VERSION 2.0

VMware Infrastructure SDK

Getting Started Guide

VMware, Inc.

3145 Porter Drive
Palo Alto, CA 94304
www.vmware.com

Please note that you can always find the most up-to-date technical documentation on our Web site at http://www.vmware.com/support/.

The VMware Web site also provides the latest product updates.

Revision 20060615 Version 2.0 Item: SDK-ENG-Q206-121

Table of Contents

CHAPTER 1

Introducing the VMware Infrastructure SDK

The VMware Infrastructure SDK allows you to build SOAP-based applications to control ESX Server hosts and virtual machines running on those hosts. You can build those applications using any development environment compatible with SOAP -- for example, Java or C#.

The goals of the VMware® VMware Infrastructure SDK (VMware Infrastructure SDK) are:

- to provide Independent Software Vendors (ISVs) a seamless way to integrate management of VMware products into their data center management solutions. ISVs include systems management vendors, enterprise management framework vendors, imaging and provisioning vendors, storage management vendors, and so on.
- to empower end users with the flexibility to automate more complex tasks, such as creation, customization, and migration of virtual machines.

With the SDK, Independent Software Vendors can:

- Complete configuration of virtual machines and hosts.
- Support discovery and inventory functions in virtual environments.
- Access real-time performance data.

- Control all virtual machine-related operations from within the systems management framework, including configuration changes and migrations.
- Trigger virtual machine operations, specifically migrations and failover operations, based on application level monitoring and specified resource entitlements for virtual machines. See VMware DRS on page 12.
- Ensure the availability of virtual machines by automatically restarting them on other hosts in the cluster, if there is available capacity. See VMware HA on page 12.

Overview of the Getting Started Guide

The purpose of this guide is to help you familiarize yourself with the object model underlying the VMware Infrastructure SDK. This guide also describes how to use the APIs to manage and monitor your hosts and virtual machines. Developers who use this manual should be familiar with the operation and management of VMware® VirtualCenter, VMware ESX Server, VMware GSX Server, and other VMware products.

Note: If you have previously used the VMware Infrastructure SDK, then refer to the *VMware Infrastructure SDK Porting Guide* for a complete list of changes from the previous version. It provides samples you can follow to port your existing 1.x client applications to SDK version 2.0.

We discuss the following topics in this guide:

- Introducing the Object Model on page 17

  This chapter introduces the object model for the VMware Infrastructure SDK. The chapter also provides detail about the group objects, that is, objects within which you store other objects. This includes Folder, Datacenter, and so on.

- Virtual Machine and Host Resources on page 31

  This chapter introduces entity objects in the inventory, that is, objects that represent components in a virtual inventory: virtual machine, host, datastore, compute resource, and resource pool.

- Managing and Monitoring on page 49

  This chapter introduces other objects that are important in the object model, but are not inventory objects: permissions, tasks, alarms, performance statistics, and so on. You can use these other objects to monitor and manage your inventory objects, as well as to obtain performance statistics for your hosts and virtual machines.

Using the Getting Started Guide

This *VMware Infrastructure SDK Getting Started Guide* contains a description of the data models, or logical structure of the VMware Infrastructure Web Service.

Note: The *VMware Infrastructure SDK Documentation Roadmap* provides a guide to usage of the documentation included with the VMware Infrastructure SDK.

The *VMware Infrastructure SDK Reference Guide* contains a complete list of the managed object types and data object types comprising the VMware Infrastructure SDK object model. It lists all the properties, methods, enumerations, and faults associated with these objects.

The *VMware Infrastructure SDK Porting Guide* describes the differences between SDK 1.x and 2.0, and provides detailed samples illustrating how to port your SDK 1.x client applications to SDK 2.0 client applications.

The *VMware Infrastructure SDK Programming Guide* includes a detailed description of VMware Infrastructure SDK concepts and how clients interact with the Web service. The guide discusses the various operations as well as how to create a simple client application, then provides basic and advanced code structures to help you build your client application. Finally, this guide discusses the different developer environments you can use, followed by a description of the sample and reference applications supplied in the VMware Infrastructure SDK package.

Intended Audience

This guide is written for programmers who are familiar with Web services concepts and principles. Readers of this manual should be comfortable with developing system administration and system monitoring programs and be familiar with general debugging techniques. In addition, developers who use this manual should be familiar with the operation and management of VMware VirtualCenter, ESX Server and GSX Server.

Note: In this release, the VMware Infrastructure SDK supports VMware VirtualCenter 1.2 clients, VirtualCenter 2.0 clients and servers, ESX Server 2.0.1, 2.1.x, 2.5.x, and 3.0, and GSX Server 3.1 and 3.2.

What's New in Version 2.0

This is a list of the major new features in VMware Infrastructure SDK 2.0.

Users of SDK 1.x should refer to the *VMware Infrastructure SDK Porting Guide* for a list of changes and improvements, along with detailed information on how to port 1.x client applications.

New Features

VMware Infrastructure SDK Is Available on Both VirtualCenter and ESX Server Hosts

The Web service is now available on both VirtualCenter and on individual ESX Server hosts. Most of the APIs supported on VirtualCenter are also supported on an ESX Server host, with the exception of a few methods, such as VMotion, which can be used only through VirtualCenter.

Complete ESX Server Host Configuration

All the features available on ESX Server through the VMware Management interface can now be accessed programmatically at ESX Server 3.0 or at VirtualCenter 2.0, including the complete customization of an ESX Server host.

ESX Server 3.0 and VirtualCenter Features Are Available Through the VMware Infrastructure Web Service

All of the new features in ESX Server 3.0 and VirtualCenter 2.0 are available through the Web service.

Support for VirtualCenter and SDK 1.x Operations

To provide SDK 1.x compatibility, VirtualCenter 2.0 and its Web service support the same interface available on VirtualCenter 1.2.

More Flexible Inventory Model

You can create a customized inventory model for hosts and virtual machines, and create multiple nested folders in the inventory hierarchy. Clients can now view hosts, virtual machines, clusters, resource pools, datacenters, networks, and datastores in the same inventory.

Resource Abstraction

VMware Infrastructure SDK now has compute resources (sharing of physical compute resources among virtual machines), resource pools (sharing of physical resources in a single host, a subset of a single host, or resources spanning multiple hosts), and clusters (groups of hosts).

PropertyCollector

The PropertyCollector interface supports a powerful filtering mechanism that enables you to retrieve specific data. The PropertyCollector lets retrieve this information for a variety of uses, including monitoring updates.

SearchIndex
The SearchIndex interface supports a mechanism for querying inventory for a specific managed entity based on a property of that entity. These properties include UUID, IP address, DNS name, or datastore path. Such searches typically return a VirtualMachine or a HostSystem.

Virtual Machine Customization
In VirtualCenter 2.0, virtual machine customization is completely decoupled from the cloning operation.

Performance Monitoring
VirtualCenter and SDK 2.0 provide higher speed performance statistics than in previous releases.

Events, Tasks and Alarms
Complete audit trails are provided for all events, tasks, and alarms.

Custom Security Roles
You can now create your own customized roles for different VirtualCenter users, instead of using predefined, fixed roles.

VMware DRS
Clients can now specify groups of virtual machines and their resource entitlements (minimum, maximum, and shares). These virtual machines are then automatically scheduled and periodically rebalanced on hosts within a group, in order to honor the resource entitlements. DRS responds to changes in the number of virtual machines and hosts, and continues to enforce the resource entitlements across the group.

VMware HA
Clients can automatically improve the availability of virtual machines on hosts participating in the cluster. Should a host fail, the HA software detects the failure. All the virtual machines running on the host are then automatically restarted on other hosts in a group, provided that they have available capacity for the additional virtual machines.

Using Web Services

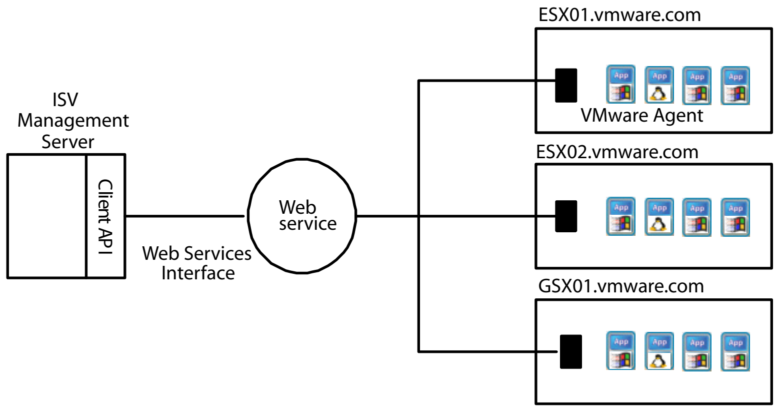
Web services are applications that have logic and functions that are accessible using standard Internet protocols and data formats such as Extensible Markup Language (XML) over Secure Hypertext Transfer Protocol (HTTPS), and SOAP (Simple Object Access Protocol). A Web service interface is defined strictly in terms of the messages that the service accepts and generates. Web services provide the following advantages:

- Industry Support — ISVs and vendors support the Web services movement. The SOAP framework is designed to be independent of any particular programming model and other implementation-specific semantics.
- Security — All client access to the Web service uses HTTPS, ensuring that management traffic is secure and encrypted.
- Interoperability — The SOAP and WSDL (Web Services Description Language) standards allow developers to program in their favorite environment, such as Java or Microsoft® Visual Studio® .NET.
- Ubiquity — Web services communicate using HTTP and XML. Any connected device that supports these technologies can both host and access Web services.
- Low Barrier to Entry — The concepts behind Web services are easy to understand, and developers can quickly create and deploy the Web services using many WSDL tool kits available on the Web.

