

vSphere Web Services SDK Programming Guide

17 APR 2018

VMware vSphere 6.7

vCenter Server 6.7

VMware ESXi 6.7



vmware®

001

WIZ, Inc. EXHIBIT - 1015
WIZ, Inc. v. Orca Security LTD.

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

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

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

Copyright © 2011–2018 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

Contents

About This Book	12
1 VMware vSphere and the vSphere Web Services SDK	14
vSphere Web Services SDK	14
SDK Developer Setup for the Web Services SDK	15
SDK Samples for the Web Services SDK	15
UML Diagrams Used in the Web Services SDK Programming Guide	15
2 vSphere Web Services API Programming Model	16
vSphere Client-Server Architecture	16
Web Services API as a Web Service	17
WSDL Files and the Client-Side Proxy Interface	17
Network Access to the vSphere Web Service	18
Language-Specific Classes and Methods	19
Mapping XML Data Types to Java and C# Data Types	20
Access to Managed Objects	20
Access to vSphere Server Data	21
Obtaining Information from a Server	21
Working with Data Structures	22
Accessing Property Values	22
Nested Properties and Property Paths in Composite Data Structures	23
xsd:anyType Arrays	23
Indexed Array and Key-Based Array Properties	26
Unset Optional Properties	26
Escape Character in Name and Path Properties	27
3 Client Applications for the Web Services API	28
vCenter Server Connections	28
Establishing a Single Sign-On Session with a vCenter Server	29
LoginByToken to vCenter Server By Using C#	29
vCenter Server Single Sign On Session Using C#	29
Persistent vCenter Server Sessions	30
Sample Code	30
Using LoginByToken in C#	30
LoginByToken to vCenter Server By Using Java	36
vCenter Server Single Sign-On Session Using Java	36
HTTP and SOAP Header Handlers in Java	36
Creating the HTTP Connection in Java	38

Using LoginByToken in Java	39
Establishing a Session with Username and Password Credentials	42
Overview of a Java Sample Application for the Web Services SDK	42
Build a Simple vSphere Client Application for the Web Services SDK	43
Java Client Example for the Web Services SDK	44
Web Server Session Token	46
Accessing the vSphere Web Services HTTP Endpoint with JAX-WS	47
Accessing the vSphere Server from a Web Services Client	49
Closing the Connection from a Web Services Client	50
Using the Java Samples as Reference	50
Multiple Versions of the vSphere API	52
Java and C# Sample Applications in the Web Services SDK	53
Java Samples in the Web Services SDK	53
C# Samples in the Web Services SDK	53
4 Datacenter Inventory	55
Inventory Overview for the Web Services SDK	55
Inventory Hierarchies and ServiceInstance	55
Folders in the Hierarchy	56
ESXi Inventory Hierarchy	57
Accessing Inventory Objects	57
Creating Inventory Objects	58
Privileges Required for Inventory Management	59
Privileges	59
Permissions	59
Managed and Standalone ESXi Hosts	60
5 Property Collector	62
Introduction to the PropertyCollector	62
Data Retrieval	63
Inventory Traversal and Object Selection	63
vSphere Data Objects for Property Collection	63
vSphere Methods for Property Collection	64
Retrieve Properties with the PropertyCollector	64
Simple Property Collector Program in Java for Web Services SDK	67
Inventory Traversal	72
TraversalSpec Traversal	72
Traverse the Inventory By Using the Property Collector	74
Inventory Traversal Example in Java	75
SelectionSpec Traversal	80
Client Data Synchronization (WaitForUpdatesEx)	86
Property Filters	87

WaitForUpdatesEx	87
Server Data Transmission	89
PropertyCollector Performance	89
SearchIndex	90

6 Authentication and Authorization 91

Objects for Authentication and Authorization Management	91
Authentication and Authorization for ESXi and vCenter Server	92
ESXi User Model	93
vCenter Server User Model	94
vSphere Security Model	94
Setting Up Users, Groups, and Permissions	96
Obtaining User and Group Information from UserDirectory	96
RetrieveUserGroups Method	97
Managing ESXi Users with HostLocalAccountManager	97
Methods Available for Local Account Management	98
Create a Local User Account on an ESXi System	98
Managing Roles and Permissions with AuthorizationManager	99
Using Roles to Consolidate Sets of Privileges	100
Modifying Sample Roles to Create New Roles	101
Granting Privileges Through Permissions	102
Authenticating Users Through SessionManager	104
Using VMware Single Sign On for vCenter Server Sessions	105
Using the Credential Store for Automated Login	105
Credential Store Libraries	106
Credential Store Methods	106
Credential Store Backing File	107
Credential Store Samples	107
Specifying Roles and Users with the Credential Store	108
Managing Licenses with LicenseManager	109

7 Hosts 110

Host Management Objects	110
Retrieving Host Information	111
Configuring and Reconfiguring Hosts	112
Managing the Host Lifecycle	112
Reboot and Shutdown	112
Using Standby Mode	112
Disconnecting and Reconnecting Hosts	113
Querying and Changing the Host Time	113
Querying Virtual Machine Memory Overhead	114

8 Storage 115

- Storage Management Objects 115
- Introduction to Storage 116
 - How Virtual Machines Access Storage 117
 - Datastores 118
- Choosing the Storage API to Use 119
 - Managed Objects for Working with Storage 120
- Configuring Disk Partitions 120
- Multipath Management 121
- Configuring iSCSI Storage 121
 - Configure the VMkernel To Support Software iSCSI 122
 - Configure iSCSI Initiators 123
- Creating and Managing Datastores 124
 - Accessing Datastores 125
 - Creating and Modifying a VMFS Datastore 125
 - Removing and Updating Datastores 127
 - Managing VMFS Datastores with HostStorageSystem 127
- Managing VMFS Volume Copies (Resignaturing) 128
 - Resignaturing Volumes with ResignatureUnresolvedVmfsVolume_Task 129
- Managing Diagnostic Partitions 129
 - Retrieving Diagnostic Partition Information 130
 - Create a Diagnostic Partition 130
- Sample Code Reference 131

9 vSphere Networks 132

- Virtual Switches 132
 - Port Groups 132
 - Virtual Machine Network Interfaces 133
 - VMkernel Network Interfaces 133
 - Physical Network Adapter (pnict) 134
- Using a Distributed Virtual Switch 134
 - Distributed Virtual Switch Configuration 135
 - Backup, Rollback, and Query Operations 136
- VMware Standard Virtual Switch 137
 - Configuring a Standard Virtual Switch 137
 - vNetwork Standard Switch Environment 138
 - Setting Up Networking with vSS 139
 - Defining the Host Network Policies 141
 - NIC Teaming 141
 - Setting Up IPv6 Networking 142
 - Adding Networking Services 143
- Sample Code Reference 144

10 Virtual Machine Configuration 146

- VirtualMachine Management Objects and Methods 146
- Creating Virtual Machines and Virtual Machine Templates 147
 - Creating a Virtual Machine Using VirtualMachineConfigSpec 147
 - Creating Virtual Machine Templates 149
 - Cloning a Virtual Machine 149
 - Converting a Template to a Virtual Machine 150
 - Accessing Information About a Virtual Machine 150
- Configuring a Virtual Machine 151
 - Name and Location 152
 - Hardware Version 153
 - Boot Options 153
 - Operating System 154
 - CPU and Memory Information 154
 - Networks 155
 - Fibre Channel NPIV Settings 156
 - File Locations 156
- Adding Devices to Virtual Machines 157
- Performing Virtual Machine Power Operations 158
- Registering and Unregistering Virtual Machines 160
- Customizing the Guest Operating System 160
- Installing VMware Tools 161
- Upgrading a Virtual Machine 162

11 Virtual Machine Management 163

- Virtual Machine Migration 163
 - Cold Migration 164
 - Migration with VMotion 164
 - Using Storage VMotion 164
- Snapshots 164
 - Creating a Snapshot 165
 - Reverting to a Snapshot 166
 - Deleting a Snapshot 166
- Linked Virtual Machines 166
 - Linked Virtual Machines and Disk Backings 166
 - Creating a Linked Virtual Machine 167
 - Instant Clone Virtual Machines 169
 - Removing Snapshots and Deleting Linked Virtual Machines 175
 - Relocating a Virtual Machine in a Linked Virtual Machine Group 175
 - Promoting a Virtual Machine's Disk 176
 - Performing Advanced Manipulation of Delta Disks 177

12 Virtual Machine Encryption APIs 178

- How Virtual Machine Encryption Protects a Datacenter 178
 - What Keys are Used 178
 - What Is Encrypted 179
 - What Is Not Encrypted 179
 - Who Can Perform Cryptographic Operations 180
 - How Can I Perform Cryptographic Operations 180
- vSphere Virtual Machine Encryption Components 180
 - Key Management Server 181
 - vCenter Server 181
 - ESXi Hosts 182
 - Encryption Process Flow 182
- Prerequisites and Required Privileges for Encryption Tasks 183
 - Cryptography Privileges and Roles 183
 - Host Encryption Mode 183
 - Encrypted vSphere vMotion 184
- API Methods for vSphere Virtual Machine Encryption 184
 - vSphere API Methods for KMS Management 184
 - API Methods to Prepare an ESXi Host 185
 - vSphere API Methods for Cryptographic Operations 186
 - SPBM API Methods for Encryption 187
- Workflows for vSphere Virtual Machine Encryption 187
 - Set Up the Key Management Server Cluster 187
 - Create an Encryption Storage Policy 190
 - Create an Encrypted Virtual Machine 193
 - Clone an Encrypted Virtual Machine 194
 - Encrypt an Existing Virtual Machine or Disk 195
 - Decrypt an Encrypted Virtual Machine or Disk 195
 - Encrypt Using Different Keys 196
 - Recrypting Encrypted Virtual Machines 197
 - Encrypted vSphere vMotion 198
 - Virtual Disk Manager 198
- Best Practices 199

13 Virtual Applications 201

- About Virtual Applications 201
 - Management Overview 201
 - Direct and Linked Children 202
 - OVF Packages 203
- Creating a VirtualApp 203
- Managing VirtualApp Children 204

Exporting a Virtual Application	204
VirtualApp and OvfManager Methods	205
VirtualApp Data Structures	206
OvfManager Data Structures	207
Example of Generating an OVF Package	208
Importing an OVF Package	209
Virtual Application Life Cycle	210
Powering a Virtual Application On or Off	210
Unregistering a Virtual Application	211
Suspending a Virtual Application	211
Destroying a Virtual Application	211

14 Resource Management 212

Resource Management Objects	212
Introduction to Resource Management	213
Resource Allocation	213
Resource Pool Hierarchies	214
Resource Pool Management Guidelines	214
Cluster Overview	215
Creating and Configuring Resource Pools	215
Understanding Expandable Reservation	216
Deleting Child Resource Pools	217
Moving Resource Pools or Virtual Machines Into a Resource Pool	217
Introduction to VMware DRS and VMware HA Clusters	218
VMware DRS	218
VMware HA	219
Creating and Configuring Clusters	219
Creating a Cluster	219
Adding a Host to a Cluster	220
Reconfiguring a Cluster	220
Managing DRS Clusters	221
Managing HA Clusters	221
Primary and Secondary Hosts	221
Failure Detection and Host Network Isolation	222
Using VMware HA and DRS Together	222

15 Tasks and Scheduled Tasks 224

Creating Tasks	224
Session Persistence	225
Cancelling a Task	225
Using TaskInfo to Determine Task Status	225
Monitoring TaskInfo Properties	226

Accessing and Manipulating Multiple Tasks	227
Gathering Data with a ViewManager Object	228
Gathering Data with a TaskManager Interface	238
Understanding the ScheduledTaskManager Interface	240
Scheduling Tasks	241
Cancelling a Scheduled Task	244
Using a TaskHistoryCollector	245
Managing the HistoryCollector	245
Sample Code Reference	246

16 Events and Alarms 248

Event and Alarm Management Objects	248
Understanding Events	249
Managing Events with EventManager	249
Event Data Objects	250
Formatting Event Message Content	251
Creating Custom Events	251
Using an EventHistoryCollector	253
Creating an EventHistoryCollector Filter	254
Managing the HistoryCollector	254
Using Alarms	254
Obtaining a List of Alarms	255
Creating an Alarm	255
Defining Alarms Using the AlarmSpec Data Object	256
Specifying Alarm Trigger Conditions with AlarmExpression	257
Specifying Alarm Actions	258
Deleting or Disabling an Alarm	259
Sample Code Reference	259

17 vSphere Performance 261

vSphere Performance Data Collection	261
PerformanceManager Objects and Methods	263
Retrieving vSphere Performance Data	265
Performance Counter Example (QueryPerf)	266
Large-Scale Performance Data Retrieval	274
Using the QueryPerf Method as a Raw Data Feed	275
Comparison of Query Methods	275
Retrieving Summary Performance Data	276
Performance Counter Metadata	276
PerfCounterInfo	276
Performance Intervals	277
ESXi Server Performance Intervals	278

	vCenter Server Performance Intervals	278
	vSphere Performance and Data Storage	279
	Modifying Historical Intervals	279
	Modifying Performance Counter Collection Levels	280
	Sample Code Reference	281
18	Diagnostics and Troubleshooting	283
	Troubleshooting Best Practices	283
	Overview of Configuration Files and Log Files	284
	ESXi Log File	285
	Virtual Machine Log Files	285
	vCenter Server Log Files	286
	Modifying the Log Level to Obtain Detailed Information	286
	Setting the Log Level on ESXi Systems	287
	Generating Logs	287
	Setting the Log Level on vCenter Server Systems	288
	Using DiagnosticManager	288
	Using the MOB to Explore the DiagnosticManager	290
	Generating Diagnostic Bundles	291
	Export Diagnostic Data By Using the vSphere Client	291
19	Managed Object Browser	292
	Using the MOB to Explore the Object Model	292
	Accessing the MOB	292
	Using the MOB to Navigate the VMware Infrastructure Object Model	293
	Using the MOB to Invoke Methods	294
	Passing Primitive Datatypes to Method	294
	Passing Arrays of Primitives to Methods	294
	Passing Complex Structures to Methods	295
	Using the MOB Along With the API Reference	298
20	HTTP Access to vSphere Server Files	300
	Introduction to HTTP Access	300
	URL Syntax for HTTP Access	301
	Datastore Access (/folder)	301
	Host File Access (/host)	302
	Update Package Access (/tmp)	303
	Privilege Requirements for HTTP Access	303
21	Sample Program Overview	304
	Java Sample Programs (JAXWS Bindings)	304
	C# Sample Programs	308

About This Book

The *vSphere Web Services SDK Programming Guide* provides information about developing applications using the VMware[®] vSphere Web Services SDK 6.7.

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

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

Revision History

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

Table 1. Revision History

Revision Date	Description
17APR2018	vSphere 6.7 - GA Added Instant Clone material to VM Management chapter. Minor updates elsewhere.
12APR2017	Removed appendix with list of permissions. Was not maintained.
15NOV2016	vSphere 6.5 - GA Added chapter about vSphere virtual machine encryption. Removed description of deprecated CIM Storage Management API. Updated information about session cookie management.
04SEP2015	Updated information about migrating VMs with VMotion across data centers.
12MAR2015	vSphere 6.0 - Rewrote "Exporting a Virtual Application" section in Virtual Applications chapter.
19SEP2013	vSphere 5.5 – Added a C# example of using LoginByToken; clarified limitation for HA clusters.
10SEP2012	vSphere 5.1 – Added information about using the SessionManager.LoginByToken method; added information about distributed virtual switches.

Table 1. Revision History (Continued)

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

Intended Audience

This book is intended for anyone who needs to develop applications using the vSphere Web Services SDK. Developers typically create client applications using Java or C# (in the Microsoft .NET environment) targeting VMware vSphere. An understanding of Web Services technology and some programming background in one of the stub languages (C# or Java) is required.

VMware Technical Publications Glossary

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

Document Feedback

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

vSphere Web Services API Programming Model

2

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

This chapter includes the following topics:

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

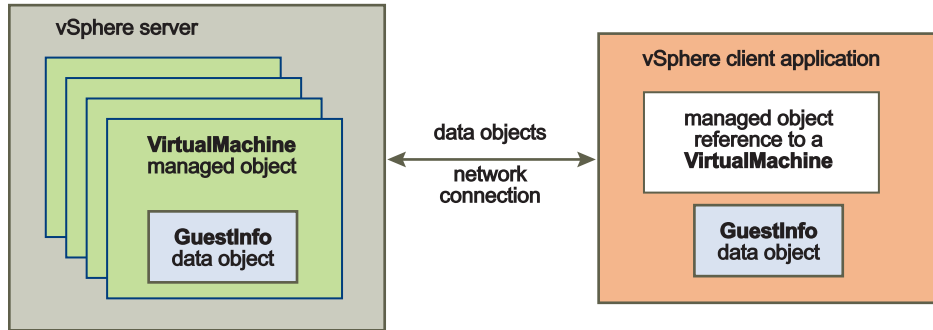
vSphere Client-Server Architecture

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

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

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

Figure 2-1. vSphere Server and Client



The VMware vSphere application model uses an asynchronous client-server communication model in most cases. Methods are nonblocking and return a reference to a Task managed object. For more information about the Task managed object, see [Tasks and Scheduled Tasks](#).

Web Services API as a Web Service

The vSphere Web Services API is a language-neutral Web service that runs on ESXi and vCenter Server systems.

The API complies with the Web Services Interoperability Organization (WS-I) Basic Profile 1.0. The WS-I Basic Profile 1.0 includes support for the following standards:

- XML Schema 1.0
- SOAP 1.1
- WSDL 1.1

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

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

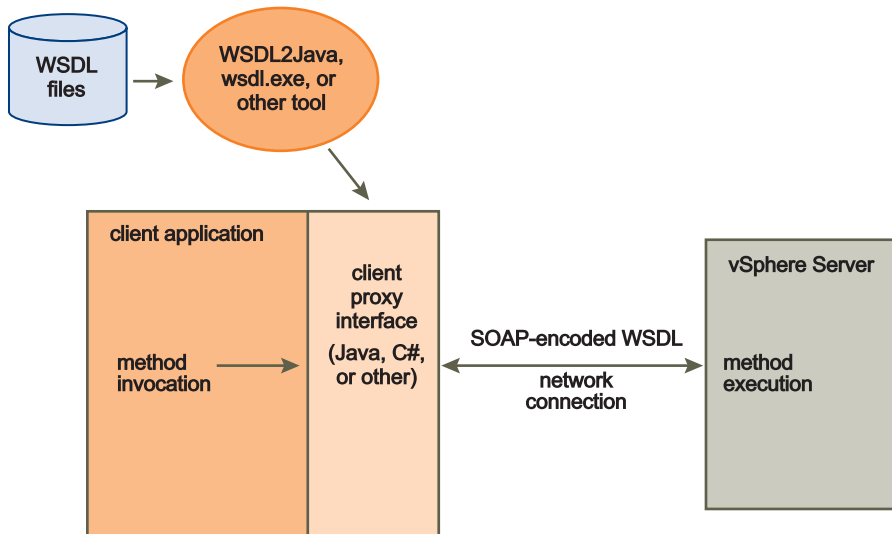
WSDL Files and the Client-Side Proxy Interface

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

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

Figure 2-2 is a representation of a client application that uses the client proxy interface to call a method. The client proxy interface is based on the WSDL definitions.

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



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

C#	<code>using VimApi;</code>
Java	<code>import com.vmware.vim25.*;</code>

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

Network Access to the vSphere Web Service

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

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

Language-Specific Classes and Methods

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

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

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

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

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

Java

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

C#

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

You can extrapolate the getter and setter methods that are available in the client proxy interface from the *vSphere API Reference*. For example, the `ScsiLun` data object has a `displayName` property. For the Java API, you can use a `setDisplayDisplayName` method to assign a string value to the property, and obtain the string value by using the `getDisplayDisplayName` method. The vSphere Web Services SDK includes Java and C# sample code that illustrates how to use the proxy interfaces. See [#unique_18](#).

