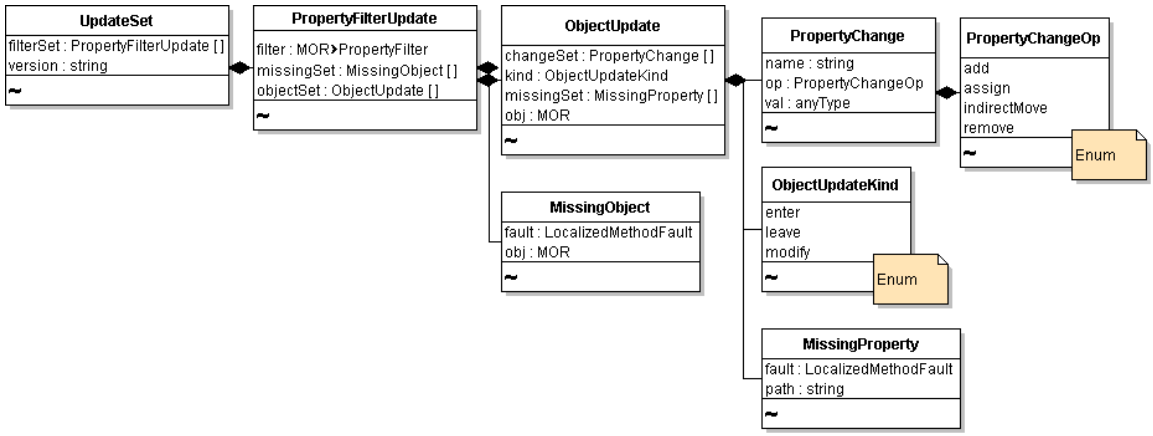
Table 8-3 lists some of the advantages and disadvantages of these two operations.

Table 8-3. PropertyCollector Operations Compared

Operation	Advantages	Disadvantages
`CheckForUpdates`	Polling mechanism that returns only properties that have changed since the version specified. Returns changed data only, so 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.
`WaitForUpdates`	Notification mechanism that blocks until an update occurs. Efficient use of network resources. The only operation of the three 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.

Both operations return `UpdateSet` data objects, the composite data structure shown in Figure 8-6. Both `CheckForUpdates` and `WaitForUpdates` apply the complete array of filters defined for the `PropertyCollector` to the server and return an `UpdateSet` object.

Figure 8-6. UpdateSet Data Object Returned by CheckForUpdates and WaitForUpdates Operations



Both `CheckForUpdates` and `WaitForUpdates` return only changed objects and properties based on the union of all filters associated with the `PropertyCollector` for the session.

The first time you invoke either of these operations after creating your filter, use an empty string (“”) for the `version` parameter, to obtain a complete set of results. The result returned contains a new version number that you use in the next call.

See the `GetUpdates.java`, `GetUpdates.cs`, `ServiceUtil.java`, and `ServiceUtil.cs` samples in the vSphere Web Services SDK package for some examples of using these operations.

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 is supporting across all sessions on the server.

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

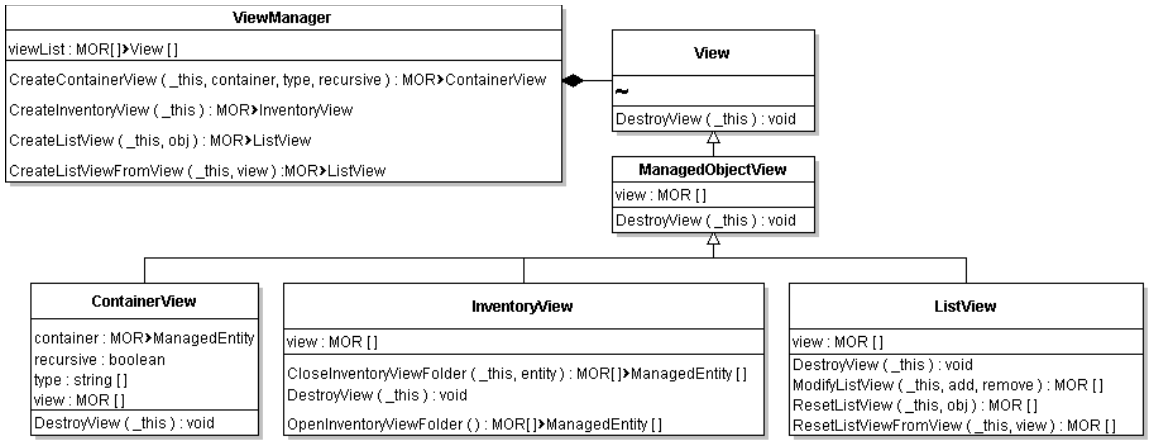
Consider using the `ViewManager` and `View` objects in conjunction with `PropertyCollector` to minimize both the overhead on `PropertyCollector` and the amount of network traffic for your client application.
See “Improving Collection Performance by Using the ViewManager and View Objects” on page 79.

Improving Collection Performance by Using the ViewManager and View Objects

The `ViewManager` service interface supports client access to server managed object content. You use the `ViewManager` to create customized objects that represent subsets, or views, of specific instances of select managed object types on the server.

The `ViewManager` provides operations for creating several different kinds of views. These views extend from the `View` and `ManagedObjectView` base types (see Figure 8-7).

Figure 8-7. ViewManager and View Managed Object Types and View Subtypes



You use the appropriate create operation of `ViewManager` to create a view. As shown in Figure 8-7, view types include `ContainerView`, `InventoryView`, and `ListView`.

For example, you use the `CreateContainerView` to create a `ContainerView`. The `ContainerView` enables you to monitor objects associated with any of the container objects supported on the server, including `Folder`, `Datacenter`, `ComputeResource`, `HostSystem`, `ResourcePool`, and `VirtualApp` objects. You use the `CreateInventoryView` to create an `InventoryView`.

The `InventoryView` provides operations for opening and closing folders on an inventory. Its `view` property contains an array of the managed object references to the objects that make up the inventory. As with `PropertyCollector`, the lifetime of any `View` object is the duration of the session, or until the object is explicitly destroyed, by invoking its `DestroyView` operation.

After you create the appropriate `View` objects for your application, instead of using a `PropertyCollector` with the entire server structure, you use `PropertyCollector` with the views. The appropriate views of server objects enable you to create much simpler `PropertyFilterSpec` objects to define your filters, and the back and forth communication between the client and server is more efficient.

See the *vSphere API Reference* for more information about `ViewManager`, about `View` managed object types and subtypes, and about how to use these objects in conjunction with `PropertyCollector`.

Using the SearchIndex to Obtain Managed Object References

The `SearchIndex` managed object is the service interface for retrieving managed entities by using specific property values, such as inventory path, datastore path, DNS name, and various other managed entity property values.

For example, if you know the IP address of a virtual machine, you can obtain its managed object reference by using the `FindByIp` operation of `SearchIndex`. You can use the `PropertyCollector` to first obtain the value of a specific property, and then use that value as input to `SearchIndex`. See the sample applications `SearchIndex.java` and `SearchIndex.cs` for more information about using `SearchIndex`.

Figure 8-8. PropertyCollector and SearchIndex Managed Objects

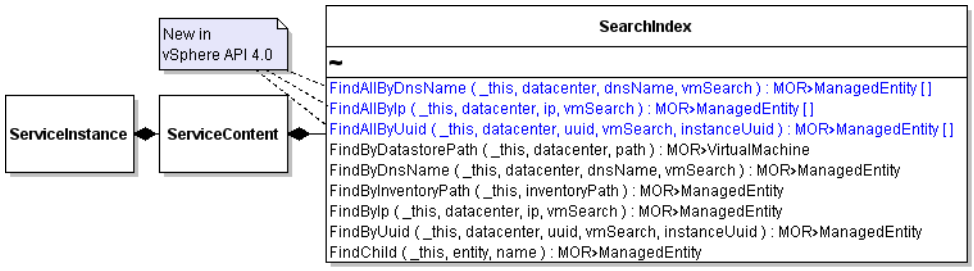


Table 8-4 lists all `SearchIndex` operations. With the exception of the `FindChild` and `FindByInventoryPath` operations, `SearchIndex` operations accept an optional `datacenter`, to limit the search to a specific `Datacenter` in the inventory. Unmanaged ESX/ESXi systems have a single datacenter, `ha-datacenter`.

Table 8-4. SearchIndex Operations

Operation	Parameter	Type	Description
`FindAllByDnsName`	`_this`	`MOR>SearchIndex`	Returns an array of references to managed entities that match the fully qualified domain name of the host or virtual machine, as configured using VMware Tools. If no entities match the `dnsName` submitted, returns an empty array.
	`datacenter`	`MOR>Datacenter`	
	`dnsName`	`string`	
	`vmSearch`	`boolean`	
`FindAllByIp`	`_this`	`MOR>SearchIndex`	Returns an array of references to managed entities that match the IP address of the host or virtual machine, as configured using VMware Tools. If no entities match the IP address, returns an empty array.
	`datacenter`	`MOR>Datacenter`	
	`ip`	`string`	
	`vmSearch`	`boolean`	
`FindAllByUuid`	`_this`	`MOR>SearchIndex`	Returns an array of references to managed entities that match the BIOS UUID specified. Set `vmSearch` to `true` to include virtual machines in the search results. If `vmSearch` is set to true, you can also set `instanceUuid` to `true` to use virtual machine instance UUID in the search results.
	`datacenter`	`MOR>Datacenter`	
	`uuid`	`string`	
	`vmSearch`	`boolean`	
	`instanceUuid`	`string`	
`FindByDatastorePath`	`_this`	`MOR>SearchIndex`	Returns a reference to a virtual machine at the specified datastore location. The path is the location of the `.vmx` file that comprises the virtual machine.
	`datacenter`	`MOR>Datacenter`	
	`path`	`string`	
`FindByDnsName`	`_this`	`MOR>SearchIndex`	Returns a reference to the managed entity that matches the fully qualified domain name of the host or virtual machine, as configured using VMware Tools. Set `vmSearch` to `true` to search for virtual machines. Otherwise, this operation returns hosts.
	`datacenter`	`MOR>Datacenter`	
	`dnsName`	`string`	
	`vmSearch`	`boolean`	
`FindByInventoryPath`	`_this`	`MOR>SearchIndex`	Returns a reference to the managed entity at the specified location in the inventory. Inventory paths must include the containing folders. For example, the inventory path to a virtual machine on an unmanaged ESX/ESXi host must include the containing default datacenter folder and the default `vmFolder`, as in this example: `ha-datacenter/vm/pubslab-linux`
	`inventoryPath`	`string`	
`FindByIp`	`_this`	`MOR>SearchIndex`	Returns a reference to the managed entity that matches the IP address of the host or virtual machine, as configured using VMware Tools. Set `vmSearch` to `true` to include virtual machines in the search. Otherwise, search includes hosts only.
	`datacenter`	`MOR>Datacenter`	
	`ip`	`string`	
	`vmSearch`	`boolean`	

Table 8-4. SearchIndex Operations (Continued)

Operation	Parameter	Type	Description
FindByUuid	_this	MOR>SearchIndex	Returns a reference to the managed entity that match the BIOS UUID specified. Set vmSearch to true to include virtual machines in the search results. If vmSearch is set to true, you can also set instanceUuid to true to use virtual machine instance UUID in the search results.
	datacenter	MOR>Datacenter	
	uuid	string	
	vmSearch	boolean	
	instanceUuid	boolean	
FindChild	_this	MOR>SearchIndex	Returns a reference to the managed entity that matches the name of the child object. The search includes only the immediate children, such as the datastoreFolder, hostFolder, networkFolder, and vmFolder of a Datacenter.
	entity	MOR>ManagedEntity	
	name	string	

See the *vSphere API Reference* for more information about these operations.

Overview of the Task Infrastructure 9

As discussed in "Asynchronous Client-Server Application Style" on page 46, many of the operations available through the vSphere API are asynchronous. Rather than blocking other callers while executing to completion, asynchronous operations return a reference to a `Task` managed object. Operations associated with an ESX/ESXi host and its virtual machines also generate one or more `Event` data objects that provide status, source, and other information about the operation. This chapter provides an overview of the `Task` and `TaskManager` managed objects as fundamental background for the programming model, and an introduction to `Event` data objects. The chapter includes these topics:

- "Sample Code Reference" on page 83
- "Understanding TaskManager and Task Managed Objects" on page 84
- "Understanding the TaskInfo Data Object" on page 85
- "Operations That Return Task Managed Objects" on page 90
- "Introduction to the Event Data Object" on page 92

NOTE In addition to the `TaskManager` and `Task` managed objects discussed in this chapter, vCenter Server provides a `ScheduledTaskManager` that supports scheduling operations in advance. See Chapter 16, "Scheduling vCenter Server Operations," on page 143 for information about scheduling server operations using the `ScheduledTaskManager`.

Sample Code Reference

Table 9-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 9-1. Sample Applications

Java SDK\samples\Axis\java\com\vmware\samples	C# SDK\samples\DotNet\cs\
`general\TaskList.java`	`TaskList`
`events\EventFormat.java`	`EventFormat`
`events\EventHistoryCollectorMonitor.java`	`EventHistoryCollectorMonitor`
`events\VMEventHistoryCollectorMonitor.java`	`VMEventHistoryCollectorMonitor`

In addition to the samples listed in Table 9-1, the helper classes associated with the sample applications use `Task` objects extensively.

Understanding TaskManager and Task Managed Objects

The TaskManager is used extensively by vSphere servers to support asynchronous operations. At runtime, various subsystems use TaskManager to instantiate Task objects in response to operation invocations from client applications.

NOTE Task and Event objects in the context of an unmanaged ESX/ESXi system are non-persistent. See Chapter 14, "Event and Task Management Using vCenter Server," on page 129 for information about tasks and events from multiple ESX/ESXi systems that are stored and managed by a vCenter Server system.

Although many operations available through the API are asynchronous and return a reference to a Task object, some operations are synchronous and do not. For example, operations available through the PerformanceManager and several other service interfaces do not return Task objects. Only those operations that include the suffix _Task in their names return Task references. See Table 9-5, "Operations that Return a Reference to a Task Managed Object," on page 90 for a list of all vSphere API 4.0 operations that return references to Task objects.

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 various 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.

A Task and its associated objects are session specific. The user account associated with the session is limited to obtaining information about the Task objects that it is authorized to view.

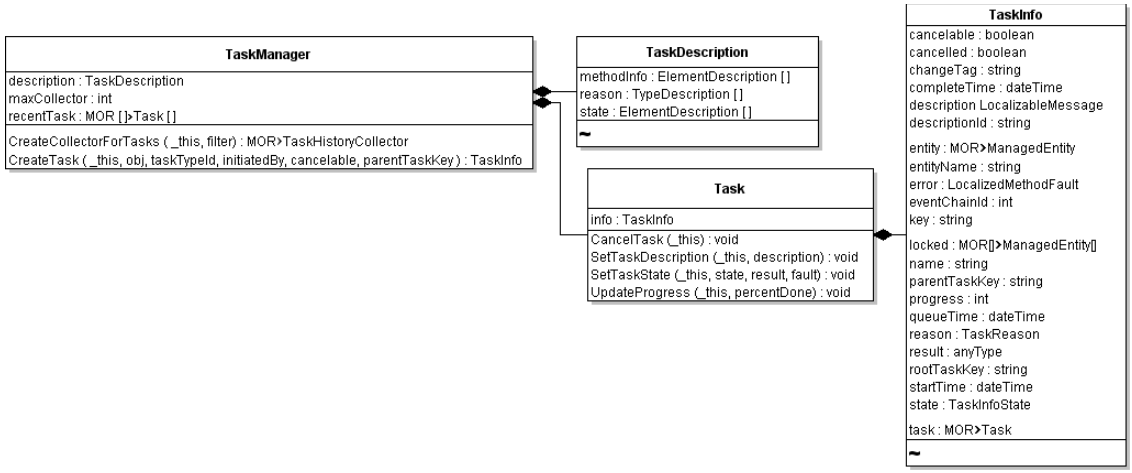
The TaskManager encompasses the properties listed in Table 9-2.

Table 9-2. TaskManager Properties

Property	Type	Description
description	TaskDescription	TaskDescription includes a methodInfo property that contains a key-based array used by TaskManager 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 include methodInfo["Folder.createVm"] and methodInfo["Folder.createClusterEx"].
maxCollector	int	Identifies the number of TaskHistoryCollector objects per client session that the server can create. This property is set by the server. ■ For ESX/ESXi, maxCollector is set to 0. A TaskHistoryCollector cannot be created on an unmanaged ESX/ESXi system. ■ For vCenter Server, maxCollector is set to 32, which specifies the number of TaskHistoryCollector objects per client session supported by the server. See "Obtaining Information about Events Using a HistoryCollector" on page 133.
recentTask	MOR[ ]>Task [ ]	Contains an array of managed object references Task objects that meet any of the following conditions: ■ The Task is queued to run. ■ The Task is running. ■ The Task completed within the past 10 minutes.

Figure 9-1 shows a UML class diagram for `TaskManager` and its associated objects.

Figure 9-1. TaskManager and Task Managed Object Types



In addition to the properties listed in Table 9-2, `TaskManager` has the following operations:

- `CreateCollectorForTasks` operation. You can use the `CreateCollectorForTasks` operation to create an object that contains all Tasks from the vCenter Server database that meet specific criteria. See "Obtaining Information about Events Using a HistoryCollector" on page 133.
- `CreateTask` operation. The `CreateTask` operation is used by all other operations available through the API, to create a managed object reference to a `Task` object. Developers creating extensions can also use the `TaskManager` to create `Task` objects for the operations supported by their extension.

Understanding the TaskInfo Data Object

The `TaskInfo` data object makes up the `info` property of a specific `Task` object. An instance of a specific `TaskInfo` object contains information about the `Task` returned by the server to your client application. Table 9-3 lists the properties of `TaskInfo` data object.

Table 9-3. TaskInfo Data Object Properties

Property	Data Type	Description
`cancelable`	`boolean`	Indicates whether the `Task` can be cancelled (true) or not (false).
`cancelled`	`boolean`	Indicates that the client has attempted to cancel (true) the `Task`.
`changeTag`	`string`	User-entered tag that identifies operations and their side effects.
`completeTime`	`dateTime`	Timestamp that identifies the date and time that the `Task` completed. Indicates neither success nor failure.
`descriptionId`	`string`	Identifies the operation, including publicly visible internal tasks. Returns values from the `TaskDescription` data object `methodInfo` property.
`entity`	`MOR>ManagedEntity`	Managed entity to which the operation applies.
`entityName`	`string`	Locale-specific managed-entity name retained for the `HistoryCollector` database.
`error`	`LocalizedMethodFault`	When `state` property is set to `fault`, identifies the fault. Otherwise, not set.
`eventChainId`	`int`	Event chain ID that leads to the associated `Task` objects.
`key`	`string`	Unique key for the `Task`.
`locked`	`MOR[ ]>ManagedEntity[]`	When the `state` property is "running," this property identifies the list of managed entities (if any) locked by the operation (using a shared lock). Otherwise, not set.

Table 9-3. TaskInfo Data Object Properties (Continued)

Property	Data Type	Description
name	string	Name of the operation that created the Task. Not set for internal tasks.
parentTaskKey	string	Tasks can be created by another Task. Identifies the key of the Task that spawned this Task. Identifies causality between Task objects.
progress	int	When state property is set to running, identifies percentage of Task completed, from 0 to 100. Otherwise, not set.
queueTime	dateTime	Timestamp that identifies date and time the Task was created.
reason	TaskReason	Source responsible for creating the Task. The source can be an alarm, schedule, system, or user.
result	anyType	When state property is set to success, contains the return value of the completed operation. Otherwise, not set.
rootTaskKey	string	A Task can create new tasks, to multiple levels. The rootTaskKey identifies the key of the Task that started the chain of tasks.
startTime	dateTime	Time stamp that indicates date and time the Task started running.
state	TaskInfoState	Runtime status of the Task specified as one the following: ■ error ■ queued ■ running ■ success
task	MOR>Task	Managed object reference to Task.

The result property of the TaskInfo data object (listed in Table 9-3 and shown in Figure 9-2) is defined as anyType. When the Task is instantiated by the server, the value of the result property is initialized as Unset. Only upon successful completion of an operation is the result property populated with the actual return type specific to the operation. Because the result property is defined as anyType, 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, the AddHost_Task (of the ClusterComputeResource managed object) returns a Task object whose info property contains a TaskInfo data object. At the start of the operation, the result property is Unset. Upon successful completion of the operation, the result property of TaskInfo contains the managed object reference of the newly added HostSystem. In the example of the CreateVM_Task shown in Table 9-4, the result is a managed object reference to a VirtualMachine whose managed object id is 64.

See the *vSphere API Reference* for details about the return type instantiated upon completion of the specific Task associated with the invoking operation.

Figure 9-2. The Result Property of a TaskInfo Data Object Can Contain Any Type

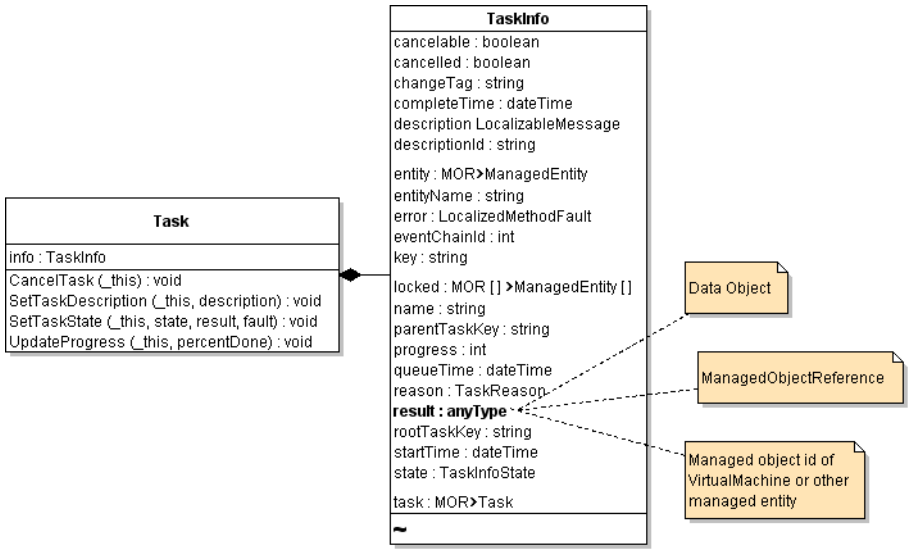


Table 9-4 lists the values obtained from a `TaskInfo` data object at the beginning and the end of the `Task` instantiated by the `CreateVM_Task` operation for creating a new virtual machine.

Table 9-4. Sample TaskInfo Values from an ESX/ESXi 4.0 System During Creation of a VirtualMachine

		Sample Values	
Property	**Datatype**	**Start of Task**	**End of Task**
cancelable	boolean	false	false
cancelled	boolean	false	false
changeTag	string	Unset	Unset
completeTime	dateTime	Unset	"2009-02-19T22:53:35.015338Z"
description	LocalizableMessage	Unset	Unset
descriptionId	string	"Folder.createVm"	"Folder.createVm"
dynamicProperty	DynamicProperty[]	Unset	Unset
dynamicType	string	Unset	Unset
entity	ManagedObjectReference:Folder	ha-folder-vm	ha-folder-vm
entityName	string	"vm"	"vm"
error	MethodFault	Unset	Unset
eventChainId	int	159740	159740
key	string	"haTask-ha-folder-vm-vim.Folder.createVm-159740"	"haTask-ha-folder-vm-vim.Folder.createVm-159740"
locked	MOR>ManagedEntity[]	Unset	Unset
name	string	"vim.Folder.createVm"	"vim.Folder.createVm"
parentTaskKey	string	Unset	Unset
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
rootTaskKey	string	Unset	Unset
startTime	dateTime	"2009-02-19T22:50:39.111604Z"	"2009-02-19T22:50:39.111604Z"

Table 9-4. Sample TaskInfo Values from an ESX/ESXi 4.0 System During Creation of a VirtualMachine (Continued)

		Sample Values	
Property	**Datatype**	**Start of Task**	**End of Task**
`state`	`TaskInfoState`	`"running"`	`"success"`
`task`	`MOR>Task`	`haTask-ha-folder-vm-vim.Folder.createVm-159740`	`haTask-ha-folder-vm-vim.Folder.createVm-159740`

Monitoring TaskInfo Properties

The values of `TaskInfo` properties change over time as the `Task` runs to completion. Depending on the requirements of your application, you might monitor one or more of the properties of `TaskInfo`.

For example, you can check the values of `startTime`, `queueTime`, `completeTime`, `progress`, `result`, and `state` as the operation progresses until completion. 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). You must also handle possible exceptions. For example, in addition to `success`, `queued`, and `running`, an operation can enter an `error` state for any number of reasons. In your code, you must handle errors and exceptions.

To monitor the state of the `Task`, use the `CheckForUpdates` operation of the `PropertyCollector`. See "Keeping Client Data in Synch with Server Objects" on page 78. 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, so you can discard the reference to it your application no longer needs the information.

Cancelling a Task

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

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

However, only a `Task` that has its `cancelable` property set to `true` and its `state` property set to `running` can be cancelled. The operation that initiates the `Task` sets the value of `cancelable` when it creates the `Task`. For example, the `CreateVM_Task` cannot be cancelled. As shown in Table 9-4, the value of the `cancelable` boolean is `false`. 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`.

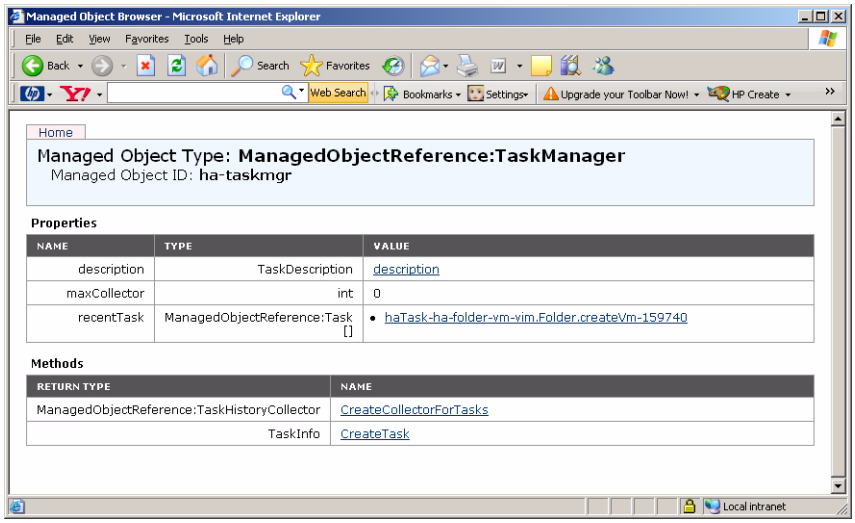
Using the TaskManager to Obtain Information about Tasks

From the moment a `Task` is queued to run until 10 minutes after it completes, a reference to the `Task` is held in the `recentTask` array of `TaskManager`. You can use this property to obtain information about the `Task` objects running on the system.