Later chapters of this book describe the WSDL data models used by the VMware Infrastructure Web Service to allow SDK clients to interact with virtual machines and their environments.

Overview of the VMware Infrastructure Web Service

The following diagram provides an overview of how you can use the VMware Infrastructure SDK to manage your virtual machines. In SDK 2.0, you can also manage your ESX Server host directly, through the host agent interface. For more information on the host agent interface, refer to the *VMware Infrastructure SDK Programming Guide*.



The VMware Infrastructure SDK comprises the following:

- Web services API to clients for managing virtual machines running on one or more hosts.
- Documentation, including this *VMware Infrastructure SDK Getting Started Guide*, the *VMware Infrastructure SDK Programming Guide*, the *VMware Infrastructure SDK Reference Guide*, and the *VMware Infrastructure SDK Porting Guide*.
- Sample code that showcases the powerful functionality that is available to the clients.

The Web services API provides access to the following functionality:

- Element Management — Basic virtual machine management functionality within the realm of a physical host machine, including complete customization of an ESX Server host.
- Virtual Computing — Virtual machine management functionality across host machines.

Element Management Operations

The VMware SDK provides the following element management operation

- Virtual machine configuration (attaching to physical resources, modifying resource shares, and so on)

- Virtual machine power operations and state management (power on/off, suspend, reset, and so on)
- Virtual machine creation and deletion (including clones and templates)
- Customizion of a virtual machine's guest operating system
- Virtual machine inventory, with multiple nested folders in the inventory hierarchy
- Virtual disk management
- Collection of virtual machine performance data
- Property collector, which enables you to find and monitor specific data on virtual machine(s), without retrieving all the available updates
- Discovery and event notification

Host Computing Operations

- Connect or disconnect a host from VirtualCenter.
- Reboot or shut down a host.
- Create or remove datastores from a host.
- Configure Internet services and firewall protection for a host.
- Configure the networking and storage systems attached to a host.

Virtual Computing Operations

The VMware SDK provides the following virtual computing operations:

- Virtual machine provisioning and deployment
- Virtual machine migration, including VMotion™
- Virtual machine management, independent of underlying physical hosts
- Distributed availability services (fail over to another host if the current host fails)
- Distributed resource scheduling (virtual machines are automatically scheduled and periodically rebalanced on hosts within the cluster)

Technical Support Resources

Standards and Reference Documentation

Refer to these Web sites/links for additional information:

- VMware Infrastructure SDK — *www.vmware.com/support/developer/vc-sdk*
- VMware ESX Server — *www.vmware.com/products/server/esx_features.html*
- VMware VirtualCenter — *www.vmware.com/products/vmanage/vc_features.html*
- W3C SOAP 1.1 Specifications — *www.w3.org/TR/SOAP*
- XML Schema — *www.w3.org/2001/XMLSchema*
- HTTPS (SSL v3) — *wp.netscape.com/eng/ssl3/ssl-toc.html*
- WSDL 1.1 — *www.w3.org/TR/wsdl*
- HTTP 1.1 — *www.ietf.org/rfc/rfc2616.txt*
- XML 1.0 — *www.w3.org/TR/REC-xml*

CHAPTER 2

Introducing the Object Model

This chapter introduces some basic features of the Virtual Infrastructure Web Service data models: that is, the logical structure of this Web service. The first three sections present key concepts for using the SDK, and the other three sections present three data models concerned with the organizational structure of the whole.

The details of each of the data types, along with the description of each of the fields, are given in the following sections. The data models are organized in the following major groups:

- Managed Objects and Data Objects on page 19 — Explains the major differences between the two kinds of composite objects.
- Managed Entity Inventory on page 21 — Describes the structure that the Web service uses to organize virtual machines and the hosts on which they run.
- VirtualCenter and Host Agent Differences on page 23 — Describes the key differences between using the SDK with VirtualCenter and with the host agent.
- Service Instance Data Model on page 24 — Describes the top-level object accessed by the Web service.
- Folder Data Model on page 27 — Describes the subdivisions used to organize managed entities.

- Datacenter Data Model on page 29 — Describes the primary organizational unit for virtual machines and hosts.

Managed Objects and Data Objects

There are two kinds of objects in the VMware Infrastructure SDK: managed objects and data objects. Both are instances of composite object types, but there are important differences.

- Managed object types are not present in the WSDL schema, and are treated in a black box fashion by clients.
- Managed objects exist only on the server, and are passed by reference in the WSDL data stream. The term "managed object reference" denotes a reference to a server-side object that can be accessed only indirectly by the client.
- Data objects are passed by value between the client and the Web service.

Accessing Data Objects

A Web service client using an interface generation toolkit typically treats data objects in an object-oriented fashion. Although the WSDL schema itself is not object-oriented, a class hierarchy can be constructed on the client side that provides properties or methods to match the data object types present in the WSDL schema. This may be the simplest way for the client to deal with data objects.

Data objects, as defined in the WSDL schema, have only properties, not methods. If you use a toolkit, the toolkit can externalize data object properties with accessor and mutator methods.

Data objects are serialized to XML and passed in client-server communications as arguments to operations that are defined in the WSDL schema as XML messages. Operations always act on managed objects, but some operations return data objects as their results.

If you do not use an interface toolkit that provides the client with object-oriented access to data objects, you can work with the contents of a data object in any form you choose on the client side. When passing data objects between client and server, you need to build or parse a SOAP message that contains the data object properties as XML elements corresponding to the message structures described in the WSDL schema that comes with the SDK.

Accessing Managed Objects

Managed object types are not present in the WSDL schema, because they are never passed in their entirety between client and server. Either the client works with a reference to a managed object, or the client works with data objects that represent parts of the managed object.

The structures of the SDK managed objects are specified in the *VMware Infrastructure SDK Reference Guide*. You may find it helpful to build object-oriented representations of managed object types for your client to use. If you do not need an entire managed object, you can choose to build only the subset of its structure with which you need to work.

A Web service client accesses managed objects in several ways. To begin, the client must specify the intended object with a managed object reference. However, because the WSDL schema is not object-oriented, the client cannot simply invoke a method on a managed object reference.

The WSDL schema presents managed object methods (usually called "operations" in SDK documentation) as messages that take a managed object reference for the first argument. If you use a toolkit to generate a client-side interface, the toolkit may present a single object (the port object) to which all the operations belong. Where an object-oriented language implicitly passes `_this` as the first argument to a method, the client must pass a `_this` reference explicitly to each operation on the port object. In the WSDL data stream, the result is a message containing a managed object reference as the first element in the message structure.

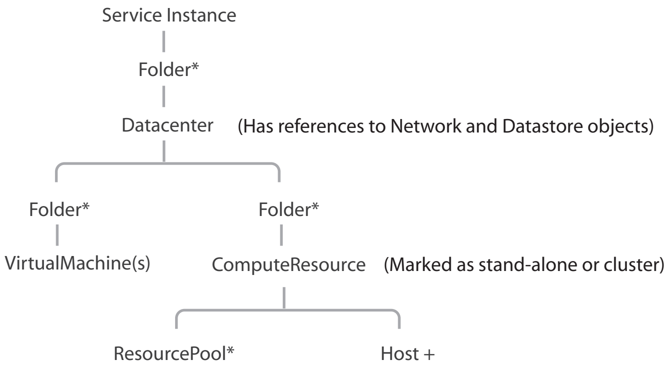
Once you have a managed object reference, you can learn about the managed object's content in one of these ways:

- The managed object may have an associated method that reports its properties. For example, the operation `CurrentTime` on the service instance object returns the current time on the server.
- The managed object may have an associated method that returns a data object contained by the managed object. For example, the operation `RetrieveServiceContent` on the service instance managed object returns the bulk of the service instance properties.
- The client can create a property collector filter used to retrieve or monitor the properties of a managed object. For example, the `ServerClock` property of the service instance can be accessed only through the property collector. Using a property collector is described in more detail in the PropertyCollector Tutorial on page 75.

Managed Entity Inventory

The VMware Infrastructure SDK manages virtual machines and host resources that are organized into an inventory hierarchy and grouped into folders. VirtualCenter imposes one hierarchical structure on the inventory; the host agent imposes a more limited version of the hierarchy, while maintaining the same general structure.

VirtualCenter Hierarchy

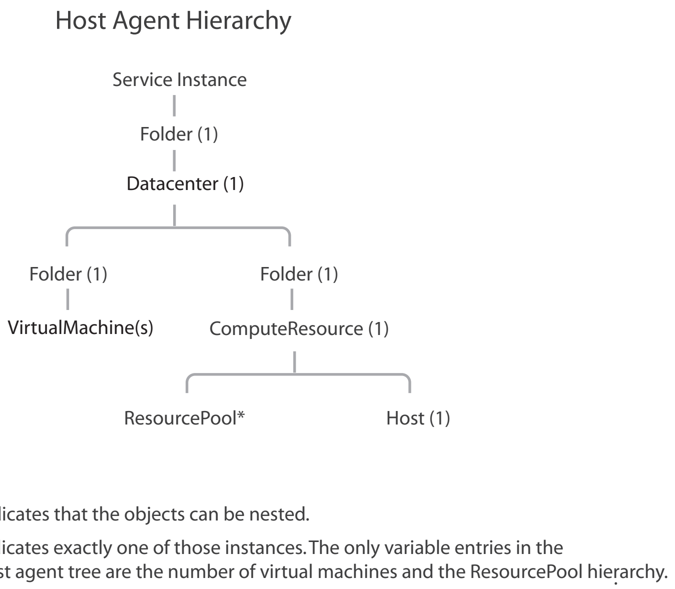


Legend

* Indicates that the objects can be nested.