Mapping XML Data Types to Java and C# Data Types

In this guide, the UML class and object diagrams use the primitive data type names such as `string` and `integer`, without the XML Schema definition namespace prefix (`xsd:`). The *vSphere API Reference* contains the complete data type name, such as `xsd:string`. The data types map to the primitive data types of the programming language used for the client application.

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

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

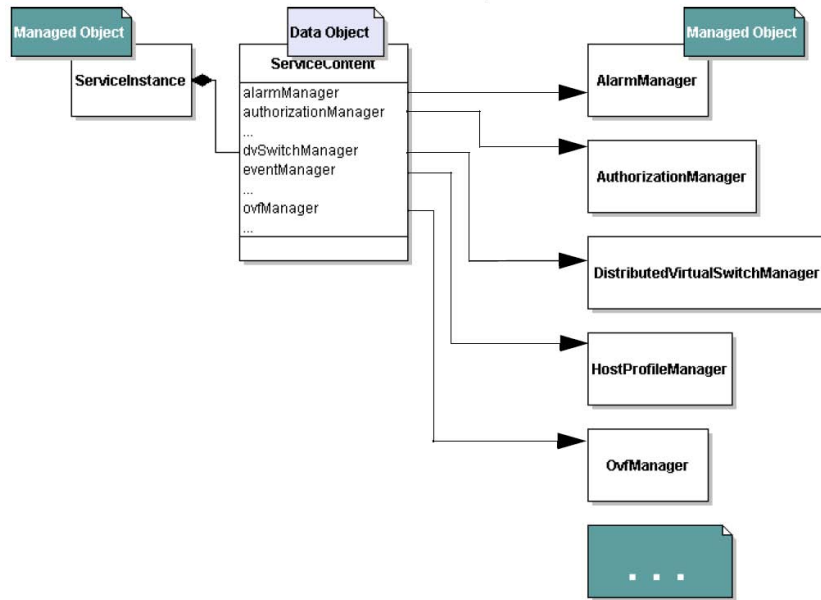
Access to Managed Objects

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

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

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

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

Figure 2-3. ManagedObjectReference Data Object

Access to vSphere Server Data

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

Obtaining Information from a Server

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

- Use an accessor (getter) method. The client proxy interface provides accessor methods for each data object property. You can use these accessor methods to obtain the values of the object. See [Language-Specific Classes and Methods](#) for information about client proxy interface accessor methods.
- Use a `PropertyCollector` to navigate to a selected point on the server and obtain values from specific properties. See [#unique_28](#) for more information about `PropertyCollector`.

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

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

Working with Data Structures

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

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

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

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

Accessing Property Values

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

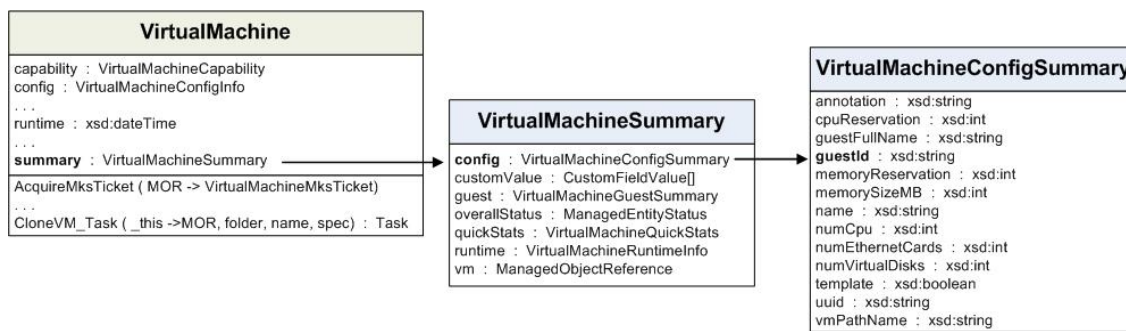
- Use dot notation to access nested properties in composite data structures.
- Cast unconstrained property values (`xsd:anyType`) to array types.
- Use keys or index values as appropriate to access array values.

Nested Properties and Property Paths in Composite Data Structures

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

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

Figure 2-4. VirtualMachine Managed Object and Nested Properties



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

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

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

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

xsd:anyType Arrays

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

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

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

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

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

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

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

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

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

Procedure

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

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

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

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

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

Procedure

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

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

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

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

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

Procedure

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

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

Indexed Array and Key-Based Array Properties

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

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

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

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

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

Unset Optional Properties

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

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

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

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

Figure 2-5. Data Object - HostFirewallInfo Properties

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

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

Escape Character in Name and Path Properties

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

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

Datacenter Inventory

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

This chapter includes the following topics:

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

Inventory Overview for the Web Services SDK

The vSphere inventory contains the following types of objects:

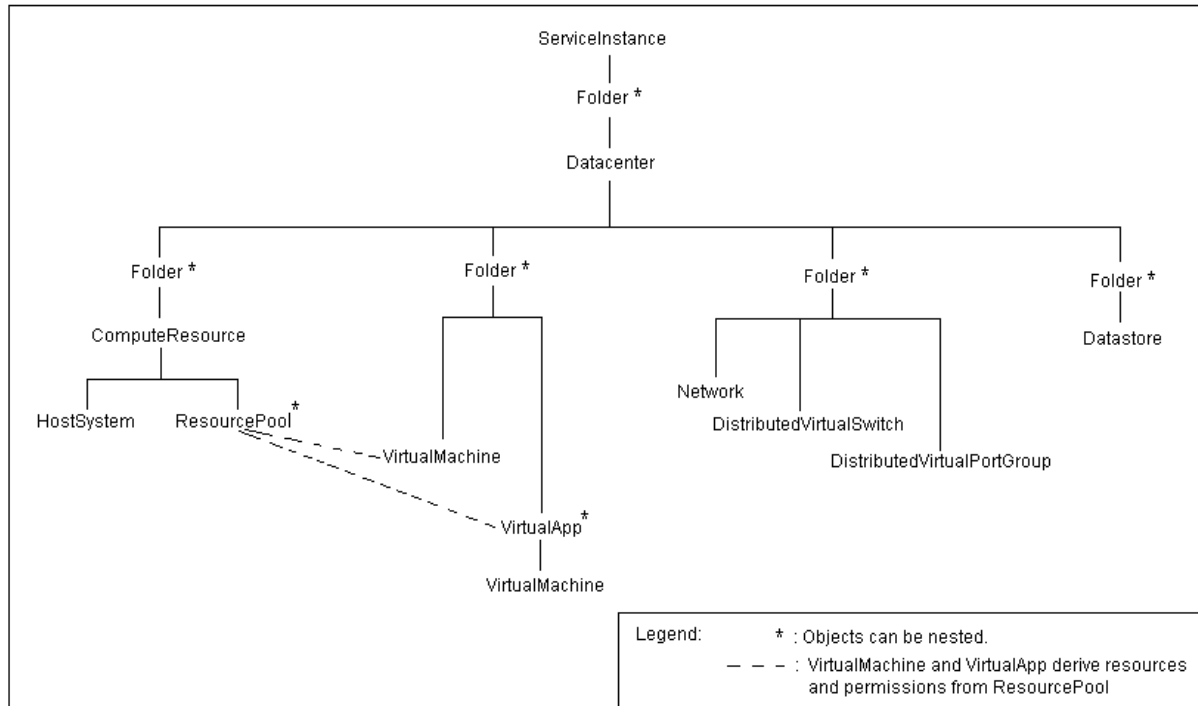
- Systems in the datacenter: Host, VirtualMachine, and VirtualApp.
- Support components: ComputeResource, Datastore, Network, and virtual devices.
- Organizational components: Folder and Datacenter

When you manage the virtual infrastructure, you access objects and their properties and methods based on their location in the inventory. Understanding the inventory structure is therefore critical for any programming task. You always start with the ServiceInstance associated with a session, which is the root object of the inventory, and traverse the inventory hierarchy from there. See [Chapter 5 Property Collector](#). How you access objects depends on whether your client application is connected to a vCenter Server or an ESXi host.

Inventory Hierarchies and ServiceInstance

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

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

Figure 4-1. vCenter Server Inventory Hierarchy

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

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

Folders in the Hierarchy

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

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

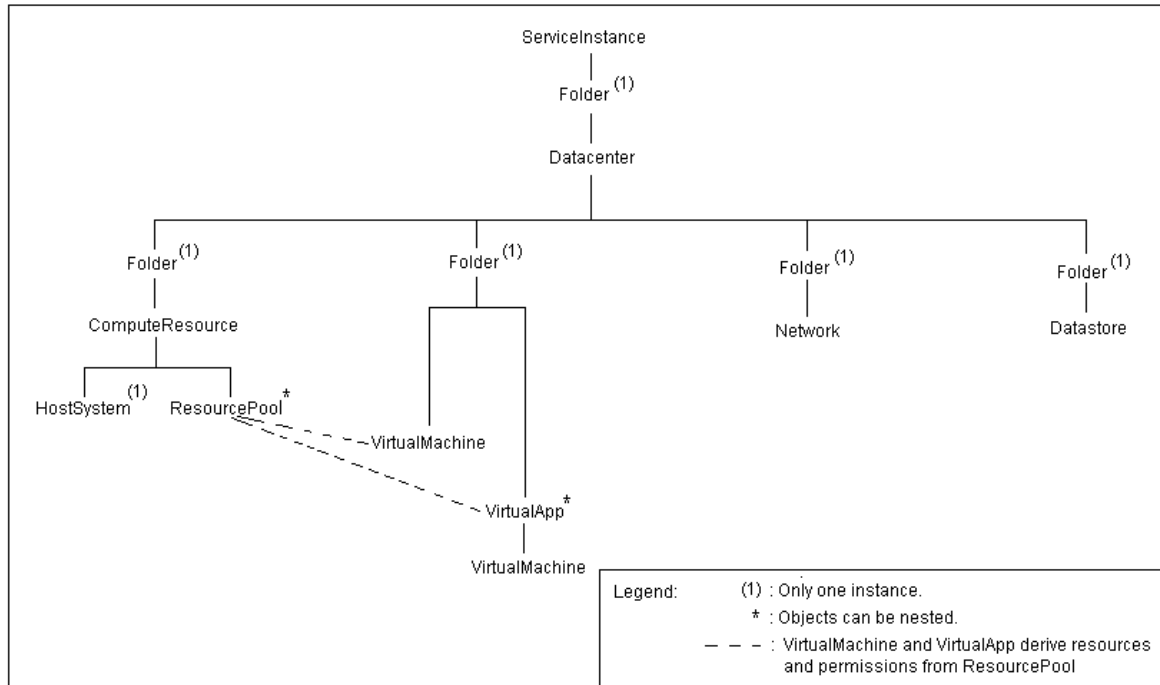
In a large deployment, the nested structure allows you to organize the objects in the datacenter into an easily manageable structure by using multiple folders and datacenters.

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

ESXi Inventory Hierarchy

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

Figure 4-2. ESXi Inventory Hierarchy

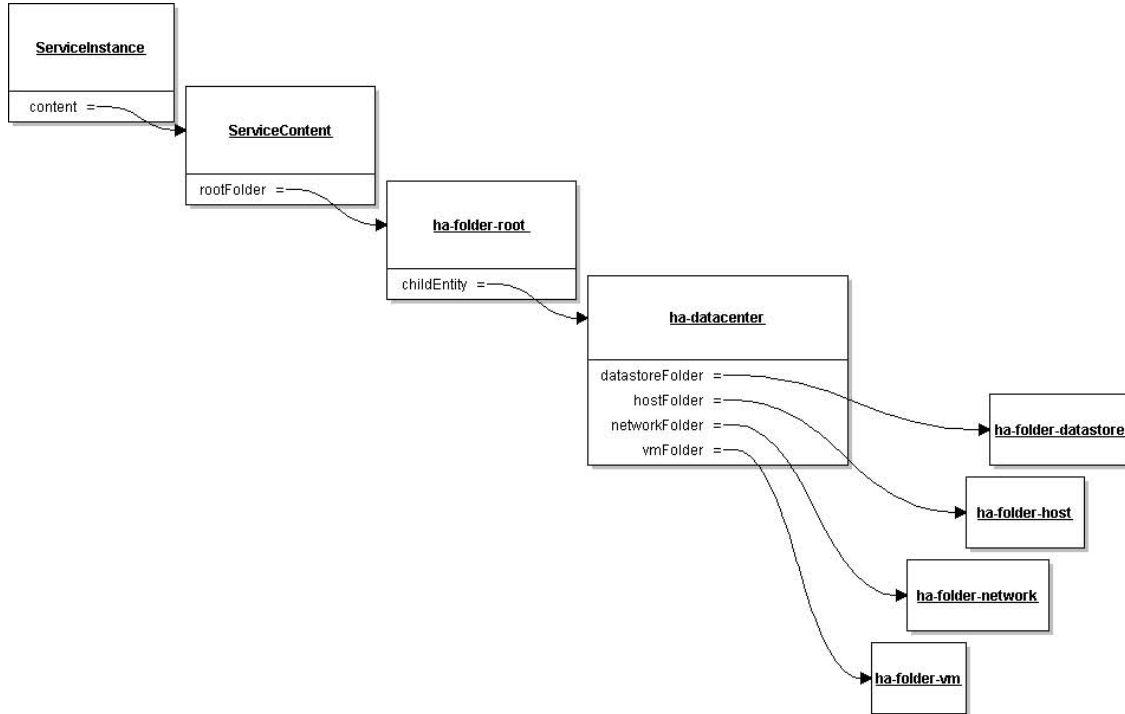


Accessing Inventory Objects

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

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

Figure 4-3 shows the `childEntity` and `folder` properties that define the default objects in the inventory of a standalone ESXi system. The inventory begins with the `ServiceContent.rootFolder` property. The `rootFolder` has a `childEntity` that consists of a managed object reference to a `Datacenter` managed object.

Figure 4-3. Instance Diagram of Root Folders in an Inventory

Creating Inventory Objects

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

- Datacenter
- DistributedVirtualSwitch
- VirtualMachine
- Cluster
- Folder

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

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

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

Privileges Required for Inventory Management

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

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

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

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

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

See [Chapter 6 Authentication and Authorization](#) for more information on authentication, authorization, roles, and user identity.

Privileges

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

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

For example:

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

Permissions

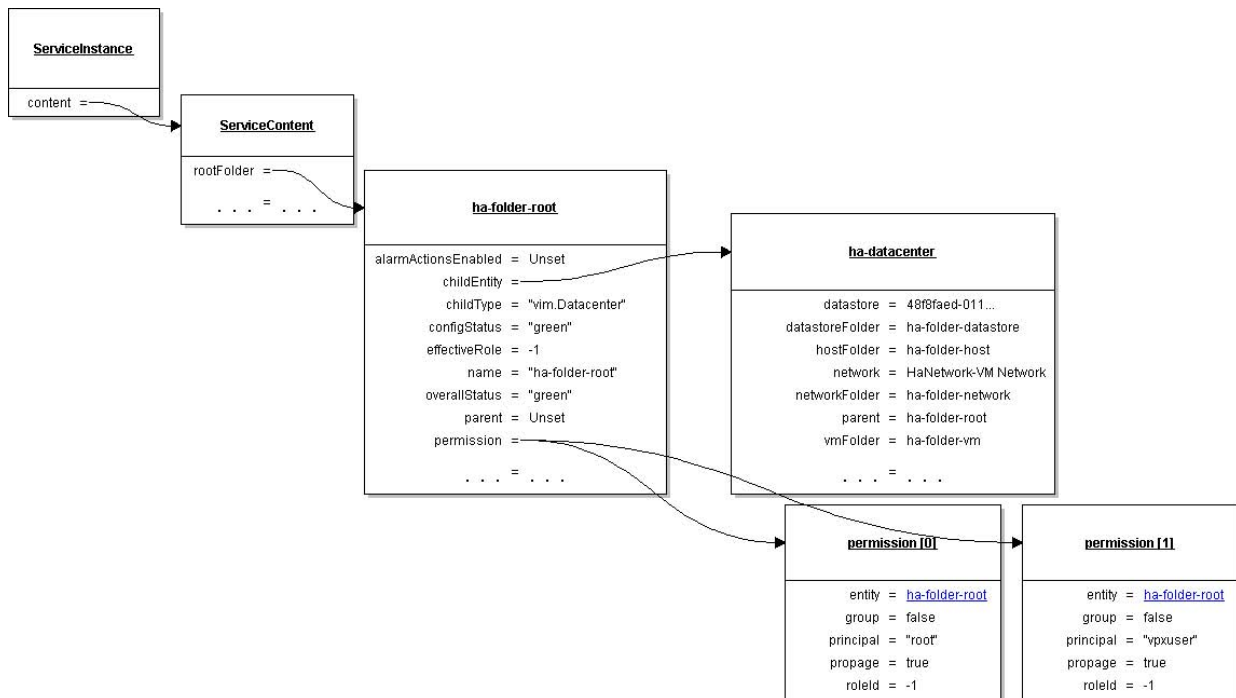
Permissions are the associations of roles with privileges on a specified managed entity. You use permissions to specify which users can access which managed entity.

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

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

Important See [Chapter 6 Authentication and Authorization](#) for a detailed discussion of privileges, permissions, and user management.

Figure 4-4. Inventory and Permissions



Managed and Standalone ESXi Hosts

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

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

The following table summarizes the differences between the number of objects that an inventory can contain. See also [Figure 4-2](#) and [Figure 4-1](#).

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

Property Collector

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

This chapter includes the following topics:

- [Introduction to the PropertyCollector](#)
- [vSphere Data Objects for Property Collection](#)
- [vSphere Methods for Property Collection](#)
- [Retrieve Properties with the PropertyCollector](#)
- [Inventory Traversal](#)
- [Client Data Synchronization \(WaitForUpdatesEx\)](#)
- [Server Data Transmission](#)
- [PropertyCollector Performance](#)
- [SearchIndex](#)

Introduction to the PropertyCollector

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

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

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

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

Data Retrieval

There are two ways to retrieve data:

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

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

Inventory Traversal and Object Selection

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

You can use vSphere view objects (for example, `ContainerView`) in filters to simplify traversal specification. A view maintains a subset of inventory objects, so if there is a change in the inventory hierarchy, you do not have to recreate the view. Use a view to specify a set of objects that the `PropertyCollector` can use for data collection.

For information about the vSphere inventory, see [Inventory Traversal and Object Selection](#).

vSphere Data Objects for Property Collection

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

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

vSphere Methods for Property Collection

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

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

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

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

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

Retrieve Properties with the PropertyCollector

The following procedure shows how to retrieve properties of managed objects by using the `RetrievePropertiesEx` method. The procedure steps are illustrated by code fragments.

This procedure shows only the task of using the `PropertyCollector`. For a description of server connection, see [Build a Simple vSphere Client Application for the Web Services SDK](#). You can also see these code fragments in the context of an end-to-end example that includes connection and authentication logic, in [Build a Simple vSphere Client Application for the Web Services SDK](#).

To do a single retrieval operation with the `PropertyCollector`, use the following steps.

Procedure

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

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

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

- 2 Create a container view for virtual machines.

`methods` is the variable for the `VimPortType` object. `VimPortType` defines the Java methods that correspond to the vSphere API methods. The `createContainerView` parameters `container` (the inventory root folder, returned by the method `sContent.getRootFolder()`) and `type` ("Virtual Machine") direct the `ViewManager` to select virtual machines, starting at the root folder. The value `true` for the `recursive` parameter extends the selection beyond the root folder so that the `ViewManager` will follow child folder paths to add virtual machines to the view. The container view provides references to all virtual machines in the inventory.

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

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

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

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

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

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

The `TraversalSpec` properties `type` and `path` determine path traversal. `TraversalSpec.type` identifies an object type. `TraversalSpec.path` identifies a property in the type object. The `PropertyCollector` uses the path object to select additional objects.