The vSphere Client application uses the `recentTask` array in this way, for example. When connected to an ESX/ESXi system, the vSphere Client displays queued, running, and completed tasks in the Recent Tasks pane. Elements from this array also display in the MOB. You can also view recent tasks in the MOB, by navigating to the `TaskManager` (on an ESX/ESXi system) using the **ServiceInstance > ServiceContent > ha-taskmgr** link.

Figure 9-3 shows an example of the `TaskManager` displayed in the MOB of an ESX 4.0 system during the `CreateVM_Task` operation. The array contains only one element, the reference to the `Task` for creating a virtual machine. Clicking any of the `recentTask` links in the MOB displays the reference to the `Task` managed object whose `info` property contains a link to the `TaskInfo` data object populated with the details about the `Task` operation, such as those listed in Table 9-4.

Figure 9-3. TaskManager as Displayed in an ESX System MOB



All completed `Task` objects age out of the `recentTasks` array after 10 minutes. However, the vCenter Server collects information about `Task` objects for all the hosts that it manages. You can obtain information about `Task` objects from vCenter Server using a `TaskHistoryCollector`. See "Obtaining Information about Events Using a HistoryCollector" on page 133.

Obtaining Information about Recent Tasks Using a PropertyCollector

To obtain the list of recent tasks programmatically, use a `PropertyCollector` to obtain references to the `TaskManager` and to all `Task` objects from the `recentTask` property of the `TaskManager`. Example 9-1 shows an excerpt from the `TaskList.java` sample that creates the `ObjectSpec`, `PropertySpec`, and a `TraversalSpec` to obtain references to all `Task` objects on the server from the `TaskList`.

Example 9-1. PropertyFilterSpec Definition to Obtain recentTask Property Values

```
private PropertyFilterSpec[] createPFSForRecentTasks(ManagedObjectReference taskManagerRef) {
    PropertySpec pSpec = new PropertySpec();
        pSpec.setAll(Boolean.FALSE);
        pSpec.setType("Task");
        pSpec.setPathSet(new String[] {"info.entity", "info.entityName", "info.name",
            "info.state", "info.cancelled", "info.error"});
    ObjectSpec oSpec = new ObjectSpec();
        oSpec.setObj(taskManagerRef);
        oSpec.setSkip(Boolean.FALSE);
    TraversalSpec tSpec = new TraversalSpec();
        tSpec.setType("TaskManager");
        tSpec.setPath("recentTask");
        tSpec.setSkip(Boolean.FALSE);
    oSpec.setSelectSet(new SelectionSpec[]{tSpec});
    PropertyFilterSpec pfSpec = new PropertyFilterSpec();
        pfSpec.setPropSet(new PropertySpec[]{pSpec});
        pfSpec.setObjectSet(new ObjectSpec[]{oSpec});
    return new PropertyFilterSpec[]{pfSpec};
}
```

Example 9-2 shows an excerpt of some of the code that obtains values for the `info` property from each `Task` object in the array.

Example 9-2. Displaying TaskInfoState Values for Each Task in the 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 9-3 shows output from a run of the program. See the source code listing for `TaskList.java` or for `TaskList.cs` in the vSphere Web Services SDK package for details.

Example 9-3. Sample Run of the TaskList Java Application

```
java com.vmware.samples.general.TaskList --url https://srv/sdk --username root --password *******
Started
Task
Operation AcquireCimServicesTicket
Name srv
Type HostSystem
State -Success
Error
======================
Ended TaskList
```

Operations That Return Task Managed Objects

Table 9-5 lists the operations available in vSphere API 4.0 that return a reference to a `Task` managed object. See the *vSphere API Reference* for more information about each of these operations.

Table 9-5. Operations that Return a Reference to a Task Managed Object

`AddHost_Task`	`PowerDownHostToStandBy_Task`
`AddStandaloneHost_Task`	`PowerOffVApp_Task`
`CheckCompatibility_Task`	`PowerOffVM_Task`

Table 9-5. Operations that Return a Reference to a Task Managed Object (Continued)

CheckHostPatch_Task	PowerOnMultiVM_Task
CheckMigrate_Task	PowerOnVApp_Task
CheckRelocate_Task	PowerOnVM_Task
CloneVApp_Task	PowerUpHostFromStandBy_Task
CloneVM_Task	ProfileCheckCompliance_Task
ComplianceManagerCheckCompliance_Task	PromoteDisks_Task
CopyDatastoreFile_Task	QueryHostPatch_Task
CopyVirtualDisk_Task	QueryVMotionCompatibilityEx_Task
CreateChildVM_Task	RebootHost_Task
CreateDVS_Task	ReconfigureCluster_Task
CreateScreenshot_Task	ReconfigureComputeResource_Task
CreateSecondaryVM_Task	ReconfigureHostForDAS_Task
CreateSnapshot_Task	ReconfigurePort_Task
CreateVirtualDisk_Task	ReconfigVM_Task
CreateVM_Task	ReconnectHost_Task
CustomizeVM_Task	RegisterChildVM_Task
DefragmentVirtualDisk_Task	RegisterVM_Task
DeleteDatastoreFile_Task	RelocateVM_Task
DeleteVirtualDisk_Task	RemoveAllSnapshots_Task
Destroy_Task	RemoveSnapshot_Task
DisableSecondaryVM_Task	Rename_Task
DisconnectHost_Task	ResetVM_Task
DVPortgroupReconfigure_Task	ResignatureUnresolvedVmfsVolume_Task
DVSAddPortgroups_Task	RevertToCurrentSnapshot_Task
DVSMerge_Task	RevertToSnapshot_Task
DVSMovePort_Task	ScanHostPatch_Task
DVSPerformProductSpecOperation_Task	ScanHostPatchV2_Task
DVSReconfigure_Task	SearchDatastore_Task
DVSRectifyHost_Task	SearchDatastoreSubFolders_Task
EagerZeroVirtualDisk_Task	ShrinkVirtualDisk_Task
EnableSecondaryVM_Task	ShutdownHost_Task
EnterMaintenanceMode_Task	StageHostPatch_Task
ExitMaintenanceMode_Task	StartRecording_Task
ExtendVirtualDisk_Task	StartReplaying_Task
GenerateLogBundles_Task	StopRecording_Task
HostProfileApply_Task	StopReplaying_Task
InflateVirtualDisk_Task	SuspendVM_Task
InstallHostPatch_Task	TerminateFaultTolerantVM_Task
InstallHostPatchV2_Task	TurnOffFaultToleranceForVM_Task
MakePrimaryVM_Task	UninstallHostPatch_Task
MigrateVM_Task	UnregisterAndDestroy_Task
MoveDatastoreFile_Task	unregisterVApp_Task

Table 9-5. Operations that Return a Reference to a Task Managed Object (Continued)

MoveHostInto_Task	UpgradeTools_Task
MoveInto_Task	UpgradeVM_Task
MoveIntoFolder_Task	ZeroFillVirtualDisk_Task
MoveVirtualDisk_Task	~

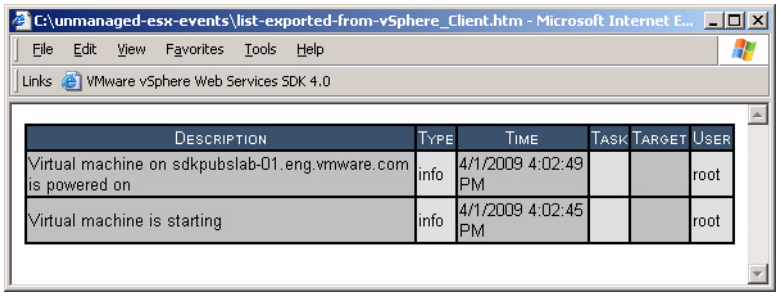
Introduction to the Event Data Object

In addition to creating Task references, many operations on hosts and virtual machines also generate one or more Event data objects as the operation executes on the server. An Event is a data object that conveys information about changes in the state of managed entities.

For unmanaged ESX/ESXi systems, Event objects are non-persistent. Events are retained only for as long as the host system's local memory can contain them. Rebooting an unmanaged ESX/ESXi host or powering off a virtual machine removes Event objects from local memory.

An unmanaged ESX/ESXi system might retain about 15 minutes worth of Event data about the host and virtual machine operations, but this can vary, depending on the processing load of the host, the number of virtual machines, and other factors that affect memory.

Figure 9-4. Events Exported from an Unmanaged ESX/ESXi System



For 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 stores the information its database.

The sample applications included in the SDK package for handling events can be used with either unmanaged ESX/ESXi or vCenter Server systems.

Using a HistoryCollector, 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 "Obtaining Information about Events Using a HistoryCollector" on page 133.

See "Understanding EventManager and the Event Data Object" on page 129 for more information.

Working with Virtual Machines 10

A `VirtualMachine` is associated with both its containing `Folder` and the `ResourcePool` from which it obtains resources at runtime, so it acquires permissions from both these objects.

Many capabilities of virtual machines require that VMware Tools is installed on the guest operating system of the virtual machine. For example, VMware Tools provides the memory-management driver that supports ballooning, and the networking driver that provides a DNS name for the virtual machine. In addition, some of the API available to virtual machines require VMware Tools. For example, the quiesce parameter of the is supported only if VMware Tools is installed on the virtual machine.

The VirtualMachine API supports programmatically upgrading the VMware Tools software on the virtual machine guest OS.

This chapter includes these topics:

- "Sample Code Reference" on page 93
- "Creating Virtual Machines" on page 93
- "Overview of Virtual Machine Operations" on page 94
- "Overview of Snapshots" on page 97

Sample Code Reference

Table 10-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 10-1. Sample Applications that Demonstrate VirtualMachine Operations

Java SDK\samples\Axis\java\com\vmware\samples	C# SDK\samples\DotNet\cs
`VMCreate.java`	`VMCreate`
`VMPowerOps.java`	`VMPowerOps`
`VMReconfig.java`	`VMReconfig`
`VMSnapshot.java`	`VMSnapshot`

Creating Virtual Machines

For unmanaged ESX/ESXi systems, you create virtual machines using the `Folder` factory object in conjunction with instances of the various data objects that define all aspects of the virtual machine, and references for the `Folder`, `ResourcePool`, and `HostSystem` managed object instances that you want to use to support the `VirtualMachine`.

The data objects you pass to the `CreateVM_Task` of the `Folder` managed object include an instance of the `VirtualMachineConfigSpec`. A `VirtualMachineConfigSpec` is a complex data object consisting of almost 50 different properties that make-up the configuration of the virtual machine you want to create. However, to create a virtual machine on an unmanaged host system you must provide only a few property values for a minimal number of properties, such as those shown in Example 10-1.

Example 10-1. Creating a VirtualMachineConfigSpec Data Object

```
ManagedObjectReference resourcePool = cb.getServiceUtil().getMoRefProp(crmor, "resourcePool");
ManagedObjectReference vmFolderMor = cb.getServiceUtil().getMoRefProp(dcmor, "vmFolder");

VirtualMachineConfigSpec vmConfigSpec = vmUtils.createVmConfigSpec(cb.get_option("vmname"),
cb.get_option("datastorename"),
Integer.parseInt(cb.get_option("disksize")),
crmor, hostmor);

vmConfigSpec.setName(cb.get_option("vmname"));
vmConfigSpec.setAnnotation("VirtualMachine Annotation");
vmConfigSpec.setMemoryMB(new Long(Integer.parseInt(cb.get_option("memorysize"))));
vmConfigSpec.setNumCPUs(Integer.parseInt(cb.get_option("cpucount")));
vmConfigSpec.setGuestId(cb.get_option("guestosid"));

ManagedObjectReference taskmor = cb.getConnection().getService().createVM_Task(
            vmFolderMor, vmConfigSpec, resourcePool, hostmor);
```

Table 10-2 lists some of the properties of the `ResourceAllocationInfo` data object that are used to allocate CPU and memory resources.

Table 10-2. ResourceAllocationInfo Data Object Properties

	Unit of Measure		
Property	**CPU**	**Memory**	**Description**
`reservation`	MHz	MB	Minimum amount of the resource guaranteed to be available if it is needed. Any unused amount of a reservation can be consumed by other virtual machines or resource pools that need it.
`limit`	MHz	MB	Maximum amount of resource that can be utilized, regardless of available resources on other virtual machines or resource pools. Use the limit setting to ensure consistent performance of virtual machines and resource pools, regardless of available resources. Set to -1 to remove all restrictions on the resource utilization. The resource is limited to available resources and shares.
`shares`			Relative amount of resource (as compared to levels of other virtual machines) that can be utilized, specified using the `SharesLevel` enumeration (high, normal, low, or custom). For custom shares, define using the shares property and
`expandableReservation`			When set to `true`, specifies that a child resource pool can grow beyond its reservation amount if its parent resource pool has unreserved resources available.

Overview of Virtual Machine Operations

Table 10-3 lists `VirtualMachine` operations available to virtual machines running on unmanaged ESX/ESXi systems. For any host managed by vCenter Server, use the API available on the vCenter Server Web service when creating or managing virtual machines. See Chapter 13, “Using vCenter Server for Virtual Machine Operations,” on page 119.

Table 10-3. Basic VirtualMachine Operations Summary

Operation	Description
`AcquireMksTicket`	Creates and returns a one-time credential for establishing a remote mouse-keyboard-screen connection to this virtual machine. this method depends on being able to retrieve TCP binding information about the server end of the client connection that is requesting the ticket. If such information is not available, the `NotSupported` fault is thrown. This method is appropriate for SOAP and authenticated connections, which are both TCP-based connections.
`AnswerVM`	Responds to a question that is blocking this virtual machine.
`CheckCustomizationSpec`	Checks the customization specification against the virtual machine configuration. For example, this is used on a source virtual machine before a clone operation to catch customization failure before the disk copy. This checks the specification's internal consistency and for compatibility with this virtual machine's configuration.
`CreateScreenshot_Task`	Create a screen shot of a virtual machine.
`CreateSnapshot_Task`	Creates a new snapshot of this virtual machine. As a side effect, this updates the current snapshot. Any % (percent) character used in this name parameter must be escaped, unless it is used to start an escape sequence. Clients may also escape any other characters in this name parameter.
`CustomizeVM_Task`	Customizes a virtual machine's guest operating system.
`DefragmentAllDisks`	Defragment all virtual disks attached to this virtual machine.
`MountToolsInstaller`	Mounts the VMware Tools CD installer as a CD-ROM for the guest operating system. You can check the status of the VMware Tools installation by checking the `guest.toolsStatus` property.
`PowerOffVM_Task`	Powers off this virtual machine. If this virtual machine is a fault tolerant primary virtual machine, this will result in the secondary virtual machine(s) getting powered off as well.
`PowerOnVM_Task`	Powers on this virtual machine. If the virtual machine is suspended, this method resumes execution from the suspend point. When powering on a virtual machine in a cluster, the system might implicitly or due to the host argument, do an implicit relocation of the virtual machine to another host. Hence, errors related to this relocation can be thrown. If the cluster is a DRS cluster, DRS is invoked if the virtual machine can be automatically placed by DRS. Because this method does not return a DRS `ClusterRecommendation`, neither VMotion nor host power operations are done as part of a DRS-facilitated power on. If you want DRS to consider such operations, use `PowerOnMultiVM_Task`. If this virtual machine is a fault tolerant primary virtual machine, its secondary virtual machines is started on system-selected hosts. If the virtual machines are in a VMware DRS enabled cluster, DRS is invoked to obtain placements for the secondaries. However, neither VMotion nor host power operations are considered for these power-ons.
`QueryUnownedFiles`	For all files that belong to the VM, check that the file owner is set to the current datastore principal user, as set by `HostDatastoreSystem.ConfigureDatastorePrincipal`.
`RebootGuest`	Issues a command to the guest operating system asking it to perform a reboot. Returns immediately and does not wait for the guest operating system to complete the operation.
`ReconfigVM_Task`	Reconfigures this virtual machine. All the changes in the given configuration are applied to the virtual machine as an atomic operation.
`RefreshStorageInfo`	Explicitly refreshes the storage information of this virtual machine, updating properties `layoutEx` and `storage`.
`RemoveAllSnapshots_Task`	Remove all the snapshots associated with this virtual machine. If the virtual machine does not have any snapshots, this operation returns success.
`ResetGuestInformation`	Clears cached guest information. Guest information can be cleared only if the virtual machine is powered off. This method can be useful if stale information is cached, preventing an IP address or MAC address from being reused.

Table 10-3. Basic VirtualMachine Operations Summary (Continued)

Operation	Description
ResetVM_Task	Resets power on this virtual machine. If the current state is poweredOn, then this method first performs powerOff(hard). Once the power state is poweredOff, then this method performs powerOn(option). 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.
RevertToCurrentSnapshot_Task	Reverts the virtual machine to the current snapshot. This is equivalent to running snapshot.currentSnapshot.revert. If no snapshot exists, then the operation does nothing, and the virtual machine state remains unchanged.
SetDisplayTopology	Sets the console window's display topology as specified.
SetScreenResolution	Sets the console window's resolution as specified.
ShutdownGuest	Issues a command to the guest operating system asking it to perform a clean shutdown of all services. Returns immediately and does not wait for the guest operating system to complete the operation.
StandbyGuest	Issues a command to the guest operating system asking it to prepare for a suspend operation. Returns immediately and does not wait for the guest operating system to complete the operation.
StartRecording_Task	Initiates a recording session on this virtual machine. As a side effect, this operation creates a snapshot on the virtual machine, which in turn becomes the current snapshot.
StartReplaying_Task	Starts a replay session on this virtual machine. As a side effect, this operation updates the current snapshot of the virtual machine.
StopRecording_Task	Stops a currently active recording session on this virtual machine.
StopReplaying_Task	Stops a replay session on this virtual machine.
SuspendVM_Task	Suspends execution in this virtual machine.
UnmountToolsInstaller	Unmounts VMware Tools installer CD.
UnregisterVM	Use the Folder.RegisterVM method to recreate a VirtualMachine object from the set of virtual machine files by passing in the path to the configuration file. However, the VirtualMachine managed object that results typically has different objects ID and may inherit a different set of permissions.
UpgradeTools_Task	Begins the tools upgrade process. Monitor the status of the installation using the guest.toolsStatus property.
UpgradeVM_Task	Upgrades this virtual machine's virtual hardware to the latest revision that is supported by the virtual machine's current host.

Working with any virtual machines from a client application requires the virtual machine to be configured properly for networking. Specifically, the appropriate version of VMware Tools must be installed and configured on the guest operating system (guest OS).

For each guest OS, such as Windows 2003 Server, Red Hat Enterprise Server Linux, or Ubuntu Linux, VMware provides a specific binary-compatible version of VMware Tools. The VMware Tools software provides many benefits, such as enhancing the display interaction for end users. It also provides the vmxnet driver needed for the virtual network interface controller (vNIC) and virtual machine network configuration. 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.

IMPORTANT Selecting **Install VMware Tools** from the vSphere Client menu only downloads the appropriate installation package to the virtual machine. You must unpack the file, run the installation script or launch the executable, and configure VMware Tools on the guest operating system of the virtual machine to enable networking, memory management, and other capabilities on the virtual machine.

Overview of Snapshots

A snapshot preserves the state and data of a virtual machine at a specific point in time.

- State includes the virtual machine's power state (powered-on, powered-off, suspended, for example).
- Data includes all the files that make-up the virtual machine, including disks, memory, and other devices, such as virtual network interface cards.

A `VirtualMachine` object provides several operations for creating and managing snapshots. These operations let you create snapshots, revert to any snapshot in the tree, and remove snapshots. You can create extensive snapshot trees that you can use to save virtual machine state at any point in time, and restore it later, if necessary.

Creating Snapshots

Creating a snapshot of a virtual machine starts with a reference (instance of `ManagedObjectReference`) to the target virtual machine. Pass the reference to the `CreateSnapshot_Task` operation, providing a `name` and `description` (both `strings`) for the snapshot.

The `CreateSnapshot_Task` operation also has two boolean properties:

- `memory`, which can be set to `true` to include the virtual machine's memory dump in the snapshot. Including memory in the snapshot will take longer (to complete the snapshot operation) and create a larger snapshot file.
- `quiesce`, which can (and should) be set to `true` so that the snapshot will include the virtual machine's disk data in a consistent state: this property quiesces the virtual machine's file system prior to initiating the operation.

See the *vSphere API Reference* for more information.

Reverting to Snapshots

After a snapshot exists on the system in a tree of snapshots, you can reinstate the snapshot by using the `RevertToSnapshot_Task` operation. See the *vSphere API Reference* for details.

Managing Storage 11

vSphere servers require dedicated storage space for the operating system files and for the components that make up the inventory. Storage includes physical, logical, and virtual constructs. This chapter includes these topics:

- "Sample Code Reference" on page 99
- "Overview of Physical and Virtual Storage Concepts" on page 99
- "Introduction to the HostStorageSystem Managed Object" on page 102
- "Introduction to the HostDatastoreSystem Managed Object" on page 106
- "Obtaining Information about Configured Storage" on page 107
- "Creating New Datastores" on page 108
- "Modifying Datastore Configuration" on page 110

Sample Code Reference

Table 11-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 11-1. Sample Applications that Demonstrate Storage and Datastore Operations

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

Overview of Physical and Virtual Storage Concepts

A virtual machine is made up of files that can be copied and moved, just as any other computer-based file. These files are stored on the physical media that backs the logical representation of the storage (the "device" is the logical representation, in this context).

The host system can discover storage devices that it has access to and can format these storage devices (or portions of them) as datastores. A datastore is an abstraction that provides access to the virtual disks of a virtual machine. A datastore is a logical container (analogous to a storage "volume") for the virtual disk and other files that encapsulate a virtual machine.

The virtual SCSI disks are typically contained in a Virtual Machine File System (VMFS) volume. VMFS is a clustered file system, optimized for virtual machines and is the default file system for the set of files that encapsulates each virtual machine on physical SCSI disks or partitions.

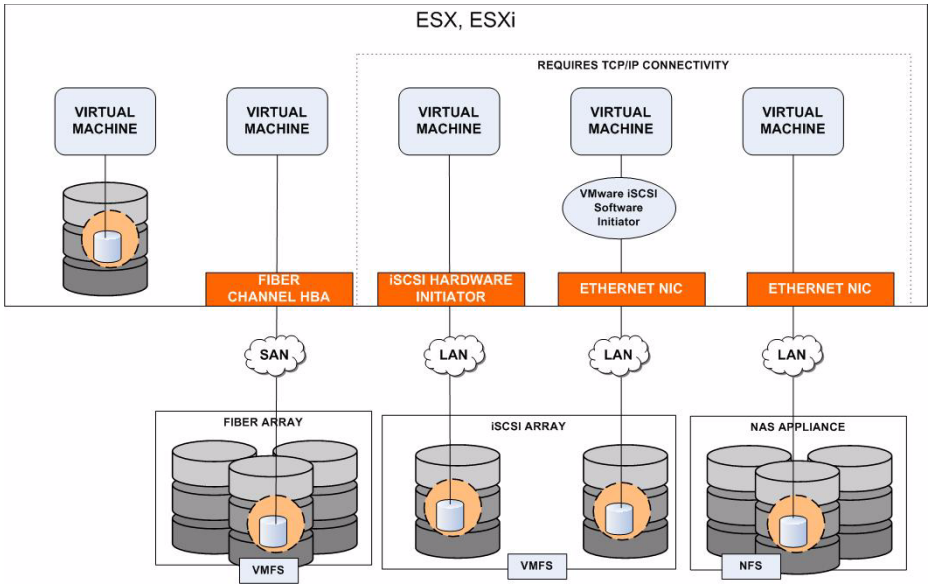
A VMFS volume contains files, directories, symbolic links, raw-device mappings (RDMs), and metadata about its contents. An RDM is a special file on a VMFS volume that mediates commands to raw devices. Raw devices can also be used to store virtual machine files, but the RDM file must be available on a VMFS file system.

Virtual machines use virtual disks for their operating system, application software, and other data files. A virtual disk is stored as a VMDK file inside a datastore.

Figure 11-1 provides a conceptual overview of the various storage architectures supported by vSphere servers, which include:

- Storage-area networks (SAN). SANs can be implemented using various technologies, including Fibre Channel (FC) and Internet SCSI (iSCSI).
- Network-attached storage (NAS). NAS storage uses file-based protocols such as NFS (network file system), NTFS (NT file system), or SMB/CIFS (Microsoft legacy server message block/common internet file system).
- Direct-attached storage (DAS). Also called local storage. Local storage has some limitations when compared to SAN and NAS systems.

Figure 11-1. Overview of Local, NAS, and SAN Storage



NOTE Figure 11-1 is for illustration purposes only. It is not a recommended configuration. Details have been over-simplified on purpose.

When you create a virtual machine using the vSphere Client, a wizard guides you through the configuration of all settings, including storage. By default, a virtual machine is created with one virtual hard disk and one virtual SCSI controller.

Virtual machines always use a virtual SCSI adapter and communicate using the SCSI command set regardless of the type of physical storage—SCSI, iSCSI, fibre channel, or file-system based. At runtime, the guest operating system running on the virtual machine issues SCSI commands to its virtual disks. The virtualization layer translates the SCSI commands into VMFS file operations.

Requirements for Storage

Any type of network-attached storage requires complete configuration of networking in the VMkernel, to support network-based access to the storage media. The VMkernel requires its own IP address.

Overview of Internet SCSI Storage

iSCSI packages SCSI commands inside internet protocol (IP) packets and sends from a host to a storage volume.

Figure 11-2. Overview of iSCSI Storage

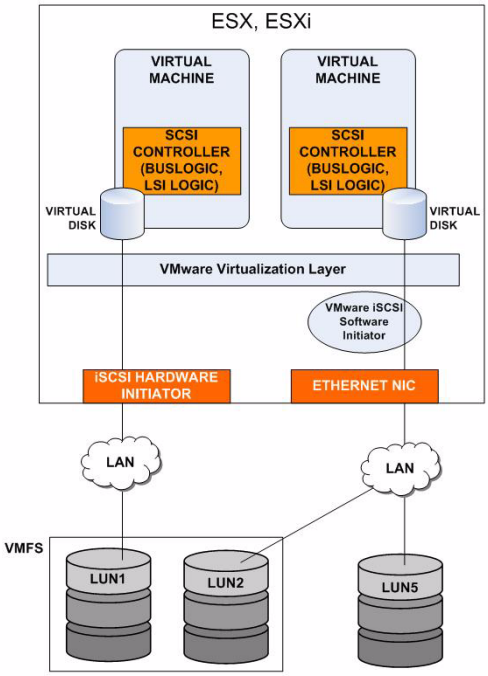


Figure 11-2 shows two different approaches to Internet SCSI (iSCSI) storage supported by vSphere servers.

- Hardware-based iSCSI uses an iSCSI host-bus adapter (HBA), which serves as the source (or "initiator") of SCSI commands.
- Software-based iSCSI uses a high-speed networking hardware, such as 1-GB Ethernet NICs and switches. ESX/ESXi has its own iSCSI initiator, built-in to the VMkernel. By default, the software iSCSI implementation in the VMkernel is disabled. (The `isSoftwareBased` property of the `HostInternetScsiHba` is set to `false`.