+ Indicates that one or more can be presented.

The host agent form of the inventory hierarchy is similar, but imposes limits on the numbers of some objects. For instance, the host agent can manage only a single host, so the number of host objects is limited to one.


Host Agent Hierarchy
Service Instance
Folder (1)
Datacenter (1)
Folder (1)
Folder (1)
VirtualMachine(s)
ComputeResource (1)
ResourcePool*
Host (1)
Legend
* Indicates that the objects can be nested.
(1) Indicates exactly one of those instances. The only variable entries in the host agent tree are the number of virtual machines and the ResourcePool hierarchy.

Managed objects that are present in the inventory are also known as *managed entities*. The managed entity types are introduced in the next two chapters in the following sections:

- Service Instance Data Model on page 24
- Folder Data Model on page 27
- Datacenter Data Model on page 29
- VirtualMachine Data Model on page 32
- HostSystem Data Model on page 37
- Datastore Data Model on page 41
- ComputeResource Data Model on page 43
- ResourcePool Data Model on page 45

VirtualCenter and Host Agent Differences

VirtualCenter is designed to deploy, monitor, and manage virtual machines across a number of hosts running ESX Server. Major features of VirtualCenter include:

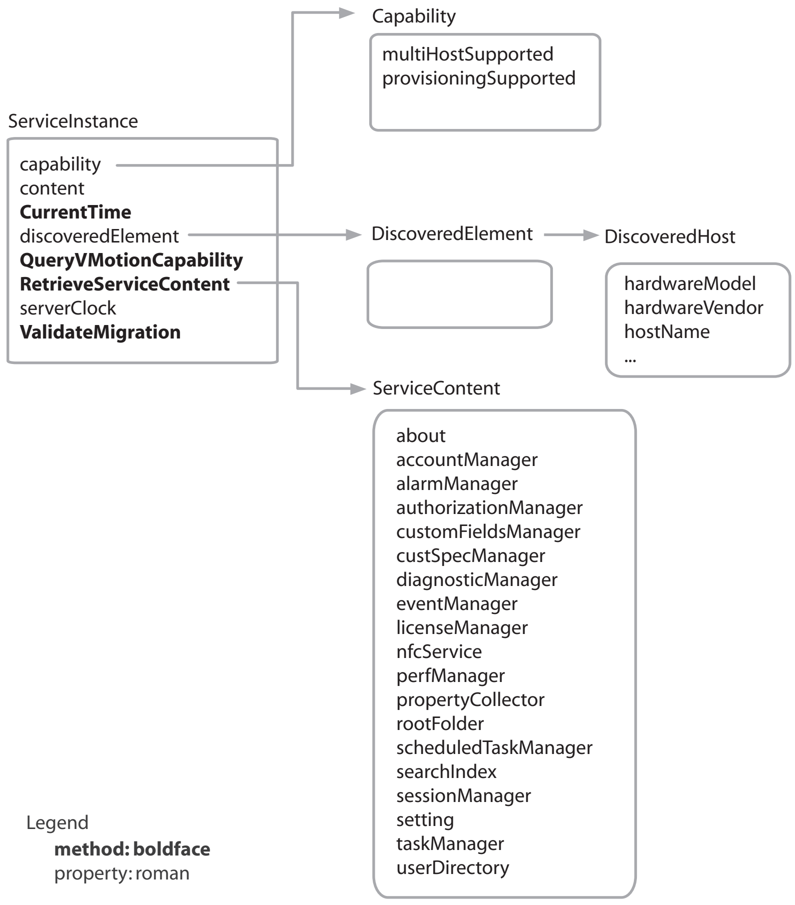
- Migration or relocation — Moving virtual machines or executing virtual machines between hosts.
- Clustering and failover — Configuring virtual machines and associated storage to fail over between hosts.
- Resource management — Backing virtual computing power with VMware HA and VMware DRS.
- Provisioning — Deploying customized virtual machines from templates.
- Monitoring — Configuring and reporting status on various conditions within the datacenter.

The host agent, which runs only on a single ESX Server host, does not provide the features listed above. The feature set of the host agent is largely a subset of the VirtualCenter feature set, but the exact features offered by the host agent depend on the product and version installed on the host.

The VMware Infrastructure SDK generally throws a `NotSupported` fault whenever you invoke an operation that is not provided by the product installed on a host involved in the operation. More information about which operations are supported by the host agent is available in the *VMware Infrastructure SDK Reference Guide*.

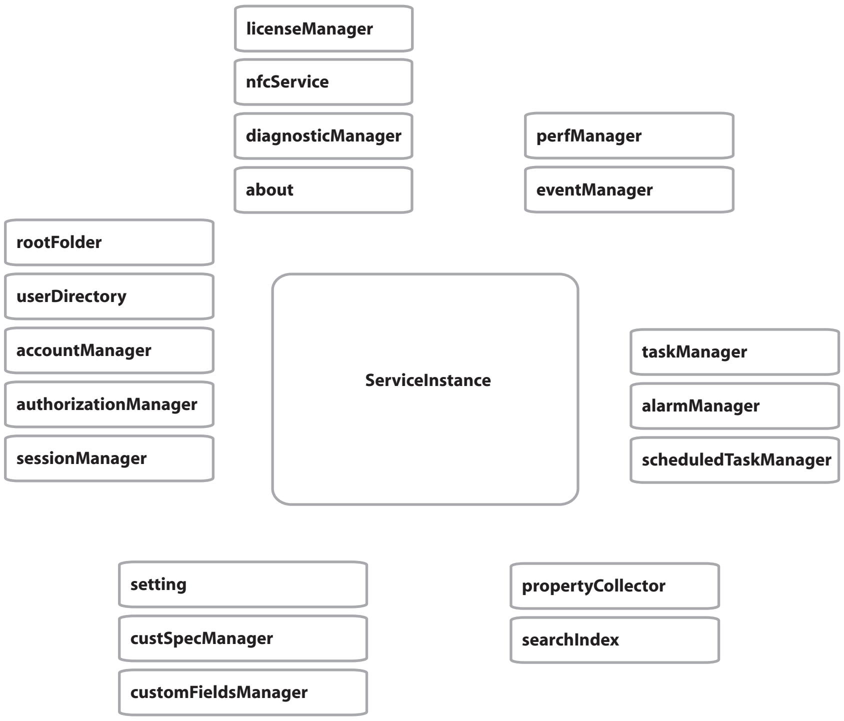
Service Instance Data Model

The service instance is the central access point for all management data. It is roughly equivalent to the Server Farm found in earlier versions of the SDK.



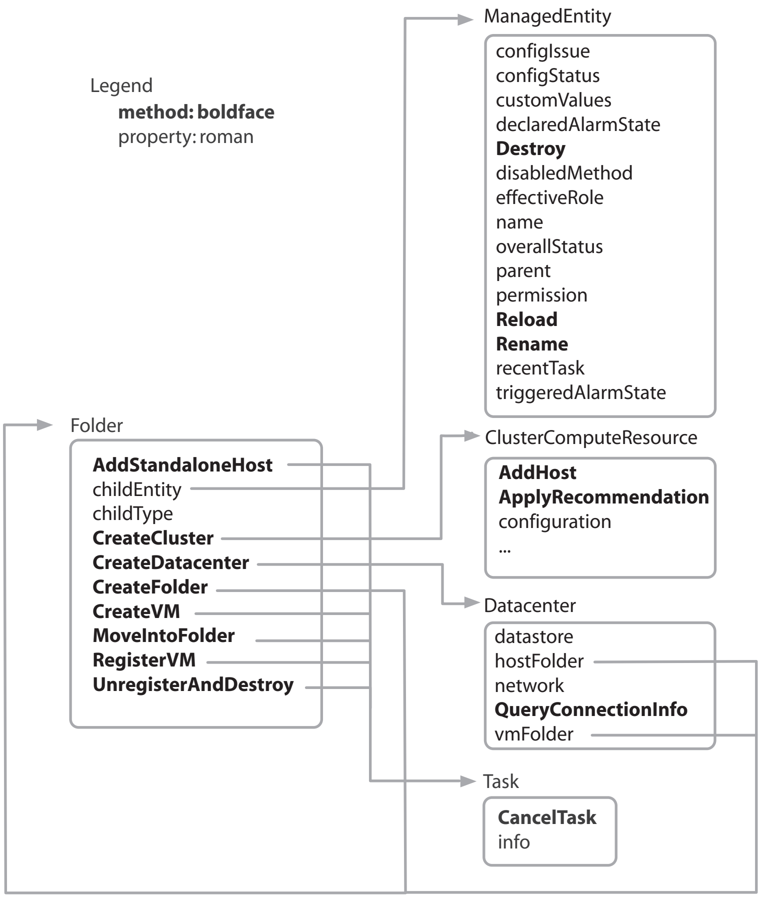
ServiceInstance has few properties and methods of its own. Most of the important properties are in ServiceContent, which you can retrieve with the RetrieveServiceContent operation. The following paragraphs describe some of the key features of ServiceInstance.

- `capability` — This is a set of flags that indicate whether or not certain major features are supported by the product that implements this service instance. For instance, the multiHostSupported capability is true for VirtualCenter but not for the host agent.
- `discoveredElement` — This is a list of host machines that VirtualCenter has discovered on the network.
- `content` — This contains the bulk of the service instance's properties, including the root folder of the inventory, the session manager, and the property collector. Most of the service instance properties are referenc.es to singleton objects used to manage objects of a given type.