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

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

5 Add the `TraversalSpec` to the `ObjectSpec.selectSet` array.

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

6 Identify the properties to be retrieved.

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

This example specifies the `VirtualMachine.name` property.

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

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

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

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

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

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

9 Retrieve the data.

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

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

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

10 Print the virtual machine names.

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

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

Simple Property Collector Program in Java for Web Services SDK

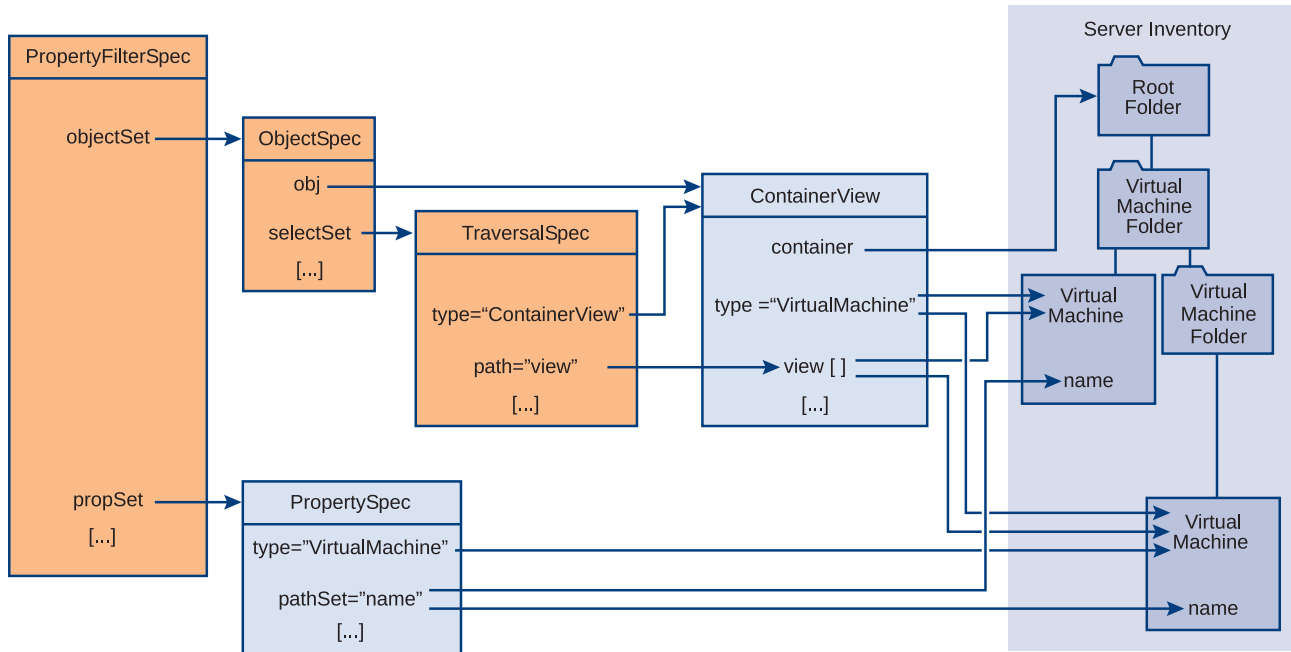
The Property Collector provides a powerful, flexible, and efficient way to collect data from vCenter Server or ESXi.

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

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

Figure 5-1 shows the objects used in [Example: Simple PropertyCollector Example \(Java\)](#). The figure represents properties that identify inventory elements directly or indirectly. It does not show all the properties for the different objects.

Figure 5-1. Property Filter Specification



Example: Simple PropertyCollector Example (Java)

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

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

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

public class PCollector {

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

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

```

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

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

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

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

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

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

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

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

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

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

```

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

//end collectProperties()

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

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

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

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

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

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

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

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

```

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

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

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

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

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

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

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

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

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

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

```

```

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

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

// retrieve data
collectProperties( vimPort, serviceContent );

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

}
}

```

Inventory Traversal

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

[Example: Simple PropertyCollector Example \(Java\)](#) uses a `ContainerView` to specify the objects that start the collection process. This is the simplest way to set up a filter, using a single reference to a view to provide the `PropertyCollector` with access to a set of objects. To select objects from the inventory, a filter includes `TraversalSpec` and possibly `SelectionSpec` objects. Use these objects to make object selections based on the references in a view, and to extend inventory traversal beyond those objects (or beyond the object specified in `ObjectSpec.obj`).

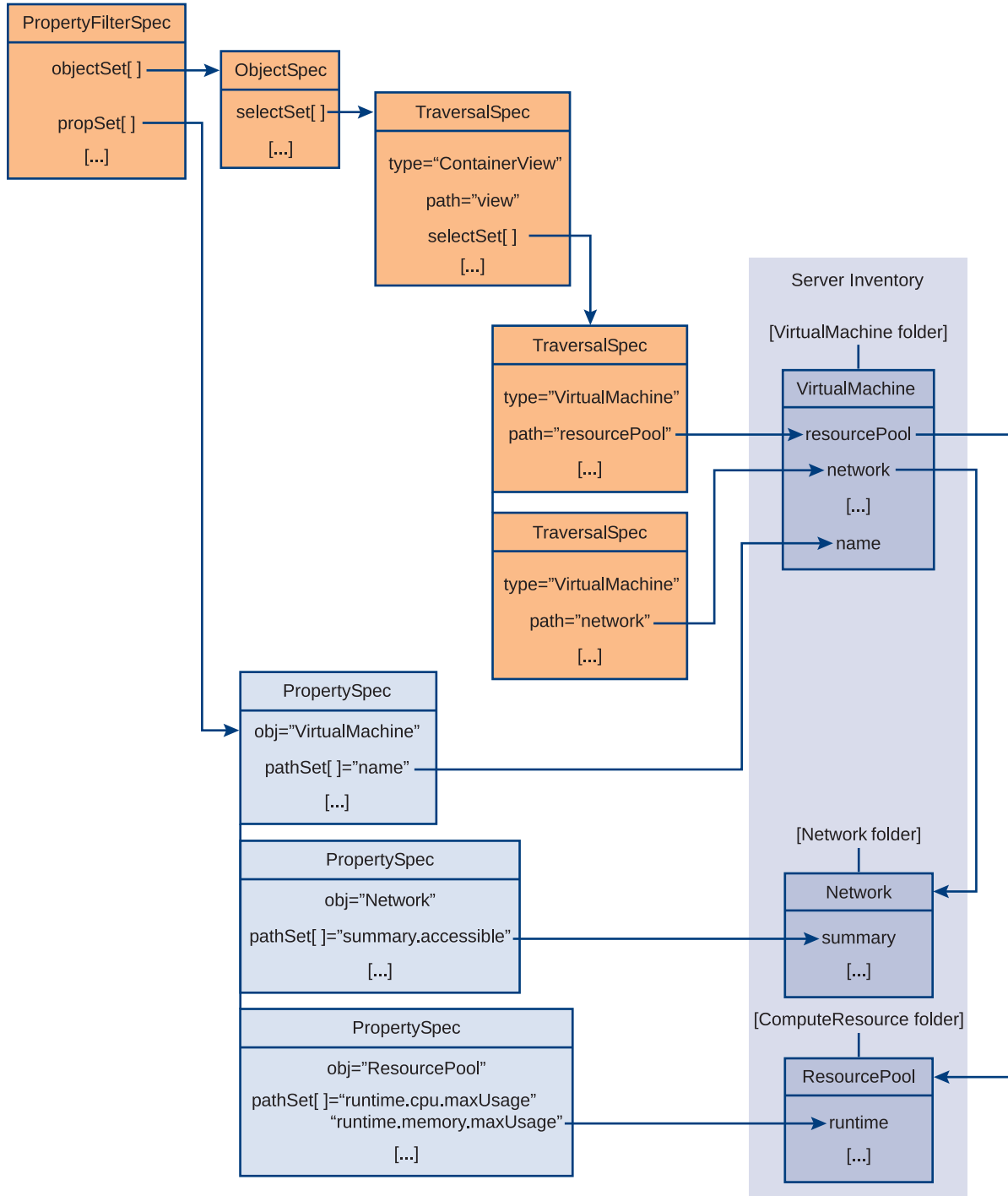
TraversalSpec Traversal

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

- `type` – identifies an inventory object type.
- `path` – specifies a managed object reference property in the type object. This property provides the traversal path extending from this object.
- `selectSet` – specifies an optional list of selection objects for additional object traversal paths. The `PropertyCollector` applies the `TraversalSpec` objects in the `selectSet` array to the result of the traversal (the target of `TraversalSpec.path`). The `selectSet` array can also contain `SelectionSpec` objects; a `SelectionSpec` is a reference to a `TraversalSpec`. See [SelectionSpec Traversal](#).
- `skip` – indicates whether to collect properties for the path object.

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

Figure 5-2 is a representation of a `PropertyFilterSpec` that defines traversal of `VirtualMachine` objects. The filter uses a `ContainerView` as a starting point. The `TraversalSpec` for the `ContainerView` specifies the `view` property for access to the view's virtual machines. The figure shows `TraversalSpec` objects that extend navigation from a `VirtualMachine` object to the associated `Network` and `ResourcePool` objects. The `PropertyCollector` applies these `TraversalSpec` objects to each of the `VirtualMachine` objects in the view list. The figure also shows the `PropertySpec` objects for collecting data from `VirtualMachine`, `Network`, and `ResourcePool` objects.

Figure 5-2. Inventory Navigation

Traverse the Inventory By Using the Property Collector

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

To define inventory traversal, use the following steps.

Procedure

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

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

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

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

Inventory Traversal Example in Java

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

Example: Inventory Traversal in Java

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

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

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

public class PCollector_traversal {

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

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

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

```

// (use a List<String> for the type parameter's string[])
List<String> vmList = new ArrayList<String>();
vmList.add("VirtualMachine");

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

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

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

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

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

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

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

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

```

```

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

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

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

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

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

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

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

```

```

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

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

//end collectProperties()

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

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

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

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

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

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

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

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

    // Variables of the following types for access to the API methods

```

```

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

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

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

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

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

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

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

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

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

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

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

```

```

// requests to maintain the session. If you do not set this to true,
// the Server will start a new session with each request.
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, url);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
// Retrieve the ServiceContent object and login
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);
vimPort.login(serviceContent.getSessionManager(),
              userName,
              password,
              null);

// retrieve data
collectProperties( vimPort, serviceContent );

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

}
}

```

SelectionSpec Traversal

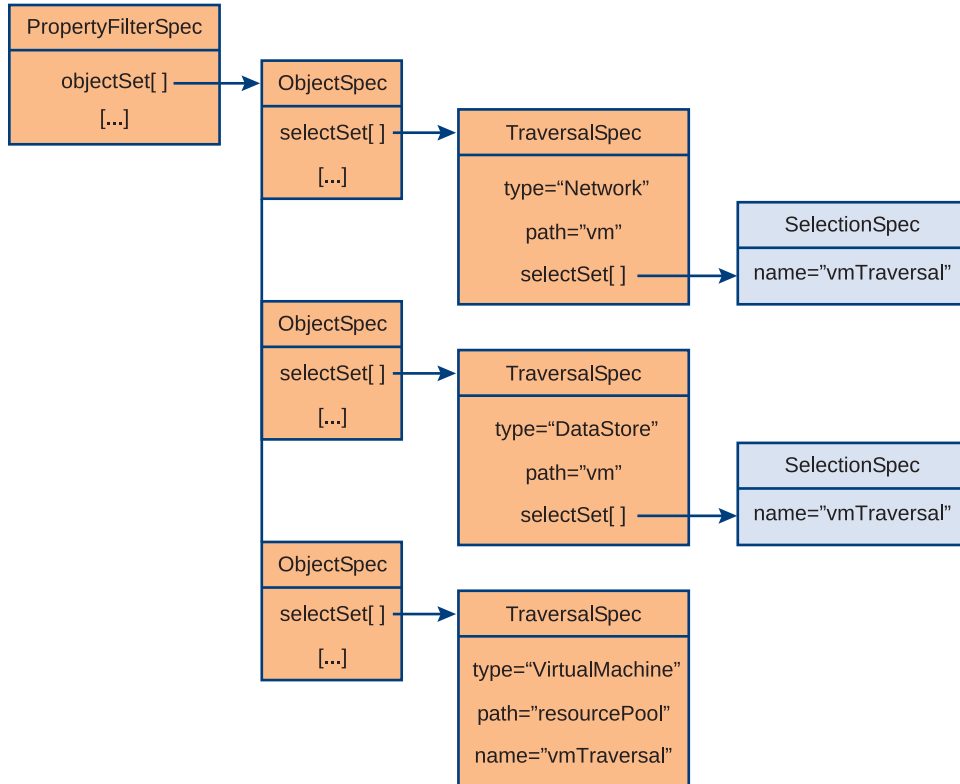
The selectSet array in ObjectSpec and TraversalSpec objects can include TraversalSpec objects and SelectionSpec objects. SelectionSpec is the base class for TraversalSpec objects.

SelectionSpec defines the name property. You can use a SelectionSpec object in a selectSet array as a reference to a named TraversalSpec object. By using SelectionSpec references, you can reuse a TraversalSpec and you can define recursive traversal.

Simple Reference SelectionSpec

Use SelectionSpec references to avoid writing duplicate TraversalSpec declarations.

The TraversalSpec identified in a SelectionSpec reference must be within the same PropertyFilterSpec. [Figure 5-3](#) shows the use of SelectionSpec references to a virtual machine TraversalSpec. The SelectionSpec references are associated with Network and Datastore traversals.

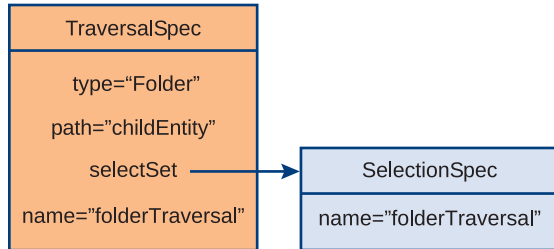
Figure 5-3. SelectionSpec Reference

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

Recursive Traversal

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

You can use recursive traversal on any inventory objects that can be nested. See [Inventory Hierarchies and ServiceInstance](#) for a general representation of the structure of an inventory. For example, on a vCenter Server, folders can nest to arbitrary depths. To describe a traversal path through a succession of folders, you can add a `SelectionSpec` to the `Folder TraversalSpec`. The `SelectionSpec` must reference the `TraversalSpec`. [Figure 5-4](#) shows a representation of a `TraversalSpec` and its associated `SelectionSpec` for nested folder traversal.

Figure 5-4. Recursive TraversalSpec and SelectionSpec

Use Recursive TraversalSpec to Traverse Nested Inventory Folders

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

To define recursive inventory traversal, use the following steps.

Procedure

- 1 Use the `SearchIndex` managed object to retrieve the managed object reference for the top-level virtual machine folder.
This folder is used as the beginning of the inventory traversal. For more information see [SearchIndex](#).
- 2 Create an `ObjectSpec` object that references the top-level virtual machine folder.
- 3 Create a `SelectionSpec` object that references the `Folder` `TraversalSpec` by name.
- 4 Create a named `TraversalSpec` for `Folder` objects.
The `TraversalSpec.path` property identifies the `Folder.childEntity` property for traversal to any child objects.
- 5 Add the `SelectionSpec` to the `TraversalSpec` to create the recursive filter.
- 6 Add the `TraversalSpec` to the `ObjectSpec`.
- 7 Create a `PropertySpec` for the `Folder` name.
- 8 Add the object and property specifications to the `PropertyFilterSpec`.
- 9 Call the `RetrievePropertiesEx` method.

Nested Folder Traversal in Java

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

Example: Nested Folder Traversal

```

import com.vmware.vim25.*;

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

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

public class nestedTraversal {

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

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

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

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

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

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

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

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

```

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

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

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

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

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

} //end collectProperties()

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

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

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

```

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

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

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

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

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

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

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

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

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

    // Initialize the contexts; the session context takes the trust manager.
    sslsc.setSessionTimeout(0);

```

```

sc.init(null, trustAllCerts, null);

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

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

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

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

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

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

// retrieve data
collectProperties( vimPort, serviceContent );

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

}
}

```

Client Data Synchronization (WaitForUpdatesEx)

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

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

Property Filters

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

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

WaitForUpdatesEx

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

Specify the following parameters when you call `WaitForUpdatesEx`:

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

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

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

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

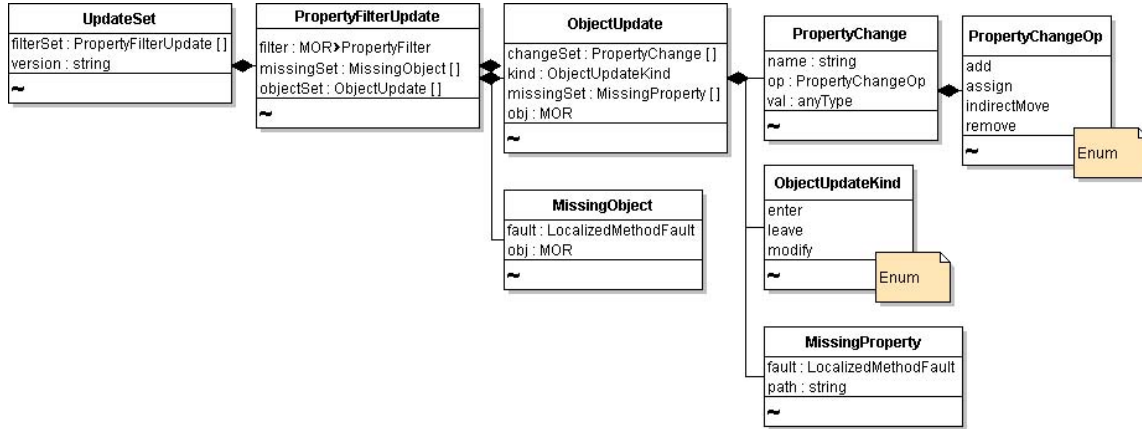
Comparing Usage of `MaxWaitSeconds`

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

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

UpdateSet Data Object from the Property Collector

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

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

Server Data Transmission

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

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

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

PropertyCollector Performance

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

- Number of objects
- Number of properties

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

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

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

SearchIndex

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

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

Storage

A virtual machine uses a virtual disk to store its operating system, program files, and other data. A virtual disk is a large physical file, or a set of files, that can be copied, moved, archived, and backed up like other files. To store and manipulate virtual disk files, a host requires dedicated storage space. ESX/ESXi supports storage in multiple ways. Hosts that are managed by a vCenter Server system can share storage.

Any type of network-attached storage requires complete configuration of networking in the VMkernel to support network-based access to the storage media. The VMkernel requires its own IP address. See [#unique_148](#).

This chapter includes the following topics:

- [Storage Management Objects](#)
- [Introduction to Storage](#)
- [Choosing the Storage API to Use](#)
- [Configuring Disk Partitions](#)
- [Multipath Management](#)
- [Configuring iSCSI Storage](#)
- [Creating and Managing Datastores](#)
- [Managing VMFS Volume Copies \(Resignaturing\)](#)
- [Managing Diagnostic Partitions](#)
- [Sample Code Reference](#)

Storage Management Objects

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

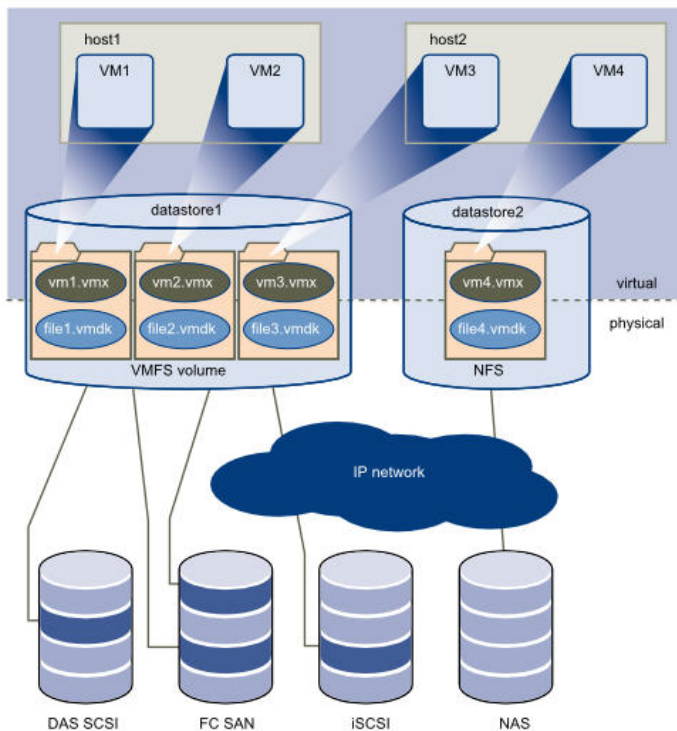
- **HostStorageSystem** – The `HostSystem.storageSystem` property is a managed object reference to the HostStorageSystem of the ESX/ESXi system. HostStorageSystem is a low-level interface that is used mainly for configuring the physical storage. See [Configuring Disk Partitions](#).