NOTE Do not use 100-MB Ethernet NICs for iSCSI-based storage: it is too slow. Use 1-GB (or faster) Ethernet.

An initiator's properties include its model name, IP address, iSCSI name, discovery methods, iSCSI alias, and a list of any targets that it has already discovered.

Introduction to Managed Objects that Support Storage

The `HostSystem` managed object has a `configManager` property that is defined as an instance of a `HostConfigManager` data object. The `HostConfigManager` data object contains close to twenty properties, each of which comprises a managed object reference to a specific service interface for the host system. A `HostConfigManager` data object has two properties associated with managing storage at the file-system level, the other at the logical (volume) management level:

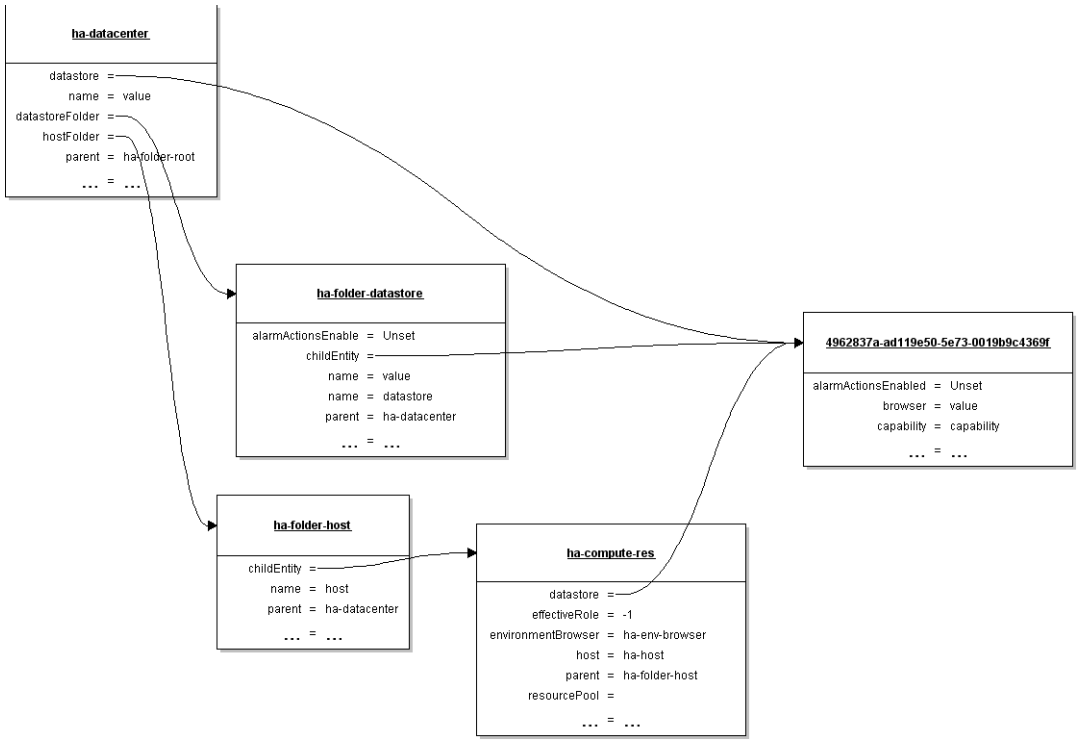
- The `datastoreSystem` property, which provides a managed object reference to the `HostDatastoreSystem` of the server. `HostDatastoreSystem` provides operations for creating, configuring, extending, and removing the datastores available for use by the host system (ESX/ESXi). `HostDatastoreSystem` is the abstraction for logical storage, the volumes (Datastore managed objects) that can be used by the host for virtual machines.

- The `storageSystem` property, which provides a managed object reference to the `HostStorageSystem` of the server. `HostStorageSystem` provides operations associated with the configuration and management of the physical storage implementation. It provides operations for creating, configuring, and managing the host's storage subsystem. A `HostStorageSystem` is the service interface to use to make storage available for virtual machines.

The `HostStorageSystem` object provides an abstraction for physical storage resources.
The `HostDatastoreSystem` provides access to storage resources as presented by the file system.

The `Datastore` managed entity provides operations for mounting datastores, browsing datastores, and obtaining information about the datastores associated with a virtual machine.

Figure 11-3. Datastore Managed Object As a ManagedEntity



Introduction to the HostStorageSystem Managed Object

The `HostStorageSystem` managed object can be used to obtain information about, configure, and manage the storage systems used by VMware vSphere servers. Specific types of storage are modelled in the vSphere management object as subtypes of the `HostHostBusAdapter` base data object type. As shown in Figure 11-4, these subtypes support Fibre Channel (FC), SCSI, internet SCSI, and directly attached storage.

Figure 11-4. HostHostBusAdapter Data Object Type and Subtypes

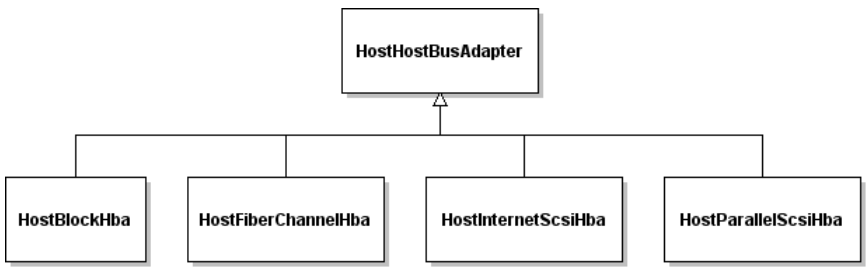


Table 11-2 lists the operations available to `HostStorageSystem` managed objects.

Table 11-2. HostStorageSystem Operations Summary

Operation	Description
`AddInternetScsiSendTargets`	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.
`AttachVmfsExtent`	Extends a VMFS by attaching a disk partition as an extent.
`ComputeDiskPartitionInfo`	Computes the disk partition information based on the specified disk layout. The server computes a new partition information object for a specific disk representing the specified layout.
`ComputeDiskPartition InfoForResize`	Computes the disk partition information to support resizing a given partition.
`DisableMultipathPath`	Disables an enabled path for a logical unit. Use the path name from `HostMultipathStateInfoPath` or `HostMultipathInfoPath`.
`EnableMultipathPath`	Enables a disabled path for a logical unit. Use the path name from `HostMultipathStateInfoPath` or `HostMultipathInfoPath`.
`ExpandVmfsExtent`	Expands a VMFS extent as specified by the disk partition specification.
`FormatVmfs`	Formats a new VMFS on a disk partition.
`QueryPathSelection PolicyOptions`	Obtains the set of path-selection-policy options. These options determine the path that can be used by a device managed by native multipathing. A `HostMultipathInfo` data object identifies the devices that are managed through native multipathing.
`QueryStorageArrayType PolicyOptions`	Obtains the set of storage-array-type policy options. These options determine the storage-array-type policies that can be used by a device managed by native multipathing. A `HostMultipathInfo` data object identifies the devices that are managed through native multipathing.
`QueryUnresolved VmfsVolume`	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 will be unbound.
`RefreshStorageSystem`	Refresh the storage information and settings to pick up any changes that might have occurred.
`RemoveInternetScsi SendTargets`	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.
`RemoveInternetScsi StaticTargets`	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.
`RescanAllHba`	Issue a request to rescan all virtual machine host bus adapters for new storage devices.
`RescanHba`	Issue a request to rescan a specific virtual machine host bus adapter for new storage devices.
`RescanVmfs`	Rescans for new VMFS instances that might have been added.
`ResolveMultiple Unresolved VmfsVolumes`	Resignature or force `mount` list of unbound VMFS volumes. VMFS volumes are bound to their underlying block-storage devices.
`RetrieveDiskPartitionInfo`	Returns an array containing information about the partitions on the host (an array of `HostDiskPartitionInfo` objects).
`SetMultipathLunPolicy`	Updates the path selection policy for a Logical Unit. Use the LUN UUID from `HostMultipathInfoLogicalUnit`.

Table 11-2. HostStorageSystem Operations Summary (Continued)

Operation	Description
`UnmountForceMounted VmfsVolume`	Unmount the 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, user may decide to keep the original VMFS UUID. Once the resolution is applied, the VMFS volume is mounted on the host for its use. User can unmount the VMFS volume if it is not being used by any registered VMs.
`UpdateDiskPartitions`	Changes the partitions on the disk by supplying a partition specification and the device name.
`UpdateInternetScsi AdvancedOptions`	Updates the advanced options the iSCSI host bus adapter or the discovery addresses and targets associated with it.
`UpdateInternetScsiAlias`	Updates the alias of an iSCSI host bus adapter.
`UpdateInternetScsi AuthenticationProperties`	Updates the authentication properties for one or more targets or discovery addresses associated with an iSCSI host bus adapter.
`UpdateInternetScsiDigest Properties`	Updates the digest properties for the iSCSI host bus adapter or the discovery addresses and targets associated with it.
`UpdateInternetScsiDiscovery Properties`	Updates the discovery properties for an iSCSI host bus adapter.
`UpdateInternetScsiIP Properties`	Updates the IP properties for an iSCSI host bus adapter.
`UpdateInternetScsiName`	Updates the name of an iSCSI host bus adapter.
`UpdateScsiLunDisplayName`	Update the mutable display name associated with a SCSI LUN. The SCSI LUN to be updated is identified using the specified UUID.
`UpdateSoftware InternetScsiEnabled`	Enables and disables software iSCSI in the VMkernel.
`UpgradeVmfs`	Upgrades the VMFS to the current VMFS version.
`UpgradeVmLayout`	Iterates over all registered virtual machines. For each VM which VMX file is located on the service console and all disks are available on VMFS3 or NAS, relocate sthe disks into directories if stored in the ROOT, and relocate the VMX file into the directory too. Events are logged for each virtual machine that is relocated.

If the `DiscoveryProperties.sendTargetsDiscoveryEnabled` flag is set to `true`, the `AddInternetScsiSendTarget` operation adds send-target entries to the storage adapter's discovery list.

To obtain managed object references, create a `PropertyFilterSpec` to obtain `HostSystem` managed objects and the appropriate property.

To enable the VMkernel to support software iSCSI

1 Obtain a managed object reference to the host system's HostStorageSystem.

2 Invoke the `UpdateSoftwareInternetScsiEnabled` operation, passing the reference to the `HostStorageSystem` and the value `true`.

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 conform to one of two alternative formats which are specific to the storage vendor's hardware (Table 11-3).

Table 11-3. Format for iSCSI Names

Name	Description	Format
EUI	**Extended unique identifier**. A 64-bit global identifier assigned by the hardware manufacturer. An EUI address includes the 24-bit company name (the "organizationally unique identifier," or OUI, assigned by the IEEE to the vendor), and a 40-bit unique ID, such as a serial number.	`eui.0123456789ABCDEF`
IQN	**iSCSI qualified name**. Begins with initials `iqn` followed by the company domain in reverse order. For example, `iqn.2000-04.com.qlogic:qle4060c.my-san-server55.1`	`iqn.yyyy-mm.com.company: san-server-name`

To configure iSCSI Initiators

1 Access the list of available HBAs on the host system by obtaining the nested value for host bus adapters configured on the host. You can do this by creating a property collector with `HostSystem` as the starting point. From the `HostSystem` `config` property, you can obtain the list (array) of host bus adapters by specifying this property path:

```
config.storageDevice.hostBusAdapter
```

This 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 a string consisting of the device name of the host bus adapter.

3 Determine the capabilities of the iSCSI Host Bus Adapter HBA.

4 Configure the initiator.

- For a hardware initiator, configure the IP address.
- For a software initiator, enable the software initiator in the VMkernel.

5 Configure the iSCSI name and alias.

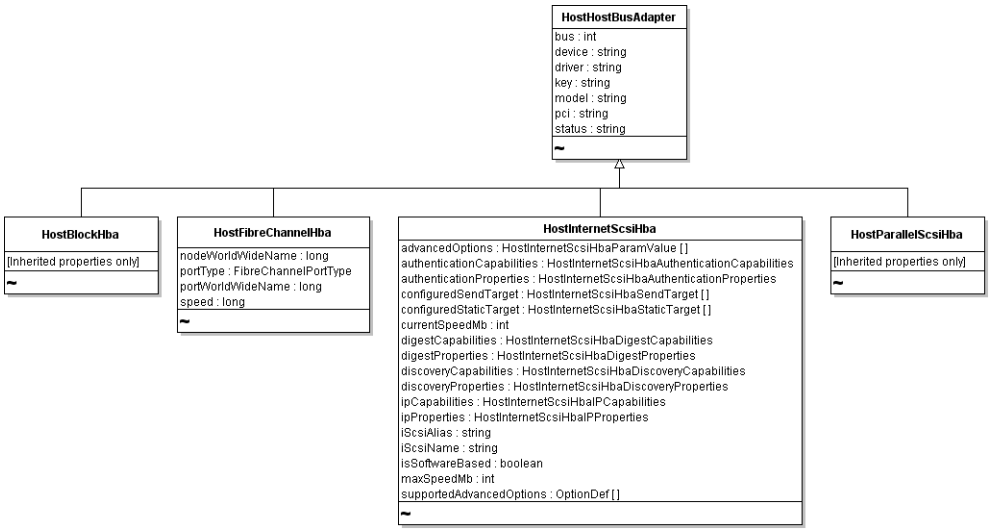
6 Configure target discovery.

7 Set the authentication information. (Skip this step if you are not using CHAP.)

8 Configure access to the targets.

9 Invoke a rescan operation on the host bus adapters. Rescan enables the HBAs to discover the new storage devices. You can either issue a rescan for a single HBA using the `RescanHba` operation with the HBA ID as a parameter or issue a rescan on all HBAs using `RescanAllHba`.

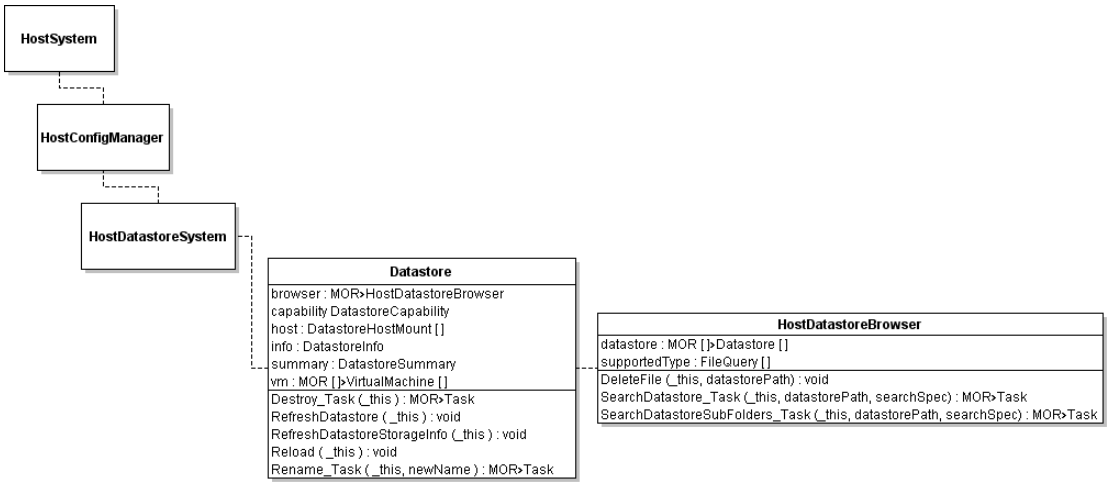
Figure 11-5. HostHostBusAdapter Data Object and Associated Data Objects



Introduction to the HostDatastoreSystem Managed Object

Figure 11-6 shows some of the relationships among `HostSystem` managed objects, datastores, and storage.

Figure 11-6. HostSystem and HostDatastoreSystem



The objects returned depend on the specifics of the query. For example, the `QueryAvailableDisksForVmfs` returns an array of references to an array of `HostVmfsDatastoreOption` data objects that represent the configured storage for the host.

Table 11-4. Summary of HostDatastoreSystem Operations

Operation	Description
`ConfigureDatastore Principal`	Configures datastore principal user for the host. All virtual machine-related file I/O is performed under this user. Use this operation if you want to the virtual machine files —configuration, disk, and so on—to be checked for proper access. Host must be placed in maintenance mode for this operation. Requires a reboot to effect changes. Does not apply to virtual machine files stored on NFS.
`CreateLocalDatastore`	Creates a new local datastore on the host system, in the specified directory.
`CreateNasDatastore`	Creates a new network-attached storage datastore using the specifications supplied in the `HostNasVolumeSpec` data object.
`CreateVmfsDatastore`	Creates a new VMFS datastore using the specifications supplied in the `VmfsDatastoreCreateSpec` data object.

Table 11-4. Summary of HostDatastoreSystem Operations (Continued)

Operation	Description
ExpandVmfsDatastore	Expands an existing VMFS datastore using the specification provided in the VmfsDatastoreExpandSpec data object (which contains the name of the extent and partition information).
ExtendVmfsDatastore	Extends an existing VMFS datastore using the specification provided in the VmfsDatastoreExtendSpec data object.
QueryAvailableDisksForVmfs	Obtain list of disks that can be used to contain VMFS datastore extents. Optionally, provide a datastore name to obtain 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 referenced by RDMs (which aren't usable for VMFS datastores).
QueryUnresolvedVmfsVolumes	Obtain 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 will be unbound.
QueryVmfsDatastoreCreateOptions	Obtain information about options for creating a new VMFS datastore for a disk.
QueryVmfsDatastoreExpandOptions	Obtain information about options for expanding the extents of an existing VMFS datastore.
QueryVmfsDatastoreExtendOptions	Obtain information about options for extending an existing VMFS datastore for a disk.
RemoveDatastore	Removes a datastore from a host.
ResignatureUnresolvedVmfs Volume_Task	Assigns a new DiskUuid to a VMFS volume but keeps its contents intact. This operation facilitates safe volume sharing across hosts. VMFS volumes are bound to their underlying block device.
UpdateLocalSwapDatastore	Choose the localSwapDatastore for this host. Any change to this setting will affect 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.

The ExpandVmfsDatastore operation increases the size of the datastore up to the full size provisioned for the datastore, if necessary.

See the *vSphere API Reference* for more information about the HostDatastoreSystem operations, including constraints and limitations.

Obtaining Information about Configured Storage

Table 11-5 lists several query operations available on HostDatastoreSystem. Use these operations to find out about existing storage configured on an ESX/ESXi system before creating new storage or modifying existing storage.

The query operations for HostDatastoreSystem are specific to the type of storage, but most work in the same general way, typically requiring the managed object reference to the HostDatastoreSystem and one or more specific data objects specifying the configuration details.

Table 11-5. HostDatastoreSystem Query Operations

Operation	Parameter	Type	Description
QueryAvailableDisks ForVmfs	_this	MOR>HostDatastoreSystem	Reference to the HostDatastoreSystem.
	datastore	MOR>Datastore	Reference to an existing datastore on which to create or add VMFS extents.
QueryVmfsDatastore CreateOptions	_this	MOR>HostDatastoreSystem	Reference to the HostDatastoreSystem used for the invocation.
	devicePath	string	The devicePath of the disk on which datastore creation options are generated. The device path is a file path that can be opened to create partitions on the disk.

Table 11-5. HostDatastoreSystem Query Operations (Continued)

Operation	Parameter	Type	Description
QueryVmfsDatastore ExtendOptions	_this	MOR>HostDatastoreSystem	Reference to the HostDatastoreSystem used to make the method call.
	datastore	MOR>Datastore	Reference to the datastore to extend.
	devicePath	string	The devicePath of the disk on which datastore creation options are generated.
	suppress Expand Candidates	boolean	Indicates whether to exclude options that can be used for either extending or expanding extent. Set to true to exclude free space available for expansion.

Free space can be used for adding an extent or expanding an existing extent. Expanding an existing extent is sometimes referred to as growing the extent.

Creating New Datastores

The HostDatastoreSystem provides several operations for creating new datastores (Table 11-6). All of the operations require a managed object reference to HostDatastoreSystem, and return a reference to the Datastore object after it is created.

Table 11-6. HostDatastoreSystem Creation Operations

Operation	Parameter	Type	Description
CreateLocalDatastore	_this	MOR>HostDatastoreSystem	Reference to the HostDatastoreSystem of the host system.
	name	string	Host-wide unique name for the datastore.
	path	string	File path to the directory in which virtual machine data (specifically, VMX and other files) will be located.
CreateNasDatastore	_this	MOR>HostDatastoreSystem	Reference to the HostDatastoreSystem of the host system.
	spec	HostNasVolumeSpec	Data object that defines the local path, remote host, and remote path for the datastore. See Table 11-7 for details.
CreateVmfsDatastore	_this	MOR>HostDatastoreSystem	Reference to the HostDatastoreSystem.
	spec	VmfsDatastoreCreateSpec	Specification for creating a datastore backed by VMFS. The data object defines the extent (if appropriate, to append to an existing VMFS datastore), the partition, and the VMFS properties for the datastore.

Table 11-7 lists the properties and data types of the HostNasVolumeSpec data object. You create an instance of this data object to define a network-attached storage based datastore on the system.

Table 11-7. HostNasVolumeSpec Data Object

Property	Type	Description and Usage
accessMode	string	Access mode for the mount point. Access modes include standard NFS file permissions.
localPath	string	Name of the NAS datastore that will be created using this specification. ■ ESX/ESXi systems: Datastore name is a component in the file system path at which the NAS volume can be found. For example, if localPath is set to nas_volume the created NAS datastore is named "nas_volume" and it can be accessed using the file system path /vmfs/volumes/nas_volume. ■ vCenter Server system: Datastore name is localPath, but the datastore name may not be shown in the file system path where the NAS volume is accessed.
password	string	Set to CIFS if the password for the CIFS server should be used. If type is NFS, this field is ignored.
remoteHost	string	Fully qualified name of the host supporting the NFS server.
remotePath	string	Remote path to the NFS mount point.
type	string	Type of network-attached storage volume. Default is NFS. Valid types include CIFS and NFS.
userName	string	If type is CIFS, the user name to use when connecting to the CIFS server. If type is NFS, this field is ignored.

To create a VMFS-based datastore, you use the CreateVmfsDatastore operation. This operation takes a VmfsDatastoreCreateSpec data object that consists of an extent, a partition, and a vmfs property, all of which are complex data objects. For example, the vmfs property is defined as an instance of the HostVmfsSpec data object. Table 11-8 lists the properties and data types of the HostVmfsSpec data object.

Table 11-8. HostVmfsSpec Data Object

Property	Type	Description and Usage
blockSizeMb	int	Optional. VMFS block size, in megabytes (MB). Controls maximum file size. Valid block size varies by VMFS version: ■ VMFS3—1 MB ■ VMFS2—1, 2, 4, 8, 16, 32, 64, 128, 256 MB
extent	HostScsiDiskPartition	Primary extent that identifies the VMFS in vmhbaI:T:L format. The data is not guaranteed to be stable across re-boots, so do not try to identify VMFS using this data. Instead, define a unique volume name on the host and use it to refer to the VMFS. Or, use the VMFS UUID (which is immutable after it is created).
majorVersion	int	VMFS version number, such as 2 or 3. Version number can be optionally changed during an upgrade. Upgraded file systems cannot be downgraded.
volumeName	string	Name of the VMFS volume.

Table 11-9 lists the properties and data types for the HostScsiDiskPartition data object.

Table 11-9. HostScsiDiskPartition Data Object

Property	Type	Description and Usage
diskName	string	Name of the SCSI disk device on which the VMFS extent resides. The name can be a the name of the HostScsiDisk data object or the canonicalName (property of ScsiLun data object).
partition	int	Number of the HostScsiDisk partition.

Defining the specification for a datastore takes a fair number of complex data objects, such as these. See the *vSphere API Reference* for more information.

Modifying Datastore Configuration

Table 11-10 lists the operations available to change the `HostDatastoreSystem` configuration.

Table 11-10. HostDatastoreSystem Modification Operations

Operation	Parameter	Type	Description
`ConfigureDatastorePrincipal`	`_this`	`MOR>HostDatastoreSystem`	Reference to the `HostDatastoreSystem` service interface.
	`userName`	`string`	Principal (user) name for the data store.
	`password`	`string`	Optional. Password for storage systems that support user-impersonation and require password.
`ExpandVmfsDatastore`	`_this`	`MOR>HostDatastoreSystem`	Reference to the `HostDatastoreSystem` service interface.
	`datastore`	`MOR>Datastore`	Reference to the `Datastore` to expand. Increases the size of an existing extent.
	`spec`	`VmfsDatastoreExpandSpec`	Data object defining the extents of the `Datastore` to expand.
`ExtendVmfsDatastore`	`_this`	`MOR > HostDatastoreSystem`	Reference to the `HostDatastoreSystem` service interface.
	`datastore`	`MOR>Datastore`	Reference to the `Datastore` to extend by adding new extents.
	`spec`	`VmfsDatastoreExtendSpec`	Data object defining the extents to add to the `Datastore`.
`RemoveDatastore`	`_this`	`MOR>HostDatastoreSystem`	Reference to the `HostDatastoreSystem` service interface.
	`datastore`	`MOR>Datastore`	Reference to the `Datastore` to remove.
`ResignatureUnresolvedVmfs Volume_Task`	`_this`	`MOR>HostDatastoreSystem`	Reference to the `HostDatastoreSystem` service interface.
	`resolutionSpec`	`HostUnresolvedVmfs ResignatureSpec`	The HUVRS data object contains a string array comprising the list of device paths, each of which specifies a VMFS extent.

12 Monitoring Performance

VMware vSphere servers have been instrumented using a comprehensive system that tracks utilization of resources by various objects. At runtime, statistical data is generated by the vSphere server's performance providers. A performance provider is any entity on the system that generates utilization or consumption information. Performance providers include all the managed entities such as `HostSystem` and `VirtualMachine` objects, and the devices associated with them, such as virtual disks.

IMPORTANT Several data objects, including `HostListSummaryQuickStats`, `VirtualMachineQuickStats`, and `ResourcePoolQuickStats`, provide near real-time summary information about performance or utilization without going through the `PerformanceManager`. These data objects do not support monitoring.

This chapter includes these topics:

- "Sample Code Reference" on page 111
- "Introduction to PerformanceManager" on page 112
- "Obtaining Statistics" on page 115
- "vCenter Server and PerformanceManager" on page 117

Sample Code Reference

Table 12-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 12-1. Sample Applications that Demonstrate Using PerformanceManager Operations

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

Introduction to PerformanceManager

The `PerformanceManager` is the service interface for obtaining statistical data about various aspects of system performance, as generated and maintained by various performance providers.