To communicate with the Web service, you need to start with the ServiceInstance object. It gives you access to the inventory, which organizes all your virtual machines and hosts. The service instance also gives you access to the various session management objects, such as the AuthorizationManager, the TaskManager, and the EventManager. The procedure to access the service instance object is described in the *VMware Infrastructure SDK Programming Guide*.

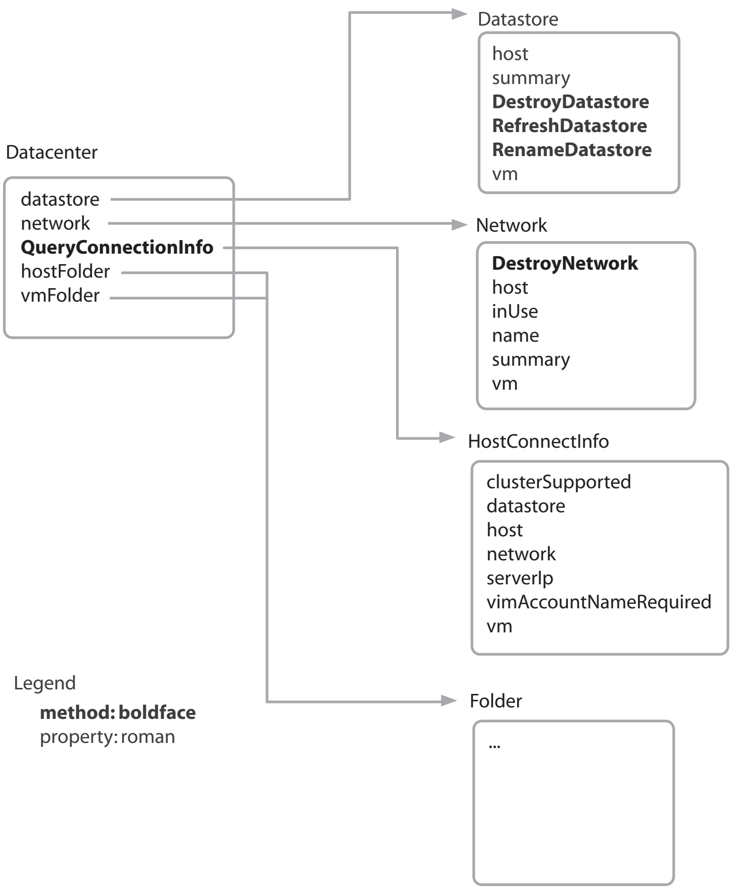
Folder Data Model



The preceding illustration shows the managed object type for organizing virtual machines and hosts in the inventory hierarchy. Folders may be nested to as many levels as you want, but the type of objects a folder may contain is determined by the values of the `childType` property. A folder may contain child objects only of types that match one of the values of its `childType` property, in addition to any nested folders it contains. For more information on the `childType` property, see the description of the Folder managed object in the *VMware Infrastructure SDK Reference Guide*.

A folder is roughly equivalent to a Farm Group or a Virtual Machine Group in earlier versions of the SDK. For more information on the differences between SDK versions, refer to the *VMware Infrastructure SDK Porting Guide*.

Datacenter Data Model



The preceding figure shows a Datacenter managed object type that groups virtual machines and host resources under a high-level organizational construct that represents a management unit. A datacenter roughly corresponds to a Farm in earlier versions of the SDK.

Key features of the Datacenter object type are:

- `datastore` — A list of references to physical file storage units.
- `hostFolder` — The root of the subtree of folders containing the physical compute resources belonging to the datacenter.

- `vmFolder` — The root of the subtree of folders containing the virtual machine objects belonging to the datacenter.
- `network` — A list of objects representing the networks, either virtual or physical, that connect hosts and virtual machines.

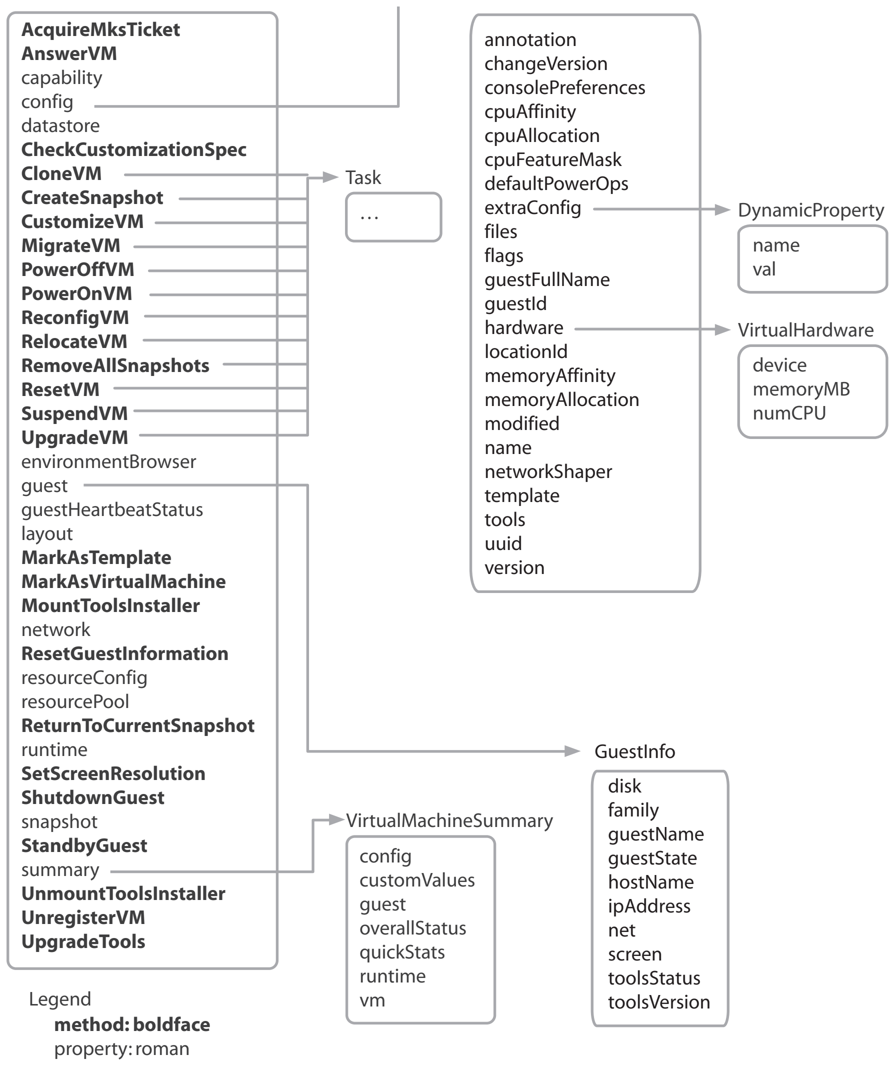
CHAPTER 3

Virtual Machine and Host Resources

This chapter discusses the Web service data models concerned with virtual machines and the physical resources that back them. This includes object types that implement resource management. The highlights of some of the major data types are given in the following sections:

- VirtualMachine Data Model on page 32 — Describes the data type used to model a virtual machine.
- HostSystem Data Model on page 37 — Describes the host machine configuration, including whether or not the machine is a non-uniform memory architecture (NUMA) machine.
- Datastore Data Model on page 41 — Describes the data types used to manage storage resources.
- ComputeResource Data Model on page 43 — Describes the basic model used to manage hosts or groups of hosts as resources for virtual machines.
- ResourcePool Data Model on page 45 — Describes an abstraction used to divide host resources among virtual machines.
- ClusterComputeResource Data Model on page 47 — Describes an extension of the ComputeResource, used for installations with VMware HA or VMware DRS.

VirtualMachine Data Model



The preceding illustration shows the managed object type used to represent a virtual machine. Most of the properties pertaining to a virtual machine are contained in sub-objects, such as VirtualMachineConfigInfo. Note that many of the operations defined on a virtual machine return a

Task object, which is described at TaskManager on page 57. Several other important aspects of the virtual machine object type are discussed in the following sections:

- VirtualMachineSummary on page 33 — An encapsulation of frequently-used basic properties.
- Information and Specification on page 34 — Specifying and viewing virtual machine configuration.
- Customization on page 35 — Customizing virtual machines during deployment.
- Power Operations on page 35 — Powering a virtual machine on and off.
- Backing Resources on page 36 — The physical computing resources used by virtual machines.

VirtualMachineSummary

The VirtualMachineSummary data object type, contained within the VirtualMachine managed object type, comprises a small number of properties that clients access most frequently. These include the current status of the virtual machine and some basic performance statistics.

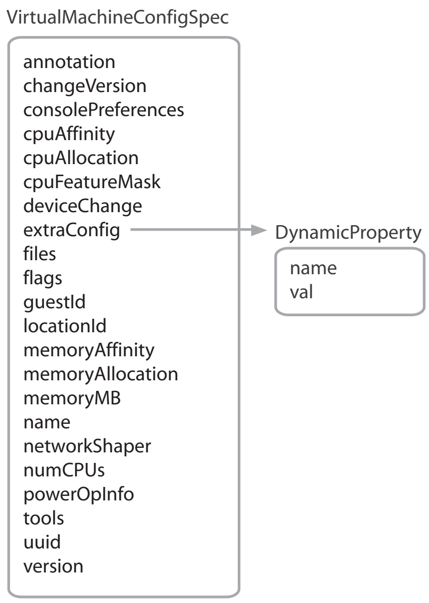
The SDK provides the VirtualMachineSummary object as a convenient way to access common properties without having to specify the properties individually to the property collector (explained at PropertyCollector on page 53). Summary objects exist for a number of other object types as well.