- **HostDatastoreSystem** – The `HostSystem.datastoreSystem` property is a managed object reference to a `HostDatastoreSystem` managed object. `HostDatastoreSystem` methods allow you to create, configure, extend, and remove datastores. While `HostStorageSystem` supports access and configuration of physical storage, `HostDatastoreSystem` supports access and configuration of logical storage through the volumes (Datastore managed objects) the host can use for virtual machines. See [Creating and Managing Datastores](#).
- **HostDatastoreBrowser** – Provides access to the contents of one or more datastores. The items in a datastore are files that contain configuration, virtual disk, and other data associated with a virtual machine.
- **Datastore** – The `Datastore` managed entity provides methods for mounting datastores, browsing datastores, and obtaining information about the datastores associated with a virtual machine. See [Creating and Managing Datastores](#).
- **HostDiagnosticPartition** – Supports creating and querying diagnostic partitions for your ESX/ESXi host. See [Managing Diagnostic Partitions](#).

Introduction to Storage

The VMware vSphere storage architecture consists of layers of abstraction that hide and manage the complexity and differences of physical storage subsystems, shown in the following illustration.

Figure 8-1. Storage Architecture



How Virtual Machines Access Storage

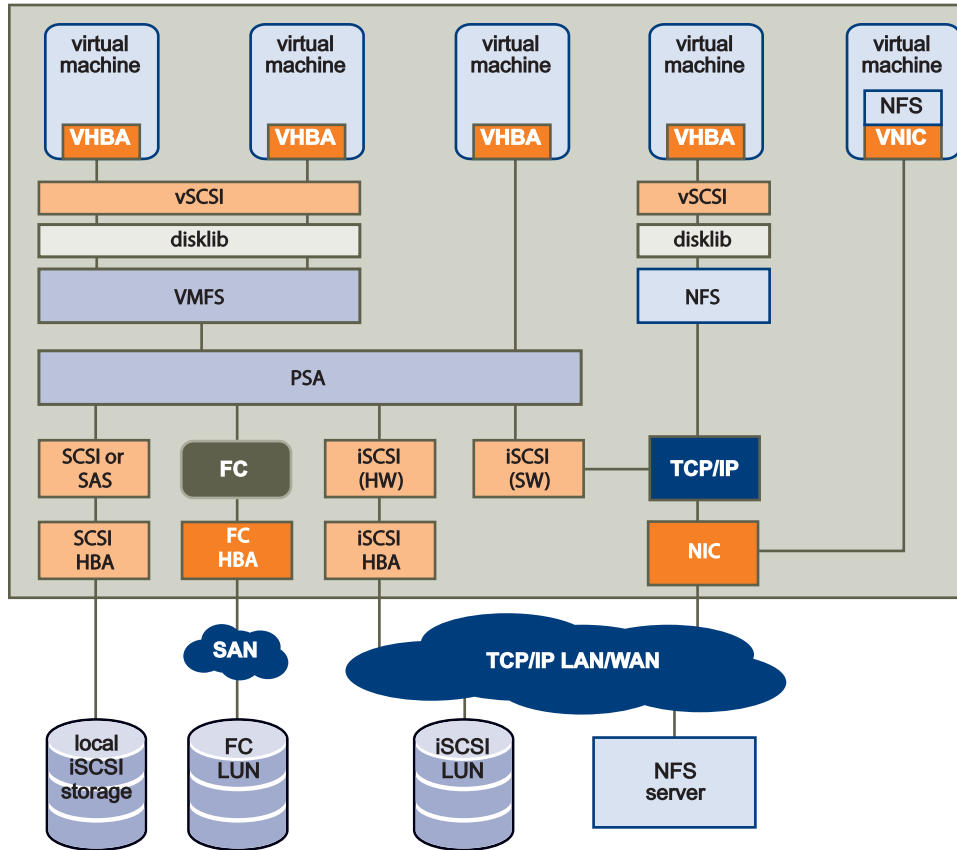
Virtual machines use virtual disks for their operating system, application software, and other data files. A virtual disk is stored as a VMDK file on a datastore. The virtual disk hides the physical storage layer from the virtual machine's operating system. Regardless of the type of storage device that your host uses, the virtual disk always appears to the virtual machine as a local SCSI device. As a result, you can run operating systems that are not certified for specific storage equipment, such as SAN, in the virtual machine.

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

To the applications and guest operating systems running on each virtual machine, the storage subsystem appears as a virtual SCSI controller connected to one or more virtual SCSI disks as shown in the top half of [Figure 8-1](#). These controllers are the only types of SCSI controllers that a virtual machine can see and access, and include the objects that extend `VirtualSCSIController`:

- `ParaVirtualSCSIController`
- `VirtualBusLogicController`
- `VirtualLsiLogicController`
- `VirtualLsiLogicSASController`

How precisely a virtual machine accesses storage depends on the setup of the host. [Figure 8-2](#) gives an overview of the different possibilities.

Figure 8-2. Storage API Architecture

Datstores

A datastore is a manageable storage entity, usually used as a repository for virtual machine files including log files, scripts, configuration files, virtual disks, and so on. vSphere supports two types of datastores, VMFS and NAS.

- If you want to use a NAS volume, mount it using `CreateNasDatastore` and unmount it using `RemoveDatastore`. The two commands are host specific, you must invoke the create and remove methods on each host on which you want to mount or unmount the datastore.
- To create a VMFS datastore, call `CreateVmfsDatastore`, passing in any existing disk. As a result of the call, the disk is formatted with VMFS and the datastore is automounted on all ESX/ESXi hosts on which the disk is visible the next time you perform a rescan. When you call `RemoveDatastore` on a VMFS datastore, the datastore is destroyed. After a rescan, the datastore is no longer available to any ESX/ESXi systems. In contrast to NAS datastores, you do not have to invoke methods for creation and removal of the datastore on each host.

An ESX/ESXi host automatically discovers the VMFS volume on attached Logical Unit Numbers (LUNs) on startup and after re-scanning the host bus adapter. When you create a VMFS datastore, the datastore label is based on the VMFS volume label. If there is a conflict with an existing datastore, the label is made unique by appending a suffix. The VMFS volume label remains unchanged.

Destroying a VMFS datastore removes the partitions that compose the VMFS volume.

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

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

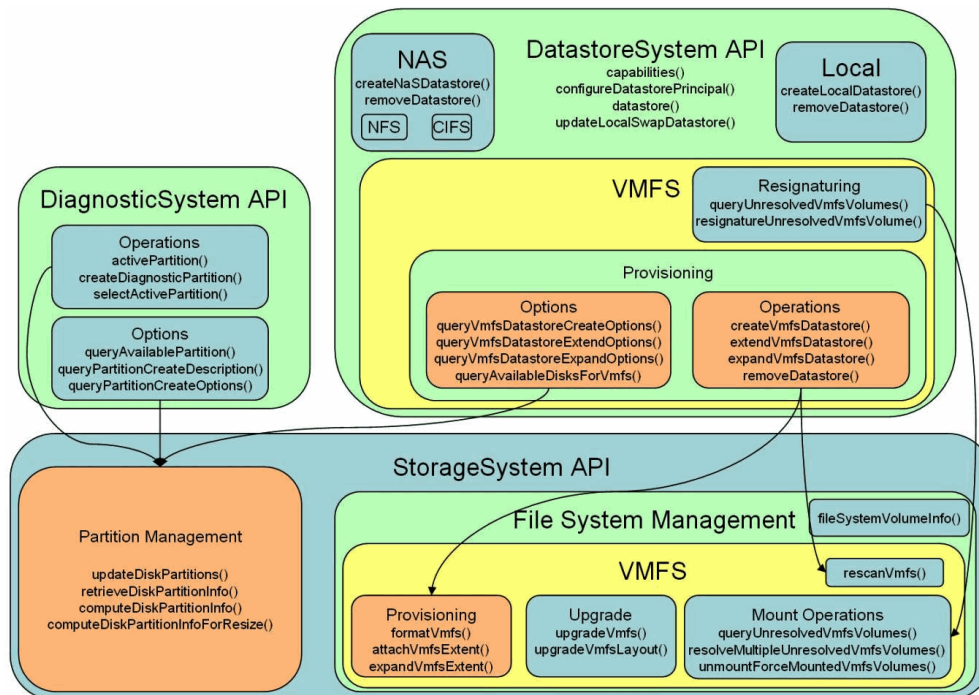
Choosing the Storage API to Use

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

The HostDatastoreSystem APIs are primarily used for managing VMFS volumes. They don't require an in-depth knowledge of storage systems, and do enforce best practices.

[Figure 8-3](#) gives an overview of the different APIs. See [Managed Objects for Working with Storage](#) to determine which tasks are commonly performed with which API.

Figure 8-3. Storage APIs



Managed Objects for Working with Storage

The Web Services API provides several managed objects that are designed to work with ESXi storage devices.

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

Configuring Disk Partitions

HostStorageSystem manages low-level storage components including HBAs, SCSI LUNs, file system volumes, and so on. You can use this API to set up the partitions before creating, extending, or expanding a VMFS file system.

- **ComputeDiskPartitionInfo** – Computes the disk partition information based on the specified disk layout. The server computes a new HostDiskPartitionInfo object for a specific disk using the layout that is specified by the HostDiskPartitionLayout object. Inside the HostDiskPartitionLayout object, you specify the list of block ranges for that partition, and optionally the total number and size of the blocks. You can then use that information inside the HostDiskPartitionSpec when updating a disk partition.
- **ComputeDiskPartitionInfoForResize** – Computes the disk partition information to support resizing a given partition. Returns the resized disk partition information as a HostDiskPartitionInfo object. You can then use that information inside the HostDiskPartitionSpec when resizing the disk partition.
- **RetrieveDiskPartitionInfo** – Allows you to specify an array of device path names that identify disks and returns an array of HostPartitionInfo objects for each of those disks.
- **UpdateDiskPartitions** – Changes the partitions on a disk by supplying a partition specification (HostDiskPartitionSpec) and device name.

After you have updated the disk partitions for the host, you must perform a rescan by using one of the following methods. Complete rescans might take a long time.

- `RefreshStorageSystem` – Refreshes the storage information and settings to pick up changes, but does not explicitly issue commands to discover new devices.
- `RescanAllHba` – Rescans all host bus adapters for new storage devices. This method might take a long time.
- `RescanHba` – Rescans a specific host bus adapter for new devices.

`HostStorageSystem` methods are also used for setting up iSCSI storage. See [Configuring iSCSI Storage](#).

Multipath Management

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

- `EnableMultipathPath` – Enables a disabled path for a device. Use the pathname from `HostMultipathStateInfoPath` or `HostMultipathInfoPath`.
- `QueryPathSelectionPolicyOptions` – Obtains the set of path-selection-policy options. These options determine the path that can be used by a device that is managed by native multipathing. A `HostMultipathInfo` data object identifies the devices that are managed through native multipathing.
- `QueryStorageArrayTypePolicyOptions` – Obtains the set of storage-array-type policy options. These options determine the storage-array-type policies that a device that is managed by native multipathing might use. A `HostMultipathInfo` data object identifies the devices that are managed through native multipathing.
- `SetMultipathLunPolicy` – Updates the path selection policy for a LUN. Specify the LUN using the LUN UUID from the `HostMultipathInfoLogicalUnit` object.
- `DisableMultipathPath` – Disables an enabled path for a device. Use the pathname from `HostMultipathStateInfoPath` or `HostMultipathInfoPath`.

Configuring iSCSI Storage

vSphere supports software iSCSI, dependent hardware iSCSI, and independent hardware iSCSI. See *Configuring iSCSI Adapters and Storage* in the *vSphere Storage* documentation for a detailed discussion.

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

- Add a dynamic or static target.
 - `AddInternetScsiSendTarget` – Adds send target entries to the host bus adapter discovery list if the `DiscoveryProperties.sendTargetsDiscoveryEnabled` flag is set to true.

- `AddInternetScsiStaticTargets` – Adds static target entries to the host bus adapter discovery list. The `DiscoveryProperty.staticTargetDiscoveryEnabled` flag must be set to true.
- Configure targets.
 - `UpdateInternetScsiAdvancedOptions` – Updates the advanced options that the iSCSI host bus adapter or the discovery addresses and targets associated with it.
 - `UpdateInternetScsiAlias` – Updates the alias of an iSCSI host bus adapter.
 - `UpdateInternetScsiAuthenticationProperties` – Updates the authentication properties for one or more targets or discovery addresses associated with an iSCSI host bus adapter.
 - `UpdateInternetScsiDigestProperties` – Updates the digest properties for the iSCSI host bus adapter or the discovery addresses and targets associated with it.
 - `UpdateInternetScsiDiscoveryProperties` – Updates the discovery properties for an iSCSI host bus adapter.
 - `UpdateInternetScsiIPProperties` – Updates the IP properties for an iSCSI host bus adapter.
 - `UpdateInternetScsiName` – Updates the name of an iSCSI host bus adapter.
 - `UpdateSoftwareInternetScsiEnabled` – Enables and disables software iSCSI in the VMkernel.
- Remove a dynamic or static target.
 - `RemoveInternetScsiSendTargets` – Removes send target entries from the host bus adapter discovery list. The `DiscoveryProperty.sendTargetsDiscoveryEnabled` must be set to true. If any of the targets provided as parameters are not found in the existing list, the other targets are removed and an exception is thrown.
 - `RemoveInternetScsiStaticTargets` – Remove static target entries from the host bus adapter discovery list. The `DiscoveryProperty.staticTargetDiscoveryEnabled` must be set to true. If any of the targets provided as parameters are not found in the existing list, the other targets are removed and an exception is thrown.

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

Configure the VMkernel To Support Software iSCSI

Before you can set up iSCSI on a system, you must create a dedicated VMkernel network interface. See. You can then enable the VMkernel to support iSCSI and configure the initiator.

To enable the VMkernel to support software iSCSI, use the following steps.

Procedure

- 1 Obtain a managed object reference to the host system's `HostStorageSystem`.
- 2 Invoke the `UpdateSoftwareInternetScsiEnabled` method, passing the reference to the `HostStorageSystem` and the value `true`.

Configure iSCSI Initiators

After you have configured the VMkernel to support software iSCSI, you can configure the storage initiators for iSCSI.

To configure iSCSI initiators, use the following steps.

Procedure

- 1 Access the list of available HBAs on the host system.

You can do this by creating a property collector with `HostSystem` as the starting point. See [#unique_28](#). From the `HostSystem.config` property, you can obtain the list (array) of host bus adapters by specifying this property path:

```
config.storageDevice.hostBusAdapter
```

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

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

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

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

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

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

- 8 Configure access to the targets.
- 9 Rescan the HBAs.

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

Creating and Managing Datastores

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

The `HostDatastoreSystem` managed object provides methods for creating and managing datastores. All `HostDatastoreSystem` methods require a managed object reference to `HostDatastoreSystem`, and return a reference to the `Datastore` object after it is created.

`HostDatastoreSystem` allows you to create and expand, query, and remove or update datastores. `HostDatastoreSystem` also allows you to configure a datastore principal for a host by calling `ConfigureDatastorePrincipal`. All virtual machine-related file I/O is performed under this user.

VMFS provisioning tasks are often performed as follows:

- 1 Call `QueryAvailableDisksForVmfs` to get the subset of disks that are well suited for holding VMFS datastores.

`QueryAvailableDisksForVmfs` obtains a list of disks that can be used to contain VMFS datastore extents. You can provide a datastore name to obtain the list of disks that can contain extents for the specified VMFS datastore. The operation does not return disks currently used by the VMFS datastore, nor does it return management LUNs and disks that are referenced by RDMs. RDM disks are not usable for VMFS datastores.
- 2 Get information about provisioning options by calling one of the following methods, passing in the selected disk:
 - `QueryVmfsDatastoreCreateOptions` – Obtains information about options for creating a new VMFS datastore on a disk. The method returns an array of `VmfsDatastoreOption` data objects.
 - `QueryVmfsDatastoreExpandOptions` – Obtains information about options for expanding the extents of an existing VMFS datastore.
 - `QueryVmfsDatastoreExtendOptions` – Obtains information about options for extending an existing VMFS datastore for a disk.

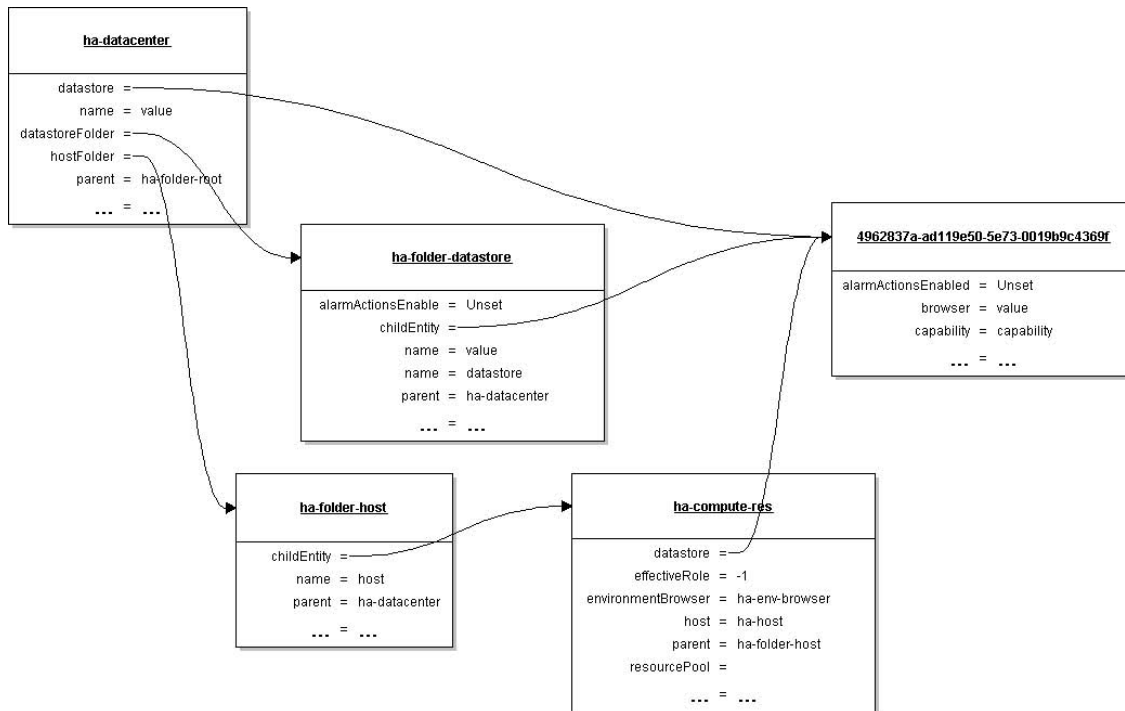
- 3 If required, change the layout by calling `HostStorageSystem.ComputeDiskPartitionInfo` and then `HostStorageSystem.UpdateDiskPartition` to resize the partition.
- 4 Call `CreateVmfsDatastore`, `ExtendVmfsDatastore`, or `ExpandVmfsDatastore` to complete the VMFS provisioning operation.