Table 12-2. PerformanceManager Properties

Property	Type	Description
`description`	`PerformanceDescription`	The description property is a composite object that includes information about the types of counters (`counterType`) and statistics (`statsType`) available on the system.
`historicalInterval`	`PerfInterval[ ]`	An array of `PerfInterval` data objects that defines the performance intervals configured for the system. ■ For an ESX/ESXi system, the array contains a single object that defines the baseline sampling period for the server. ■ For a vCenter Server, this array contains four objects. See "Historical Intervals" on page 117 for more information.
`perfCounter`	`PerfCounterInfo[ ]`	An array of `PerfCounterInfo` data objects that contains all the metadata associated with the performance metrics generated on the system.

In addition to the properties listed in Table 12-2, `PerformanceManager` provides several query operations for obtaining metadata that defines the statistics, and for obtaining statistical values about performance.

Table 12-3. PerformanceManager Operations

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

See the *vSphere API Reference* for VMware Web Services SDK 4.0 for complete `PerformanceManager` documentation.

Overview of Counter Groups and the PerfCounterInfo Data Object

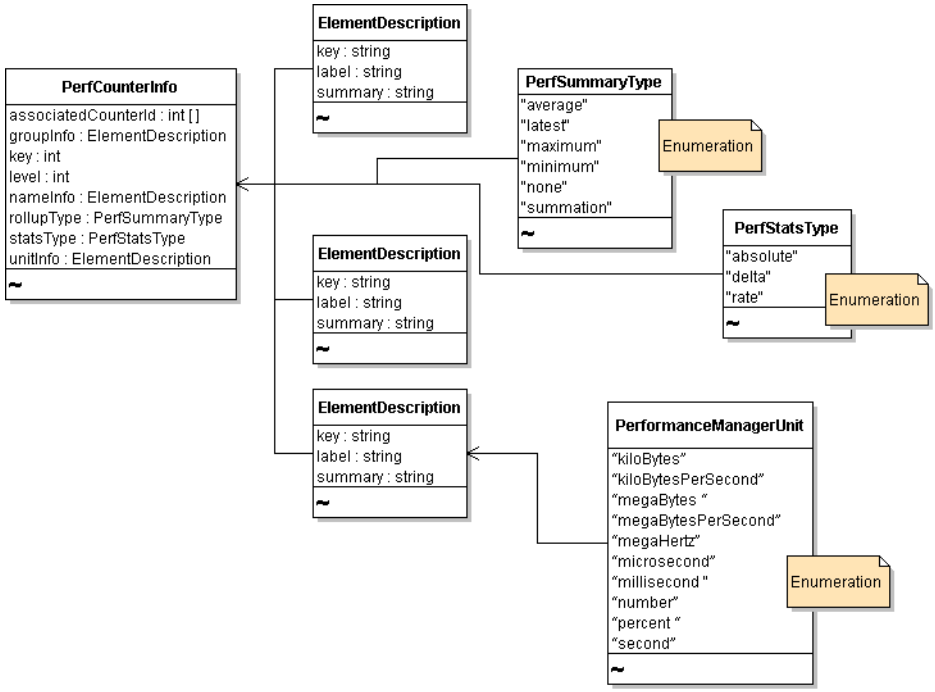
The performance monitoring system was designed for maximum flexibility. Each performance provider has its own set of counters that provides metadata about the performance data it might generate. Counters are organized by groups of finite system resources, such as memory, CPU, disk, and so on. The *vSphere API Reference* includes documentation for the following counter groups:

- CPU
- Disk I/O Counters
- Management Agent
- Memory
- Network
- Storage Utilization Counters
- Virtual Machine Operations

Some of these counter groups, such as storage utilization, are new in vSphere 4.0. The counter groups and specific counters used on any system depend on the specific configuration. Not all counter groups are used, and you might find available counters that are also not used. Disregard any extraneous counters.

The counters are modeled in the system using the `PerfCounterInfo` data object. Instances of this data object provide the metadata for the runtime statistical values.

Figure 12-1. PerfCounterInfo Data Object



The `PerfCounterInfo` data object includes a description, the unit of measure that the counter represents, and the other attributes listed in Table 12-4.

Table 12-4. PerfCounterInfo Data Object Properties

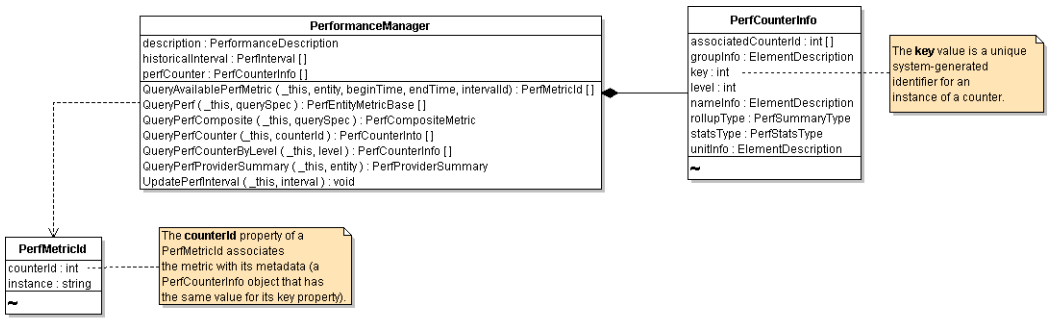
Property	Type	Description
`associatedCounterId`	`int [ ]`	An array of integers that lists counter IDs (`PerfCounterInfo.key` property) for other counters related to this one. This property is not used in current releases.
`groupInfo`	`ElementDescription`	Contains the name of the resource group to which this counter belongs, such as disk, cpu, or memory.
`key`	`int`	A unique integer that identifies the counter. Also called the counter ID. The value is not static. The counter key on an ESX/ESXi system might not be the same as the counter key for the same counter on the vCenter Server system managing the ESX/ESXi host. However, the system maps the keys from ESX/ESXi to vCenter Server systems transparently.
`level`	`int`	A number from 1 to 4 that identifies the level at which this counter will be aggregated with others of its kind into vCenter Server systems.
`nameInfo`	`ElementDescription`	A descriptive name for the counter that comprises several nested properties.
`rollupType`	`PerfSummaryType`	Identifies the function that controls how data is converted as it is consolidated from one interval into the next interval. Some counters specify absolute values, such as the total number of seconds that the system has been running continuously since startup, and so no conversion of values occurs. The `PerfSummaryType` is an enumeration containing valid constants for this property.

Table 12-4. PerfCounterInfo Data Object Properties (Continued)

Property	Type	Description
`statsType`	`PerfStatsType`	Identifies the type of statistical data that the value represents over the course of the interval, such as an average, a rate, the minimum value, the maximum value, summation, latest, or absolute. The `PerfStatsType` is an enumeration containing valid constants for this property.
`unitInfo`	`ElementDescription`	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.

Each counter has `key` property that identifies the metric generated by the system. The value of key is unique and is not static—it might change between system reboots, for example. At runtime, the `key` property of a `PerfCounterInfo` instance is used to populate the value of the `counterId` property of a specific `PerfMetricId` instance (Figure 12-2).

Figure 12-2. PerfCounterInfo Key Matches a PerfMetricId CounterId



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

Sampling Periods and Intervals

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

Table 12-5. Default Sampling Periods and Retention

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

The historicalInterval is defined as an array of PerfInterval data objects. For an ESX/ESXi system, the array contains a single PerfInterval object with the properties listed in Table 12-6.

Table 12-6. Values of PerfInterval Data Object from an ESX/ESXi System

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

See "Historical Intervals" on page 117 for information about intervals on managed hosts using vCenter Server.

Obtaining Statistics

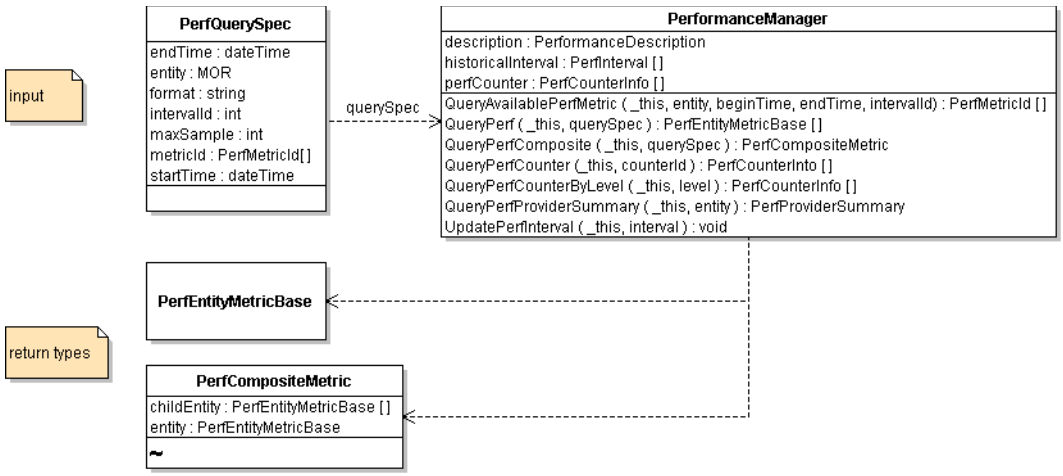
You use the QueryPerf and the QueryPerfComposite operations of PerformanceManager to obtain statistical data about performance. These operations behave differently, in the following ways:

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

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

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

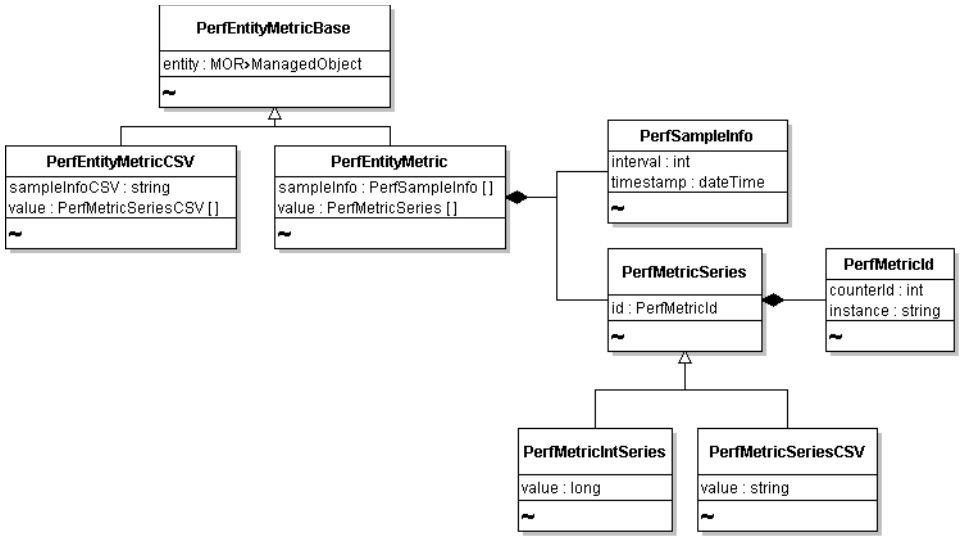
Figure 12-3. Define a PerfQuerySpec to Obtain Statistics



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

QueryPerfComposite is designed for efficient client-server communications. In theory, it generates less network traffic (than QueryPerf) because it returns a large-grained object, a PerfCompositeMetric data object that contains all the data (see Figure 12-4, "PerfEntityMetricBase Data Object," on page 116).

Figure 12-4. PerfEntityMetricBase Data Object



Before using QueryPerf or QueryPerfComposite, you use the QueryAvailablePerfMetric operation before either of these operations, to retrieve information about the metrics that are available for a specified managed entity within a given time frame, so you can specify the metrics you want to obtain.

Table 12-7 lists the parameters for the QueryAvailablePerfMetric operation.

Table 12-7. QueryAvailablePerfMetric Operation

Parameter	Type	Description
_this	MOR>PerformanceManager	Reference to the PerformanceManager.
entity	MOR>ManagedEntity	Reference to the managed entity for which available metrics should be obtained.
beginTime	dateTime	Optional. Server time from which to start gathering available metrics. If not specified, first available metrics in the system are returned.
endTime	dateTime	Optional. Server time at which to stop gathering available metrics. If not specified, returned metrics includes all metrics, up to the most recent.
intervalId	int	Optional. Identifier of the interval for which to return available metrics. Valid intervals include: ■ The refresh rate returned by QueryProviderSummary operation ■ One of the HistoricalIntervals If not specified, system returns available historical metrics.

Use the optional begin and end time, or specify an interval to obtain information about the metrics available for the interval. For unmanaged ESX/ESXi systems, use the refreshRate, which you can obtain from the service provider.

vCenter Server and PerformanceManager

A vCenter Server system aggregates performance data from all the ESX/ESXi systems that it manages. The amount of data aggregated depends on the level setting configured for the vCenter Server. The level settings are reflected in the `historicalInterval` property of the `PerformanceManager` of the vCenter Server system. `historicalInterval` is an array of `PerfInterval` data objects that define four different level settings, 1 through 4.

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

- 5-minute samples for the past day
- 30-minute samples for the past week
- 2-hour samples for the past month
- 1-day samples for the past year

Data older than a year is purged from the vCenter Server database.

Historical Intervals

Table 12-8 lists the default values of the `historicalInterval` property from a vCenter Server system.

Table 12-8. Values of PerfInterval Data Objects from a vCenter Server System

key	name	sampling period	length	enabled	level
1	Past Day	300	86400	TRUE	1
2	Past Week	1800	604800	TRUE	1
3	Past Month	7200	2592000	TRUE	1
4	Past Year	86400	31536000	TRUE	1

Table 12-9 lists some examples of the types of statistics that can be collected at various levels. The actual counters can vary from version to version of the product.

Table 12-9. Statistics Level Settings

Level	Description
1	Counters defined with `average` rollup type for CPU, Memory, Disk, and Network; plus counters for System Uptime, System Heartbeat, and DRS (Distributed Resource Scheduler, tracked in the `clusterServices` group). Does not include counters for devices.
2	Counters defined with `average`, `summation`, and `latest` rollup types for CPU, memory, disk, and network; plus counters for system uptime, system heartbeat, and DRS (`clusterServices`). Does not include counters for devices.
3	Counters defined with `average`, `summation`, and `latest` rollup types for CPU, memory, disk, network, and all devices; plus counters for system uptime, system heartbeat, and DRS (`clusterServices`).
4	All counters defined for all entities and devices, for every rollup type, including `minimum` and `maximum`.

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

Modifying Historical Intervals

VMware recommends that you do not modify the default historical intervals. The `PerfInterval` data objects in the `historicalInterval` array build incrementally on each object in the array. Modifying the specifics of any given object in the array can result in problems. Also, changes to an interval are global and apply to all entities in the system.

If you must modify an interval to meet a special requirement, use the `UpdatePerfInterval` operation of `PerformanceManager` and follow these guidelines.

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

Optimizing Query Response Time

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

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

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

The optimization is available in VirtualCenter Server 2.5 and subsequent systems.

Using vCenter Server for Virtual Machine Operations

13

VMware vCenter Server supports comprehensive deployment, management, monitoring, and other services for multiple ESX/ESXi systems. vCenter Server runs as a database-backed service on a variety of Windows operating systems. The software licensing and installation process includes an instance of Microsoft SQL Server. vCenter Server also supports integration with Oracle Database Server.

A vCenter Server persists status, configuration, performance, and other information about each virtual machine, host, cluster, virtual application, and other entities that it manages to its database. Historical performance, event, task, and other data is also preserved in the vCenter Server database.

The API available on the vCenter Server includes several service interfaces that are not supported on unmanaged ESX/ESXi systems, including `AlarmManager` and `ScheduledTaskManager`. These service interfaces have been available in prior releases of vCenter Server, but with vCenter Server 4.0 and vSphere API 4.0, they support many new capabilities. For example, control over `Alarm` setting is more granular than in prior releases, false alarms or unnecessary repeated triggering of an alarm has been resolved. See "Using the Alarm Infrastructure" on page 137.

Many new features of vSphere 4.0 are available only through the API available on the vCenter Server Web service, including `ClusterProfileManager`, `ProfileComplianceManager`, `HostProfileManager`, `IpPoolManager`, `HostSnmpManager`, `VirtualMachineCompatibilityChecker`, and `VirtualMachineProvisioningChecker`, among others.

vCenter Server provides a range of options for deploying and configuring virtual machines, and for organizing virtual assets. vCenter Server provides container objects, such as folders and datacenters, for organizing virtual assets in an inventory comprising multiple ESX/ESXi systems. vCenter Server supports deploying multiple virtual machines by using a template that codifies a standard configuration. This chapter provides information about these approaches. It includes these topics:

- "Sample Code Reference" on page 120
- "Overview of the VirtualMachine Managed Object Type" on page 120
- "Using Templates to Deploy Standard Virtual Machines" on page 124
- "Migrating Virtual Machines Using VMotion" on page 125
- "Moving Storage Dynamically Using Storage VMotion" on page 127

Sample Code Reference

Table 13-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

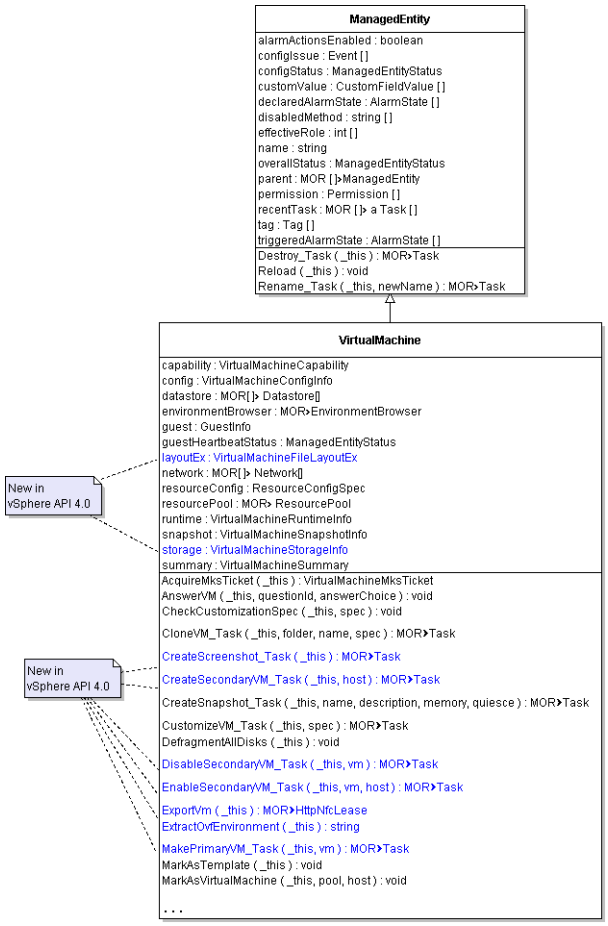
Table 13-1. Sample Applications Demonstrating VirtualMachine Deployment Using vCenter Server Systems

Java	C#
SDK\samples\Axis\java\com\vmware\samples\vm	**SDK\samples\DotNet\cs**
`VMClone.java`	`VMClone`
`VMDeltaDisk.java`	`RecordSession`
`VMLinkedClone.java`	`VMPowerOps`
`VMPromoteDisks.java`	`VMotion`
`VMRelocate.java`	
`VMotion.java`	
`VMotion25.java`	

Overview of the VirtualMachine Managed Object Type

A `VirtualMachine` managed object provides operations for managing virtual machines, such as creating snapshots of its state, creating new instances of itself by cloning, and participating with other virtual machine instances in fault-tolerant deployments.

Figure 13-1. VirtualMachine Extends ManagedEntity Type



A VirtualMachine managed entity has close to fifty operations, many of which are new in vSphere API 4.0 Figure 13-1 shows only some of the details. See the *vSphere API Reference* for complete information.

Table 13-2. VirtualMachineConfigSpec Data Object

Property	Type	Description and Usage
alternateGuestName	string	Full name for guest. Used only if guestId is specified as other or other-64.
annotation	string	User-provided description of the virtual machine. Because this property is optional in the virtual machine configuration, it is necessary to pass an explicit empty string in a ConfigSpec object to remove an annotation that is already present in the VirtualMachineConfigInfo for a virtual machine.
bootOptions	VirtualMachineBoot Options	Settings that control the boot behavior of the virtual machine. These settings take effect during the next power-on of the virtual machine.
changeTracking Enabled	boolean	Setting to control enabling/disabling changed block tracking for the virtual disks of this VM. This may only be set if the changeTrackingSupported capability is true for this virtual machine. Any change to this property will take effect the next time the virtual machine powers on, resumes from a suspended state, performs a snapshot create/delete/revert operation or migrates while powered on.
changeVersion	string	If specified, the changes are only applied if the current changeVersion matches the specified changeVersion. This field can be used to guard against updates that have happened between when configInfo is read and when it is applied.
consolePreferences	VirtualMachine ConsolePreferences	Legacy console viewer preferences that are used with power operations. For example, power on.
cpuAffinity	VirtualMachine AffinityInfo	Affinity settings for CPU.
cpuAllocation	ResourceAllocation Info	Resource limits for CPU.
cpuFeatureMask	VirtualMachine CpuIdInfoSpec[]	Specifies the CPU feature compatibility masks.
cpuHotAddEnabled	boolean	Indicates whether virtual processors can be added to the virtual machine while it is running, or not. This attribute can only be set when the virtual machine is powered-off.
cpuHotRemove Enabled	boolean	Indicates whether virtual processors can be removed from the virtual machine while it is running, or not. This attribute can only be set when the virtual machine is powered-off.
deviceChange	VirtualDevice ConfigSpec[]	Set of virtual devices being modified by the configuration operation.
extraConfig	OptionValue[]	Additional configuration information for the virtual machine. This describes a set of modifications to the additional options. An option is removed if the key is present but the value is not set or the value is an empty string. Otherwise, the key is set to the new value. Configuration keys that would conflict with parameters that are explicitly configurable through other fields in the ConfigSpec object are silently ignored.
files	VirtualMachine FileInfo	Information about virtual machine files.
flags	VirtualMachine FlagInfo	Additional flags for a virtual machine.
ftInfo	FaultTolerance ConfigInfo	Fault tolerance settings for this virtual machine.
guestId	string	Short guest operating system identifier.

Table 13-2. VirtualMachineConfigSpec Data Object (Continued)

Property	Type	Description and Usage
instanceUuid	string	128-bit UUID of a virtual machine, represented as a hexadecimal string. This identifier is used by the vCenter Server system to uniquely identify all virtual machine instances in the vSphere environment, including those that may share the same SMBIOS UUID. Normally, this property is not set by a client, allowing the vSphere environment to assign or change it when vCenter Server detects an identifier conflict between virtual machines. This identifier can be modified even when a virtual machine is powered on.
locationId	string	128-bit hash based on the virtual machine's configuration file location and the UUID of the host assigned to run the virtual machine. Normally, this property is not set by a client, allowing the Virtual Infrastructure environment to assign a location ID when the virtual machine is created. However, if the virtual machine's configuration file has been manually moved, it may be desirable to clear this property, setting it to an empty string, so the property is regenerated.
memoryAffinity	VirtualMachine AffinityInfo	Affinity settings for memory.
memoryAllocation	ResourceAllocation Info	Resource limits for memory.
memoryHotAddEnabled	boolean	Indicates whether memory can be added to the virtual machine while it is running, or not. This attribute can only be set when the virtual machine is powered-off.
memoryMB	long	Size of a virtual machine's memory, in MB.
name	string	Display name of the virtual machine. Any % (percent) character used in this name parameter must be escaped, unless it is used to start an escape sequence. Clients may also escape any other characters in this name parameter. Snapshots of virtual machines that have spaces in their names and are associated with ESX Server 2.x systems are not supported. Therefore, if you want the option to take snapshots of this virtual machine and you are associating it with an ESX Server 2.x systems, do not use spaces in the name.
networkShaper	VirtualMachine NetworkShaperInfo	Resource limits for network.
npivDesiredNodeWwns	short	The NPIV node WWNs to be extended from the original list of WWN numbers. This property should be set to the aggregate of existing numbers plus new numbers. Desired Node WWNs should always be greater than the existing number of node WWNs
npivDesiredPortWwns	short	The NPIV port WWNs to be extended from the original list of WWN numbers. This property should be set to the aggregate of existing plus new numbers. Desired Node WWNs should always be greater than the existing number of port WWNs
npivNodeWorldWideName	long[]	The NPIV node WWN to be assigned to a virtual machine. This property should only be used or set when the value of npivWorldWideNameOp property is set. Otherwise, an InvalidVmConfig fault is thrown. If the specified node WWN is currently being used by another virtual machine, a VmWwnConflict fault is thrown. For more information about WWN, see npivNodeWorldWideName.
npivOnNonRdmDisks	boolean	This property is used to check whether the NPIV can be enabled on the Virtual machine with non-RDM disks in the configuration, so this is potentially not enabling NPIV on VMFS disks. This property is also used to check whether RDM is required to generate WWNs for a virtual machine.

Table 13-2. VirtualMachineConfigSpec Data Object (Continued)

Property	Type	Description and Usage
npivPortWorldWideName	long[]	The NPIV port WWN to be assigned to a virtual machine. This property should only be used or set when the value of npivWorldWideNameOp property is set. Otherwise, an InvalidVmConfig fault is thrown. If the specified port WWN is currently being used by another virtual machine, a VmWwnConflict fault is thrown. For detail description on WWN, see npivPortWorldWideName.
npivTemporaryDisabled	boolean	This property is used to enable or disable the NPIV capability on a specific virtual machine on a temporary basis. When this property is set NPIV Vport will not be instantiated by the VMX process of the virtual machine. When this property is set port WWNs and node WWNs in the VM configuration are preserved.
npivWorldWideNameOp	string	The flag to indicate what type of NPIV WWN operation is going to be performed on the virtual machine. If unset, it indicates no change to existing NPIV WWN assignment (or not assigned) in the virtual machine.
npivWorldWideNameType	string	This property is used internally in the communication between the vCenter Server and ESX/ESXi system to indicate the source for npivNodeWorldWideName and npivPortWorldWideName when npivWorldWideNameOp is set. This property should only be set by the vCenter Server. If this property is set in a call to a vCenter Server, an InvalidVmConfig fault will always be thrown. In a call to an ESX/ESXi system, an InvalidVmConfig fault is thrown if the value of npivWorldWideNameOp is not set to set.
numCPUs	int	Number of virtual processors in a virtual machine.
powerOpInfo	VirtualMachine DefaultPowerOpInfo	Configuration for default power operations.
preserveSwapOnPowerOff	boolean	Whether to keep the virtual machine's swap file when it is powered off. This may only be set to true if the swapPreservationSupported capability is true for this virtual machine.
swapDirectory	string	A datastore path to a directory that may be used to store the swap file for this virtual machine. This may only be set if the swapPlacementSupported capability is true for this virtual machine. Any change to this property will take effect the next time the virtual machine powers on, resumes from a suspended state, or migrates while powered on.
swapPlacement	string	Virtual machine swap file placement policy. This may only be set if the swapPlacementSupported capability is true for this virtual machine. Any change to this policy will take effect the next time the virtual machine powers on, resumes from a suspended state, or migrates while powered on.
tools	ToolsConfigInfo	Configuration of VMware Tools running in the guest operating system.
uuid	string	128-bit SMBIOS UUID of a virtual machine represented as a hexadecimal string in 12345678–abcd–1234–cdef–123456789abc format. Normally, this property is not set by a client, allowing the vSphere environment to assign a UUID when the virtual machine is created. However, in some rare cases, such as a manual copy of a virtual machine, it may be necessary to set this property.