The VirtualMachineSummary object includes a reference to the containing VirtualMachine object. The reference is a convenience for clients who monitor only the VirtualMachineSummary (using the PropertyCollector). A client can use the reference to invoke an operation on the VirtualMachine object.

Information and Specification

The VirtualMachineConfigInfo sub-object contains information about the virtual machine's current configuration. For example, the `annotation` property is a user-supplied string that describes this particular virtual machine; the `files` property contains information about files associated with the virtual machine; and the `uuid` property is a unique BIOS identifier for the virtual machine.

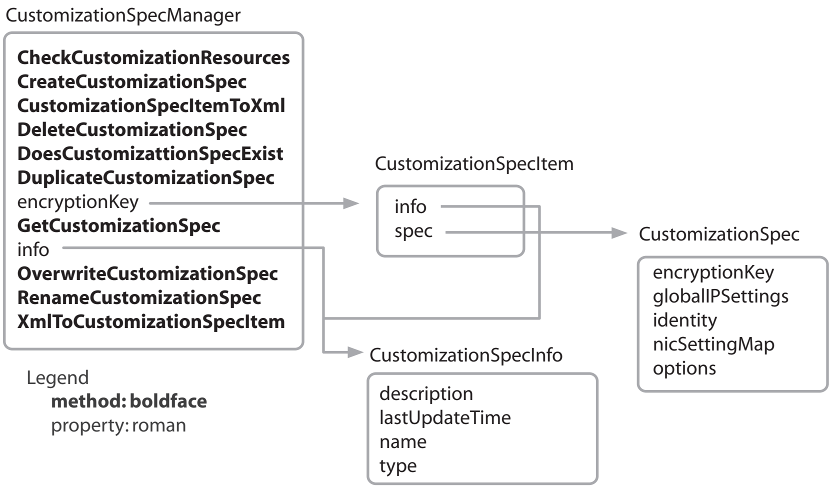
Some of the properties can be modified by the client using a VirtualMachineConfigSpec object, which contains many of the same properties present in VirtualMachineConfigInfo. The VirtualMachineConfigSpec object is passed by the client when creating or modifying virtual machines. The VirtualMachineConfigSpec object type is shown below



Note that the DynamicProperty data object type allows for user-defined properties.

Customization

The CustomizationSpec is used to specify modifications to be made to the guest operating system during deployment. The CustomizationSpecManager, reached by means of the service instance, allows the client to create, modify, and delete customization specifications. Specifications can be referenced by name when needed.



Power Operations

The SDK supports four hard power operations and two soft power operations. The hard power operations include:

- PowerOnVM_Task — Starts a virtual machine (equivalent to pressing the power button on powered-off hardware).
- PowerOffVM_Task — Stops a virtual machine (equivalent to pressing the power button on powered-on hardware).
- SuspendVM_Task — Suspends virtual machine execution, to resume later at the same point. There is no equivalent hardware operation.
- ResetVM_Task — Resets a virtual machine (equivalent to pressing the reset button on powered-on hardware).

The soft power operations include:

- ShutdownGuest — Initiates a shutdown of the guest operating system. Some guest operating systems complete their shutdown by powering off the virtual machine.

- StandbyGuest — Initiates a power-saving standby mode in the guest operating system; does not affect virtual machine power state.

Backing Resources

Virtual machine resources are backed by physical resources — CPUs, memory, and disk volumes, for example — on their hosts. The VirtualHardware and EnvironmentBrowser subobjects are key components of the mapping between virtual and physical resources.

The EnvironmentBrowser subobject identifies hardware resources, such as disk volumes and CPUs, available for use by the virtual machine. A client can use the environment browser to identify which backing resources to assign to the virtual machine.

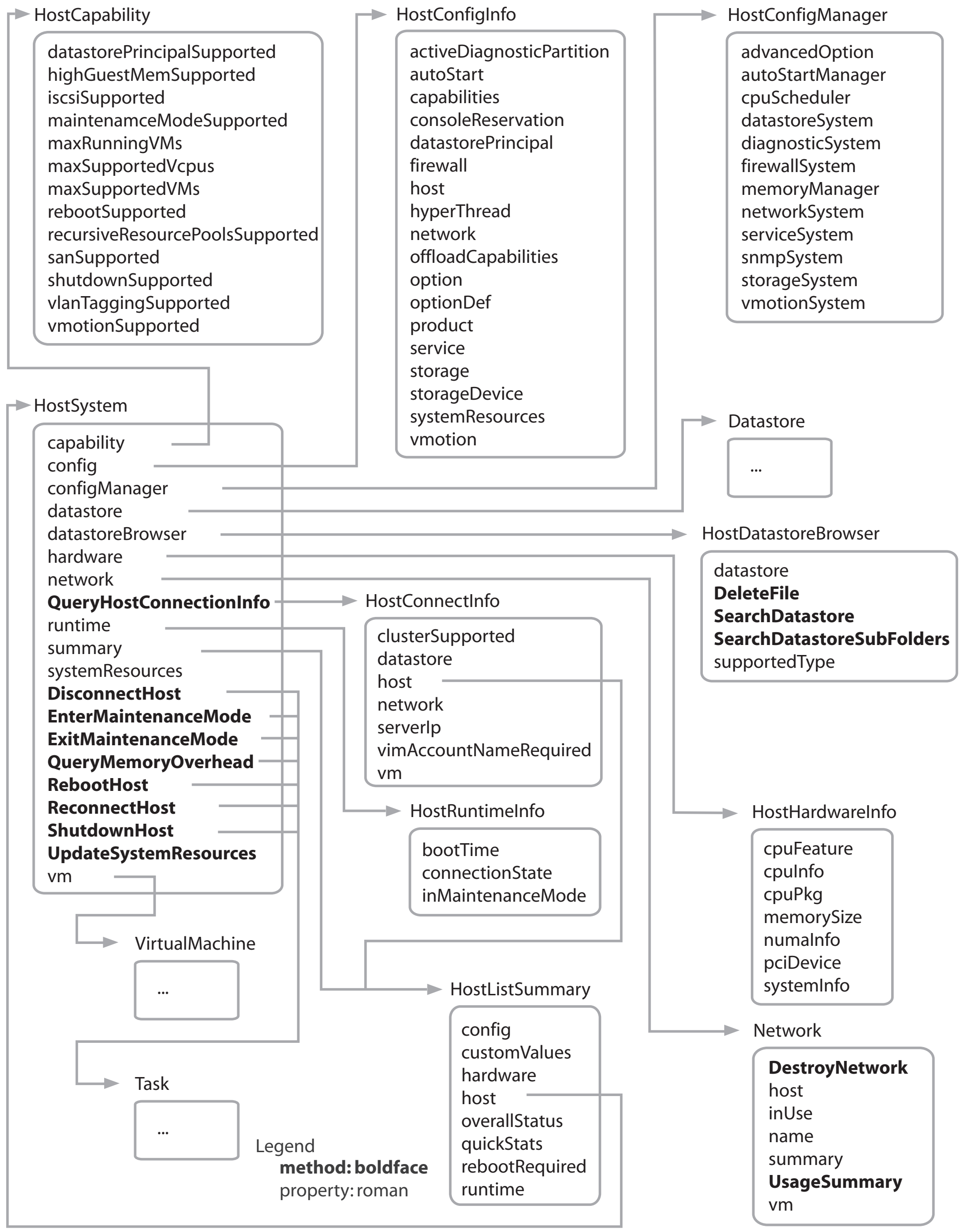
The VirtualHardware array lists the backing resources currently assigned to the virtual machine. For instance, the `device` property presents an array of virtual device objects, each of which identifies the physical device on the host that is responsible for performing the functions requested of the virtual device.

HostSystem Data Model

The following illustration shows the HostSystem data type exported by the Web service. This data type is used to manage the hardware subsystems that support a virtual machine.

Most of the properties pertaining to a host system are contained in subobjects, such as HostHardwareInfo. Note that many of the operations defined on a host return a Task object, which is described at TaskManager on page 57. Several other important aspects of the host system object type are discussed in the following sections:

- HostListSummary on page 39 — An encapsulation of frequently-used basic properties.
- HostCapability on page 39 — A description of the capabilities of the host hardware and software combination.
- HostConfigInfo on page 39 — The configuration information for the host.
- HostHardwareInfo on page 39 — The Hardware characteristics of the host.
- HostDatastoreBrowser on page 40 — The interface to access files in the datacenter.


HostCapability
datastorePrincipalSupported
highGuestMemSupported
iscsiSupported
maintenamceModeSupported
maxRunningVMs
maxSupportedVcpus
maxSupportedVMs
rebootSupported
recursiveResourcePoolsSupported
sanSupported
shutdownSupported
vlanTaggingSupported
vmotionSupported
HostConfigInfo
activeDiagnosticPartition
autoStart
capabilities
consoleReservation
datastorePrincipal
firewall
host
hyperThread
network
offloadCapabilities
option
optionDef
product
service
storage
storageDevice
systemResources
vmotion
HostConfigManager
advancedOption
autoStartManager
cpuScheduler
datastoreSystem
diagnosticSystem
firewallSystem
memoryManager
networkSystem
serviceSystem
snmpSystem
storageSystem
vmotionSystem
HostSystem
capability
config
configManager
datastore
datastoreBrowser
hardware
network
QueryHostConnectionInfo
runtime
summary
systemResources
DisconnectHost
EnterMaintenanceMode
ExitMaintenanceMode
QueryMemoryOverhead
RebootHost
ReconnectHost
ShutdownHost
UpdateSystemResources
vm
Datastore
...
HostDatastoreBrowser
datastore
DeleteFile
SearchDatastore
SearchDatastoreSubFolders
supportedType
HostConnectInfo
clusterSupported
datastore
host
network
serverIp
vimAccountNameRequired
vm
HostRuntimeInfo
bootTime
connectionState
inMaintenanceMode
HostHardwareInfo
cpuFeature
cpuInfo
cpuPkg
memorySize
numaInfo
pciDevice
systemInfo
VirtualMachine
...
HostListSummary
config
customValues
hardware
host
overallStatus
quickStats
rebootRequired
runtime
Network
DestroyNetwork
host
inUse
name
summary
UsageSummary
vm
Task
...
Legend
method: boldface
property: roman

HostListSummary

The HostListSummary data object type, contained within the HostSystem managed object type, comprises a small number of properties that clients access most frequently. These include the basic hardware configuration, current status of the host system, and some basic performance statistics.