Accessing Datastores

The following diagram illustrates how you can access or specify datastores.

- Each Datacenter managed object has a `datastore` property that contains an array of datastores.
- Each Datacenter managed object has a `datastoreFolder` property that is a reference to the folder (or folder hierarchy) that contains the datastores for this datacenter.
- Each Datacenter managed object has a `hostFolder` property that is a reference to the folder (or folder hierarchy) that contains the compute resources, including hosts and clusters, for this datacenter. Each `HostSystem` or `ComputeResource` has a `datastore` property that is an array of `Datastore` managed objects.

Figure 8-4. Datastore Managed Object



For more information about the hierarchy of managed objects, see [Chapter 4 Datacenter Inventory](#).

Creating and Modifying a VMFS Datastore

A datastore is a manageable storage entity, usually used as a repository for virtual machine files including log files, scripts, configuration files, virtual disks, and so on.

VMFS is a proprietary file system VMware designed for virtual machines. VMFS is well suited for storing a small number of large data files like virtual disks. These files are mostly used by a single host. VMFS differs from other filesystem formats like FAT16/FAT32 and so on in that it can be accessed by multiple hosts connected to the same SAN LUN.

You can set up a VMFS datastore on any SCSI-based storage device that the host can access. VMFS volume creation, extension, and expansion requires first partitioning operations and the VMFS volume operations.

Set Up Disk Partitions

The Web Services API enables you to manage disk partitions on ESXi host storage devices.

To set up the disk partitions, use the following steps.

Procedure

- 1 Call `HostStorageSystem.RetrieveDiskPartitionInfo` to retrieve information about existing partitions.
- 2 Call `HostStorageSystem.ComputeDiskPartition`, passing in the desired disk layout. The server computes a new partition information object for a specific disk representing the desired layout and returns a `HostDiskPartitionInfo` object that you can use in the `HostDiskPartitionSpec` you pass into `UpdateDiskPartitions`.
- 3 Call `HostStorageSystem.UpdateDiskPartitions` to update partitions by passing in a `HostDiskPartitionSpec`.

Create the VMFS Datastore

After you create disk partitions on host storage, you can configure VMFS datastores.

To create a VMFS datastore, use the following steps.

Procedure

- 1 Configure and install any third-party adapter your storage requires and rescan the adapters by calling `HostStorageSystem.RescanAllHba`.
- 2 Call `HostDatastoreSystem.QueryAvailableDisksForVmfs` for information about disks that can be used to contain VMFS datastore.

This method filters out disks that are currently in use by an existing VMFS unless the VMFS using the disk is one being extended. It will also filter out management LUNs and disks that are referenced by RDMS. These disk LUNs are also unsuited for use by a VMFS. The method returns an array of `HostScsiDisk` objects.
- 3 Call `HostDatastoreSystem.QueryVmfsDatastoreCreateOptions` for information about options for creating a new VMFS datastore. The call returns an array of `VmfsDatastoreCreateOption` data objects that allow you to access the UUIDs of suitable data stores.
- 4 (Optional) If no suitable partitions for your VMFS volume exist, you might have to create them. Use the `ComputeDiskPartitionInfo` and `UpdateDiskPartitions` methods in `HostStorageSystem`.

5 Create the datastore.

- Call `HostDatastoreSystem.CreateVmfsDatastore` to create a VMFS datastore. The method takes a `VmfsDatastoreCreateSpec` data object that consists of a partition, a `HostVmfsSpec`, and an optional extent. The `HostVmfsSpec` allows you to specify the block size, extent, major version, and volume name for the VMFS.
- Call `HostDatastoreSystem.CreateNasDatastore` to create a network-attached storage based datastore.

You can later expand and extend the VMFS datastore by calling one of the following methods.

- Call first `QueryVmfsDatastoreExpandOptions` and then `ExpandVmfsDatastore` to expand an existing VMFS datastore using the specification provided in the `VmfsDatastoreExpandSpec` data object (which contains the name of the extent and partition information). `ExpandVmfsDatastore` increases the size of the datastore up to the full size provisioned for the datastore, if necessary.
- Call first `QueryVmfsDatastoreExtendOptions` and then `ExtendVmfsDatastore` to extend an existing VMFS datastore using the specification provided in the `VmfsDatastoreExtendSpec` data object.

Removing and Updating Datastores

The Web Services API has methods to update and remove datastores.

- `RemoveDatastore` – Removes a datastore from a host.
- `UpdateLocalSwapDatastore` – Choose the `localSwapDatastore` for this host. Any change to this setting affects virtual machines that subsequently power on or resume from a suspended state at this host, or that migrate to this host while powered on. Virtual machines that are currently powered on at this host are not affected.

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

Managing VMFS Datastores with `HostStorageSystem`

In most cases, the `Datastore` methods are appropriate for creating and managing VMFS datastores. However, in some cases the following `HostStorageSystem` commands are used instead:

- `AttachVmfsExtent` – Extends a VMFS by attaching a disk partition as an extent.
- `ExpandVmfsExtent` – Expands a VMFS extent as specified by the disk partition specification.
- `FormatVmfs` – Formats a new VMFS on a disk partition based on the `HostVmfsSpec` that you pass in. Returns a `HostVmfsVolume` that represents the new VMFS file system. The `HostVmfsVolume` includes the block size, list of partition names of the disk's VMFS extents, and other information including the UUID.

This command is a low-level API you can use to partition disks explicitly. In most cases, the `Datastore` VMFS commands are more suitable.

- `RescanVmfs` – Rescans for new VMFS instances.
- `UpgradeVmfs` – Upgrades the VMFS to the current VMFS version.

Update and Upgrade with `HostStorageSystem`

You can use the `HostStorageSystem` managed object to alter characteristics of storage units.

- `HostStorageSystem.UpdateScsiLunDisplayName` – Update the mutable display name associated with a SCSI LUN. The SCSI LUN to be updated is identified using the LUN UUID.
- `HostStorageSystem.UpgradeVmLayout` – Iterates over all registered virtual machines. For each virtual machine, upgrades the layout and logs an event. After the method has been called, the information in the `VirtualMachineFileLayout` data object data object is correct.

Managing VMFS Volume Copies (Resignaturing)

By default, ESX/ESXi hosts mount all VMFS datastores. Each VMFS datastore that is created in a partition on a LUN has a unique UUID that is stored in the file system superblock. In addition, the LUN ID of the source LUN is unique and is stored in the VMFS metadata.

When a LUN is replicated or a copy is made, the resulting LUN copy is identical, byte-for-byte, with the original LUN. As a result, if the original LUN contains a VMFS datastore with UUID X, the LUN copy appears to contain an identical VMFS datastore, or a VMFS datastore copy, with exactly the same UUID X. ESX/ESXi can determine whether a LUN contains the VMFS datastore copy, and considers the copy unresolved and does not mount it automatically.

To make the data on the LUN copy available, you can either force mount the copy if you are sure the original is not in use, or you can resignature the copy. When you perform datastore resignaturing, consider the following points:

- Datastore resignaturing is irreversible because it overwrites the original VMFS UUID.
- The LUN copy that contains the VMFS datastore that you resignature is no longer treated as a LUN copy, but instead appears as an independent datastore with no relation to the source of the copy.
- A spanned datastore can be resignatured only if all its extents are online.
- The resignaturing process is crash and fault tolerant. If the process is interrupted, you can resume it later.
- You can mount the new VMFS datastore without a risk of its UUID colliding with UUIDs of any other datastore, such as an ancestor or child in a hierarchy of LUN snapshots.

See the *vSphere Storage* documentation for additional information.

Resignaturing Volumes with ResignatureUnresolvedVmfsVolume_Task

The easiest way to resignature unresolved volumes is by using the `HostDatastoreSystem.ResignatureUnresolvedVmfsVolume_Task` method. The method assigns a new `DiskUuid` to a VMFS volume, but keep its contents intact. The method supports safe volume sharing across hosts and is appropriate in most cases.

You can instead use the low-level `HostStorageSystem` methods to find, force mount, or unmount unresolved volumes:

- `HostStorageSystem.QueryUnresolvedVmfsVolume` – Obtains the list of unbound VMFS volumes. For sharing a volume across hosts, a VMFS volume is bound to its underlying block device storage. When a low-level block copy is performed to copy or move the VMFS volume, the copied volume is unbound.
- `HostStorageSystem.ResolveMultipleUnresolvedVmfsVolumes` – Resignatures or force mounts unbound VMFS volumes. This method takes a `HostUnresolvedVmfsResolutionSpec` data object as input. The `HostUnresolvedVmfsResolutionSpec.resolutionSpec` property is an array of `HostUnresolvedVmfsResolutionSpec` data objects that contain a `HostUnresolvedVmfsResolutionSpecVmfsUuidResolution` enumeration. The enumeration is either `forceMount` or `resignature`.
- `UnmountForceMountedVmfsVolume` – Unmounts a force mounted VMFS volume. When a low-level block copy is performed to copy or move the VMFS volume, the copied volume is unresolved. For the VMFS volume to be usable, a resolution operation is applied. As part of resolution operation, you might decide to keep the original VMFS UUID. Once the resolution is applied, the VMFS volume is mounted on the host for its use. This method allows you to unmount the VMFS volume if it is not used by any registered virtual machines.

Managing Diagnostic Partitions

Your host must have a diagnostic partition (dump partition) to store core dumps for debugging and for use by VMware technical support.

For more information about diagnostic bundles, see [Generating Diagnostic Bundles](#). For information about how to collect diagnostic partitions for a purple screen fault in ESXi, see the VMware knowledge base article at <http://kb.vmware.com/kb/1004128>. <http://kb.vmware.com/kb/1004128>.

A 100MB diagnostic partition for each host is recommended. If more than one ESX/ESXi host uses the same LUN as the diagnostic partition, that LUN must be zoned so that all the ESX/ESXi host can access it. Each host needs 100MB of space, so the size of the LUN determines how many servers can share it. Each ESX/ESXi host is mapped to a diagnostic slot. VMware recommends at least 16 slots (1600MB) of disk space if servers share a diagnostic partition. You can set up a SAN LUN with FibreChannel or hardware iSCSI. SAN LUNs accessed through a software iSCSI initiator are not supported.

Caution If two hosts that share a diagnostic partition fail and save core dumps to the same slot, the core dumps might be lost. To collect core dump data, reboot a host and extract log files immediately after the host fails. If another host fails before you collect the diagnostic data of the first host, the second host does not save the core dump.

Retrieving Diagnostic Partition Information

The `HostDiagnosticSystem` managed object allows you to retrieve information in several ways.

- Retrieve the `HostDiagnosticPartition` object from the `HostDiagnosticSystem.activePartition` property to examine the properties of the active partition.
- Call the `HostDiagnosticPartition.QueryAvailablePartition` method to retrieve a list of available diagnostic partitions, in order of suitability.
- Call the `HostDiagnosticPartition.QueryPartitionCreateOptions` method to retrieve a list of disks with sufficient space to contain a diagnostic partition of the specified type. The choices are returned in order of suitability.

Create a Diagnostic Partition

Creating a diagnostic partition requires that you find a suitable partition using one of the query methods. You can then retrieve a creation specification, and perform the actual creation.

To create a diagnostic partition, use the following steps.

Procedure

- 1 Find a suitable partition by calling `HostDiagnosticPartition.QueryAvailablePartition` or `HostDiagnosticPartition.QueryPartitionCreateOptions`.
- 2 Call `HostDiagnosticPartition.CreateDiagnosticPartition`, passing in a `HostDiagnosticPartitionCreateSpec`, which includes information about the diagnostic type, id, storage type, and so on.

On success, this method creates the partition and makes the partition the active partition if specified in the `active` parameter. On failure, the diagnostic partition might exist, but will not be active even if the partition was supposed to be made active.

Sample Code Reference

The following table lists the sample applications included with the vSphere Web Services SDK that demonstrate some of the topics discussed in this chapter.

Java (SDK\vsphere-ws\java\JAXWS\samples\com\vmware\)	C# (SDK\vsphere-ws\dotnet\cs\samples\)
scsilun\SCSILunName.java	SCSILunName\SCSILunName.cs
--	SCSILunName\SCSILunName.csproj
--	SCSILunName\SCSILunName2008.csproj
--	SCSILunName\SCSILunName2010.csproj
httpfileaccess\GetVMFiles.java	GetVirtualDiskFiles\GetVirtualDiskFiles.cs

vSphere Networks

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

This chapter includes the following topics:

- [Virtual Switches](#)
- [Using a Distributed Virtual Switch](#)
- [VMware Standard Virtual Switch](#)
- [Sample Code Reference](#)

Virtual Switches

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

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

To configure a vSphere network you perform the following operations:

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

Port Groups

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

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

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

Virtual Machine Network Interfaces

When you create a virtual machine, you include a `VirtualMachineConfigSpec`, which, in turn, includes a `VirtualDeviceConfigSpec`. The device property of `VirtualDeviceConfigSpec` is a `VirtualDevice` data object.

One of the available virtual devices is `VirtualEthernetCard`. You can use one of the subtypes of `VirtualEthernetCard` to specify the virtual card to use and to specify the MAC address and whether wake-on-LAN is enabled for this virtual card. See [Adding Devices to Virtual Machines](#). A limited number of adapters is supported. KB article 1001805 (<http://kb.vmware.com/kb/1001805>) discusses available network adapters and which adapter is appropriate in which situation.

VMkernel Network Interfaces

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

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

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

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

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

Physical Network Adapter (pnict)

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

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

Using a Distributed Virtual Switch

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

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

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

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

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

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

- Backup, restore, and rollback for a VMware distributed virtual switch and its associated portgroups.
- Maximum Transmission Unit (MTU) configuration.

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

Distributed Virtual Switch Configuration

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

Procedure

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

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

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

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

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

- 3 If you have created additional uplink portgroups, use the `ReconfigureDvs_Task` method to add the portgroup(s) to the `DVSConfigSpec.uplinkPortgroup` array.
- 4 Retrieve physical NIC device names from the host (`HostSystem.config.network.pnic[].device`).

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

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

When you add a host to a distributed virtual switch

(`DistributedVirtualSwitch.config.host`), the host automatically creates a proxy switch.

The proxy switch is removed automatically when the host is removed from the distributed virtual switch.

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

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

Backup, Rollback, and Query Operations

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

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

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

- To rollback the portgroup configuration, use the `DistributedVirtualPortgroup.DVPortgroupRollback_Task` method to determine if the portgroup configuration has changed. If it has changed, use the `ReconfigureDVPortgroup_Task` method to complete the rollback operation.

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

VMware Standard Virtual Switch

Network setup for ESXi hosts can consist of several parts:

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

Configuring a Standard Virtual Switch

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

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

`HostNetworkSystem` properties allow you to access `HostNetCapabilities` and `HostNetworkInfo` data objects, and access and modify the `HostNetworkConfig` data object.

`HostNetworkSystem` includes methods for retrieving and changing the network configuration. See the *API Reference* for a complete list of methods and the permissions required to run them.

- `HostNetworkConfig` – Allows you to specify the network configuration for the host. You can apply the configuration by running the `HostNetworkSystem.UpdateNetworkConfig` method.

- Network – Represents a network accessible by either hosts or virtual machines. This can be a physical network or a logical network, such as a VLAN.

When you add a host to a vCenter Server system, or when you add a virtual machine to an ESX/ESXi host, a Network is added automatically.

- `HostSystem.QueryHostConnectionInfo` and `Datacenter.QueryConnectionInfo` both return a `HostConnectInfo` data object, which describes the current network configuration.

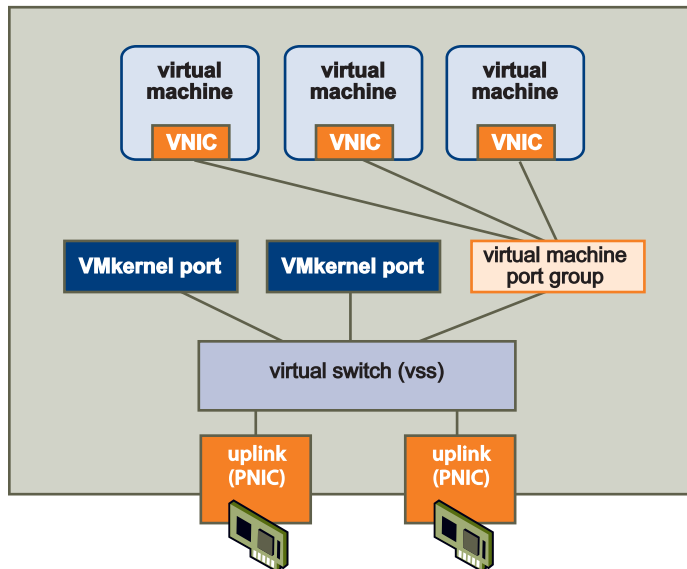
`HostSNMPSystem` – Supports SNMP setup. See [Setting Up SNMP](#).

vNetwork Standard Switch Environment

A vNetwork Standard Switch (vSS) can route traffic internally between virtual machines and can link virtual machines to external networks.

The following diagram shows the elements of a vSS environment.

Figure 9-1. vSS Environment



Virtual Switches

At the center of networking with vSS is the virtual switch itself. The vSS can send network traffic between virtual machines on the same host (private network) or network traffic to an external network (public network). The public network uses the Ethernet adapter associated with the physical host (uplink adapter).

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

Setting Up Networking with vSS

You can use the `HostNetworkSystem` managed object to access and manipulate the elements of an ESXi network.

Retrieving Information About the Network Configuration

You can retrieve information about the network configuration as follows:

- The properties of the `HostNetworkConfig` object, which you access through `HostNetworkSystem.networkConfig`, allow you to retrieve configuration information. This information is comprehensive and includes the physical adapters, virtual switches, virtual network interfaces, and so on.

You can also use `HostNetworkConfig` to make changes to the configuration.

- The properties of the `HostNetworkInfo` object, which you access through `HostNetworkSystem.networkInfo`, allow you to retrieve runtime information.

Adding a Standard Virtual Switch

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

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

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

To add a virtual switch, use the following steps.

Procedure

- 1 Obtain information about the current networking configuration.

You can use a property collector to retrieve the `HostNetworkSystem` managed object and several of its properties, such as `networkInfo`.

- 2 Define a `HostVirtualSwitchSpec` that specifies the attributes of the virtual switch. You can specify the number of ports (56 to 4088 on ESXi systems) and the `HostNetworkPolicy`. See “Defining the Host Network Policies” on page 122.
- 3 Call `HostNetworkSystem.AddVirtualSwitch` to add a virtual switch. Specify a unique name and a `HostVirtualSwitchSpec` that defines the switch attributes.

The following fragment from `AddVirtualSwitch.java` illustrates this.

Example: Adding a Virtual Switch

```

vswitchId = vSwitch42;
...
ManagedObjectReference nwSystem = configMgr.getNetworkSystem();
HostVirtualSwitchSpec spec = new HostVirtualSwitchSpec();
spec.setNumPorts(8);
service.addVirtualSwitch(nwSystem, vswitchId, spec);
System.out.println( " : Successful creating : " + vswitchId);

```

Adding a Virtual Port Group

Port groups allow you to differentiate between different kinds of traffic passing through a virtual switch. You can also use port groups as a boundary for communication or for security policy configuration. The default port groups for ESXi systems are Management Network and VM Network.

When you create a port group, you can specify a VLAN ID for it. VLANs are an important part of ESXi networking because they allow you to group traffic. For example, you could create separate network segments for VMotion, for management and for development. Using VLANS, you only need to have a separate uplink adapter for each network segment and a single virtual switch connecting to that adapter. That setup can greatly reduce the number of switches you need.

To add a virtual port group, use the following steps.