Table 13-2. VirtualMachineConfigSpec Data Object (Continued)

Property	Type	Description and Usage
vAssertsEnabled	boolean	Indicates whether user-configured virtual asserts are triggered during virtual machine replay. This setting takes effect during the next replay of the virtual machine. Enabling this functionality can potentially cause some performance overhead during virtual machine execution.
version	string	The version string for this virtual machine. This is used only while creating a new virtual machine, and can be updated by invoking UpgradeVM_Task for this virtual machine.
vServiceConfig	VmConfigSpec	Configuration of VService metadata for a virtual machine
vServiceConfigRemoved	boolean	Set to true, if the VService configuration should be removed

Using Templates to Deploy Standard Virtual Machines

A template is instance of a VirtualMachine managed object that has its template property set to true. Templates do not consume resources and their permissions derive from their containing folder only, not from their associated resource pool as do VirtualMachine objects.

Table 13-3 lists advanced operations available through the VirtualMachine managed object.

Table 13-3. Advanced VirtualMachine Operations Summary

Operation	Description
CloneVM_Task	Creates a clone of this virtual machine. If the virtual machine is used as a template, this method corresponds to the deploy command. Any % (percent) character used in this name parameter must be escaped, unless it is used to start an escape sequence. Clients may also escape any other characters in this name parameter.
ExportVm	Obtains an export lease on this virtual machine. The export lease contains a list of URLs for the virtual disks for this virtual machine, and a ticket giving access to the URLs.
ExtractOvfEnvironment	Returns the OVF environment for a virtual machine. If the virtual machine has no VService configuration, an empty string is returned. Also, sensitive information is omitted, so this method is not guaranteed to return the complete OVF environment.
MarkAsTemplate	Marks a VirtualMachine object as being used as a template. Note: A VirtualMachine marked as a template cannot be powered on.
MarkAsVirtualMachine	Clears the isTemplate flag and reassociates the virtual machine with a resource pool and host.
RemoveSecondaryVM_Task	Removes the specified secondary virtual machine from this fault tolerant group.

Creating Templates

You use a template to create multiple virtual machines having the same characteristics, such as amount of resources allocated to CPU and memory and type of virtual hardware attributes supported.

The only property that distinguishes a template from a virtual machine is the template property of the VirtualMachineConfigInfo data object that comprises the config property of a VirtualMachine instance: A template is an instance of a VirtualMachine object that has its template property set to true.

You use one of these two general ways to create a template:

- 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 MarkAsTemplate operation on the virtual machine instance. You use the managed object reference of the virtual machine to invoke this operation. Invoking this operation sets the config.template property to true, and the virtual machine is disabled.
- If you want to use an existing virtual machine as a template, yet also keep the virtual machine, use the CloneVM_Task operation of the virtual machine first, to create a duplicate of the virtual machine

Example 13-1 demonstrates the cloning approach, which makes the virtual machine inoperative.

Example 13-1. Cloning a Virtual Machine to Use as a Template

```
...
VirtualMachineRelocateSpec vmRelocSpec = new VirtualMachineRelocateSpec();
vmRelocSpec.setTransform(VirtualMachineRelocateTransformation.sparse);

VirtualMachineCloneSpec vmCSpec = new VirtualMachineCloneSpec();
vmCSpec.setLocation(vmRelocSpec);
vmCSpec.setTemplate(Boolean=TRUE);
vmCSpec.setPowerOn(Boolean=FALSE);

ManagedObjectReference taskMoRef = my_conn.cloneVM_Task(vmMoRef, folderMoRef, new
               String("Templ_1"), vmCSpec);
...
```

Example 13-1 uses the `VirtualMachineCloneSpec`, whose properties provide information about where the datastore information is located, the target host, the resource pool, and the kind of transformation (flat or sparse) to perform on the disks. If datastore information is not provided, defaults are provided. See the `VirtualMachineRelocateSpec` in the *vSphere API Reference* for more information.

Reverting a Template to Its Active Virtual Machine Status

You change a template back to an operational virtual machine by using the appropriate operation:

- Invoke the `MarkAsVirtualMachine` operation on the template. This clears the boolean property that defines the virtual machine as a template and reassociates the virtual machine with the resource pool and host to which it had originally been associated. However, the template no longer exists.
- To retain the template, clone the template by invoking the `CloneVM_Task` operation on the template. In the `VirtualMachineCloneSpec` (the spec parameter) for the operation, set the `template` property to `false`.

Migrating Virtual Machines Using VMotion

VMotion has been a capability of VMware servers for several releases of the product, but has many new features and capabilities supported in vSphere 4.0, including support for distributed virtual switches. VMotion has dependencies on both networking infrastructure and the host system hardware.

Migrating a virtual machine from one host to another requires properly configured networking of all virtual machines and source and target hosts. The virtual machine can be in any state, including powered on, powered off, or suspended.

Table 13-3 lists operations available through the `VirtualMachine` managed object for migrating or relocating virtual machines.

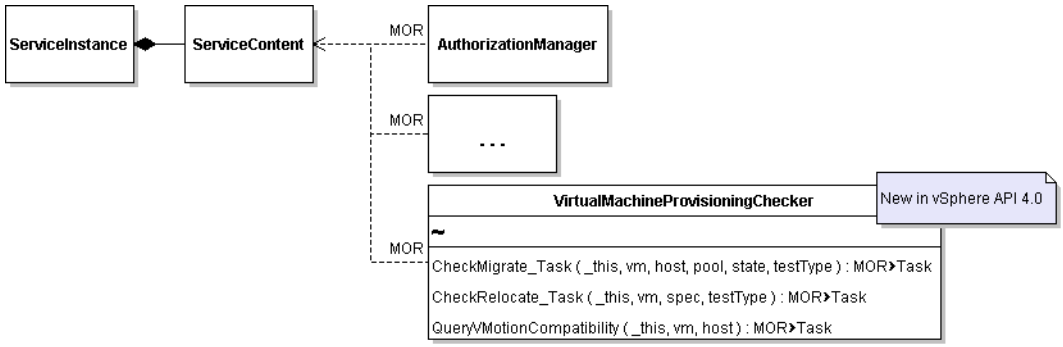
Table 13-4. VirtualMachine Operations for VMotion

Operation	Description
`MigrateVM_Task`	Migrates a virtual machine to a specific `ResourcePool` or `HostSystem`. The principal associated with the session invoking the operation must have the appropriate privileges for the state of the virtual machine: ■ Migrating a virtual machine that is running (`state` is `poweredOn`) requires the `Resource.HotMigrate` privilege. ■ Migrating a virtual machine that is shutdown or suspended (`state` is `poweredOff` or `suspended`) requires the `Resource.ColdMigrate` privilege.
`RelocateVM_Task`	Relocates the virtual disks associated with the specified virtual machine to a new location. Optionally, also moves the virtual machine to a different host. For Storage VMotion when the virtual machine is powered on, the principal invoking this operation requires the `Resource.HotMigrate` privilege and the `Datastore.AllocateSpace` privilege on the target datastore.

Checking the Validity of a Migration In Advance

As of vSphere 4.0, you use the `VirtualMachineProvisioningChecker` to determine if a proposed migration is viable or not.

Figure 13-2. VirtualMachineProvisioningChecker Service Interface



The `CheckMigrate_Task` operation determines whether a proposed `MigrateVM_Task` operation is possible. See the *vSphere API Reference* for more information.

Using the MigrateVM_Task Operation

The `MigrateVM_Task` operation moves all the configuration files that make-up a virtual machine. The disk files that make up the virtual machine remain in their original location.

Table 13-5 lists the parameters for the `MigrateVM_Task` operations. You can use this operation with virtual machines that are running or not. You use the state parameter to ensure that the virtual machine is moved only if it is in the state you specify.

Table 13-5. MigrateVM_Task Operation

Parameter	Type	Description
`_this`	`MOR`	Managed object reference of the virtual machine that you want to migrate to a new location.
`pool`	`MOR>ResourcePool`	Managed object reference to the target `ResourcePool` for the virtual machine. If this parameter is not set, the virtual machine stays assigned to its current `ResourcePool`. You must specify a host within the same compute resource.
`host`	`MOR>HostSystem`	Managed object reference to the target `HostSystem` for the virtual machine. The `HostSystem` must be associated with the object. The target host to run the virtual machine. This must specify a host that is a member of the `ComputeResource` object indirectly specified by the pool. For a stand-alone host or a cluster with VMware DRS, the property can be unspecified and the system selects a default from the same `ComputeResource` object as the `ResourcePool` specified by the pool parameter.
`priority`	`VirtualMachineMovePriority`	Priority of the migration. The priority value is one of the `VirtualMachineMovePriority` enumerated type values, specifically, `defaultPriority`, `highPriority`, or `lowPriority`.
`state`	`VirtualMachinePowerState`	Optional. If specified, the virtual machine is moved to the target location only if its state matches the specified state. The value of state is one of the `VirtualMachinePowerState` enumerated type values, specifically, `poweredOff`, `poweredOn`, or `suspended`.

Moving Storage Dynamically Using Storage VMotion

Storage VMotion allows you to move a running virtual machine from one VMFS volume to another. Storage VMotion was introduced in VMware Infrastructure 3.5, and has full support in vSphere 4.0.

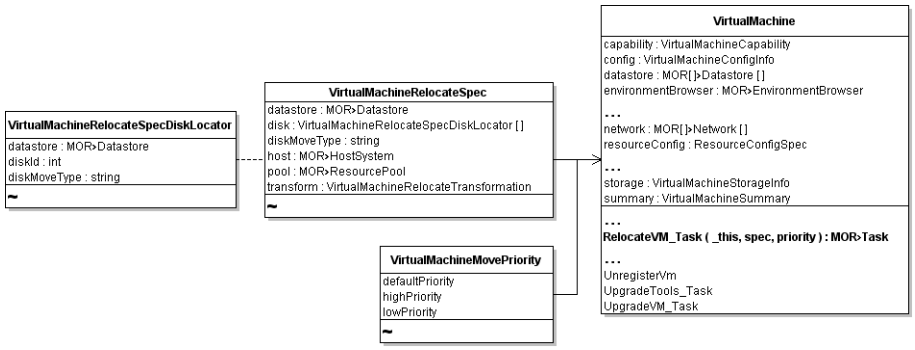
Neither the virtual machine nor its associated storage need to be taken offline. All datastore types are supported, including local storage, VMFS, and NAS (network attached storage). Storage VMotion uses the `RelocateVM_Task` operation (of `VirtualMachine`). `RelocateVM_Task` operation moves virtual disks to a new storage location, specified in the `VirtualMachineRelocateSpec` data object.

Table 13-6. RelocateVM_Task Operation

Parameter	Type	Description and Usage
`_this`	`ManagedObjectReference`	Reference to the `VirtualMachine` used to make the method call.
`spec`	`VirtualMachineRelocateSpec`	The specification of where to relocate the virtual machine.
`priority`	`VirtualMachineMovePriority`	The priority of the operation, specified as one of the three possible `VirtualMachineMovePriority` enumerations: `defaultPriority`, `highPriority`, `lowPriority`.

Figure 13-3 shows a partial UML diagram of `VirtualMachine` and some of the data objects that are used with `RelocateVM_Task`.

Figure 13-3. VirtualMachine RelocateVM_Task Operation



The data objects and enumeration to the left of the VirtualMachine managed object type represent inputs to the operation. Table 13-7 lists the properties of the `VirtualMachineRelocateSpec` data object.

Table 13-7. VirtualMachineRelocateSpec Data Object

Property	Type	Description and Usage
`datastore`	`MOR>Datastore`	Optional. Target datastore for the virtual machine. Defaults to current datastore if this parameter is not specified.
`disk`	`VirtualMachine RelocateSpecDiskLocator[]`	Optional. List that specifies datastore location for each virtual disk.
`diskMove Type`	`string`	Optional. Specifies the manner in which to move the virtual disk to the target datastore using one of the `VirtualMachineRelocateDiskMoveOptions` enumerations: ■ `createNewChildDiskBacking` ■ `moveAllDiskBackingsAndAllowSharing` ■ `moveAllDiskBackingsAndDisallowSharing` ■ `moveChildMostDiskBacking` Setting this property requires that the `deltaDiskBackingsSupported` property of `HostCapability` be set to `true`. If not specified, the move type defaults to `moveAllDiskBackingsAndDisallowSharing`. Applies to all the disks associated with the virtual machine. To override and specify the move type on a per-disk basis, use the `diskMoveType` property of `VirtualMachineRelocateSpecDiskLocator`.

Table 13-7. VirtualMachineRelocateSpec Data Object (Continued)

Property	Type	Description and Usage
`host`	`MOR>HostSystem`	Optional. Target host for the virtual machine. If neither host nor resource pool is specified, the current host is used. if resource pool is specified, and the target pool represents a stand-alone host, the host is used. if resource pool is specified, and the target pool represents a DRS-enabled cluster, a host selected by DRS is used. if resource pool is specified and the target pool represents a cluster without DRS enabled, an `InvalidArgument` exception be thrown.
`pool`	`MOR>ResourcePool`	Target resource pool for the virtual machine. ■ Optional for relocating or cloning to a virtual machine. Defaults to current resource pool of virtual machine unless specified. ■ Ignored for cloning to a template. ■ Required to clone from a template to a virtual machine.
`transform`	`VirtualMachine Relocate Transformation`	Optional. Specifies how to transform virtual disks during copy. (`VirtualMachineRelocateTransformation` is an enumerated data type consisting of `flat` and `sparse` options.) Any invalid transformations are ignored by the storage sub-system.

To move the disk files, use the `RelocateVM_Task` operation. See the samples listed in "Sample Code Reference" on page 120 for examples of using these operations.

Event and Task Management Using vCenter Server

14

An Event is a message generated by managed entities during or after various operations. The EventManager is the service interface that you can use to define your own events. This chapter includes these topics:

- "Sample Code Reference" on page 129
- "Understanding EventManager and the Event Data Object" on page 129
- "Creating Custom Events" on page 132
- "Obtaining Information about Events Using a HistoryCollector" on page 133

Sample Code Reference

Table 14-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 14-1. Sample Applications that Demonstrate Event and EventManager

Java SDK\samples\Axis\java\com\vmware\samples\events	C# SDK\samples\DotNet\cs\
EventFormat.java	EventFormat
EventHistoryCollectorMonitor.java	EventHistoryCollectorMonitor
VMEventHistoryCollectorMonitor.java	VMEventHistoryCollectorMonitor

Understanding EventManager and the Event Data Object

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

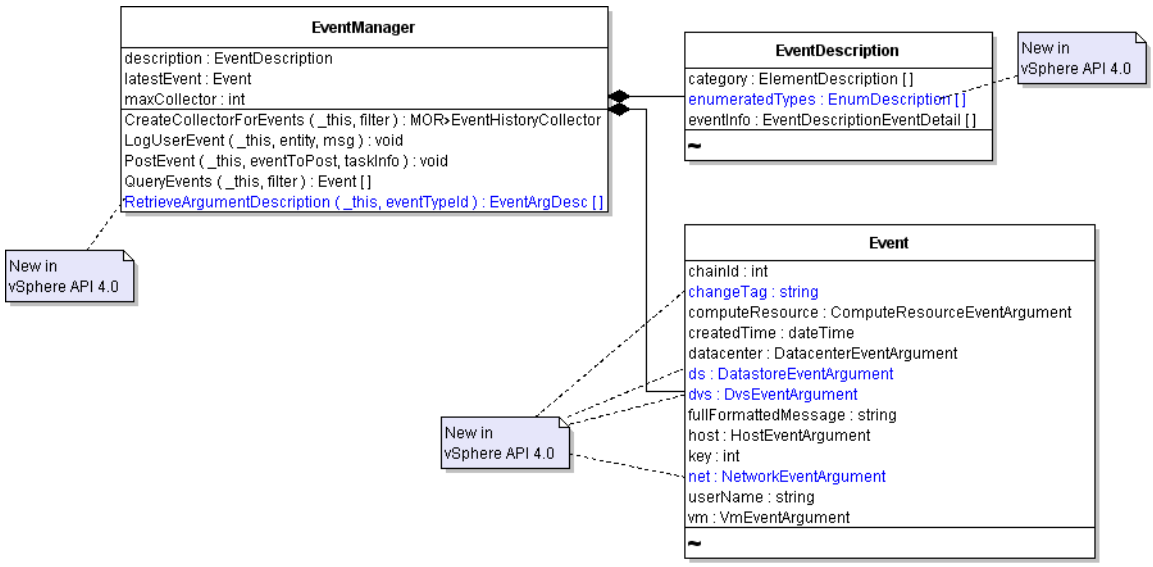
Information from Event objects generated on an unmanaged ESX/ESXi system display in the Events tab of the vSphere Client. For managed hosts, information from Event objects displays in the Tasks & Events tab of vSphere Client. For developers, the EventManager provides access to Event data objects.

EventManager Managed Object

The `EventManager` is the service interface for working with the event infrastructure. Figure 14-1 shows `EventManager` and some of its related objects. An `EventManager` has these properties:

- A `description` property defined as an instance of a `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 server.

Figure 14-1. EventManager Managed Object and Event Data Object



In vSphere API 4.0, the event infrastructure includes many new properties and data objects to support new capabilities of vSphere 4.0.

For example, `changeTag`, `ds`, `dvs`, and `net` are all new properties designed to support enhancements to the `Alarm` and `Event` infrastructure, or to support new capabilities, such as `DistributedVirtualSwitch` managed object.

See the *vSphere API Reference* for more information.

Event Data Objects

The `Event` data object has many subtypes that define the specific kinds of events that the system generates. Figure 14-2, "Event Data Object is a Base Type Used by Many Generations of Subtypes," on page 131, shows only a few of the subtypes that extend the `Event` data object. In many cases, the subtype contains no additional properties beyond those of the `Event` base type, but in other cases, the subtype provides additional information.

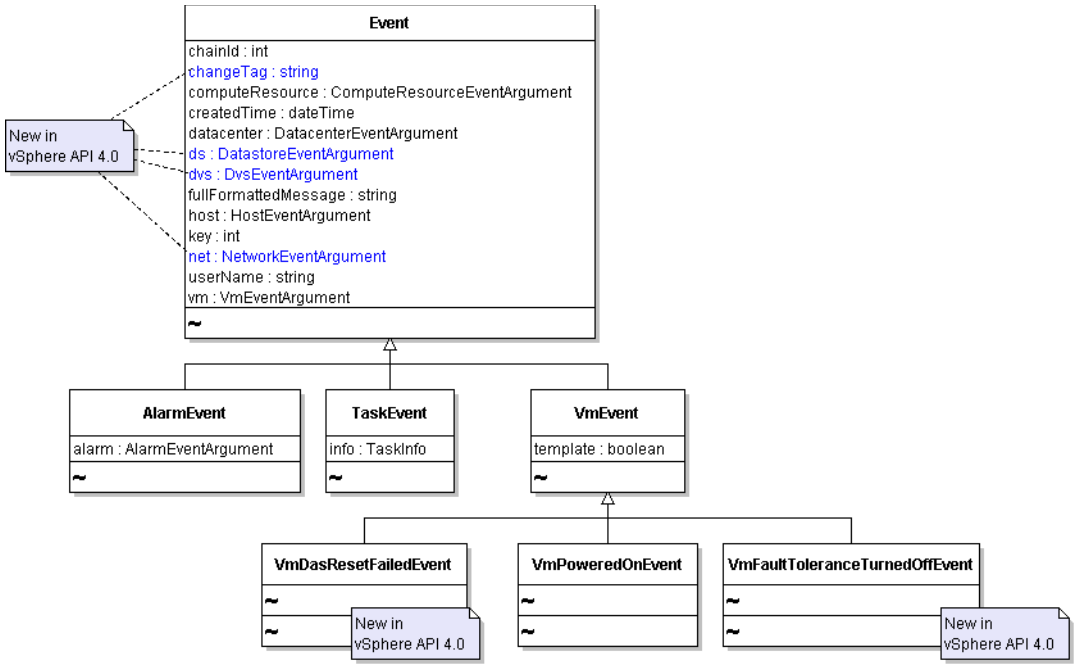
For example, the `TaskEvent` is a subtype of `Event`. In addition to containing all the properties of the `Event` type that are appropriate for the system, a `TaskEvent` also includes an `info` property that is defined as an instance of a `TaskInfo` object. The `TaskEvent` contains information such as the `entityName` of the object that generated the event. See "Understanding the TaskInfo Data Object" on page 85 for more information about the `TaskInfo` data object.

These are some examples of common event objects output 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
```

The VmPoweredOnEvent is also shown in Figure 14-2. As shown at the console, the event data objects are not formatted and do not provide any context information.

Figure 14-2. Event Data Object is a Base Type Used by Many Generations of Subtypes



Formatting Event Message Content

To format an event message, you can use the pre-defined string contained in the fullFormattedMessage property of the Event object.

You can also format an Event message based on the contextual information. At runtime, the events generated by the system are populated with the appropriate values for their source, such as the managed object reference of the compute resource associated with the event. The properties containing arguments associated with the source of an event include the computeResource, datacenter, ds, dvs, host, net, and vm properties of the Event data object (see Figure 14-1, "EventManager Managed Object and Event Data Object," on page 130).

You use the properties of an Event object in conjunction with the EventDescriptionEventDetail data object to format event message content appropriate as output for your application. Table 14-2 lists the properties of EventDescriptionEventDetail data object.

Table 14-2. EventDescriptionEventDetail Data Object Type

Property	Type	Description
category	string	Category of the Event, such as info, error, user, or warning.
description	string	A localized description of the Event designed for presentation to end-users. For example, the English description of the VmPoweredOnEvent might read VM Powered On. Use when creating event-based alarms to which end-users must respond.
formatOnComputeResource	string	Description pattern that is appropriate in the context of a specific cluster. For example, an Event generated by powering on a virtual machine in a cluster using this format the following text pattern: {vm.name} on {host.name} is powered on
formatOnDatacenter	string	Description pattern that is appropriate in the context of a specific datacenter. For example, a renaming event that uses this format uses the following pattern: Renamed {vm.name} from {oldName} to {newName} The {oldName} and {newName} are properties of the source VmRenamedEvent object.

Table 14-2. EventDescriptionEventDetail Data Object Type (Continued)

Property	Type	Description
formatOnHost	string	Description pattern that is appropriate in the context of a specific host. For example, an Event generated by powering on a virtual machine uses the following pattern: {vm.name} is powered on
formatOnVm	string	Description pattern that is appropriate in the context of a specific virtual machine. For example, an Event generated by powering on a virtual machine uses the following pattern: Virtual machine on {host.name} is powered on
fullFormat	string	Complete description pattern that includes all contextual information about the source of the Event, including datacenter, host name, and virtual machine name. For example, the following is a fully formatted text pattern for an event generated by starting a virtual machine: {vm.name} on {host.name} in {datacenter.name} is powered on
key	string	Type of event being described.

The name property in any of the formatted output is derived from computeResource, datacenter, ds, dvs, host, net, and vm properties of the Event object.

For examples of defining an event message based on context, see the EventFormat.java or EventFormat.cs samples in the vSphere Web Services SDK package.

Creating Custom Events

The EventManager LogUserEvent operation enables you to create custom Event objects to augment the pre-defined events or providing markers that facilitate browsing Event history. You can associate your custom Event with any managed entity.

To define a custom Event

1 Obtain the managed object reference to the EventManager.