The SDK provides the HostListSummary object as a convenient way to access common properties without having to specify the properties individually to the property collector. Summary objects exist for a number of other object types as well.

HostCapability

The HostCapability data object type, contained within the HostSystem managed object type, publishes certain features that depend on the specific hardware or software on the host. For instance, some hosts have VMotion capability; others do not. When a host lacks certain capabilities, the client may receive a NotSupported fault in response to SDK operations that require those capabilities.

The HostCapability data object type, like some others in the SDK, is extensible. When a client requests the capabilities of servers whose version is newer than the client, the server may reply with capabilities that are unknown to the client. These are present in the reply as dynamic properties, which are name-value pairs.

HostConfigInfo

The HostConfigInfo data object type, contained within the HostSystem managed object type, publishes a set of properties useful for describing the host configuration to a user. The configuration properties include storage configuration, hyperthreading, and network configuration.

The HostConfigInfo data object type also contains the HostNetCapability data object type, which publishes network-related capabilities of the host, such as NIC teaming. The HostNetCapability data object type, like the HostCapability data object type, is extensible with dynamic properties.

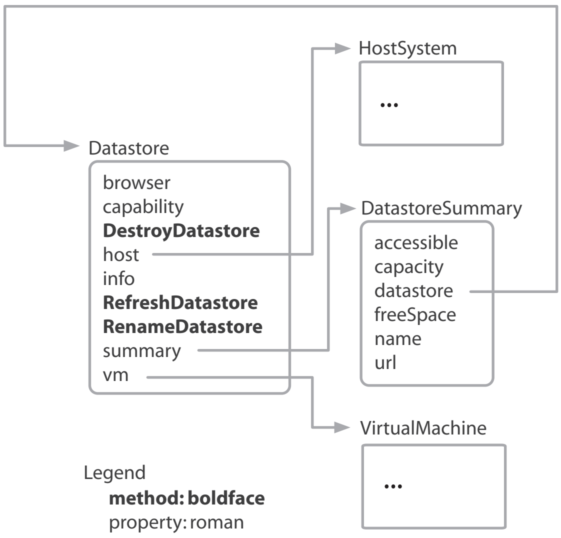
HostHardwareInfo

The HostHardwareInfo data object type, contained within the HostSystem managed object type, describes the specifics of the host system's hardware devices and characteristics. This is where you find CPU speeds, chip types, PCI devices, NUMA characteristics, and other hardware details not present in the HostListSummary.

HostDatastoreBrowser

The HostDatastoreBrowser managed object type, contained within the HostSystem managed object type, provides an interface to list files on a set of datastores. A client can present this information to a user who needs to decide where to put virtual machine files.

Datastore Data Model



The preceding illustration shows the Datastore data model exported by the Web service. This data type is used to catalog the storage devices available to host systems within a datacenter. All access to files or virtual machines in the SDK uses a path name prefixed with the name of the datastore (in square brackets) containing the files. For example, a virtual machine configuration file can be referenced as:

```
[data1]vmdir/linux/vm25/debian.vmx
```

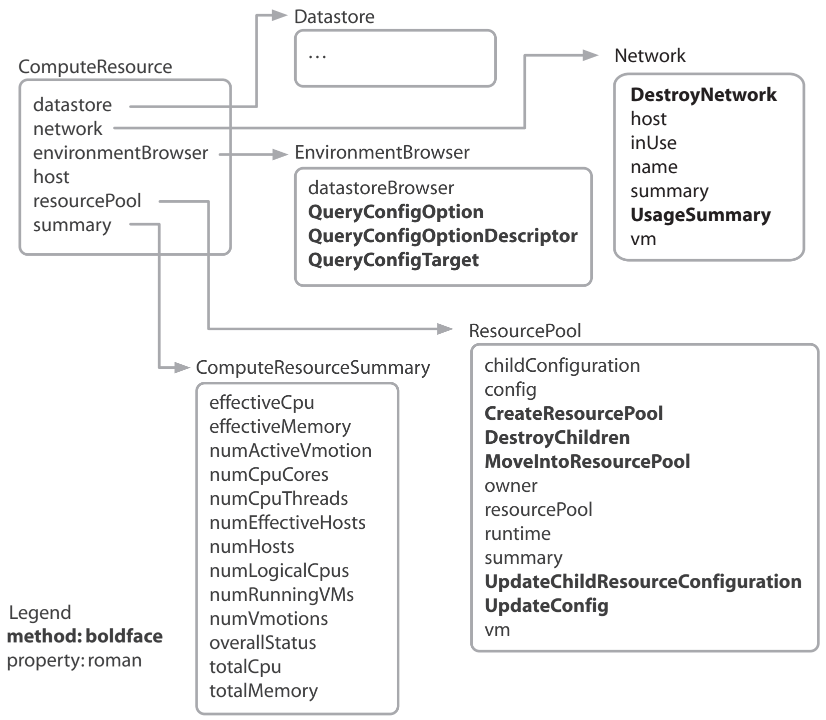
A Datastore object contains a DatastoreSummary object, similar in purpose to the summary objects contained by HostSystem and VirtualMachine objects. The DatastoreSummary object contains the capacity and free space information for the datastore. It also contains a reference to the full Datastore object that contains the DatastoreSummary object.

The Datastore, HostSystem, Datacenter, ComputeResource, and VirtualMachine data objects are linked as follows:

- Each HostSystem data object keeps a list of references to Datastore objects representing the datastores mounted by the host.
- Each Datastore data object keeps a list of references to HostSystem objects representing the hosts that have mounted that datastore.
- Each Datacenter object keeps a list of references to the datastores it manages.

- Each Datastore object keeps a list of references to VirtualMachine objects representing the virtual machines stored on the datastore.
- Each ComputeResource object keeps a list of references to datastore objects available to it.

ComputeResource Data Model



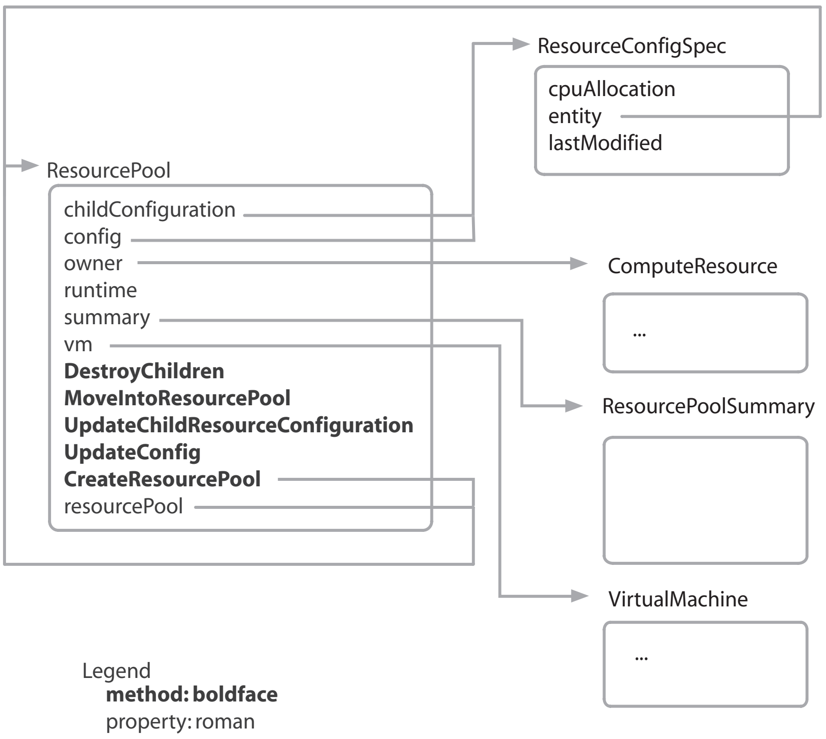
The preceding illustration shows the ComputeResource data model exported by the Web service. A ComputeResource managed object represents either a single host or a cluster of hosts available for backing virtual machines.

A ComputeResource contains:

- A list of hosts in the ComputeResource.
- A list of datastores available to hosts in the ComputeResource.
- A list of network objects available to hosts in the ComputeResource.
- A ComputeResourceSummary data object, containing current usage status and information on the resources available for virtual machines.
- An EnvironmentBrowser object that allows the client to browse files on datastores, HardwareInfo objects, and ConfigOption objects.

- A root ResourcePool for the ComputeResource. See ResourcePool Data Model on page 45.