Procedure

- 1 Define a `HostPortgroupSpec`. For each port group, you can specify the network policy, the VLAN ID, and the virtual switch to which the port group belongs.
- 2 Call `HostNetworkSystem.AddPortGroup`, passing in the `PortGroupSpec`.

Adding a VMkernel Network Interface

VMkernel network interfaces provide the network access for the VMkernel TCP/IP stack. You must create new VMkernel ports for your ESXi system if you plan on using VMotion, VMware FT, or iSCSI and NAS storage. A VMkernel port consists of a port on the virtual switch and a VMkernel interface.

To add a VMkernel Network Interface to your ESXi system, use the following steps.

Procedure

- 1 Create a `HostVirtualNicSpec` data object. Inside the object, you can specify the IP configuration in a `HostIpConfig` data object. For vSS, specify the portgroup property. For vDS, specify the `distributedVirtualPort` property.
- 2 Call `HostNetworkSystem.AddVirtualNic`, passing in the `HostVirtualNicSpec`.

- 3 You can then use the VMkernel network interface for software iSCSI or NAS, or call the `HostVmotionSystem.SelectVnic` method to use this VMkernel NIC for VMotion.

Example: Adding a VMkernel Network Interface, a code fragment from the `AddVirtualNic` example, illustrates this. The sample retrieves the IP address from the command line using the `cb.get_option` call.

Example: Adding a VMkernel Network Interface

```
private HostVirtualNicSpec createVNicSpecification() {
    HostVirtualNicSpec vNicSpec = new HostVirtualNicSpec();
    HostIpConfig ipConfig = new HostIpConfig();
    ipConfig.setDhcp(false);
    ipAddr = cb.get_option("ipaddress");
    ipConfig.setIpAddress(ipAddr);
    ipConfig.setSubnetMask("255.255.255.0");
    vNicSpec.setIp(ipConfig);
    return vNicSpec;
}

....

HostVirtualNicSpec vNicSpec = createVNicSpecification();
service.addVirtualNic(nwSystem, portGroup, vNicSpec);
```

Defining the Host Network Policies

When you configure host networks, you can define specific policies for the network. The `HostNetworkPolicy` data object type describes network policies for both virtual switches and port groups. If the settings are not specified for the port group explicitly, the port group inherits policy settings from the virtual switch with which it is associated.

The policies are defined by the following data objects available as properties of `HostNetworkPolicy`.

- `HostNicTeamingPolicy` – Defines the connection to the physical network. This includes failure criteria, active and standby NICs, and failover and load balancing information. See [NIC Teaming](#).
- `HostNetworkSecurityPolicy` – Defines the security policies for the network. See the *ESXi Configuration Guide*.
- `HostNetworkTrafficShapingPolicy` – Establishes parameters for three traffic characteristics: average bandwidth, peak bandwidth, and maximum burst size.

You can also specify the VLAN policy by assigning an integer to the `HostPortgroupSpec.vlanid` property. The VMkernel takes care of tagging and untagging the packets as they pass through the virtual switch. See the `HostPortgroupSpec` and `HostNetworkPolicy` data objects in the *API Reference*.

NIC Teaming

Virtual machines connect to the public network through a virtual switch, which, in turn, connects to the physical network interface (pnic). When the physical adapter or the adapter's network connection fails, connectivity for the associated virtual switch and all port groups and virtual machines is lost.

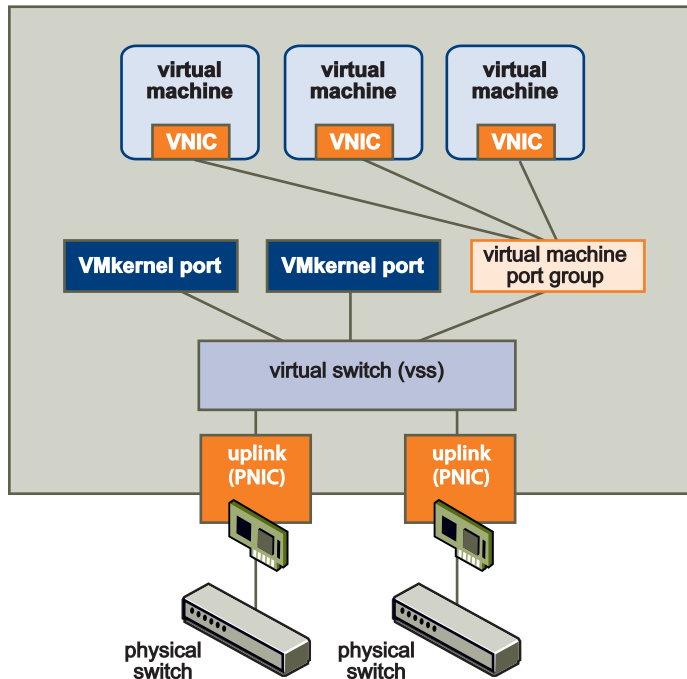
To resolve this issue, you can set up your environment so each virtual switch connects to two uplink adapters. Each uplink adapter connects to two different physical switches. The teams can then either share the load of traffic between physical and virtual networks among some or all of its members, or provide passive failover in the event of a hardware failure or a network outage.

You set up NIC teaming by setting the `HostNetworkPolicy`. The path to the `HostNicTeamingPolicy` is:

```
HostConfigSpec.network.vswitch[].spec.policy.nicTeaming
```

If you specify NIC teaming for a virtual switch, the `HostVirtualSwitchSpec.bridge` property must be set to `HostVirtualSwitchBondBridge`.

Figure 9-2. NIC Teaming



Setting Up IPv6 Networking

vSphere supports both Internet Protocol version 4 (IPv4) and Internet Protocol version 6 (IPv6) environments. With IPv6, you can use vSphere features such as NFS in an IPv6 environment.

An IPv6-specific configuration in vSphere involves providing IPv6 addresses, either by entering static addresses or by using DHCP for all relevant vSphere networking interfaces. IPv6 addresses can also be configured using stateless autoconfiguration sent by router advertisement.

You can set up IPv6 networking for a host by changing the `HostIpConfig.ipV6Config` property, which is a `HostIpConfigIpV6AddressConfiguration` data object. `HostIpConfigIpV6AddressConfiguration` allows you to specify whether auto-configuration is enabled, whether DHCP for IPv6 addresses is enabled, and an array of IPv6 addresses (`HostIpConfigIpV6Address` data objects).

`HostIpConfigIPv6Address` allows you to specify all aspects of the IPv6 address including the state of the address, the address (unless DHCP is enabled), life time, operation, origin, and prefix length. See the *API Reference*. The following code fragment illustrates setting the VMkernel NIC to get an automatic IPv6 address from router advertisements and through DHCP. The user provides the IP address on the command line when calling the program from which the fragment is taken. The sample retrieves the address using the `cb.get_option` utility applications call.

Example: IPv6 Setup

```
private HostVirtualNicSpec createVNicSpecification() {
    HostVirtualNicSpec vNicSpec = new HostVirtualNicSpec();
    HostIpConfig ipConfig = new HostIpConfig();

    //setting the vnic to get an automatic ipv6 address from router advertisements
    // and through dhcp

    ipV6Config = new HostIpConfigIPv6AddressConfiguration();
    ipV6Config.setAutoConfigurationEnabled(true);
    ipV6Config.setDhcpV6Enabled(true);
    ipConfig.setIpV6Config(ipV6Config);
    vNicSpec.setIp(ipConfig);
    return vNicSpec;
}
....
```

Adding Networking Services

You can set up network services for your ESXi system by using `HostConfigManager` properties and methods.

Adding an NTP Service

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

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

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

Setting Up the IP Route Configuration

You can use the `HostNetworkSystem.UpdateIPRouteConfig` method to specify the IP route configuration for an ESXi system. The method takes a `HostIPRouteConfig` data object as an argument. In this object, you can specify the default gateway address and the IPv6 gateway address.

Setting Up SNMP

Simple Network Management Protocol (SNMP) allows management programs to monitor and control networked devices. vCenter Server and ESXi systems include different SNMP agents:

- The SNMP agent included with vCenter Server can send traps when the vCenter Server system is started or when an alarm is triggered on vCenter Server. The vCenter Server SNMP agent functions only as a trap emitter and does not support other SNMP operations such as GET.
- ESXi 4.0 includes an SNMP agent embedded in the host daemon (`hostd`) that can send traps and receive polling requests such as GET requests.

Versions of ESX released before ESX/ESXi 4.0 included a Net-SNMP-based agent. You can continue to use this Net-SNMP-based agent in ESX 4.x with MIBs supplied by your hardware vendor and other third-party management applications. However, to use the VMware MIB files, you must use the embedded SNMP agent. To use the NET-SNMP based agent and embedded SNMP agent at the same time, make one of the agents listen on a nondefault port. By default, both agents use the same port.

The SDK supports SNMP agent configuration through the `HostSnpSystem` managed object. This object includes two methods, `ReconfigureSnpAgent` and `SendTestNotification`.

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

A `HostSnpSystemAgentLimits` data object in the `HostSnpSystem.limits` property specifies limits of the agent.

Sample Code Reference

The following table lists the sample applications included with the vSphere SDK that demonstrate how to use some of the managed objects discussed in this chapter.

Java	C#
(SDK\vsphere-ws\java\JAXWS\samples\com\vmware\host)	(SDK\vsphere-ws\dotnet\cs\samples\)
AddVirtualNic.java	AddVirtualNic\AddVirtualNic.cs
	AddVirtualNic\AddVirtualNic.csproj
	AddVirtualNic\AddVirtualNic2008.csproj

Java	C#
	AddVirtualNic\AddVirtualNic2010.csproj
AddVirtualSwitch.java	AddVirtualSwitch\AddVirtualSwitch.cs
	AddVirtualSwitch\AddVirtualSwitch.csproj
	AddVirtualSwitch\AddVirtualSwitch2008.csproj
	AddVirtualSwitch\AddVirtualSwitch2010.csproj
AddVirtualSwitchPortGroup.java	AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup.cs
	AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup.csproj
	AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup2008.csproj
	AddVirtualSwitchPortGroup\AddVirtualSwitchPortGroup2010.csproj
RemoveVirtualNic.java	RemoveVirtualNic\RemoveVirtualNic.cs
	RemoveVirtualNic\RemoveVirtualNic.csproj
	RemoveVirtualNic\RemoveVirtualNic2008.csproj
	RemoveVirtualNic\RemoveVirtualNic2010.csproj
RemoveVirtualSwitch.java	RemoveVirtualSwitch\RemoveVirtualSwitch.cs
	RemoveVirtualSwitch\RemoveVirtualSwitch.csproj
	RemoveVirtualSwitch\RemoveVirtualSwitch2008.csproj
	RemoveVirtualSwitch\RemoveVirtualSwitch2010.csproj
RemoveVirtualSwitchPortGroup.java	RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup.cs
	RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup.csproj
	RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup2008.csproj
	RemoveVirtualSwitchPortGroup\RemoveVirtualSwitchPortGroup2010.csproj

Virtual Machine Configuration

A virtual machine is a software computer that, like a physical computer, runs an operating system and applications. Virtual machines are compatible with all standard x86 computers. Each virtual machine encapsulates a complete computing environment and runs independently of underlying hardware.

This chapter includes the following topics:

- [VirtualMachine Management Objects and Methods](#)
- [Creating Virtual Machines and Virtual Machine Templates](#)
- [Configuring a Virtual Machine](#)
- [Adding Devices to Virtual Machines](#)
- [Performing Virtual Machine Power Operations](#)
- [Registering and Unregistering Virtual Machines](#)
- [Customizing the Guest Operating System](#)
- [Installing VMware Tools](#)
- [Upgrading a Virtual Machine](#)

VirtualMachine Management Objects and Methods

Virtual machines are the central elements of your vSphere environment.

You create a virtual machine by calling `Folder.CreateVM_Task`, and configure the virtual machine by using properties and methods of the `VirtualMachine` managed object. Most of the properties point to data objects that the methods use as input. [Figure 10-1](#) shows some of the properties and methods.

Client applications commonly access and manipulate the following virtual machine related objects:

- `VirtualMachine` – Managed object used for most virtual machine manipulation. Includes methods that create templates, clones, or snapshots of a virtual machine, perform power operations and guest OS management, and install VMware Tools.
- `VirtualMachineConfigInfo` – Data object which allows you to retrieve configuration-specific information from a virtual machine.
- `VirtualMachineCloneSpec` – Data object which allows you to specify virtual machine properties for a clone operation. Argument to `VirtualMachine.CloneVM_Task`.

Figure 10-1. VirtualMachine Managed Object with Some Properties and Methods

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

Creating Virtual Machines and Virtual Machine Templates

To create a virtual machine, you use the `Folder.CreateVM_Task` method. The method takes a `VirtualMachineConfigSpec` data object as input argument. `VirtualMachineConfigSpec` allows you to specify the attributes of the virtual machine you are creating.

If you need several identical virtual machines, you can convert an existing virtual machine to a template and create multiple copies (clones) from the template. You can also create multiple virtual machines by cloning an existing virtual machine directly.

Creating a Virtual Machine Using VirtualMachineConfigSpec

Use the `Folder.CreateVM_Task` method to create a virtual machine by specifying its attributes. You must specify either a host or a resource pool (or both). The virtual machine uses the CPU and memory resources from the host or resource pool.

Calling the CreateVM_Task Method

Create a virtual machine by calling the `Folder.CreateVM_Task` method with the following arguments:

- `_this` —Folder where you want to place the virtual machine.
- `config`— `VirtualMachineConfigSpec` data object that specifies CPU, memory, networking, and so on. See [Specifying Virtual Machine Attributes with VirtualMachineConfigSpec](#)

- `pool`— Resource pool for the virtual machine to draw resources from.
- `host`— `HostSystem` managed object that represents the target host on which to run the virtual machine. If you invoke this method on a standalone host, omit this parameter. If the target host is part of a VMware DRS cluster, this parameter is optional; if no host is specified, the system selects one.

Important All objects must be located in the same datacenter.

Specifying Virtual Machine Attributes with `VirtualMachineConfigSpec`

The actual customization of the virtual machine happens through the properties of the `VirtualMachineConfigSpec` that is passed in as an argument to `Folder.CreateVM_Task`. For example, you can specify the name, boot options, number of CPUs, and memory for the virtual machine. All properties of `VirtualMachineConfigSpec` are optional to support incremental changes. See the *API Reference*.

The following example fragment from the `VMCreate` sample program illustrates how to define a `VirtualMachineConfigSpec`.

Example: Defining a `VirtualMachineConfigSpec` Data Object

```
VirtualMachineConfigSpec vmConfigSpec = new VirtualMachineConfigSpec();
...
vmConfigSpec.setName("MyVM");
vmConfigSpec.setMemoryMB(new Long(Integer.parseInt(500)));
vmConfigSpec.setNumCPUs(Integer.parseInt(4));
vmConfigSpec.setGuestId(cb.get_option("guestosid"));
...
```

The VMware SDK `SDK/samples/Axis/java/com/vmware/apputils/vim/VMUtils.java` sample defines a more comprehensive virtual machine that also includes a Floppy, CD-ROM, disk, and virtual NIC. See [Configuring a Virtual Machine](#) for a discussion of commonly set properties.

When you create a virtual machine, the virtual machine files are added at the virtual machine's storage location. See [Table 10-1](#).

Additional Configuration Information

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

Creating Virtual Machine Templates

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

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

Cloning a Virtual Machine

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

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

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

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

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

Example: Cloning a Virtual Machine

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

String clonedName = cloneName;