```
..
ManagedObjectReference _svcRef = new ManagedObjectReference();
ServiceContent _sic = my_conn.retrieveServiceContent(_svcRef);
ManagedObjectReference eMgrRef = _sic.getEventManager();
...
```

2 Obtain the managed object reference to the entity with which you are associating the Event.

3 Invoke the LogUserEvent operation using both these references and a string consisting of the Event message for the msg parameter of the operation.

User-defined Event objects display in the vSphere Client among the other events on the system, with the prefix User logged event: followed by the text submitted in your msg parameter. In other client applications, such as in the console-based Event sample applications, custom events display as com.vmware.vim.GeneralUserEvent objects.

Obtaining Information about Events Using a HistoryCollector

An `EventHistoryCollector` lets you gather information about the events that have been generated by the server. You create an `EventHistoryCollector` using the `CreateCollectorForEvents` operation of `EventManager`.

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. The `EventFilterSpec` includes properties, such as `eventTypeId`, that 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 a `EventHistoryCollector` object.

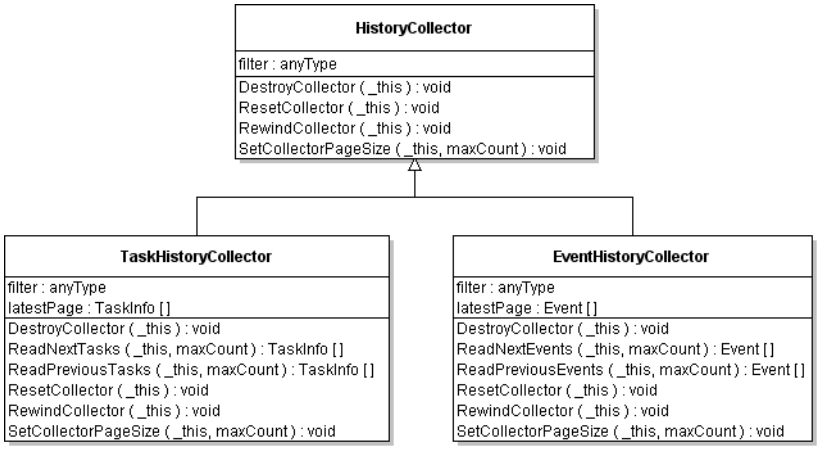
The `EventHistoryCollector` returned consists of the collection of objects from the vCenter Server database.

The `EventHistoryCollector` consists of an array of one or more `Event` data objects that meet the criteria specified. The `latestPage` property of the `EventHistoryCollector` object has a property that consists of the 1000 most recent objects in the collection.

Overview of HistoryCollector Managed Objects

An `EventHistoryCollector` is one of the specific subtypes of the `HistoryCollector` managed object type. The `TaskHistoryCollector` is another. You define the specification to use as the filter and pass it to the `CreateTaskHistoryCollector` operation of the `TaskManager`. Figure 14-3 shows the relationship of `HistoryCollector` to its two subtypes.

Figure 14-3. TaskHistoryCollector and EventHistoryCollector Descend from HistoryCollector



In addition to the `filter` property that they inherit from the `HistoryCollector` base type, the `EventHistoryCollector` and `TaskHistoryCollector` each has a `latestPage` property for their respective object types. The `HistoryCollector` managed object type provides operations for managing the life-cycle and scrollable view of a collection.

Table 14-3. Operations Common to HistoryCollector and Its Descendents

Operation	Description
`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`.

Table 14-3. Operations Common to HistoryCollector and Its Descendents (Continued)

Operation	Description
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 comprising a maximum of 1000 objects of the appropriate type (Task, Event). The array is sorted by creation date and time of the objects.

Table 14-4 lists the properties of TaskFilterSpec. To create a TaskHistoryCollector, you use one or more properties of TaskFilterSpec data object to define filter that meets your requirements. For example, rather than returning all Task objects associated with virtual machines, you might create a filter to collect only those 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.

Table 14-4. TaskFilterSpec Data Object

Property	Type	Description and Usage
alarm	MOR >Alarm	Optional. Use to create a filter based on specific instance of an Alarm.
entity	TaskFilterSpecByEntity	Optional. Use to create a filter based on specific managed entity types. Default returns values for all managed entities.
eventChainId	int[]	Optional. Use to create a filter that returns only Task objects associated by a specific chain of events.
parentTaskKey	string[]	Optional. Use to create a filter that returns only those Task objects that have the specified parentTaskKey.
rootTaskKey	string[]	Optional. Use to create a filter that returns only those Task objects that contain the specified rootTaskKey.
scheduledTask	MOR >ScheduledTask	Optional. Use to create a filter that returns a reference only to the specified ScheduledTask. Default returns all Task objects.
state	TaskInfoState[]	Optional. Use to create a filter that returns only those Task objects in the specified state: ■ error ■ queued ■ running ■ success
tag	string[]	Optional. Use to create a filter that returns Task objects with the specified tag. Include an empty string ("") in the array to return Task objects without a tag.
time	TaskFilterSpecByTime	Optional. Use to create a filter that returns Task objects with the specified time. If not specified, the collector includes all Task objects.
userName	TaskFilterSpecByUsername	Optional. Use to create a filter that returns Task objects for a specified user account. If not provided, then the tasks belonging to any user are collected.

Before you can create a TaskHistoryCollector, you must create an instance of the filter that defines the specifications of the Task objects that you want to collect. 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, state of the Task, or any of the other properties listed in Table 14-4.

By default, when a HistoryCollector is instantiated, the objects in the collection are retrieved in order of oldest to newest. To access the content of a HistoryCollector, you use a PropertyCollector to obtain the items from the latestPage property.

New Objects Appended to a HistoryCollector

Once a `HistoryCollector` exists, 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.

Deleting a HistoryCollector

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.

See the *vSphere API Reference* for complete information about `HistoryCollector`, `EventHistoryCollector`, and `TaskHistoryCollector`.

Using the Alarm Infrastructure 15

VMware vCenter Server provides a comprehensive alarm infrastructure for automating actions and sending various types of notification in response to specified server conditions. You can create alarms that respond to specific conditions, server state, or a specified event.An alarm must have an expression that evaluates to a true-false value. The alarm infrastructure includes the `AlarmManager` service interface, the `Alarm` managed object type, and many associated objects that make up the data structures required to support defining the server conditions that must exist for an alarm to become active. The alarm infrastructure integrates with other server components, such as events and performance counters. With the release of vSphere 4.0, the alarm infrastructure provides finer-grained control over alarm threshold settings than in prior releases. Many `Alarms` exist by default on vCenter Server.

This chapter includes these topics:

- "Sample Code Reference" on page 137
- "Overview of AlarmManager and Alarm Managed Objects" on page 137

Sample Code Reference

Table 15-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

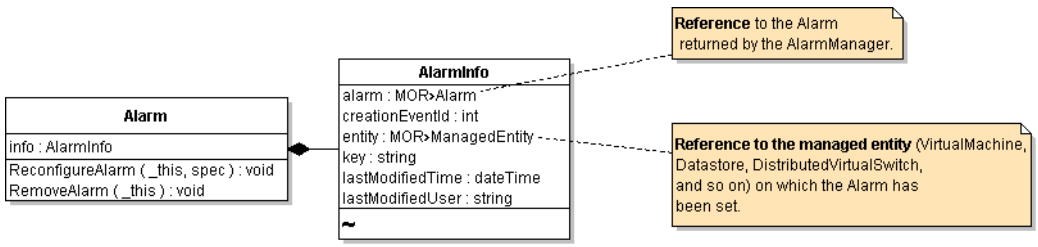
Table 15-1. Sample Applications that Use Alarm

Java	C#
SDK\samples\Axis\java\com\vmware\samples\alarms	**SDK\samples\DotNet\cs**
`VMPowerStateAlarm.java`	`VMPowerStateAlarm`

Overview of AlarmManager and Alarm Managed Objects

The `AlarmManager` is the service interface for creating, setting, and managing alarms. The `AlarmManager` instantiates an `Alarm` populated with the properties you specify and returns a managed object reference to the `Alarm`.

Figure 15-1. Alarm Managed Object



Obtaining a List of Alarms

The `AlarmManager` provides operations for obtaining information about existing alarms. You use the `GetAlarm` operation to obtain an array of references to all the `Alarm` managed objects defined for a specific managed entity. The parameters for the operation are a reference to the `AlarmManager` object and an optional reference to a specific 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.

As shown in Figure 15-1, an `Alarm` is a managed object with operations for its own reconfiguration and removal. It also has an `info` property that consists of an `AlarmInfo` data object. As with the Task managed object and TaskInfo data object, you can obtain information about the `Alarms` that are active on the system by collecting the properties of the `AlarmInfo` data object.

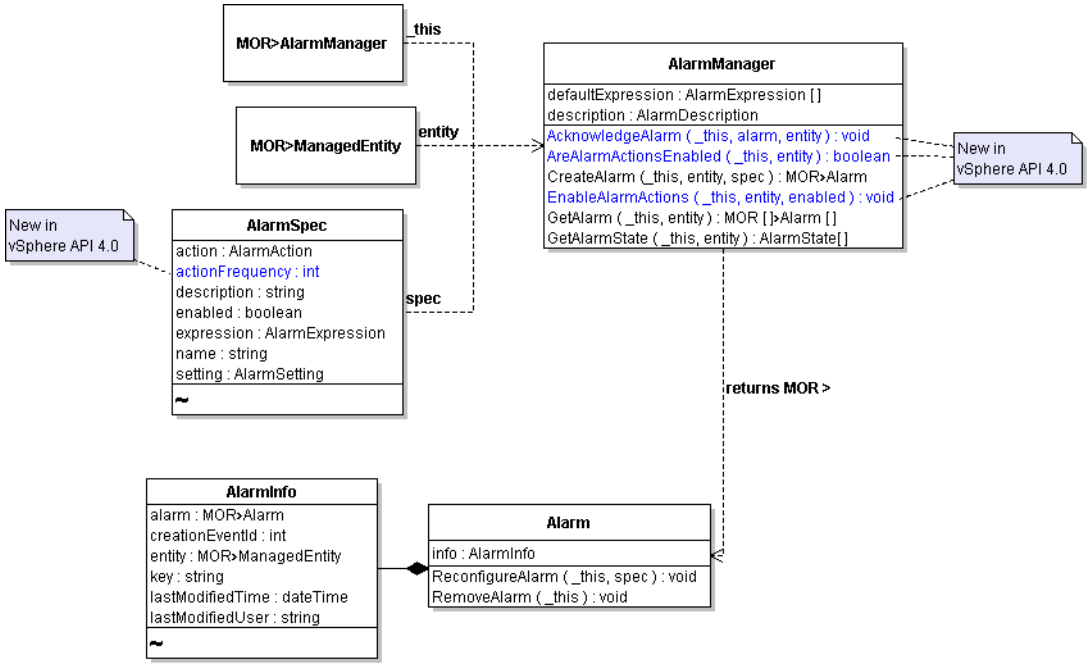
When you create the `Alarm`, you associate it with one or more specific managed entities, such as virtual machines, hosts, distributed virtual switches, datastores, and so on. When the conditions defined for the `Alarm` occur on the system, the `Action` specified for the alarm starts. The action might be sending an email, running a script, or invoking an operation of one of the service interfaces.

Defining an Alarm

Several different data objects provide the components needed to specify the settings for an alarm. The alarm infrastructure integrates with other infrastructure services, such as the performance monitoring system and the event infrastructure.

For example, you can define an alarm that monitors the values of a specific metric generated by one of the performance providers on the system, by specifying the metric in the expression you define. You might want to create an Alarm that sends an alert email message when CPU usage on a specific virtual machine exceeds 99% for more than 30 minutes.

Figure 15-2. CreateAlarm Operation Returns a Reference to an Alarm Managed Object



To create an Alarm on a specific managed entity

1 Obtain a managed object reference to the `AlarmManager` of the vCenter Server.

2 Obtain a managed object reference of the entity on which you want to set the `Alarm`.

3 Populate the AlarmSpec data object with the details for the Alarm. Many of these properties are nested data objects that you must create in your code before you can instantiate the AlarmSpec data object. See Table 15-2, "AlarmSpec Properties," on page 139 for more information about this data object type.

4 Pass the references and the data object to the CreateAlarm operation, as its parameters. The system returns a managed object reference to the Alarm, as shown in Figure 15-2.

Table 15-2. AlarmSpec Properties

Property	Type	Description
action	AlarmAction	Action to initiate when the Alarm becomes active.Specify one of the Action subtypes, including: ■ CreateTaskAction ■ MethodAction ■ RunScriptAction ■ SendEmailAction ■ SendSNMPAction See "Overview of the AlarmAction Data Object" on page 141.
actionFrequency	int	Number of seconds that the Alarm remains in the state required to initiate the specified Action.
description	string	Required. Description of the Alarm.
enabled	boolean	Required. Set to true to enable the Alarm. Set to false to disable the Alarm.
expression	AlarmExpression	Required. One or more of the AlarmExpression data objects combined in a way that evaluates to a true-false expression. ■ AndAlarmExpression ■ OrAlarmExpression ■ EventAlarmExpression ■ MetricAlarmExpression ■ StateAlarmExpression. See "Overview of the AlarmExpression Data Object" on page 139.
name	string	Required. Name of the Alarm.
setting	AlarmSetting	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.

Overview of the AlarmSpec Data Object

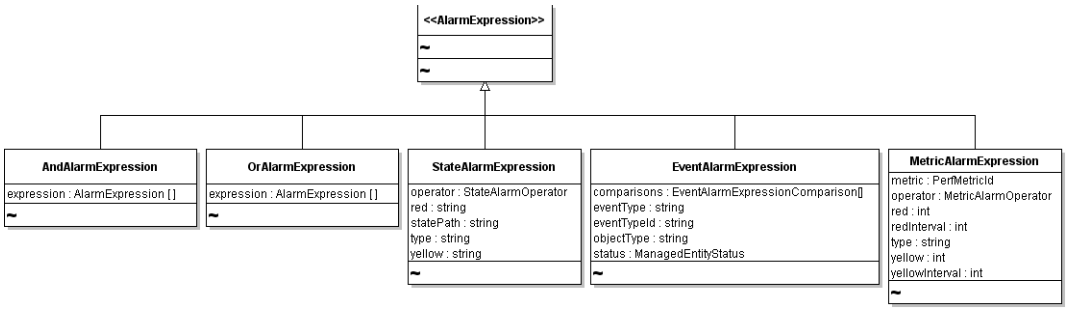
You set an Alarm on an entity by creating the AlarmSpec data object. The AlarmSpec data object includes properties for all aspects of an Alarm, including its expression and the action to take when the expression evaluates to true. As shown in Table 15-2, the expression and the action properties are defined as instances of two other complex data object types, AlarmExpression and AlarmAction, both of which are abstract data object types.

Overview of the AlarmExpression Data Object

You specify the conditions that trigger the action defined for an Alarm by providing an AlarmExpression data object for the expression property of the AlarmSpec data object. The AlarmExpression data object is an abstract type. It has several subtypes (see Figure 15-3, "AlarmExpression Abstract Data Type and Its Subtypes," on page 140).

The `AlarmExpression` subtypes allow you to specify thresholds on objects, state of objects, or specify specific events to monitor. You use the `AlarmExpression` data object type to specify the conditions under which you want the `Alarm` to become active.

Figure 15-3. AlarmExpression Abstract Data Type and Its Subtypes

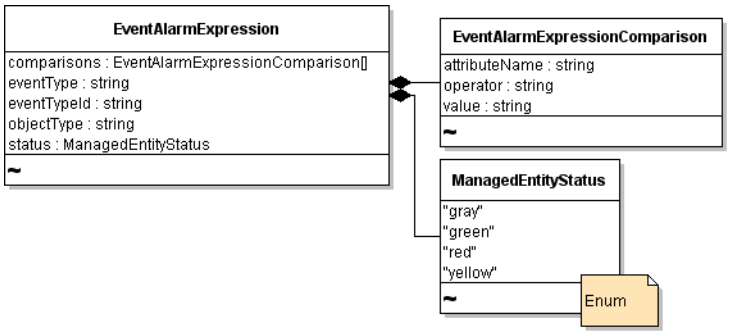


By using the appropriate subtype of `AlarmExpression`, you might set alarms for a variety of conditions, states, or events, such as these examples:

- Power state of a virtual machine or the state of a distributed virtual switch, by using a `StateAlarmExpression` data object
- Resource utilization metrics that exceed a specified limit, by using a `MetricAlarmExpression`
- Events such as power on or power off of primary or secondary virtual machines in a fault-tolerant cluster, by using the `EventAlarmExpression`

You create instances of the `StateAlarmExpression`, `EventAlarmExpression`, or `MetricAlarmExpression` data objects specify thresholds that trigger the alarm. To use a specific type of event as the basis for an alarm configuration, you use the `EventAlarmExpression`. The `EventAlarmComparison` data object lets you specify the property of the `Event` that should trigger the alarm and the operator to use as the basis for comparison.

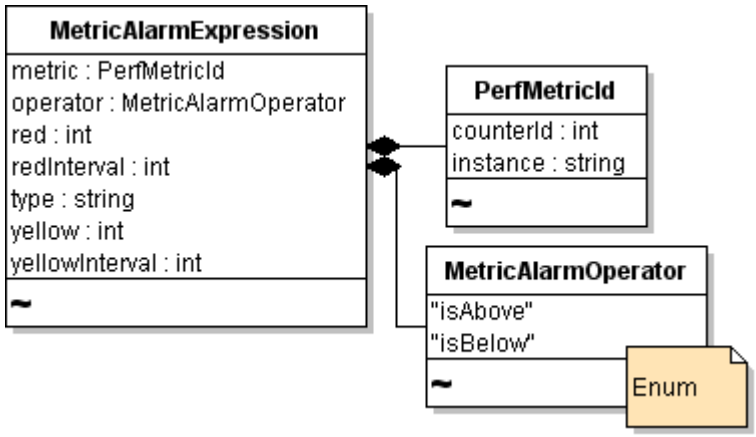
Figure 15-4. EventAlarmExpression



You can set an alarm to monitor specific performance metrics by using the `MetricAlarmExpression` data object. You set the `metric` property to the `PerfMetricId` of a specific performance metric that you want to monitor on the system.

Set the `red` or `yellow` properties to identity the level at which the metric value moves from green, to yellow, to red status.

Figure 15-5. MetricAlarmExpression Data Object



You must define either the `red` or `yellow` property in the `MetricAlarmExpression` data object that you create. You can also specify both properties, if appropriate for your application. For example, use the `yellow` property to specify a threshold value that signifies a warning condition, and use the `red` property to specify a threshold value that signifies a more serious condition. Use each of these properties in conjunction with the `isAbove` or `isBelow` `MetricAlarmOperator` enumerations to complete the definition of the threshold.

In conjunction with `red` and `yellow` properties, you use either `redInterval` or `yellowInterval` properties. The `redInterval` and `yellowInterval` properties are both new in vSphere API 4.0. These properties enable you to set the number of seconds that the performance metric must be in `red` or `yellow`, respectively, before the expression becomes `true`, thus triggering the defined action.

Using one or more instances of the `AndAlarmExpression` and the `OrAlarmExpression` data objects, you combine the objects into a complete expression that evaluates to a true-false value.

Overview of the AlarmAction Data Object

The system can respond to the alarms set on any object in many different ways:

- By invoking operations of other service interfaces
- By running a script
- By sending an email message to a specific email account
- By generating an SNMP trap

To invoke an operation of one of the service interfaces, you create a `MethodAction` data object. To run a script, you create an instance of the `RunScriptAction` data object that specifies the fully qualified path to the shell script on the vCenter Server. To send an email message to a system administrator, you use the `SendEmailAction` data object. And so on.

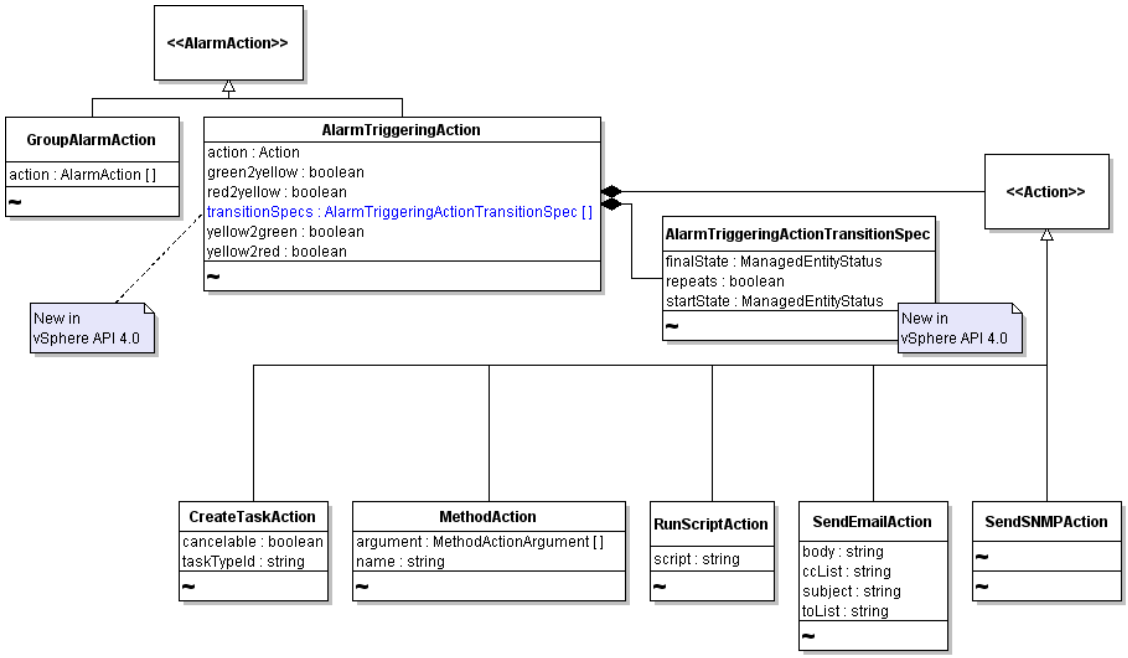
You specify the actions that the system should take by setting the `action` property of the `AlarmSpec` data object to the `AlarmAction` data object defined for the purpose. You use the `AlarmAction` data object in conjunction with one or more `AlarmTriggeringActionTransitionSpec` data objects.

The `AlarmAction` data object is an abstract type that has two descendent objects, `AlarmTriggeringAction` and `GroupAlarmAction`. The `GroupAlarmAction` data object is an array version of the `AlarmAction` base type, meaning that 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 `AlarmTriggeringAction` has an `action` property and a `transitionSpecs` property.
The `transitionSpecs` property is new in vSphere API 4.0. The `transitionSpecs` property consists of an instance of an `AlarmTriggeringActionTransitionSpec`, also new in vSphere API 4.0.

The `action` property of the `ScheduledTaskSpec` is defined as an instance of the `Action` data object.
The Action data object is an abstract type that has several descendent objects that model specific behaviors.

Figure 15-6. AlarmAction and Action Abstract Data Object Types and Subtypes



In vSphere API 4.0, the `AlarmTriggeringAction` data object has a new property, `transitionSpecs`, that provides discrete control over the timing, thresholds, and frequency of an `Alarm`. Rather than using `green2yellow`, `red2yellow`, `yellow2green`, and `yellow2red` properties to trigger alarms, use the `AlarmTriggeringActionTransitionSpec` to define state changes with complete control.

`AlarmTriggeringActionTransitionSpec` has properties that enable you to define a starting status and a final status for the state of the `Alarm`. The state of an alarm is contained in an `AlarmState` data object. You can also limit the number of `Alarm` objects actually triggered to a single `Alarm`, by specifying `false` for the `repeats` property of the `AlarmTriggeringActionTransitionSpec`.

For example, you can use the `MethodAction` data object type to invoke an operation on the server. The `MethodAction` data object contains these properties:

- `name`—Use the `name` property of `MethodArgument` to specify the name of the operation that you want to invoke at the scheduled time.
- `argument`—Use the `argument` property to pass the parameters required by the operation, if any. The argument property is defined as an array of `MethodArgumentAction` data objects.

Depending on the entity associated with the alarm, the `argument` property of `MethodAction` might not be needed. For example, the `PowerOffVM_Task` operation normally requires the managed object reference of the virtual machine you want to shutdown. If your alarm action is associated with a container object, such as a `Folder` or `Datacenter`, a parameter specifying a specific virtual machine reference is not needed.

When an alarm is activated, it generates an `Event` that gets posted to the `Event` history database. The action initiated by the `Alarm` might also post its own `Event` to the database, depending on the `Action` type.

See the *vSphere API Reference* for complete information.

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.

You can disable the `Alarm` rather than deleting it. Obtain a managed object reference to the `AlarmManager` and the entity on which the `Alarm` is set, and invoke the `EnableAlarmActions` operation, passing the value `false` for the `enabled` parameter. The `EnableAlarmActions` operation is new in vSphere API 4.0.

Scheduling vCenter Server Operations

16

The ScheduledTaskManager enables you to schedule operations on vCenter Server. The operation might be scheduled for a one-time run or for a regular repeating schedule. The ScheduledTaskManager is supported on vCenter Server. This chapter includes these topics:

- "Sample Code Reference" on page 143
- "ScheduledTaskManager Managed Object" on page 143
- "Creating a ScheduledTask Object" on page 144
- "Defining the Schedule for the vCenter Server Operation" on page 145

Visibility of many of the objects and return types discussed in this chapter depends on the permissions associated with user account using the client application. A visible entity is one for which the principal associated with the session has view privileges. For more information about permissions, see "Overview of User Models and Server Access Control Concepts" on page 33.

Sample Code Reference

Table 16-1 lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Table 16-1. Sample Applications that Demonstrate Using ScheduledTaskManager

Java SDK\samples\Axis\java\com\vmware\samples\scheduling	C# SDK\samples\DotNet\cs\
DeleteOneTimeScheduledTask.java	DeleteOneTimeScheduledTask
OneTimeScheduledTask.java	OneTimeScheduledTask
WeeklyRecurrenceScheduledTask.java	WeeklyRecurrenceScheduledTask

ScheduledTaskManager Managed Object

You can use ScheduledTaskManager to define actions to occur on vCenter Server at many different times:

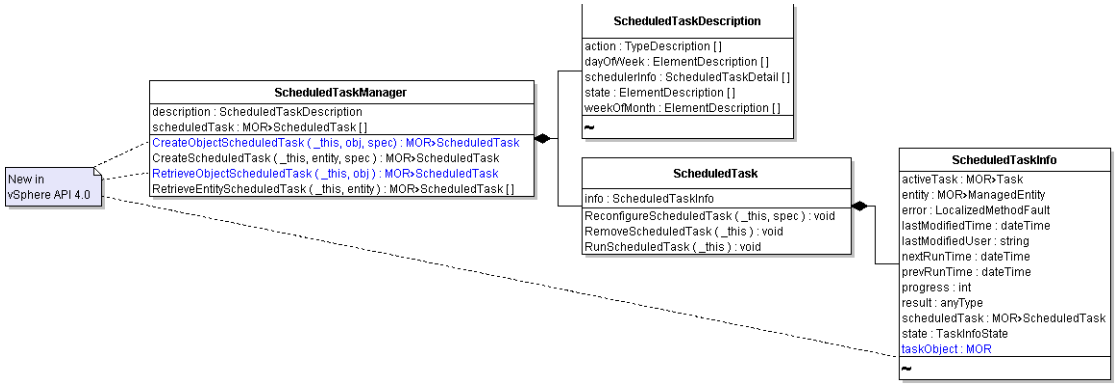
- When the vCenter Server system starts up operations, such as after a reboot
- At a specific time and day
- At hourly, daily, weekly, or monthly intervals

The type of actions you specify to occur at the scheduled time include running scripts on the server or invoking operations using one of the other service interfaces on the vCenter Server system. You apply the scheduled action to an entity in the inventory, such as a specific virtual machine or a host in a datacenter. ScheduledTask objects display in the vSphere Client Task & Events tab.

With vSphere 4.0, you can apply scheduled tasks to any managed object, not just managed entities, by using the CreateObjectScheduledTask operation. See the *vSphere API Reference* for details.

Figure 16-1 shows the ScheduledTaskManager service interface and associated data objects for its properties.

Figure 16-1. ScheduleTaskManager and ScheduledTask Managed Object Types



The 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, including its progress, state, previous and next runtimes, and the 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 (*_Task) operations), the managed object reference to the Task populates the activeTask property of ScheduledTaskInfo.

Retrieving ScheduledTasks for a Specific Managed Entity

You can use the RetrieveEntityScheduledTask operation obtain an array of all actions scheduled for a specific managed entity.

Table 16-2. RetrieveEntityScheduledTask Operation

Parameter	Type	Description
ScheduledTaskManager	MOR>ScheduledTaskManager	Reference to the ScheduledTaskManager.
entity	MOR>managed entity	Optional. Entity whose ScheduledTasks you want to obtain. If no entity is specified, this operation returns all ScheduledTasks that are visible to the principal associated with the session.

Creating a ScheduledTask Object

A ScheduledTask is a managed object type. It is created by the ScheduledTaskManager based on the specifications you provide in a ScheduledTaskSpec data object (see Figure 16-2, “Using ScheduledTaskManager to Create a ScheduledTask,” on page 145).

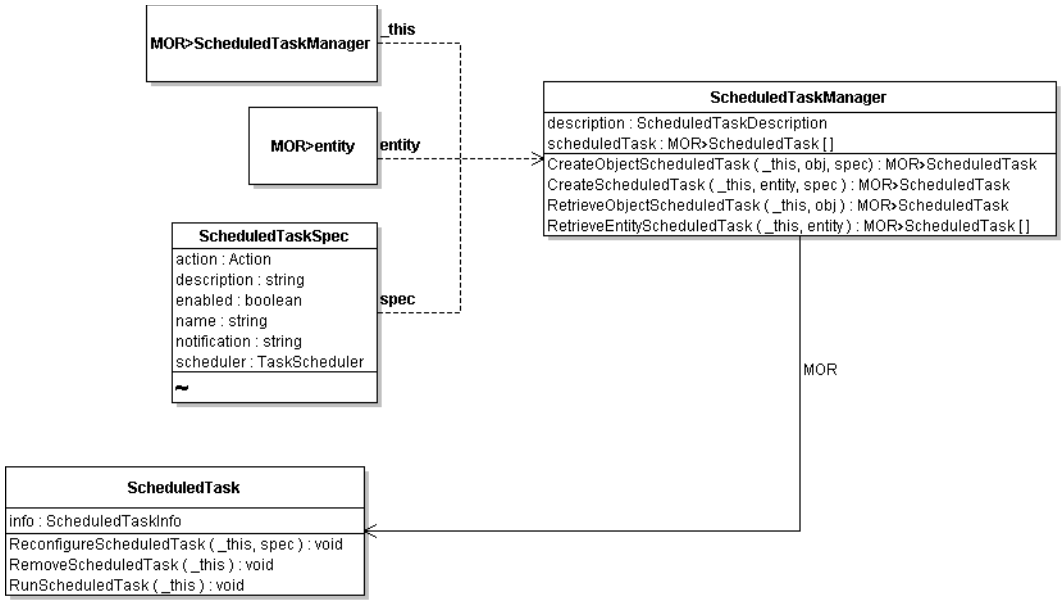
You create a ScheduledTask by invoking the CreateScheduledTask operation with the managed object reference of the ScheduledTaskManager, the managed object reference of the managed entity on which to apply the schedule, and an instance of the ScheduledTaskSpec object that defines the schedule and specifies the action to take at the specified time.

You can schedule actions on any entity in the inventory. The ScheduledTask object is associated with the entity you specify, taking into account the relative location of the entity in the inventory hierarchy. 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.
- If you specify a node object in the inventory, such as a virtual machine, the schedule applies only to the virtual machine.

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`. See "Understanding Container Objects and the Inventory" on page 26.

Figure 16-2. Using ScheduledTaskManager to Create a ScheduledTask



Defining the Schedule for the vCenter Server Operation

The `ScheduledTaskSpec` data object contains all the information to create a `ScheduledTask`.

Table 16-3. ScheduledTaskSpec Data Object Properties

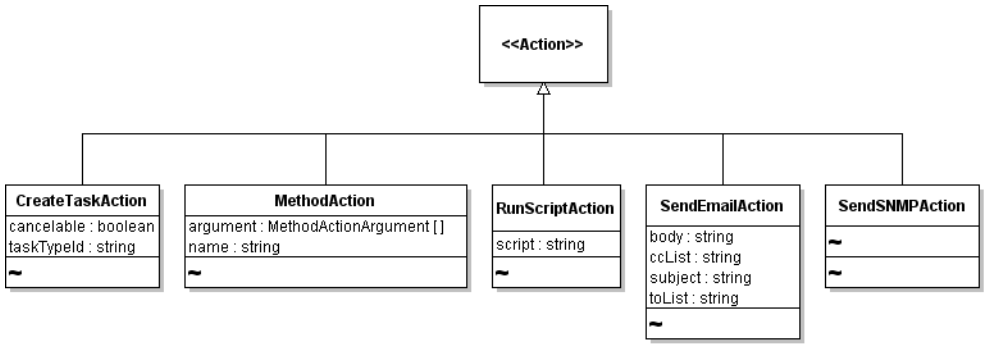
Property	Type	Description
`action`	`Action`	Specifies the action to take when the ScheduledTask runs. The Action data object is an abstract type that is extended by several specific action types. See "Defining the Actions" on page 146 for more information about the `Action` data object.
`description`	`string`	Description of the `ScheduledTask`.
`enabled`	`boolean`	Set to `true` to enable the ScheduledTask. Set to `false` to disable the ScheduledTask.
`name`	`string`	Name of the ScheduledTask.
`notification`	`string`	Optional. Use to specify the email address for sending notification messages about the ScheduledTask. To use notifications, the vCenter Server must have an SNMP email gateway configured. By default, this property is set to an empty string.
`scheduler`	`TaskScheduler`	Specifies the time, frequency, and other details of the schedule. The `TaskScheduler` data object is the base type for a number of specific schedule objects. See "Scheduling Recurring Operations" on page 146.

Both `action` and `schedule` properties of a `ScheduledTaskSpec` object consist of two other data objects, an `Action` and `TaskScheduler` data objects, respectively. See "Scheduling Recurring Operations" on page 146 for an overview of the `TaskScheduler` data object and its subtypes.

Defining the Actions

The `Action` data object is an abstract type. You define the specific action to occur on the server according to your schedule by using of the specific subtypes of `Action` data object (see Figure 16-3).

Figure 16-3. Action Data Object and Its Subtypes



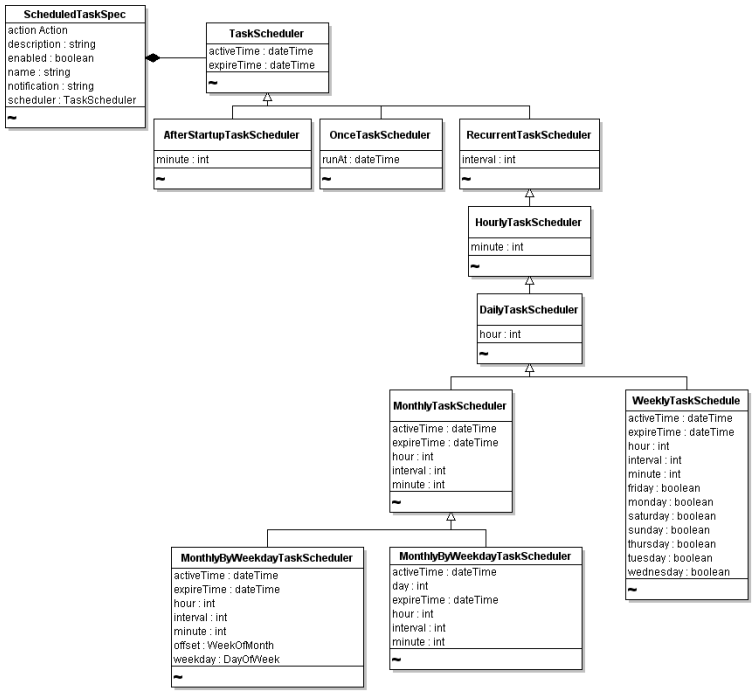
The `Action` data objects are also used by the `Alarm` infrastructure. See "Overview of the AlarmAction Data Object" on page 141 for more information about using `Action` subtypes.

See the *vSphere API Reference* for more information.

Scheduling Recurring Operations

The `TaskScheduler` data object is a base type that has many subtypes. To specify the times, days, or frequency of scheduled tasks, create the appropriate instances of `TaskScheduler` subtypes that you need and set the `scheduler` property of the `ScheduledTaskSpec`.

Figure 16-4. TaskScheduler Data Object Type and Subtypes



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 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 16-4 provides some usage information about the `TaskScheduler` subtypes shown in Figure 16-4. The examples in the table are Java code snippets.

Table 16-4. 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. Use the `minute` property to specify the number of minutes. 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 thirty 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.setWednesday(false);`
	`wts.setThursday(false);`
	`wts.setFriday(false);`
	`wts.setSaturday(false);`
	`wts.setSunday(true);`
	`dts.setMinute(30);`
	`dts.setHour(4);`

`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 30 minutes past noon	`MonthlyByDayTaskScheduler mbdts = new MonthlyByDayTaskScheduler();`
	`mbdts.setDay(31);`
	`mbdts.setInterval(3);`
	`mbdts.setMinute(30);`
	`mbdts.setHour(14);`

Table 16-4. TaskScheduler Data Object Subtypes (Continued)

TaskScheduler subtype	Usage
`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 30 minutes past midnight:	`MonthlyByWeekdayTaskScheduler mbwts = new MonthlyByWeekdayTaskScheduler();`
	`mbwts.setOffset(WeekOfMonth.last);`
	`mbwts.setWeekday(DayOfWeek.wednesday);`
	`mbwts.setHour(4);`
	`mbwts.setMinute(30);`

As of vSphere API 4.0, 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.

For example, the code in Example 16-1 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) timezone. For a server in the Eastern European Summer Time (EEST) zone, the setting is read by the system as 3:15 pm.

Example 16-1. ScheduledTask 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);
...
```