ResourcePool Data Model



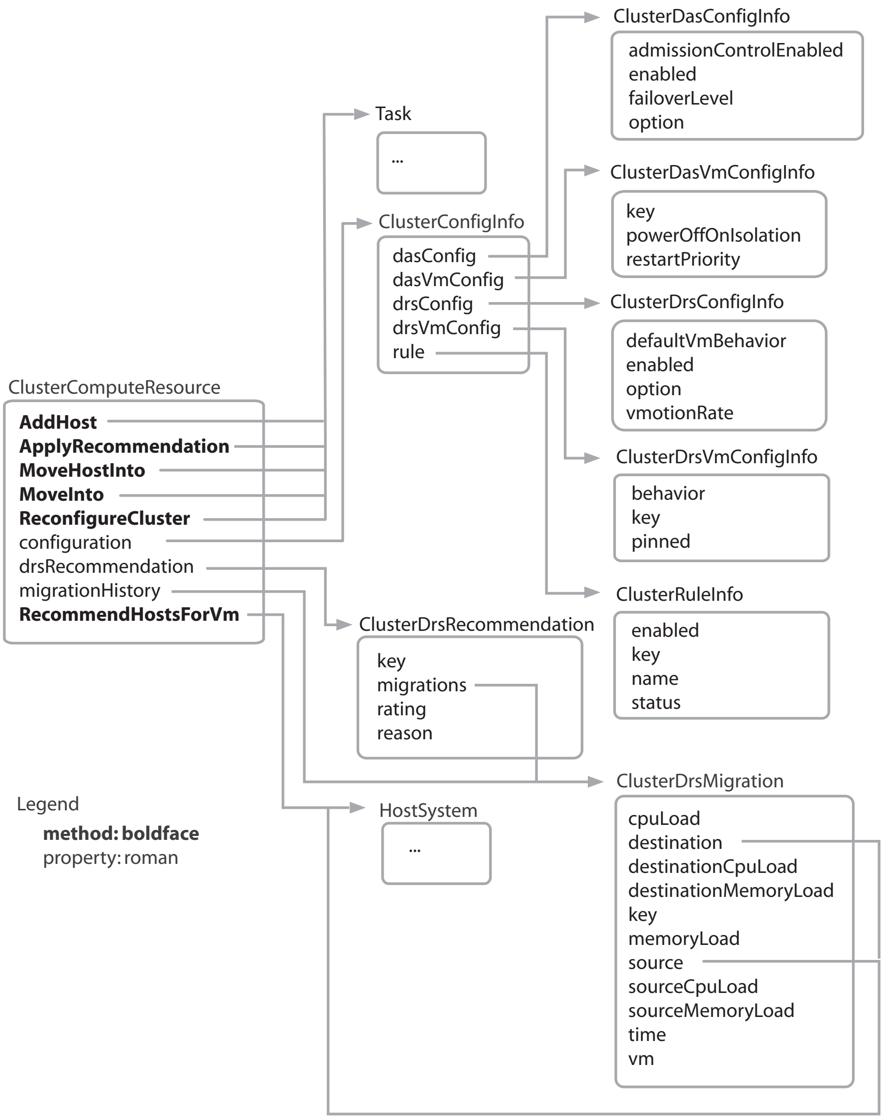
The preceding illustration shows the ResourcePool data model exported by the Virtual Infrastructure Web Service. A resource pool is a way to create arbitrary divisions of CPU and memory resources that can be made available to virtual machines. A virtual machine must be allocated to a resource pool before it can run.

Resource pools are configured with absolute values for minimum and maximum quantities of each resource. This allows a system administrator to guarantee service levels for a set of virtual machines using that resource. Resource pools can also be configured in terms of shares, which allow a system administrator to specify the relative importance of virtual machines using a resource pool.

A compute resource always has at least one resource pool associated with it. The "root" resource pool represents all of the CPU and memory resources available from the host or the aggregate of hosts in the compute resource. In some environments, the root pool can be divided between child resource pools, which can be subdivided to arbitrary depths, allowing the flexibility to implement complex resource allocation policies between competing needs.

Note: Virtual machines, as well as resource subdivisions, are known as “children” of a resource pool.

ClusterComputeResource Data Model



The preceding illustration shows the ClusterComputeResource data model exported by the Virtual Infrastructure Web Service. Some installations support extending the concept of a ComputeResource to include more than one host. When that happens, the ComputeResource managed object is extended to become a ClusterComputeResource managed object. ClusterComputeResource adds a number of operations and properties that support VMware HA and VMware DRS, including:

- AddHost_Task and MoveInto_Task are the operations for building a cluster by adding host machines.

  Note: There is no operation to remove hosts from a cluster; you can only remove the cluster as a whole, using the Destroy_Task operation. Then create a new cluster using the CreateCluster operation of the Folder managed object.
- The RecommendHostsForVm operation considers resource usage on the hosts in the cluster to arrive at a choice of which host is the best fit for powering on a virtual machine.
- The ApplyRecommendation operation migrates a set of virtual machines between hosts in the cluster to achieve more efficient resource usage. Recommendations are prepared in the background by the VMware DRS service and stored in the drsRecommendation property of ClusterComputeResource.
- Both VMware DRS and VMware HA can be configured by the user. To make changes to the cluster configuration, you invoke the ReconfigureCluster_Task operation, passing a ClusterConfigSpec data object that describes the changes.
- The current configuration of the ClusterComputeResource (for both VMware DRS and VMware HA) is available in the ClusterConfigInfo property.

CHAPTER 4

Managing and Monitoring

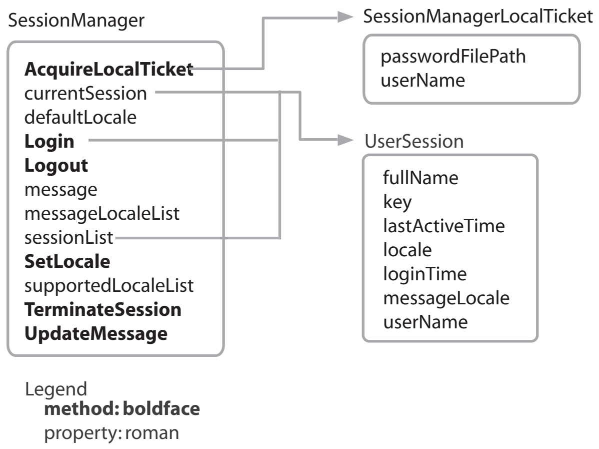
The following sections discuss some of the Virtual Infrastructure Web Service objects concerned with managing the VMware infrastructure environment. The details of each of the data types, along with the descriptions of key fields, are given in the following sections. The data types are organized in the following major sections:

- Sessions, Authorization, and Permissions on page 50 — Describes the object types exported by the Web service to handle security services.
- PropertyCollector on page 53 — Retrieves and monitors properties.
- EventManager on page 55 — Tracks the history of significant state changes to managed entities.
- TaskManager on page 57 — Describes the object types used to track asynchronous operations.
- Scheduled Tasks on page 59 — Describes the object types used to manage scheduled future operations.
- Alarms on page 60 — Describes the object types used to manage conditional operations.
- PerformanceManager on page 62 — Describes the object types used to collect performance metrics.

Sessions, Authorization, and Permissions

Security for a computing installation must satisfy two basic needs: identification and access control. In the SDK, identification is provided by the SessionManager object. Access control is provided by a system of permissions, described at Permissions System on page 50.

SessionManager



The preceding illustration shows the SessionManager object. The SessionManager allows a client to create a session, which identifies a particular user. A session also defines the lifetime and visibility of certain objects, like PropertyCollector filters. Session-specific objects are not visible outside the session in which they were created. When the session terminates, all session-specific objects are destroyed.

SessionManager has Login and Logout operations that allow users to create and end sessions. Sessions can also be ended by the system administrator, using the TerminateSession operation.

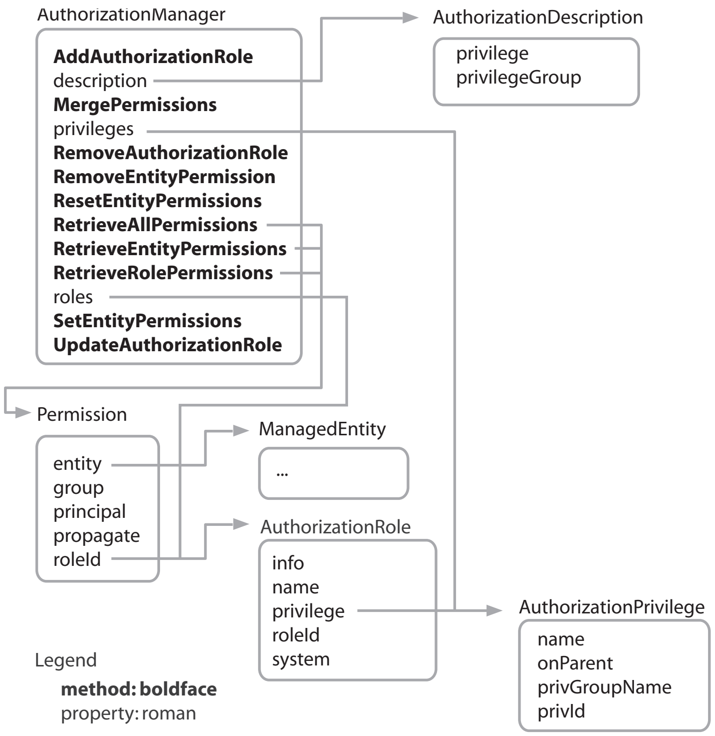
The system administrator may choose to allow local sessions, which accommodate users already logged on to a host. A client running on behalf of such a local user may invoke the AcquireLocalTicket operation, which returns a one-time user name and password that allows the client to log on without asking the user for a password. This feature is useful for utilities that run at a host-local terminal window or at the VMware Service Console.