ManagedObjectReference cloneTask
    = service.cloneVM_Task(vmRef, vmFolderRef, clonedName, cloneSpec);
```

The `VirtualMachine.CloneVM_Task` method takes the source virtual machine, target folder, name, and `VirtualMachineCloneSpec` as arguments.

The `VirtualMachineCloneSpec` data object includes the location, power state, and whether the clone should be a template. The location, in turn, is a `VirtualMachineRelocateSpec` data object that specifies the target location (datastore, disk, and host or resource pool) and any transformation to be performed on the disk.

Converting a Template to a Virtual Machine

You can change a template back to an operational virtual machine.

- To convert the template to a virtual machine, call the `MarkAsVirtualMachine` method on the template. You must specify a resource pool and, optionally, a host for the virtual machine. Host and resource pool must be under the same `ComputeResource`. When the operation completes, the template no longer exists.
- To keep the template, clone the template by calling the `CloneVM_Task` method on the template. In the `VirtualMachineCloneSpec` (the `spec` parameter), set the `template` property to `false`.

Accessing Information About a Virtual Machine

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

Checking Default Files

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

Table 10-1. Default Files

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

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

File Extension	Usage	File Description
.vmsd	vmname.vmsd	Virtual machine snapshot file.
.vmsn	vmname.vmsn	Virtual machine snapshot data file.
** .delta.vmdk		Snapshot difference file. A number preceding the extension increases with more snapshots.
** .vmdk		Metadata about a snapshot.
-Snapshot#.vmsn		Snapshot of virtual machine memory. Snapshot size is equal to the size of your virtual machine's maximum memory.

Checking Default Devices

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

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

Caution Do not try to change default device properties using the `VirtualMachineConfigSpec.deviceChange` method discussed in [Adding Devices to Virtual Machines](#), because the `deviceChange` method is not applicable to default device properties.

Configuring a Virtual Machine

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

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

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

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

Property	VirtualMachineConfigSpec Path
<code>VirtualDevice.unitNumber</code>	<code>deviceChange.device.unitNumber</code>
<code>VirtualDevicePciBusSlotInfo.pciSlotNumber</code>	<code>deviceChange.device.slotInfo.pciSlotNumber</code>

Property	VirtualMachineConfigSpec Path
VirtualDiskFlatVer1BackingInfo.diskMode	deviceChange.device.backing.diskMode
VirtualDiskFlatVer2BackingInfo.diskMode	
VirtualDiskRawDiskMappingVer1BackingInfo.diskMode	
VirtualDiskSeSparseBackingInfo.diskMode	
VirtualDiskSparseVer1BackingInfo.diskMode	
VirtualDiskSparseVer2BackingInfo.diskMode	
VirtualDiskFlatVer2BackingInfo.digestEnabled	deviceChange.device.backing.digestEnabled
VirtualDiskSeSparseBackingInfo.digestEnabled	
VirtualMachineConfigSpec.changeTrackingEnabled	changeTrackingEnabled
VirtualEthernetCard.addressType	deviceChange.device.addressType
VirtualEthernetCard.macAddress	deviceChange.device.macAddress
VirtualEthernetCard.wakeOnLanEnabled	deviceChange.device.wakeOnLanEnabled
VirtualSCSIController.sharedBus	deviceChange.device.sharedBus
VirtualSerialPort.yieldOnPoll	deviceChange.device.yieldOnPoll
VirtualUSBController.autoConnectDevices	deviceChange.device.autoConnectDevices
VirtualUSBXHCIController.autoConnectDevices	
VirtualUSBController.ehciEnabled	deviceChange.device.ehciEnabled
VirtualMachineVideoCard.useAutoDetect	deviceChange.device.useAutoDetect
VirtualMachineVideoCard.videoRamSizeInKB	deviceChange.device.videoRamSizeInKB
VirtualMachineVideoCard.numDisplays	deviceChange.device.numDisplays
VirtualMachineVideoCard.use3dRendererSupported	deviceChange.device.use3dRendererSupported

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

Name and Location

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

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

The location of the virtual machine is determined implicitly during creation because you call a `Folder.CreateVM_Task` method and specify resource pool and optional target host the virtual machine should belong to. See [#unique_230](#) for a discussion of resource pools and virtual machine location.

Hardware Version

The hardware version of a virtual machine indicates the lower-level virtual hardware features a virtual machine supports, such as BIOS, number of virtual slots, maximum number of CPUs, maximum memory configuration, and other hardware characteristics.

For a newly created virtual machine, the default hardware version is the most recent version available on the host where the virtual machine is created. To increase compatibility, you might want to create a virtual machine with a hardware version older than the highest supported version. You can do so by specifying the `VirtualMachineConfigSpec.version` property during virtual machine creation. For existing virtual machines, call the `VirtualMachine.UpgradeVM_Task` method.

The hardware version of a virtual machine can be lower than the highest version supported by the ESXi host it is running on under the following conditions:

- You migrate a virtual machine to a newer version of ESXi after it was created on a host that was running an earlier version of ESXi.
- You create a virtual machine on a newer version of ESXi by using an existing virtual disk that was created on a host that was running an earlier version of ESXi.
- You add a virtual disk created on a host that was running an earlier version of ESXi to a virtual machine created on a newer version of ESXi.

Virtual machines with hardware versions lower than 4 can run on ESX/ESXi 4.x hosts but have reduced performance and capabilities. In particular, you cannot add or remove virtual devices on virtual machines with hardware versions lower than 4 when they reside on an ESX/ESXi 4.x host. To make full use of these virtual machines, upgrade the virtual hardware.

Boot Options

You can control a virtual machine's boot behavior by setting the `VirtualMachineConfigSpec.bootOptions` property. The `VirtualMachineBootOptions` data object in that property allows you to specify the following properties:

- `bootDelay` – Delay before starting the boot sequence, in milliseconds.
- `bootRetryDelay` – Delay before a boot retry, in milliseconds. This property is only considered if the `bootRetryEnabled` property is set to `true`.
- `bootRetryEnabled` – If set to `true`, a virtual machine that fails to boot tries again after the `bootRetryDelay` time period has elapsed.
- `enterBIOSSetup` – If set to `true`, the virtual machine enters BIOS setup the next time it boots. The virtual machine resets this flag to `false` so subsequent boots proceed normally.

Operating System

The guest operating system that you specify affects the supported devices and available number of virtual CPUs. You specify the guest operating system in the following two properties:

- `guestosid` – Specify one of the constants in the `VirtualMachineGuestOsIdentifier` as a string.
- `alternateGuestName` – Full name for the guest operating system. Use this property if `guestosid` is one of the values of `VirtualMachineGuestOsIdentifier` starting with `other*`.

CPU and Memory Information

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

CPU and Memory Resource Allocation

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

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

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

CPU and Memory Modification for Running Virtual Machines

Set `CpuHotAddEnabled` and `CpuHotRemoveEnabled` to specify whether virtual processors can be added to or removed from a virtual machine while the virtual machine is running. Set `MemoryHotAddEnabled` to specify whether memory can be added while the virtual machine is running.

Number of CPUs

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

CPU Processors and Memory Affinity

If your virtual machine is on an ESXi system, and if you have a license that supports Symmetric Multiprocessors (SMP), you can configure the virtual machine to have multiple virtual CPUs by setting `cpuAffinity` and `memoryAffinity`. You define a set of integers that represents the processors (for CPU) and NUMA nodes (for memory). If you are reconfiguring the affinity setting and leave the array empty, any existing affinity is removed. See the *Resource Management Guide* for a discussion of NUMA nodes and affinity.

CPU Features

You can use the `VirtualMachineConfigSpec.cpuFeatureMask[].info` property to represent the CPU features requirements for a virtual machine or guest operating system. See the `HostCpuIdInfo` data object discussion in the *API Reference* for a detailed discussion.

Networks

You configure network settings so that a virtual machine can communicate with the host and with other virtual machines.

Virtual Network Interfaces

You can add a virtual network interface to a virtual machine using a subclass of `VirtualEthernetCard`, you can set the `addressType` to `Manual`, `Generated`, or `Assigned`. If you choose `Assigned`, you can specify a MAC address explicitly.

For more information about virtual devices in general, see [Adding Devices to Virtual Machines](#).

The number of virtual network interfaces depends on the hardware version you specify for a virtual machine. Hardware version 7 virtual machines support up to ten virtual NICs. Hardware version 4 virtual machines support up to four virtual NICs.

Virtual Machine MAC Address

Upon virtual machine creation, ESXi or vCenter Server systems assign each virtual network interface its own unique MAC address. The first three bytes of the MAC address that is generated for each virtual network adapter consists of a manufacturer-specific Organizationally Unique Identifier (OUI). The MAC address-generation algorithm produces the other three bytes. vSphere generates MAC addresses that are checked for conflicts. After the MAC address has been generated, it does not change unless the virtual machine is moved to a different location.

All MAC addresses that have been assigned to virtual network interfaces of running and suspended virtual machines on a given physical machine are tracked. The MAC address of a powered off virtual machine is not checked against those of running or suspended virtual machines. It is possible that a virtual machine acquires a different MAC address after a move.

The *ESXi Configuration Guide* discusses virtual machine MAC addresses in detail.

Fibre Channel NPIV Settings

N-port ID virtualization (NPIV) supports sharing a single physical FC HBA port among multiple virtual ports, each with unique identifiers. This capability lets you control virtual machine access to LUNs on a per-virtual machine basis.

Each virtual port is identified by a pair of world wide names (WWNs): a world wide port name (WWPN) and a world wide node name (WWNN). These WWNs are assigned by vCenter Server. For detailed information on how to configure NPIV for a virtual machine, see the *Fibre Channel SAN Configuration Guide*.

NPIV support is subject to the following limitations:

- NPIV must be enabled on the SAN switch. Contact switch vendors for information about enabling NPIV on their devices.
- NPIV is supported only for virtual machines with RDM disks. Virtual machines with regular virtual disks continue to use the WWNs of the host's physical HBAs.
- Virtual machines on a host have access to a LUN using their NPIV WWNs if the physical HBAs on the ESXi host have access to a LUN using its WWNs. Ensure that access is provided to both the host and the virtual machines

You can set up NPIV with the `VirtualMachineConfigSpec` properties that start with `npiv`.

File Locations

File locations for a virtual machine are specified in the following properties:

- `VirtualMachineConfigSpec.files` is a `VirtualMachineFileInfo` data object that allows you to specify the log directory, snapshot directory, suspend directory, and configuration file location. Most locations have a default that you can change as needed.
- `VirtualMachineConfigSpec.locationID` is a 128-bit hash based on the virtual machine's configuration file location and the UUID of the host the virtual machine is assigned to. This property is not usually set by developers; however, clearing this property by setting it to an empty string is recommended if you move the virtual machine.

If a virtual machine's `VirtualMachineCapability.swapPlacementSupported` property is true for a virtual machine, you can specify a value for the `VirtualMachineConfigSpec.swapPlacement` property. The value must be one of the values of the `VirtualMachineConfigInfoSwapPlacementType` enumeration, as a string.

Adding Devices to Virtual Machines

You can add devices to a virtual machine during creation using the `VirtualMachineConfigSpec.deviceChange` property, which is a `VirtualDeviceSpec`. You specify the host device that the virtual device should map to by using a backing object. A backing object represents the host device associated with a virtual device.

- Backing option objects – You can find out which devices the host supports by extracting the relevant backing option object.
- Backing information object – The backing information object allows you to supply data for virtual device configuration. You access a `VirtualDeviceBackingInfo` object as follows:

```
VirtualMachineConfigSpec.deviceChange[].device.backing
```

To add a device to a virtual machine, you must first find out which devices are supported on the corresponding ESXi host, and then specify a `VirtualDevice` object. Perform these tasks to add a device to a virtual machine:

Procedure

- 1 Find out which devices your ESXi system supports by calling the `QueryConfigOption` method, which you can access through the `VirtualMachine.environmentBrowser` property. The method returns a `VirtualMachineConfigOption` data object that specifies what the ESXi supports. For example, `VirtualMachineConfigOption.hardwareOptions` includes information about supported CPU and memory and an array of `VirtualDeviceOption` data objects.

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

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

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

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

3 Specify connection information for the device.

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

```
VirtualDeviceConnectInfo vdc = new VirtualDeviceConnectInfo();
// Allow the guest to control whether the virtual device is connected?
vdc.setAllowGuestControl(false);
// Is the device currently connected?
vdc.setConnected(true);
// Connect the device when the virtual machine starts?
vdc.setStartConnected(true);
```

4 Define the controller key, the virtual device key, and the unit number.

You define these items with the integer properties: `controllerKey`, `key`, and `unitNumber`. See the `VirtualDevice` data object in the *API Reference*.

5 Specify device Information.

The `deviceInfo` property is a `Description` data object that has a `name` property and a `summary` property. You can supply a string value for each, describing the device.

```
Description vddesc = new Description();
vddesc.setLabel('CD-ROM Device cdrom0');
vddesc.setSummary('The CD-ROM device for this virtual machine.');
```

6 Specify the virtual device as the device property of a `VirtualDeviceConfigSpec`.

7 Specify the `VirtualDeviceConfigSpec` as the `deviceChange` property to the `VirtualMachineConfigSpec` that you pass in to a `Folder.CreateVM_Task` or `VirtualMachine.ReconfigVM_Task` method.

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

```
VirtualDevice vd = new VirtualDevice();
vd.setBacking(vcpbi);
vd.setConnectable(vdc);
vd.setControllerKey(257);
vd.setDeviceInfo(vddesc);
vd.setKey(2);
vd.setUnitNumber(25);
```

Performing Virtual Machine Power Operations

Just like physical machines, virtual machines have power states.

- **Powered on** – The virtual machine is running. If no OS has been installed, you can perform OS installation as you would for a physical machine.

- **Powered off** – The virtual machine is not running. You can still update the software on the virtual machine's physical disk, which is impossible for physical machines.
- **Suspended** – The virtual machine is paused and can be resumed; like a physical machine in standby or hibernate state.

Important Before you power on a virtual machine, you must make sure that the host has sufficient resources. You must have enough memory for the virtual machine, and some memory overhead. See [Querying Virtual Machine Memory Overhead](#).

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

- **PowerOnVM_Task** – Powers on a virtual machine. If the virtual machine is suspended, this method resumes execution from the suspend point.
- **PowerOffVM_Task** – Powers off a virtual machine.
- **ResetVM_Task** – Resets power on this virtual machine. If the current state is poweredOn, ResetVM_Task first performs a hard powerOff operation. After the power state is poweredOff, ResetVM_Task performs a powerOn operation.

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

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

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

Important Power operations might affect other virtual machines that are participating in a DRS cluster or VMware HA. See [#unique_230](#) for information about DRS clusters and VMware HA.

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

Registering and Unregistering Virtual Machines

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

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

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

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

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

The `RemoveManagedObject.java` sample illustrates unregistering a virtual machine.

Customizing the Guest Operating System

You install the guest operating system on the virtual machine just as you would install it on a physical machine. Afterwards, you can use the Web Services API to retrieve information and perform some customization if VMware Tools is installed on top of the guest operating system.

`VirtualMachine` includes the following methods for managing the guest operating system:

- `ShutdownGuest` and `RebootGuest` shut down and reboot the guest OS, and `StandbyGuest` puts the guest in hibernate mode. In each case, you perform the action on the guest OS. For example, you might shut down Windows but leave the virtual machine running.
- `ResetGuestInformation` clears cached guest information. Guest information can be cleared only if the virtual machine is powered off. Use this method if stale information is cached, preventing reuse of an IP address or MAC address.
- `SetScreenResolution` sets the console screen size of the guest operating system. When you call this method, the change is reflected immediately the virtual machine console you can access in the vSphere Client.

You can customize the identity and network settings of the guest OS with the `CustomizationSpec` data object that is a parameter to `VirtualMachine.CustomizeVM_Task`. The `CustomizationSpec` is also a property of the `VirtualMachineCloneSpec` you pass in when cloning a virtual machine.

The settings you customize with this method are primarily virtual machine settings, but because the virtual machine and the guest OS share the information, you are also customizing the guest OS with this method.

The `CustomizationSpec` allows you to set the following properties:

- `encryptionKey` – Array of bytes that can be used as the public key for encrypting passwords of administrators.
- `globalIPSettings` – Contains a `CustomizationGlobalIPSettings` data object which specifies a list of DNS servers and a list of name resolution suffixes for the virtual network adapter.
- `identity` – Allows you to specify the network identity and settings, similar to the Microsoft Sysprep tool.
- `nicSettingMap` – Custom IP settings that are specific to a particular virtual network adapter.
- `options` – Optional operations (either `LinuxOptions` or `WinOptions`).

Installing VMware Tools

VMware Tools is a suite of utilities that enhances the performance of a virtual machine's guest operating system and improves virtual machine management. For each guest OS, VMware provides a specific binary-compatible version of VMware Tools. The SDK requires that you install VMware Tools, or some operations related to the guest operating system fail.

Important You must install the guest operating system before you install VMware Tools.

With VMware Tools installed on the guest OS, the virtual machine obtains its DNS (domain name server) name and an IP address and is therefore reachable over the network.

`VirtualMachine` includes three methods for automating installation and upgrade of VMware Tools.

- `MountToolsInstaller` – Mounts the VMware Tools CD installer as a CD-ROM for the guest operating system. To monitor the status of the tools installation, check `GuestInfo.toolsStatus`. Check `GuestInfo.toolsVersionStatus` and `GuestInfo.toolsRunningStatus` for related information.
- `UnmountToolsInstaller` – Unmounts the VMware Tools installer CD.
- `UpgradeToolsTask` – Performs an upgrade of VMware Tools. This method assumes VMware Tools has been installed and is running. The method takes one argument, `InstallerOptions`, which allows you to specify command-line options passed to the installer to modify the installation procedure for tools.

Use the `ToolsConfigInfo` data object in `VirtualMachineConfigSpec.toolsInfo` property to specify the settings for the VMware Tools software running on the guest operating system.

Upgrading a Virtual Machine

You can upgrade virtual machine hardware by running the `VirtualMachine.UpgradeVM_Task` method. The method upgrades this virtual machine's virtual hardware to the latest revision that is supported by the virtual machine's current host. You can specify the version number as an argument. This method is useful if you want to run your virtual machine on a newer hypervisor that supports newer versions of the hardware.

Virtual Machine Management

Virtual machines can perform like physical computers and can be configured like physical computers. Virtual machines also support special features that physical computers do not support. This chapter discusses some of these features: migrating virtual machines, using snapshots, and using linked virtual machines.

For more information about configuring virtual machines, see the chapter [Configuring a Virtual Machine](#).

This chapter includes the following topics:

- [Virtual Machine Migration](#)
- [Snapshots](#)
- [Linked Virtual Machines](#)

Virtual Machine Migration

Migration is the process of moving a virtual machine from one host or storage location to another. Copying a virtual machine creates a new virtual machine. It is not a form of migration. vSphere supports the following migration types:

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

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

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

Cold Migration

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

Migration with VMotion

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

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

- Uses `QueryVMotionCompatibility_Task` to check two hosts are compatible.
- Uses `CheckMigrate_Task` to check whether migration is feasible. For example, if two hosts are not compatible, virtual machines cannot be migrated from one to the other.
- Uses `CheckRelocation_Task` to check whether relocation is possible.

The sample performs the migration if the hosts are compatible.

Using Storage VMotion

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

You can place the virtual machine and all its disks in a single location, or select separate locations for the virtual machine configuration file and each virtual disk. The virtual machine remains on the same host during Storage VMotion.

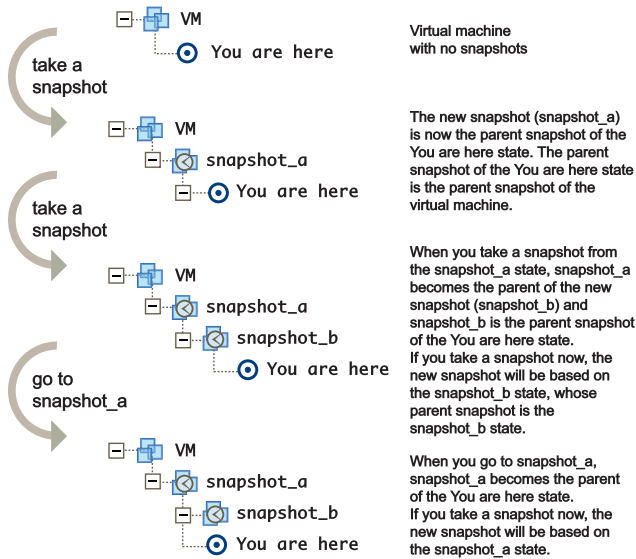
To perform storage VMotion, you use the `VirtualMachine.RelocateVM_Task` method. The `RelocateVMSpec` passed in to the method allows you to specify the target datastore and target host or resource pool.

Snapshots

A snapshot is reproduction of the virtual machine just as it was when you took the snapshot. The snapshot includes the state of the data on all virtual machine disks and the virtual machine power state (on, off, or suspended). You can take a snapshot when a virtual machine is powered on, powered off, or suspended.

When you create a snapshot, the system creates a delta disk file for that snapshot in the datastore and writes any changes to that delta disk. You can later revert to the previous state of the virtual machine.

The `VirtualMachine` object has methods for creating snapshots, reverting to any snapshot in the tree, and removing snapshots.

Figure 11-1. Virtual Machine Snapshots

When you revert a virtual machine, the virtual machine returns to the parent snapshot of the virtual machine (that is, the parent of the current You are here state).

Snapshot hierarchies can become fairly complex. For example, assume that, in the example in [Figure 11-1](#), you revert to snapshot_a. You might then work with and make changes to the snapshot_a virtual machine, and create a new snapshot, creating, in effect, a branching tree.

Creating a Snapshot

The `VirtualMachine.CreateSnapshot_Task` method creates a new snapshot of a virtual machine. As a side effect, the current snapshot becomes the parent of the new snapshot.

The method allows you to specify a name for the snapshot and also requires you set the `memory` and `quiesce` properties:

- `memory` – If `true`, a dump of the internal state of the virtual machine (basically a memory dump) is included in the snapshot. Memory snapshots consume time and resources, and take a while to create. When set to `false`, the power state of the snapshot is set to powered off.
- `quiesce` – If `true` and the virtual machine is powered on when the snapshot is taken, VMware Tools is used to quiesce the file system in the virtual machine. This ensures that a disk snapshot represents a consistent state of the guest file systems. If the virtual machine is powered off or VMware Tools is not available, the quiesce flag is ignored.

The `VMSnapshot.java` example calls this method as follows:

```
ManagedObjectReference taskMor = service.createSnapshot_Task(
    vmMor, snapshotName, desc, false, false);
```

The method returns MOR to a Task object with which to monitor the operation. The `info.result` property in the Task contains the newly created `VirtualMachineSnapshot` upon success.

Reverting to a Snapshot

When you revert to a snapshot, you restore a virtual machine to the state it was in when the snapshot was taken. The `VirtualMachine.RevertToSnapshot_Task` allows you to specify a target host and whether the virtual machine should be powered on.

If the virtual machine was running when the snapshot was taken, and you restore it, you must either specify the host to restore the snapshot to, or set the `SupressPowerOn` flag to `true`.

Deleting a Snapshot

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

Linked Virtual Machines

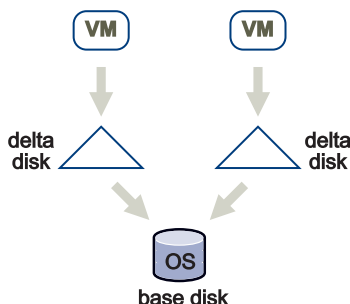
Linked virtual machines are two or more virtual machines that share storage and support efficient sharing of duplicated data.

Linked Virtual Machines and Disk Backings

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

Linked virtual machines can be created from a snapshot or from the current running point. After you create a set of linked virtual machines, they share the base disk backing and each virtual machine has its own delta disk backing, as shown in [Figure 11-2](#).

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



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

Limitation for HA Clusters

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

- Clusters that contain ESXi 5.0 or earlier hosts – If a cluster has eight or fewer hosts, then linked virtual machines restart properly. However, if the cluster has more than eight hosts and any of the hosts are ESXi 5.0 or earlier, HA might not be able to restart a virtual machine after it fails. HA is not aware that virtual machines in the linked clone group are subject to the eight host limit. In this case, when HA responds to a failure, it might try to restart the virtual machine on a host that cannot participate in the group due to the maximum host limit. HA will attempt failover five times to different hosts. Thus, in clusters with 13 or more hosts, it is possible that HA will never try a host that is associated with the linked clone group.
- Clusters that contain only ESXi 5.1 or later hosts – The maximum host limit for a linked clone group is the maximum number of hosts allowed in a cluster. In this case, the number of hosts in the cluster does not affect the ability to restart failed virtual machines.

Creating a Linked Virtual Machine

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

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

Creating a Linked Virtual Machine From a Snapshot

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

Procedure

- 1 To create the snapshot, call the `CreateSnapshot_Task` method for the virtual machine. The virtual machine can be in any power state. The following pseudo code creates a snapshot named `snap1`. The code does not include a memory dump. VMware Tools is used to quiesce the file system in the virtual machine if the virtual machine is powered on.