Deleting a Scheduled Task

To delete a `ScheduledTask` object you need its managed object reference to pass to the `RemoveScheduledTask` operation of the `ScheduledTask`. The `RemoveScheduledTask` operation cancels only the current run of the `ScheduledTask`. It does not cancel subsequent runs of the `ScheduledTask`.

To cancel an upcoming run of a `ScheduledTask`, you use the `ReconfigureScheduledTask` with a new `ScheduledTaskSpec` data object containing the new specifications for the schedule.

To cancel a `ScheduledTask` that spawns its own `Task` and cancel the `Task` as it runs, you create a `PropertyCollector` to obtain the reference to the `Task` and invoke its `CancelTask` operation. The `Task` must be cancellable. See "Overview of the Task Infrastructure" on page 83.

Managed Object Privileges Reference

A

VMware vSphere components are secured through a system of privileges, roles, and permissions (see Chapter 4, "Overview of User Models and Server Access Control Concepts," on page 33 for more information). This section lists privileges required to perform various operations, and privileges required to read properties. It includes these topics:

- "Privileges Required to Invoke Operations" on page 149
- "Privileges Required to Read Properties" on page 156
- "Privileges Defined for the Administrator Role" on page 158

Privileges Required to Invoke Operations

Table A-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 A-1.

Table A-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	
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

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

Operation	Privilege	VS	ESX/ESXi
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
CreateLocalDatastore	Host.Config.Storage	X	X
CreateNasDatastore	Host.Config.Storage	X	X
CreatePerfInterval	Performance.ModifyIntervals	X	X
CreateResourcePool	Resource.CreatePool	X	X
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

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

Operation	Privilege	VS	ESX/ESXi
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
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	
Login	System.Anonymous	X	X

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

Operation	Privilege	VS	ESX/ESXi
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
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

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

Operation	Privilege	VS	ESX/ESXi
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
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

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

Operation	Privilege	VS	ESX/ESXi
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
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

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

Operation	Privilege	VS	ESX/ESXi
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
UnregisterAndDestroy_Task	Folder.Delete[1]	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

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

Operation	Privilege	VS	ESX/ESXi
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	

1. The privilege is also required on the parent managed entity.

Privileges Required to Read Properties

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

Table A-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

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

Object	Property	Privilege
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
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
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

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

Object	Property	Privilege
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 A-3.

Table A-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
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

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

Privilege	Privilege
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
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

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

Privilege	Privilege
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
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

Diagnostics and Troubleshooting

B

This section provides information about using the `DiagnosticManager` service interface to facilitate troubleshooting. This appendix includes these topics:

- "Best Practices for Troubleshooting" on page 161
- "Overview of Configuration Files and Log Files" on page 162
- "Modifying the Log Level to Obtain Detailed Information" on page 164
- "Understanding and Using the DiagnosticManager" on page 167

Best Practices for Troubleshooting

Approach troubleshooting and problem-solving systematically, and take notes so you can trace your steps. Follow these industry-standard guidelines whenever you need to resolve an issue with your client application.

- Don't 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 in between client and server?
- Use the appropriate logging facilities for your programming language to capture runtime information for the client-application. See the `Log.cs` and `Log.java` files for C# and Java log class implementations, as used by the various SDK sample applications. These files are located in the `\samples` subdirectory of the vSphere Web Services SDK:
  - C# client logging example: `\SDK\samples\DotNet\cs\ClientUtils\Log.cs`
  - Java client logging example: `\SDK\samples\Axix\java\com\vmware\apputils\Log.java`
- Use the tools provided by VMware for analysis and to facilitate debugging. The following tools provide access to logging information. They require ESX or vCenter Server administrator privileges.
  - Use the VMware vSphere Client. The vSphere Client provides a graphical user interface that enables administrators to examine all ESX and vCenter Server log file, including log files for virtual machines, change the log level settings if necessary. The vSphere Client provides menu selections for creating reports that summarize configuration information, performance, and other details, and for exporting diagnostic bundles. The vSphere Client application maintains its own local log files.
  - Use the MOB. The MOB provides direct access to live runtime server-side objects. You can use the MOB to drill-down into the various objects and obtain property values. See Chapter 2, "Using the Managed Object Browser to Explore the Object Model," on page 16 for an overview of the MOB.
  - Use the API. The vSphere object model provides a DiagnosticManager service interface that enables 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.

Overview of Configuration Files and Log Files

ESX and vCenter Server both have various configuration files that control the behavior of the system. Most configuration file settings are set during installation but can be modified after installation. The log files capture messages generated by the kernel and various sub-systems and services. Both ESX and vCenter Server services maintain their own log files. Table B-1 lists various log files or reports, their locations and associated configuration files.

Table B-1. Server and System Logs Reference

Description	Log location	Filename or names	Configuration file
ESX service log	/var/log/vmware/	hostd.log [hostd-0.log, ...hostd-9.log]	config.xml
VirtualCenter 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	/var/log/vmkernel		syslog.conf, logrotate.conf
VMkernel Alerts and Availability report	/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 are the most relevant log files:

- hostd.log, which can be used as something of a SOAP monitor (when set to trivia log level as "Generating Logs" on page 166)
- vpxa.log (file located on a managed ESX host system, placed by the vCenter Server)
- vmware.log

In addition to viewing log files in real time, using the vSphere Client (Administration > System Logs tab), you can also generate reports and complete diagnostic bundles.

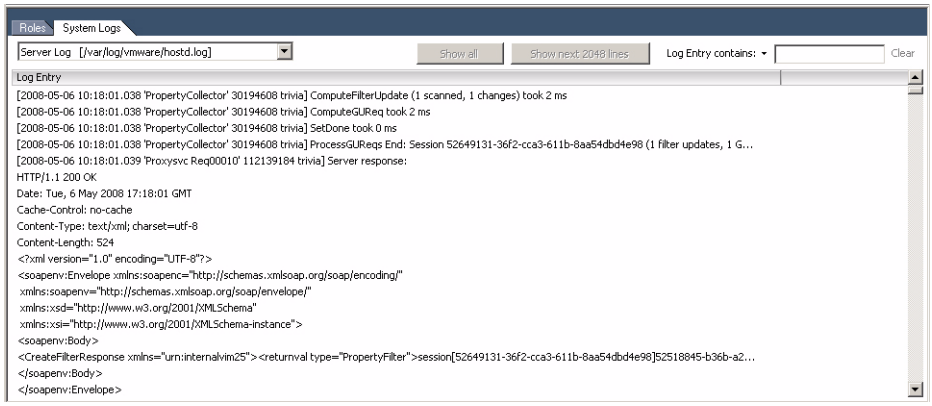
To generate a diagnostic bundle using the vSphere Client, select File > Export > Export Diagnostic Data.

For complete information about ESX and vCenter Server configuration and systems administration, see the vSphere documentation library, in particular, the *vSphere Basic System Administration* manual.

ESX Log File

The ESX log (hostd.log) captures information of varying specificity and detail, depending on the log level. Each request to the server is logged.

Figure B-1. ESX Log (hostd.log) Displayed using vSphere Client



The raw text form of an ESX (`hostd`) log file is shown in Example B-1.

Example B-1. Sample ESX Log (`hostd.log`) File 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 virtual machine running on a host system 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. Here are some points to note about virtual machine logs:

- ESX maintains six log files that rotate at each power-cycle (the default) or at a configured file size.
- ESX can be configured to maintain a specific number of log files. When the limit is reached, the oldest file is automatically deleted.
- VMware recommends a log file size of 500 KB.
- Messages generated by VMware Tools messages are logged separately.

Example B-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 systems.

By default, system files and folders are not visible in the Windows Explorer of File Manager. To locate VMware log files, you might need to change the setting. From the Windows Explorer, select Tools > Folder Options > View properties page and select the “Show hidden files and folders” option.

Modifying the Log Level to Obtain Detailed Information

The service running on ESX systems that generates log information is called `hostd` (host daemon). The default log level setting for `hostd` `info` (Table B-2). The vCenter Server logs are controlled by settings through the vSphere Client. This section provides information about changing log levels from the defaults.

Log Level Settings

The amount of information captured in the log files varies, depending on the level setting.

Table B-2. Log Level Settings

Log Level Setting	Description
None	Disables logging.
Error	Logging limited to error messages only.
Warning	Error messages plus warning messages are logged.
Info	Default setting for logging on ESX 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. See Example B-5, "SOAP Message Contained in Generated ESX Log," on page 167. Use for debugging and to facilitate client application development only. Not recommended for production environments.

Setting the Log Level on ESX Systems

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

Example B-3. ESX 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 `trivia` to obtain SOAP message content to use in debugging.

You can also use this technique to analyze an operation invoked using the he vSphere Client and then read through the extensive log files to extrapolate how you might accomplish the same tasks in your own code.

NOTE The process described below requires a system restart.

To change the log level in config.xml for hostd

1 Connect to ESX using putty or other secure shell.

2 Navigate to the location of the config.xml file:

```
cd /etc/vmware/hostd
```

3 Use vi or editor of your choice to open the config.xml file:

```
vi config.xml
```

The config.xml file opens for editing (Example B-3).

4 Scroll through the file to the <log> element. For example:

```
<log>
      <directory>/var/log/vmware/</directory>
      <name>hostd</name>
      <outputToConsole>false</outputToConsole>
      <level>info</level>
</log>
```

5 Change `<level>info</level>` to `<level>trivia</level>`.

6 Save and close the file.

7 Navigate to the init.d directory:

```
cd /etc/init.d
```

8 Restart the host agent:

```
./mgmt-vmware restart
```

After the service restarts, the new log level is in effect. Example B-4 shows the entire process.

Example B-4. Editing the ESX Configuration File and Restarting the Server

```
login as: root
root@sdkpubslab-01.eng.vmware.com's password:
Last login: Tue May  6 03:40:58 2008 from 10.16.250.244
[root@sdkpubslab-01 root]# cd /etc/vmware/hostd
[root@sdkpubslab-01 hostd]# vi config.xml
...
[root@sdkpubslab-01 hostd]# cd /etc/init.d
[root@sdkpubslab-01 init.d]# ./mgmt-vmware restart
Stopping VMware ESX Management services:
   VMware ESX Host Agent Watchdog                   [  OK  ]
   VMware ESX Host Agent                            [  OK  ]
Starting VMware ESX Management services:
   VMware ESX Host Agent (background)               [  OK  ]
   Availability report startup (background)               [  OK  ]
[root@sdkpubslab-01 init.d]#
```

Generating Logs

If you are connected to ESX, you can use the tail command to explicitly create a log file that captures the complete detail of any 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 (starts capturing new log data from the end of the file), 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. After the operation completes, stop the tail process by entering:

```
<ctrl><c>
```

The file contains the SOAP message content and other log messages sent and received by `hostd` during the execution. Example B-5 shows an excerpt of a SOAP message (`hostd`) associated with creating a new virtual machine.

Example B-5. SOAP Message Contained in Generated ESX Log

```
<soap:Body>
  <CreateVM_Task xmlns="urn:internalvim25">
    <_this type="Folder">ha-folder-vm</_this>
    <config>
      <name>TestVM_06</name>
      <version>vmx-04</version>
      <guestId>rhel3_64Guest</guestId>
      <alternateGuestName>Red Hat Enterprise Linux 3 (64-bit)</alternateGuestName>
      <files>
        <vmPathName>[sdkpubslab-01:storage1]</vmPathName>
      </files>
      <tools>
        <afterPowerOn>true</afterPowerOn>
        <afterResume>true</afterResume>
        <beforeGuestStandby>true</beforeGuestStandby>
        <beforeGuestShutdown>true</beforeGuestShutdown>
        <beforeGuestReboot>true</beforeGuestReboot>
      </tools>
      <numCPUs>1</numCPUs>
      <memoryMB>384</memoryMB>
...
</config>
    <pool type="ResourcePool">ha-root-pool</pool>
  </CreateVM_Task>
...
```

Be sure to change the log level back to info after obtaining the information you need from the log file.

Setting the Log Level on vCenter Server System

To change log-level settings on vCenter Server, you must use the vSphere Client. Connect to the vCenter Server instance.

To set logging level for vCenter Server using the VMware vSphere Client

1 Choose Administration and click Server Settings > Logging Options.

2 Choose Trivia from the pop-up menu and click OK.

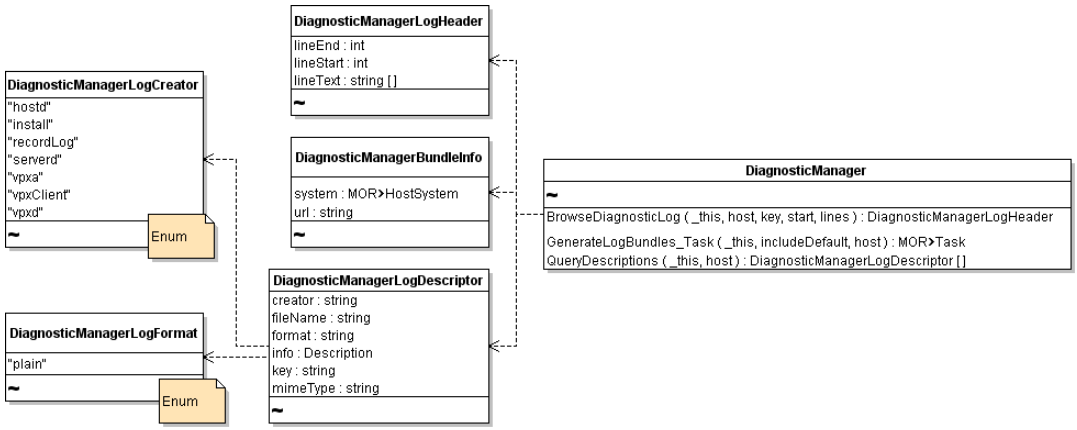
Understanding and Using the DiagnosticManager

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 "Log Level Settings" on page 165 for details.

The `DiagnosticManager` is a singleton managed object that works service-wide, rather than on a per-session basis. The `DiagnosticManager` has no properties, but provides operations for obtaining information about the logs and how they have been defined, and for initiating the process of generating a diagnostic bundle that can be sent to VMware support for analysis.

Figure B-2 shows a UML class diagram for `DiagnosticManager`. The `DiagnosticManager` is available on both ESX and vCenter Server systems.

Figure B-2. DiagnosticManager Managed Object and Associated Data Objects



As shown in Figure B-2, the three operations available through `DiagnosticManager` are:

- `BrowseDiagnosticLog`
- `GenerateLogBundleTask`
- `QueryDescriptions`

See the *vSphere API Reference* for complete documentation.

The `creator` of the log is the system or sub-system that controls the specific log. For example, `hostd` and `vpxa` are two examples of log creators.

The `creator` value is populated from the `DiagnosticManagerLogCreator` enumeration. Table B-3 lists all string values currently available from the `DiagnosticManagerLogCreator` enumeration that can populate the `creator` property of the `DiagnosticManagerLogDescriptor` data object.

Table B-3. DiagnosticManagerLogCreator Enumeration

Name	Description
`hostd`	Host agent
`install`	Installation
`recordLog`	System Record Log
`serverd`	Host server agent
`vpxa`	VirtualCenter agent
`vpxClient`	Virtual infrastructure client
`vpxd`	VirtualCenter service

Table B-4. ESX 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 B-4. ESX 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 directly from the ESX or vCenter Server instance using the MOB. See "Navigating the Object Model in the MOB" on page 17 for information about using the MOB to navigate to the `ServiceContent` data object, if necessary.

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.

The link name is the managed object ID for the managed object reference. The managed object ID name varies depending on whether you are accessing a vCenter Server or ESX system:

- For ESX, `ha-diagnosticsmanager` is the managed object ID.
- For vCenter Server, `DiagMgr` is typically the managed object ID.

Click the link to the reference to display the managed object reference to the `DiagnosticManager` in the MOB (Figure B-3). `DiagnosticManager` provides three operations that enable you to obtain information about the descriptions currently available in the log file and log file content.

Figure B-3. Managed Object Reference to DiagnosticManager

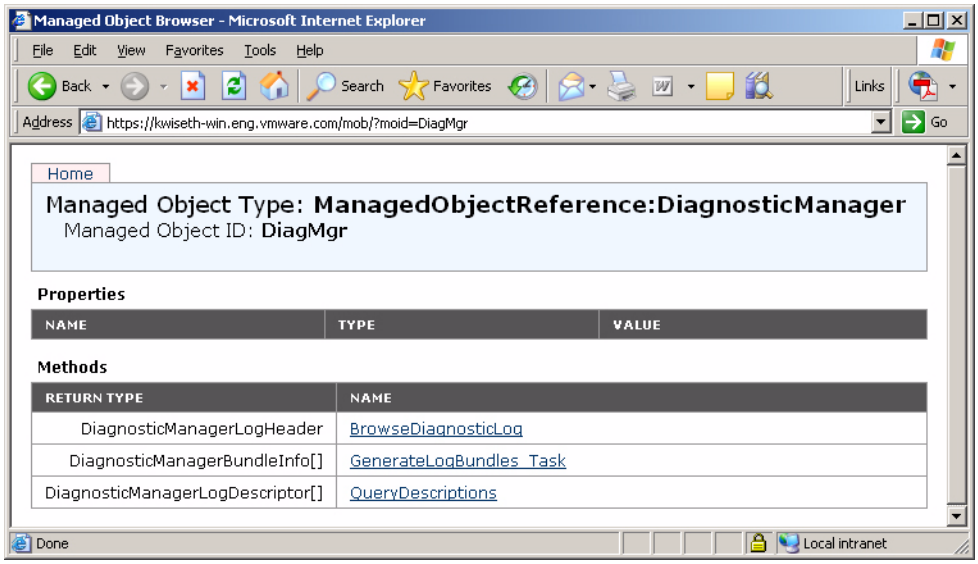
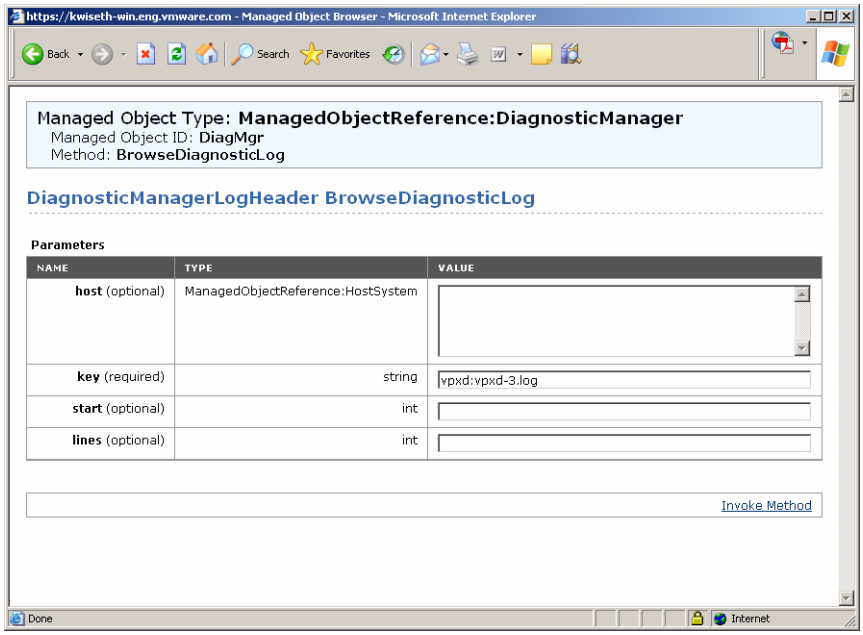


Figure B-3 shows the `DiagnosticManager` for a vCenter Server instance.