```
myVm.CreateSnapshot("snap1", "snapshot for creating linked virtual machines", False, True)
```

- 2 To create the linked virtual machine, specify the snapshot you created and use a `VirtualMachineRelocateDiskMoveOptions.diskMoveType` of `createNewDeltaDiskBacking`, as illustrated in [Example: Creating a Linked Virtual Machine from a Snapshot](#). Creating linked virtual machines from a snapshot works with virtual machines in any power state.

Example: Creating a Linked Virtual Machine from a Snapshot

```

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

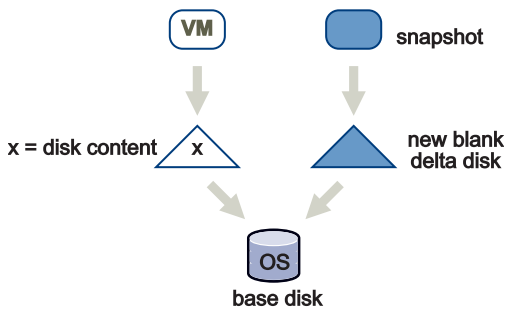
cloneSpec = new VirtualMachineCloneSpec()
cloneSpec.powerOn = False
cloneSpec.template = False
cloneSpec.location = relSpec
cloneSpec.snapshot = myVm.snapshot.currentSnapshot

myVm.Clone(myVm.parent, myVm.name + "-clone", cloneSpec)

```

The result is a virtual machine with the same base disk as the original, but a new delta disk backing.

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

**Creating a Linked Virtual Machine From the Current Running Point**

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

For more information about cloning a virtual machine, see [Example: Creating a Linked Virtual Machine from a Snapshot](#).

Example: Creating a Linked Virtual Machine from the Current Running Point

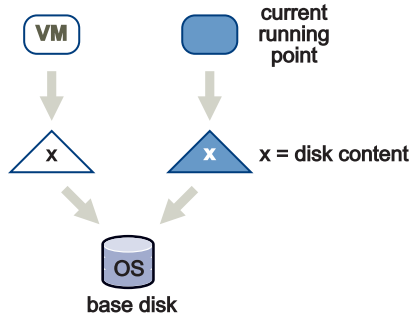
```

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

cloneSpec = new VirtualMachineCloneSpec()
cloneSpec.powerOn = False
cloneSpec.template = False
cloneSpec.location = relSpec

myVm.Clone(myVm.parent, myVm.name + "-clone", cloneSpec)

```

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

Instant Clone Virtual Machines

The instant clone operation creates a running virtual machine that continues running from the current state of the source virtual machine. There is no delay to power on the cloned virtual machine.

An instant clone virtual machine is a kind of linked clone with these characteristics:

- The clone starts out powered on and running.
- The clone replicates the memory of the source virtual machine.
- The clone shares memory pages with the source virtual machine as long as they remain unmodified.

The instant clone operation facilitates rapid deployment of virtual machines for low-latency response to peak computing demand. It also uses memory and disk space as efficiently as possible.

Because the clone replicates the running state of the source virtual machine, it usually requires customization of some network properties in the guest operating system to eliminate conflicts with the source. An instant clone virtual machine may also require new backings for some dedicated virtual devices, such as serial ports, parallel ports, or optical drives.

Instant Clone Terminology

The instant clone feature has its own set of terminology, which uses some terms in ways specific to the feature.

stun	To stop the clock of a virtual machine temporarily, preventing guest operations during an instant clone operation.
freeze	To stop the clock of a virtual machine indefinitely, allowing a series of instant clone operations that reproduce the exact process state of the source virtual machine.
suspend	Save the state of a virtual machine so applications can resume, and put it to sleep.
resume	Continue running the virtual machine that was suspended, including its applications.

unstun	To release a stunned source virtual machine, allowing I/O operations to proceed.
source	A virtual machine that is cloned during an instant clone operation.
instant clone	1 An operation that quickly creates a new virtual machine that is an exact copy of a source virtual machine. 2 A new virtual machine created by an instant clone operation.

Run State of the Instant Clone Source

An instant clone can be created either from a source virtual machine in a frozen state, or from the current running point of a source virtual machine. Certain characteristics of the process and of the result can affect your choice of frozen virtual machine or current running point for the source virtual machine.

Advantages of Cloning from the Current Running Point

An instant Clone created from the current running point of a source virtual machine does not require an agent running in the guest operating system to invoke the frozen state.

If you create an instant clone from the current running point, the source virtual machine continues running from that point after a very brief delay, whereas a frozen source virtual machine requires intervention by the host to thaw it.

Advantages of Cloning from a Frozen Virtual Machine

A source virtual machine in the frozen state allows you to deploy any number of instant clones in exactly the same state, because the guest operating system retains its state after the clone operation.

When you create instant clones from the current running point of a source virtual machine, each cloning operation adds another delta disk to the source virtual machine. Each delta disk adds a small performance penalty, but a long chain of delta disks causes a noticeable performance penalty for virtual disk accesses.

The Instant Clone Process from a Frozen Source

The process to create an instant clone from a frozen source virtual machine requires one-time initial preparation, but after that it proceeds faster than creating an instant clone from the current running point. The source virtual machine must be booted and running, in a state that you want to replicate to other virtual machines. VMware Tools must be installed and running in the guest operating system.

Freezing the Source Virtual Machine for an Instant Clone Operation

Before you start the instant clone operation, you issue a VMware Tools command in the guest operating system, which freezes the source virtual machine. This is only needed one time, prior to the first instant clone operation. After that, the source virtual machine remains frozen. When VMware Tools receives the command to freeze the virtual machine, it passes the command through a back door to the host hypervisor.

The command to freeze the virtual machine depends on the guest operating system.

- For a Linux guest operating system, either of the following commands will freeze the virtual machine:
 - `vmware-rpctool "instantclone.freeze"`
 - `vmtoolsd -cmd "instantclone.freeze"`
- For a Windows guest operating system, the following command will freeze the virtual machine:
 - `C:\Program Files\VMware\VMware Tools\vmtoolsd.exe --cmd "instantclone.freeze"`

When a virtual machine is frozen, the hypervisor prevents future context switches to the guest operating system, which in effect stops the virtual machine's clock, although the virtual machine status continues to indicate that it is running. You can determine whether the virtual machine is frozen by examining the `runtimeinfo.InstantCloneFrozen` flag.

How the Instant Clone Operation Works

The instant clone operation involves the following sequence of steps.

Stunning the Instant Clone Source Virtual Machine

When you issue an API command to initiate an instant clone operation, the host begins by doing a stun operation on the source virtual machine. The stunned condition lasts a very short time, and is not directly visible to the API user.

The stunned condition blocks all I/O operations of the source virtual machine while the instant clone is created. The stun is redundant in the case of a frozen virtual machine. During the time the source virtual machine is stunned, the host creates a delta disk for the source virtual machine, which remains empty as long as the source virtual machine is stunned or frozen.

Copying Virtual Disks During the Instant Clone Operation

Then the host creates an empty delta disk for the instant clone virtual machine. This delta disk links to the base disk of the source virtual machine. More precisely, if the source virtual machine already had one or more delta disks, the instant clone delta disk links to the second delta disk in the disk chain of the source. The second delta disk is the delta disk that was the topmost delta disk when the stun operation was invoked.

Copying Memory During the Instant Clone Operation

The host creates a new virtual memory paging file for the instant clone virtual machine, whose page tables link to the paging file of the source virtual machine. When the instant clone virtual machine is complete and is allowed to run instructions, all its memory pages will be shared with the source virtual machine until either virtual machine does a write to memory.

Configuring the Instant Clone Virtual Machine

Then the host creates a virtual machine configuration file for the instant clone, which links to its delta disk and memory paging file. At this point, the instant clone virtual machine is capable of running instructions, and the host unstuns it. Both the memory and the virtual disks are shared with the source virtual machine, so the instant clone is identical to the source virtual machine, from the guest operating system point of view.

When you create virtual machines with identical guest operating systems, the network settings conflict. Usually you will want to customize these settings for each instance of the guest operating system.

Limitations of Instant Clones

The instant clone feature provides good performance and scalability, but it has a few limitations:

- The instant clone feature is not available when you connect directly to ESXi. Instant clone depends on functionality available only in vCenter Server.
- Delta disks created during the instant clone operation are not visible in the snapshot trees of the virtual machines. They cannot be managed with the `VirtualMachineSnapshot` managed object. Instead, you must manage the delta disks using the `VirtualMachine.Reconfigure()` method.
- The instant clone operation is partly parallel and partly serialized. You can start several operations concurrently, but part of the process is exclusive and requires serialization of requests.
- The instant clone operation responds to errors by backing out changes, such as delta disks, to restore the previous state of the file system and the source virtual machine. However, there is a small window of failure at the end of the operation where the cloned virtual machine is independent and cannot be backed out to recover from an error. This is a low risk situation.

Guest Customization of Instant Clone Virtual Machines

The instant clone operation produces a virtual machine whose memory is exactly the same as the source virtual machine. This can cause conflicts in network addressing, if two different virtual machines respond concurrently with the same network identity. The conflicting network attributes are usually the host name, IP address, and MAC address.

In the case of an instant clone derived from a frozen source virtual machine, a conflict might not immediately arise, because the source virtual machine is not active on the network. The instant clone simply assumes the identity of the source virtual machine. However, a second clone derived from the same frozen virtual machine could create a network conflict by duplicating the same identity as the first instant clone.

For an instant clone derived from the current running point of the source virtual machine, the problem is more acute because the source virtual machine resumes operation simultaneously with the instant clone.

To avoid network conflicts, you customize the instant clone to adopt a new identity, before allowing both the source virtual machine and the instant clone to run concurrently. In a situation where temporary network disruption is tolerable, you can ignore the conflict while you customize the network settings in the guest operating system of the instant clone.

Otherwise, you have two options to avoid the race condition that results when both virtual machines resume running from the same point. You can either disable the virtual NIC before the instant clone operation, or else you can specify that the virtual NIC in the instant clone will be disabled at the time when it resumes running after the operation.

Avoiding Network Identity Collisions after Instant Clone Operations

To avoid a network conflict between a source virtual machine and its instant clone, you must disable at least one of the two conflicting virtual NICs until you complete network customization in the guest operating system of the instant clone. Here are some common approaches:

If the source virtual machine is frozen, its virtual NIC is effectively disabled, and you can reconfigure the guest network settings of the instant clone at any time before the source virtual machine is thawed.

If you derive the instant clone from the current running point of the source virtual machine, you can set the `connected` property of the virtual NIC backing to `false`, which disables it in the source before you start the instant clone operation. Then you re-enable the virtual NIC after the instant clone operation completes. After you customize the network settings in the guest operating system of the instant clone, you must re-enable the virtual NIC in the instant clone as well. This approach requires that you tolerate a network interruption until the instant clone operation is complete.

To avoid a network interruption, you can issue the instant clone operation with an `InstantCloneSpec` that specifies that the virtual NIC in the instant clone will be disconnected. The virtual NIC in the source virtual machine is not affected. After you complete guest operating system customization in the instant clone, you set the `connected` property of its virtual NIC back to `true`, and the instant clone operates under its new network identity.

In any case where you change the network identity settings in the guest operating system of the instant clone, you must reset the network stack for the virtual NIC to cause it to adopt the new settings. You do the reset from the guest operating system, as with a physical machine. The way to reset a NIC differs, depending on the operating system.

Resetting the Network Stack in a Linux Virtual Machine

When you change network settings in an instant clone, you must reset the network stack of the virtual NIC to adopt the new settings. For a Linux guest operating system, you must unbind and rebind the network driver so it adopts a new MAC address.

The following shell script rebinds all network drivers:

```
#!/bin/bash
for NETDEV in /sys/class/net/*
do
    DEV_LABEL=`basename \"$NETDEV\"`
    DEV_DRIVR=`basename \"$NETDEV\"`
    echo $DEV_LABEL > /sys/bus/pci/drivers/$DEV_DRIVR/unbind
    echo $DEV_LABEL > /sys/bus/pci/drivers/$DEV_DRIVR/bind
done
```

If the guest operating system was configured to use DHCP, you must also force a DHCP refresh after the MAC addresses refresh has completed.

Tip: Avoid resetting the network stack in the guest operating system while you are working over a network connection, such as SSH. You can run the shell commands to reset the stack as part of a customization script installed in the guest operating system of the source virtual machine before running the instant clone operation.

Resetting the Network Stack in a Windows Virtual Machine

When you change network settings in a Windows virtual machine, you must disable and re-enable the network object. There are several ways to do that in Windows.

Reset the Network Stack in a Windows Virtual Machine Using the Control Panel

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest operating system, you must disable and re-enable the network connection so it adopts a new MAC address. You can do this using the Control Panel.

Procedure

- 1 In the Control Panel, navigate to **Control Panel > Networks and Internet > Network Connections**.
- 2 Right-click the network adapter icon and choose **disable**.
- 3 Right-click the network adapter icon and choose **enable**.
- 4 Repeat these steps for each network adapter.

Reset the Network Stack in a Windows Virtual Machine Using PowerShell

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest operating system, you must disable and re-enable the network connection so it adopts a new MAC address. To perform this task with PowerShell, use the `Restart-NetAdapter` cmdlet.

The cmdlet is described in more detail at <https://technet.microsoft.com/en-us/itpro/powershell/windows/netadapter/restart-netadapter>.

Procedure

- 1 List the virtual network adapters in the guest operating system.

Get-NetAdapter

The names of the virtual network adapters appear in the first column of output.

- 2 Restart the network adapter representing the external interface.

Restart-NetAdapter -Name "Ethernet0"

- 3 If you have more than one virtual interface, repeat the previous step for each one.

Reset the Network Stack in a Windows Virtual Machine Using netsh

When you change network settings in a Windows virtual machine, you must reset the network stack of the virtual NIC to adopt the new settings. For a Windows guest operating system, you must disable and re-enable the network connection so it adopts a new MAC address. To perform this task with netsh, use the interface command.

Run the following commands in a DOS shell.

Procedure

- 1 List the network interfaces.

```
netsh interface show interface
```

- 2 Choose the interface that represents the external network connection and disable it with a netsh command.

```
netsh interface set interface name="Ethernet0" admin=DISABLED
```

- 3 Re-enable the interface with a netsh command.

```
netsh interface set interface name="Ethernet0" admin=ENABLED
```

- 4 Repeat the previous two commands if you have more than one NIC that needs to be reset.

Removing Snapshots and Deleting Linked Virtual Machines

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

- Snapshot removal – During snapshot removal, the snapshot metadata is also removed, and the virtual machine from which the snapshot was taken is no longer shown as having snapshots. If you remove a snapshot while connected to the ESXi host directly, shared disks are not consolidated and unnecessary levels of delta disks might result. If you remove a snapshot while connected to a vCenter Server system, shared disks are not consolidated, but unshared disks are consolidated.
- Virtual machine deletion – When you delete a virtual machine by directly connecting to the ESXi host, shared disks are not deleted. When you delete a virtual machine by connecting to a vCenter Server system, shared disks are not deleted, but unshared disks are deleted.

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

Relocating a Virtual Machine in a Linked Virtual Machine Group

You can move the virtual machines in a linked virtual machine group between datastores and save storage. The contents of the delta disk might not be as important as the contents of the base, and you can save storage by removing the delta disk.

For more information, see [Figure 11-3](#).

Example: Relocating a Linked Virtual Machine

```
relSpec = new VirtualMachineRelocateSpec()
relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.moveChildMostDiskBacking
relSpec.datastore = localDatastore
myVm.Relocate(relSpec)
```

You can relocate multiple linked virtual machines to a new datastore, but keep all shared storage during the relocation. To achieve the relocation, relocate the desired virtual machines one by one, giving the option to allow reattaching to an existing disk, as shown in [Example: Relocating Multiple Linked Virtual Machines](#).

Example: Relocating Multiple Linked Virtual Machines

```
relSpec = new VirtualMachineRelocateSpec()
relSpec.diskMoveType = VirtualMachineRelocateDiskMoveOptions.moveAllDiskBackingsAndAllowSharing
relSpec.datastore = targetDatastore
myVm.Relocate(relSpec)
```

Promoting a Virtual Machine's Disk

Promoting a virtual machine's disk improves performance.

Important You can use the PromoteDisks API only when connected to a vCenter Server system.

You can use PromoteDisks to copy disk backings or to consolidate disk backings.

- **Copy** – If the `unlink` parameter is `true`, any disk backing that is shared by multiple virtual machines is copied so that this virtual machine has its own unshared version. Files are copied into the home directory of the virtual machine. This setting results in improved read performance, but higher space requirements. The following call copies and shares disks, and then collapses all unnecessary disks.

```
myVm.PromoteDisks(True, [])
```

- **Consolidate** – If the `unlink` parameter is `false`, any disk backing that is not shared between multiple virtual machines and not associated with a snapshot is consolidated with its child backing. The net effect is improved read performance at the cost of inhibiting future sharing. The following call eliminates any unnecessary disks:

```
myVm.PromoteDisks(False, [])
```

Promoting a virtual machine's disk might also be useful if you end up with disk backings that are not needed for snapshots or for sharing with other virtual machines.

Both uses of `PromoteDisks` take an optional second argument, which allows you to apply the method to only a subset of disks. For example, you can unshare and consolidate only the virtual disk with key 2001 as follows:

```
for any of my VMs in dev
  if (dev.key == 2001)
    disk2001 = dev

myVm.PromoteDisks(True, [disk2001])
```

Performing Advanced Manipulation of Delta Disks

For advanced manipulation of delta disks, you can use `VirtualDeviceConfigSpec` methods such as `VirtualDeviceConfigSpec.create` and `VirtualDeviceConfigSpec.add`.

Together `add` and `create` allow you to create a blank delta disk on top of an existing disk. For the `VirtualDeviceConfigSpec` you specify `add` to prepare the VM for a new delta disk, then `create` to initiate the file operation for the delta disk whose parent property is an existing disk. These methods create a new delta disk whose parent is the pre-existing disk.

One use case is adding a delta disk on top of an existing virtual disk in a virtual machine without creating a snapshot. [Example: Adding a Delta Disk Backing](#) illustrates how to add the delta disk for the first virtual disk in the virtual machine.

Example: Adding a Delta Disk Backing

```
disk = None
for any of my VMs in dev
  if (VirtualDisk.isinstance == dev):
    disk = dev

# Remove the disk
removeDev = new VirtualDeviceConfigSpec()
removeDev.operation = "remove"
removeDev.device = disk

# Create a new delta disk which has the
# original disk as its parent disk
addDev = new VirtualDeviceConfigSpec()
addDev.operation = "add"
addDev.fileOperation = "create"
addDev.device = copy.copy(disk)
addDev.device.backing = copy.copy(disk.backing)
addDev.device.backing.fileName = "[" + disk.backing.datastore.name + "]"
addDev.device.backing.parent = disk.backing

spec = new VirtualMachineConfigSpec()
spec.deviceChange = [removeDev, addDev]
```

For working code with a similar use case, see the JAXWS sample program `VMDeltaDisk.java` in the SDK under `vsphere-ws/java`.