The vCenter Server `DiagnosticManager` tracks multiple ESX host systems, so 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 (Figure B-4).

Figure B-4. Using the MOB to Invoke DiagnosticManager Operations

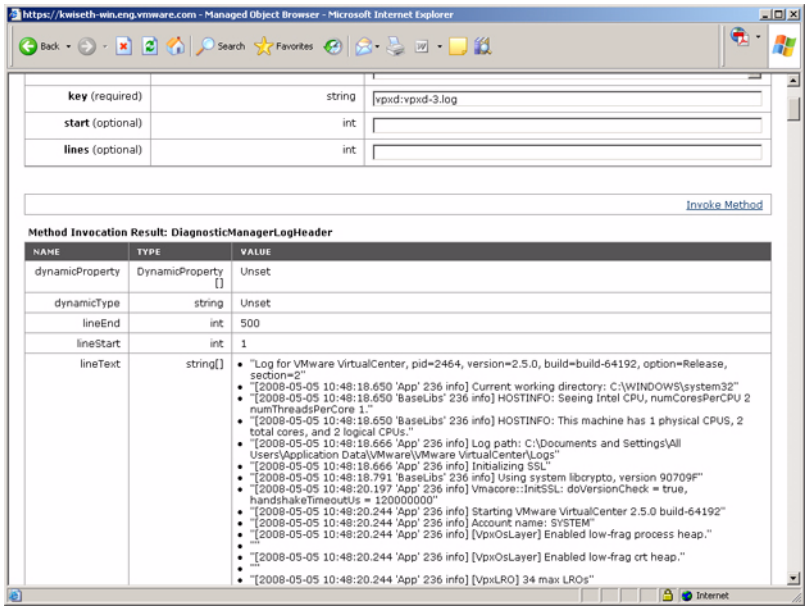


When you click the **Invoke Method** link, the contents of the log file for the selected host are returned by the vCenter Server system(Figure B-5). The contents are returned as a string array for the `lineText` property of `DiagnosticManagerLogHeader`.

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 the a diagnostic bundle created through the `GenerateLogBundles_Task`
- Available in the hostd.log
- Returned to a client application that you write

Figure B-5. DiagnosticManager Operations Available Through the MOB



The vSphere Client provides graphical-user interface access to these same operations. Example B-6 shows the content of a log file when the log level is set to `verbose`.

Example B-6. Sample Verbose Log File Data

```
...[2008-05-06 09:44:42.005 'TaskManager' 15227824 info] Task Created :
               haTask-ha-folder-vm-vim.Folder.createVm-755
[2008-05-06 09:44:42.005 'PropertyJournal' 19266480 verbose] PropertyJournalImpl::HasChanged:
               Overflow [N5Vmomi19PropertyJournalImplE:0xbb3cac0] recentTask
...'vm:/vmfs/volumes/47c5b0d6-16d11128-a260-0019b9c4369f/TestVM_05/TestVM_05.vmx' 15227824
               verbose] Adding task: haTask-ha-folder-vm-vim.Folder.createVm-755
[2008-05-06 09:44:42.196 'ha-host' 15227824 verbose] ModeMgr::Begin: op = normal, current =
               normal, count = 1
[2008-05-06 09:44:42.201 'ha-eventmgr' 15227824 info] Event 17 : Creating TestVM_05 on host
               sdkpubslab-01.eng.vmware.com in ha-datacenter
[2008-05-06 09:44:42.218 'HostsvcPlugin' 15227824 verbose] CreateEntry '112'
[2008-05-06 09:44:42.222 'ResourcePool ha-root-pool' 15227824 verbose] Added child 112 to pool
[2008-05-06 09:44:42.222 'Vmsvc' 15227824 verbose] Create VM initiated [112]:
               /vmfs/volumes/47c5b0d6-16d11128-a260-0019b9c4369f/TestVM_05/TestVM_05.vmx...
```

Table B-5 lists the `DiagnosticManagerLogDescriptor` object properties and values. All files are VCenter Server logs in plain format.

Table B-5. vCenter Server DiagnosticManagerLogDescriptor Values

creator	file name	format	info.label	key	mime type
vpxd	vpxd-0.log	plain	VirtualCenter Server Log	vpxd:vpxd-0.log	text/plain
vpxd	vpxd-1.log	plain	VirtualCenter Server Log	vpxd:vpxd-1.log	text/plain
vpxd	vpxd-2.log	plain	VirtualCenter Server Log	vpxd:vpxd-2.log	text/plain
vpxd	vpxd-3.log	plain	VirtualCenter Server Log	vpxd:vpxd-3.log	text/plain
vpxd	vpxd-4.log	plain	VirtualCenter Server Log	vpxd:vpxd-4.log	text/plain
vpxd	vpxd-5.log	plain	VirtualCenter Server Log	vpxd:vpxd-5.log	text/plain
vpxd	vpxd-6.log	plain	VirtualCenter Server Log	vpxd:vpxd-6.log	text/plain
vpxd	vpxd-7.log	plain	VirtualCenter Server Log	vpxd:vpxd-7.log	text/plain
vpxd	vpxd-8.log	plain	VirtualCenter Server Log	vpxd:vpxd-8.log	text/plain
vpxd	vpxd-9.log	plain	VirtualCenter Server Log	vpxd:vpxd-9.log	text/plain
vpxd	vpxd-index	plain	VirtualCenter Server Log	vpxd:vpxd-index	text/plain

Generating Diagnostic Bundles

Typically, diagnostic bundles are generated at the request of VMware technical support, by customers having problems. However, these diagnostic bundles can also provide a quick way for developers to 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 Launch the vSphere Client and connect to the ESX or vCenter Server system.

2 Select Administration > Export Diagnostic Data.

- For vCenter Server, you can filter for specific hosts whose logs you want to export and the location for storing the log files.
- For ESX host systems, you can specify the location for the bundle.

3 Click OK.

To collect ESX VMkernel files

If the VMkernel fails, error messages typically display briefly, and then the virtual machine reboots.

If you specified VMware core dump partition as part of your virtual machine configuration, the VMkernel also generates a core dump and error log.

In rare circumstances, very serious VMkernel problems can freeze the server without raising error messages or dumping the system core.

To use the command line to collect ESX log files

1 Use putty or other SSH tool to connect to the service console.

2 Navigate to the `/usr/bin` subdirectory.

3 Run the `vm-support` script:

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

4 This script collects all relevant ESX 3 system log files, configuration files, and other server details into a compressed file (the diagnostic bundle) whose name includes the date and time.

To obtain a managed object reference to the `DiagnosticManager`, first obtain a managed object reference to `ServiceContent`. Each `ServiceContent` data object has a `diagnosticManager` property, which is defined as a managed object reference to an instance of `DiagnosticManager` specific for the session.

Deploying Client Applications

C

This appendix explains how to deploy client applications. It includes these topics:

- "Understanding the Client Application Deployment Model" on page 173
- "Credential Store Administration Tool for Non-Interactive Clients" on page 173

Understanding the Client Application Deployment Model

The vSphere Web Services SDK includes WSDL files and other software artifacts that you can use to develop client applications. Included with the SDK are pre-compiled client-side libraries containing the proxy code needed to interact with the Web services API.

The VMware vSphere servers (ESX/ESXi, vCenter Server, for example) targeted by any client applications you develop must be appropriately licensed. However, you do not need any special license from VMware to create software using the VMware vSphere Web Services SDK for such environments.

Deploy your applications as you would any Web-services based client.

Credential Store Administration Tool for Non-Interactive Clients

The vSphere Web Services SDK includes a Java administration tool (command-line interface) that can be used by VMware vSphere administrators to manage the credential store for any client applications or agents. If you have an application that starts-up, automatically—a Windows or other service, for example, or a cron job that launches an application that must connect to ESX/ESXi or vCenter Server, you can use the Credential Store Administration Tool to store the user account and its password locally.

For example, say you are using VCB (VMware Central Backup) in your environment, and you want to configure a cron job to initiate the backup process each night at 3:00 a.m. to launch each night

The tool is contained in the unpacked vSphere Web Services SDK download file at this location:

```
%SDKHOME%\samples\Axis\java\com\vmware\security\credstore\CredentialStoreAdmin.java
```

Guidelines for Using the CredentialStoreAdmin Tool

The CredentialStoreAdmin tool provides complete access to the credential store backing file on the local machine. This is the actual text-based XML file that is created, opened, manipulated, and closed by the interfaces and implementations provided in the various credstore classes in the vSphere Web Services SDK.

IMPORTANT Never use the `CredentialStoreAdmin` tool to store the root or Administrator user account and password in the credential store. Create special user accounts for any automated or agent software and create special roles (or use appropriately privileged, non-Administrator roles) to apply to these user accounts. Accounts created expressly for the purpose of running automated applications or agents are the only type of account that should generally be stored in the credential store.

Before using the `CredentialStoreAdmin` tool, be sure you fully understand its behavior.

CAUTION The `CredentialStoreAdmin` tool can display user accounts and passwords stored in the credential store in clear text, at the console prompt. Do not allow this tool to be used by non-administrator users. Keep this tool in a password-protected subdirectory that only you or appropriate administrator can access.

CredentialStoreAdmin Tool and the Backing File

The `CredentialStoreAdmin` tool lets you access the local credential store backing file. The backing file is an XML-formatted text file containing hosts (target servers) and associated user names, with their respective passwords (Example C-1).

Example C-1. Populated Credential Store Backing File

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <viCredentials>
   <version>1.0</version>
    - <passwordEntry>
        <host>myTestServer</host>
        <username>root</username>
        <password>8Ofo8/bh8IPLnlD/YpB+mYV/jhIYm4V4+ulVwSVgK2BGPPV3U65i5IC2xxOYVYVnQ+2sgbzdvnT4brs
                        +pu+HhqkYdrXuyCs3NceT5mrrTPP5m0XEo1ZsZEup6t1PSVcO53ddrN+dnZqEUqNA8FFDhV4B
                        OKTkBAGQhCnRb0GBA0c=</password>
    </passwordEntry>
        - <passwordEntry>
        <host>192.168.25.14></host>
        <username>user94</username>
        <password>ChsJCQ0eS0pIeqB+6K6uv/gvaxqmzJJvL0ltudLnqBOz+2WQGvAadQ9ZS6gdGg6jWGTTOuoS+OE1Zsx
                        A37dv1MqRpvabWzAK0YQIPwysKLijkF9Nc7yP8/GQCQBL1nBXOSZLk4HTemZMOHzZE7ojMqu4
                        l/GUI1PfS9zNs4m44dY=</password>
    </passwordEntry>
   </viCredentials>
```

The hosts in the file can be specified by server name or by IP address—both IPv4 and IPv6 are supported. At runtime, when you're accessing the credential store and trying to obtain the password for the user name associated with the host, you must specify the host in the same form as it is stored in the backing file—hostname or IP address.

For example, if you store by hostname, you must enter the hostname in the application. Or if you use IP address, you must use the same style of IP address as is in the file (IPv6, IPv4).

Table C-1. Specify the Server by Hostname or by IP Address

Host	IPv4	IPv6
prod-server.mycompany.com prod-server	192.168.28.124	2001:0db8:3c4d:0015:0000:0000:abcd:ef12

You do not need to use fully qualified domain names (FQDN) if your local system can find the server. If you do use FQDN to specify server names, you must use the same names with your application code as with the CredentialStoreAdmin tool.

Passwords are obfuscated (Example C-1) but not encrypted, so it's important that users keep their credential store files protected on their local file systems.

CAUTION Never email the credential store backing file unless you encrypt the file. The credential store client APIs do not encrypt the contents of this file.

Furthermore, the `CredentialStoreAdmin` tool displays the password in clear text, not in its obfuscated form, so be careful when using the tool to display passwords.

The Java runtime (JRE) must be in local path to run the tool.

Perl and PowerShell versions of this command-line tool are also available with the VMware vSphere SDK for Perl and with the VMware vSphere PowerCLI.

To run the Java Credential Store Administration Tool on a client machine

1 Install the CredentialStoreAdmin tool from the vSphere Web Services SDK into a directory on the client machine that will run the agent software.

"Installing the tool" means to copy the contents of the `\credstore` subdirectory to a directory on the local machine and set the path so that the Java runtime can load the class and run it.

2 On the client machine, open a command prompt.

3 Navigate to the subdirectory containing the Credential Store administration tool.

4 Launch the CredentialStoreAdmin tool from a command prompt as follows:

```
java com.vmware.security.credstore.CredentialStoreAdmin add –h <servername> –u <username> –p
<password>
```

Table C-2 lists all the commands available to the `CredentialStoreAdmin` tool.

In addition to the arguments listed in the table, each of these commands can take the -f argument, to specify the name of the backing file. Use the -f option to create a Credential Store backing file using a name of your choice, including directory path.

If you create a credential store using a custom filename, you must pass the filename to the command when you invoke it.

Table C-2. CredentialStoreAdmin Command Summary

Command	Arguments		Description
`clear`	~	~	
`add`	`–h`	`host`	For the specified host name, adds to the local credential store the specified user account and password.
	`–u`	`username`	
	`–p`	`password`	
`get`	`–h`	`host`	For the specified host name and user name, obtains the password from the credential store.
	`–u`	`username`	
`help`	`<command>`		Displays at the command prompt the same basic text as captured in this table. Currently, entering a command name with the help command has no effect on the output of the -help command.
`list`	`––showpw`		Displays at the command prompt the contents of the credential store. Without any arguments, displays all host names with each user account. ■ Use the `––showpw` argument to display the passwords in clear text. (Be mindful of the security issues associated with doing this.) The `showpw` argument uses two dashes. ■ Use the `–h` argument with the specific hostname to obtain the user name stored in the credential store for that host. Enter the hostname in the same
	`–h`	`host`	
`remove`	`–h`	`host`	For the specified host name and user name, deletes the user name and password from the credential store.
	`–u`	`username`	

Each time a client application or the CredentialStoreAdmin tool itself accesses the credential store—whether to read its contents or write new data to it—the backing file is locked. After the read or write completes, the file is stored again to disk. This ensures that readers and writers both have a complete file to work with and do not start reading from a mid-point, or corrupt the data in the file by writing to a file that's open for a read.

Index

A

access, limiting **70**
accessing property values **49**
accessor methods **52**
accounts **70**
Action object **142**, **146**
Active Directory **37**
AddAuthorizationRole operation **42**
Administrator role **158**
Alarm managed object **137**, **138**
AlarmAction object **141**, **142**
AlarmExpression object **139**, **140**
AlarmManager object **137**
alarms **137**, **138**
    actions **146**
    deleting **142**
    disabling **142**
    list **138**
    samples **137**
AlarmSpec
    object **139**
    properties **139**
API version supported **64**
applying permission to an entity **71**
asynchronous applications **46**
authenticating users **39**
authentication **33**
authorization **33**
AuthorizationManager
    limiting access **70**
    managed object **40**
    operations **41**, **43**
    overview **39**
automating client application login **67**

B

backing file **174**

C

C#
    datatypes,map to XML **53**
    helper classes **65**
cancelling tasks **88**
changing permissions **43**
CheckForUpdates operation **79**
checking migration validity **126**
client applications
    characteristics **46**
    deploying **173**
    pattern **59**
    patterns **59**
client data, synchronizing **78**
client library **51**
client-side proxy code **51**
cloning virtual machines **125**
closing server connection **60**
collection performance **79**
complex content **55**
config.xml file **165**
configuration files **162**, **166**
configured storage **107**
connection object **59**
container object **26**
cookie **63**
counter groups **112**
CreateAlarm operation **138**
CreateFilter **57**, **77**
CreateUser sample **69**
creating a VirtualMachineConfigSpec **94**
creating custom events **132**
creating datastores **108**
creating new user accounts **70**
creating roles **41**
creating ScheduledTask object **144**
creating snapshots **97**
creating templates **124**
creating UserAccount objects **71**
creating virtual machines **93**
credential store **69**
    administration tool **173**
    backing file **68**, **174**
    backing file structure **68**
    client API libraries **67**
    client API methods **68**
    CreateUser sample **69**
    default locations **70**
    file format **68**
    introduction **67**
    least privilege **69**
    SimpleAgent sample **69**

CredentialStoreAdmin
commands **175**
tool **70**

D

Datacenter managed object type **26**
Datastore managed object **102**
datastores
configuration **110**
creating **108**
datatypes for MOB usage **58**
default log level setting **165**
deleting alarms **142**
deleting HistoryCollector **135**
deleting permissions **43**
deleting roles **41**
deleting scheduled tasks **148**
deploying client applications **173**
deploying virtual machines **124**
deployment model **173**
deployment sample **120**
deprecated items **51**
diagnostic bundles **171**
DiagnosticManager **167**
log descriptor values **168**
managed object **168**
operations **170**
operations available through the MOB **170**
DiagnosticManagerLogCreator enumeration **168**
diagnostics **161**
disabling alarms **142**
distributed virtual switch **12**

E

enabling single sign-on support **37**
Escape characters **53**
ESX/ESXi
config.xml file **165**
configuration file **166**
events **92**
inventory **30**
log **163**
log file **163**
log level **165**
PerfInterval **115**
sample log **163**
user model **35**
Event
object **129**, **130**
objects **92**, **130**, **131**
samples **129**
EventAlarmExpression **140**
EventDescriptionEventDetail data object type **131**
EventHistoryCollector **133**
EventManager
object **129**, **130**
samples **129**
events
creating **132**
exported from ESX/ESXi system **92**
formatting messages **130**
management **129**
message content **131**
examples
generated accessor and mutator methods **52**
returned object displayed in MOB **54**

F

file access **34**
file format, credential store **68**
Folder object **27**, **30**
folder object **26**
format for iSCSI names **105**

G

generated accessor and mutator methods **52**
generating logs **166**
granting privileges **42**
group permissions **44**
groups **37**
management **36**
membership **44**
permissions **44**

H

helper classes **65**
C#
sample applications **65**
java
sample applications **65**
sample applications **64**
historical intervals **117**
HistoryCollector **133**
appending objects **135**
deleting **135**
operations **133**
overview **133**
host name **174**
host provisioning **12**
hostd.log **163**
HostDatastoreSystem
creation operations **108**
modification operations **110**
query operations **107**

HostDatastoreSystem object **106**
HostDatastoreSystem operations **106**
HostHostBusAdapter data object **102**, **106**
HostLocalAccountManager
  object **35**
  operations **36**
HostNasVolumeSpec data object **109**
HostScsiDiskPartition data object **109**
HostStorageSystem
  object **102**
  operations **103**
HostSystem object **106**
HostVmfsSpec data object **109**
HTTP-based file access **34**

I

Import statements **51**
includes **51**
intervals **117**
inventory **25**, **26**, **30**
  introduction **30**
  permissions **29**
  privileges **28**
  root folders **27**
invoking operations through MOB **53**
IP address **174**
iSCSI
  names **105**
  overview **101**

J

java
  datatypes, map to XML **53**
  helper classes **65**
  TaskList application **90**

L

limit **94**
limiting access **70**
local storage **100**
local user accounts **38**
locale, setting **60**
localization **60**
log descriptor values **168**
log files
  overview **162**
  sample **171**
  vCenter Server **164**
  virtual machines **163**
log level
  default settings **165**
  ESX system **165**
  modifying **164**
  settings **165**
  vCenter Server system **167**
login **67**
logs **163**
  generating **166**
  server **162**
  system **162**

M

managed object browser **17**
managed object references **24**
managed objects **101**
  handling local user accounts **38**
ManagedEntity **26**
  descendents **13**
  ManagedEntity subtypes **25**
  permission objects **29**
ManagedObjectReference
  default methods **52**
  diagram **46**
  Type property **46**
  valid string values **52**
  XML schema **55**
managing storage **99**
mapping WSDL to client-side proxy code **51**
mapping XML datatypes to java and C# datatypes **53**
maximum allowed resource usage **94**
messages (event), formatting **130**
MetricAlarmExpression data object **141**
Microsoft Active Directory **37**
MigrateVM_Task operation **126**
migrating virtual machines **125**, **126**
minimum available resources **94**
mo_ref **22**
MOB **17**, **54**, **58**
  invoking DiagnosticManager operations **170**
  invoking operations **53**
  navigating object model **17**
  passing complex datatypes to operations **55**
  passing primitive datatypes to operations **54**
monitoring performance **111**
MOR (managed object reference, abbreviation for) **22**
MOR>ManagedObject **22**
MoRef **22**
multiple API versions **63**
mutator methods **52**

N

NAS storage overview **100**
nested data objects **57**
nested properties **50**, **52**
nested TraversalSpec **76**
new features **12**

O

ObjectContent data object **76**, **77**
ObjectSpec definition as XML Schema **57**
obtaining statistics **115**
operations
HistoryCollector **133**
passing complex datatypes **55**
passing primitive datatypes **54**
privileges required **149**
required privileges for **149**
optimizing query response time **118**

P

passing complex datatypes to operations **55**
passing primitive datatypes to operations **54**
PerfCounterInfo
data object **113**
matching PerfMetricId CounterId **114**
overview **112**
properties **113**
PerfEntityMetricBase data object **116**
PerfInterval data objects
ESX/ESXi system **115**
vCenter Server system **117**
PerfMetricId CounterId **114**
performance **79**, **111**
performance counter groups **112**
PerformanceManager
operations **112**
overview **117**
properties **112**
samples **111**
PerfQuerySpec **115**
permissions **29**, **42**, **43**, **44**, **71**
changing **43**
deleting **43**
group membership **44**
setting **43**
users and groups **44**
physical storage **99**
powering on VM **148**
predefined user roles **41**
privileges **28**, **40**, **42**
Administrator role **158**
and security management **28**
definition format **28**
ESX/ESXi
operations **149**
examples **28**
format **28**, **34**
invoking operations **149**
operations, required for **149**
properties, required for **156**
reading object properties **156**
vCenter Server operations **149**
profiles **12**
property values **49**
PropertyCollector **77**, **89**
managed object type **74**, **81**
operations **74**
operations **74**
operations, compared **78**
performance **79**
samples **73**
PropertyFilter managed object type **74**
PropertyFilterSpec **56**, **75**, **89**
introduction **75**
XML schema definition **56**
PropertySpec XML schema **56**
provisioning **12**

Q

QueryAvailablePerfMetric operation **116**

R

recent tasks **89**
recentTask Array **90**
recurring operations **146**
recursion **76**
RelocateVM_Task operation **127**
removing permissions **43**
reservation **94**
ResourceAllocationInfo, properties **94**
restarting server **166**
RetrieveEntityScheduledTask operation **144**
RetrieveProperties operation **76**
return datatypes **50**
revert to snapshot **97**
roles **40**, **41**
and security management **40**
creating **42**
modifying **42**
root folders **27**

S

sample applications **64**
sample ESX log **163**
samples
Alarm **137**
datastore operations **99**
Event **129**
EventManager **129**
multiple API versions **64**
overview **83**
PerformanceManager **111**

PropertyCollector **73**
ScheduledTaskManager **143**
SearchIndex **73**
storage operations **99**
verbose log file **171**
virtual machine deployment **120**
VirtualMachine **93**
sampling intervals **114**
sampling periods **114**
ScheduledTask **144**, **145**
cancelling **88**
deleting **148**
power on VM **148**
retrieving **144**
ScheduledTaskManager
creating ScheduledTask **145**
samples **143**
ScheduledTaskSpec object **145**
ScheduleTaskManager **144**
scheduling recurring operations **146**
scheduling vCenter Server operations **145**
SDK 2.5 client applications **14**
SDK 2.5 clients and vSphere 4.0 servers **13**
SearchIndex **80**
object **81**
operations **81**
samples **73**
security **69**
server access control **33**
server connection, closing **60**
server logs **162**
ServiceContent
data object **18**, **19**
property names **20**
return types **20**
values **20**
ServiceInstance
object **23**
properties **18**
session token **63**
session tokens **59**, **60**, **63**
SessionManager managed object **39**
setting permissions **43**
shares **94**
simple client application **62**
SimpleAgent sample **69**
single sign-on support **37**
snapshots **97**
special characters **53**
specifying server **174**
statistics **115**
statistics level settings **117**
storage **101**, **107**
requirements **100**
storage VMotion **127**
supported version **64**
synchronizing client data **78**
system logs **162**
system privileges **34**
system roles **40**, **41**

T

target server, specifying **174**
task infrastructure **83**
task management **129**
Task object **84**, **85**
TaskFilterSpec data object **134**
TaskHistoryCollector **133**
TaskInfo
object **85**
properties **88**
properties **85**
TaskInfoState values **90**
TaskList java application **90**
TaskManager **88**
object **84**, **85**
properties **84**
tasks, cancelling **88**
TaskScheduler subtypes **147**
templates **124**, **125**
retrieving **125**
tokens **60**, **63**
TraversalSpec **76**
troubleshooting **161**

U

Unified Modeling Language **21**
Update operation **77**
UpdateAuthorizationRole operation **42**
updating roles **41**
user accounts **38**, **70**
user management **36**
user models **33**
User permissions **44**
UserAccount object **71**
UserDirectory **37**
UserDirectory managed object **37**
UserManager
operations **38**
server source data **37**
users
overview **37**
permissions **44**

V

vCenter Server
DiagnosticManagerLogDescriptor values **171**
inventory **30**
log files **164**
scheduling operations **145**
user model, overview **36**
virtual machine operations **119**
version information **64**
versions **51**, **64**
sample using multiple versions **64**
VI SDK 2.5-based clients **14**
View object **80**
View objects **79**
ViewManager object **80**
ViewManager objects **79**
vimServiceVersions.xml **63**
virtual machines
cloning **125**
creating **93**
deployment using vCenter Server **120**
log files **163**
managing **93**
migrating **126**
migration **125**
operations **119**
powering on **148**
templates **124**
virtual storage **99**
VirtualMachine
ConfigSpec data object **121**
nested properties **50**
operations **95**
operations **124**
operations for VMotion **125**
overview **50**
RelocateVM_Task operation **127**
samples **93**
VirtualMachine managed object
overview **120**
VirtualMachineConfigSpec **94**
VirtualMachineProvisioningChecker service interface **126**
VirtualMachineRelocateSpec data object **127**
VMkernel availability report **164**
VMotion **125**
vNetwork Distributed Switch *See* distributed virtual switch **12**
vSphere 4.0 servers and SDK 2.5 clients **13**
vSphere Client
connected to ESX/ESXi **16**
Connected to vCenter Server **16**
vSphere components **11**
vSphere object model **15**

W

WaitForUpdates operation **79**
Web Server, reusing session token **63**
web service
connecting **59**
overview **45**, **50**
WSDL
file **45**
map to client-side proxy code **51**

X

XML schema **55**
ManagedObjectReference **55**
PropertyFilterSpec **56**
PropertySpec **56**
XML Schema Primitives to .NET mapping **53**
XML Schema Primitives to java mapping **53**