Permissions System

When a client has a session in progress, the session associates the client with a specific user. The SDK can validate that user against the access control in effect for operations on managed objects.

Access control attached to an object allows the SDK to check that the user invoking the operation is granted the privilege to perform the operation on that object. If the user has a valid permission to access the managed object in the desired way, the operation proceeds; otherwise, it throws a NoPermission fault or a ManagedObjectNotFound fault.

The access control model for the SDK is based on the Permissions object. Every managed entity has one or more Permissions objects attached to it. Permissions may attach directly to a managed entity, or may be inherited from a parent entity in the inventory tree.



The above illustration shows the Permission data model. A Permission object associates a user with a privilege to perform an operation on the object to which the Permission object is attached. A Permission object must contain three things:

- A managed entity reference
- A user name or group name
- A role

The managed entity reference identifies the managed object to which the permission applies. However, some operations do not operate directly on a single managed object. In those cases, the permissions on the ServiceInstance object (at the root of the inventory hierarchy) apply.

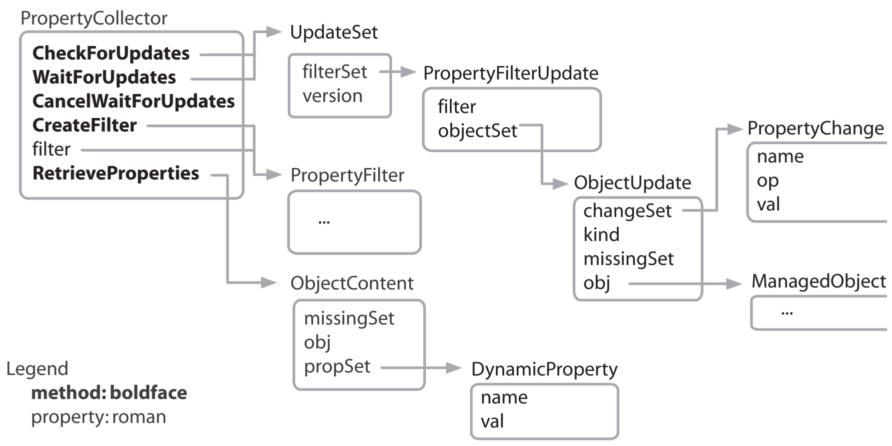
The user name of a Permission object must match the user name of the current session in order for the client to obtain access to the managed entity. Or, if the permission contains a group name, the user must be a member of the group to obtain access to the managed entity.

A role is a collection of privileges. For convenience in managing privileges, the SDK provides pre-defined roles such as “Administrator” and “Virtual Machine Power User” that group privileges into collections commonly needed by users performing functions appropriate to the role name. Clients (if they have appropriate permissions) may use the AuthorizationManager to define new roles for their own installations.

The AuthorizationManager object provides methods to create, alter, and delete Permission objects and roles. AuthorizationManager also allows a client to view the list of privileges (which is fixed) and the list of roles that are currently defined.

For installations without VirtualCenter, the UserDirectory object provides a way for a client to browse the set of users and groups on the host. UserDirectory has only one method, RetrieveUserGroups, which is used to search for either user names or group names. The matches can be exact or partial (substring) matches, or the client can request the list of all users on the machine.

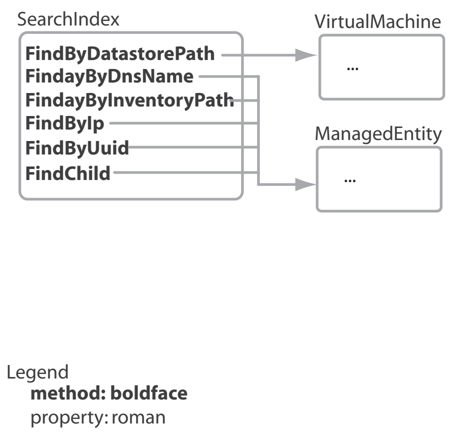
PropertyCollector



The above illustration shows the PropertyCollector data model. Property collectors are the primary means of retrieving properties of managed objects. The ServiceInstance provides a PropertyCollector to each client session.

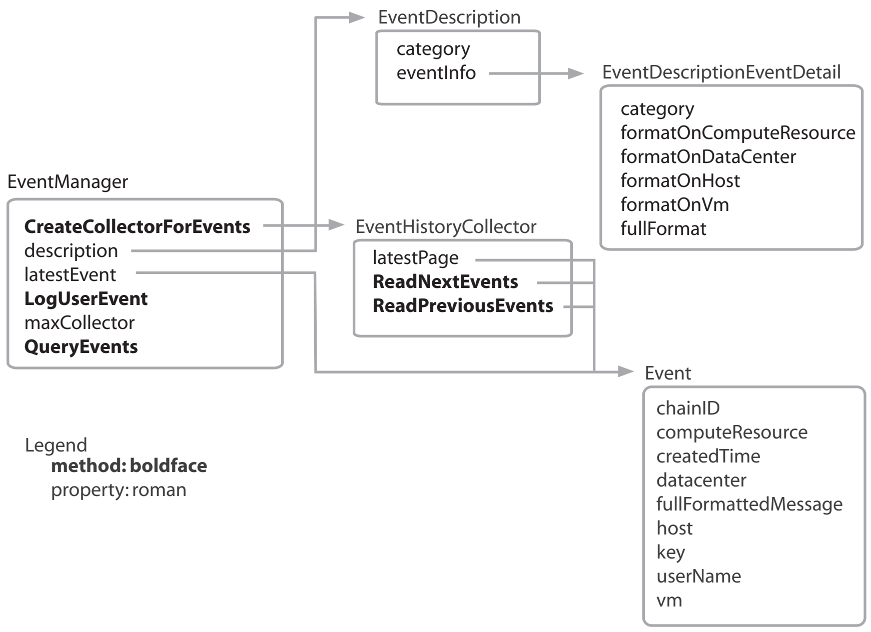
Clients can use the PropertyCollector to create any number of session-specific property filters, which last until destroyed or until the client's session terminates. While the property filters exist, the client can use several methods to retrieve the desired properties. More information on using the PropertyCollector is available in the PropertyCollector Tutorial on page 75 and in the *VMware Infrastructure SDK Programming Guide*.

SearchIndex



The above illustration shows the SearchIndex data model. The SearchIndex provides the primary means of retrieving managed entities when you can identify the entity by means of specific property values such as inventory path, datastrore path, DNS name, UUID, IP address, or as the child of a parent entity. See the *VMware Infrastructure SDK Programming Guide* for more information.

EventManager



The preceding illustration shows the EventManager and Event data types exported by the Web service. Event objects record significant state changes of managed entities, such as:

- Powering a virtual machine on or off.
- Deploying a new virtual machine.
- Reconfiguring a compute resource.
- Adding a new host to VirtualCenter.

The EventManager is accessed from the ServiceInstance. The EventManager contains references to the most recent events. A more general way to view past events is by using an event history collector.

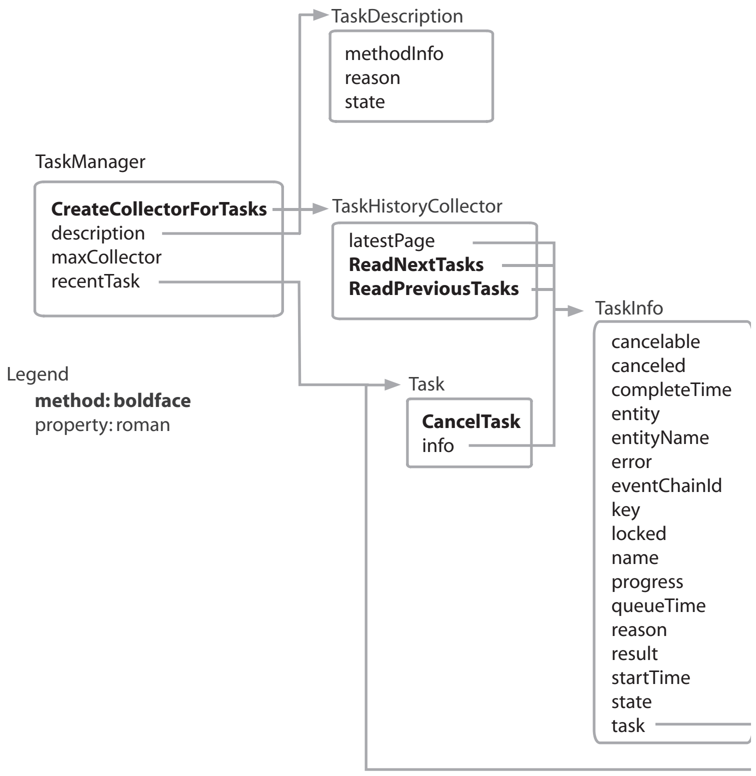
An event history collector is created to filter events from the entire database of past events. The specification for creating the event filter can select events during a specific time range, belonging to a particular user, or associated with certain other objects, such as alarms. See Alarms on page 60.

Event objects have specialized content that depends on the source of the event. For example, a fault event contains a reference to the fault that caused it; a scheduled task event contains a reference to the responsible scheduled task and the managed entity associated with the scheduled task. See Scheduled Tasks on page 59.

All event objects have properties to connect them to an associated host, virtual machine, datacenter, and compute resource. They also contain a time stamp, an event ID, and a formatted message describing the event. Message strings can be locale-specific. The locale-specific strings used to create events are stored in the EventManager's description array.

The EventManager also lets you log a user-defined event. You can use the EventManager to save historical information to supplement the predefined events, or to provide markers when browsing event history.

TaskManager



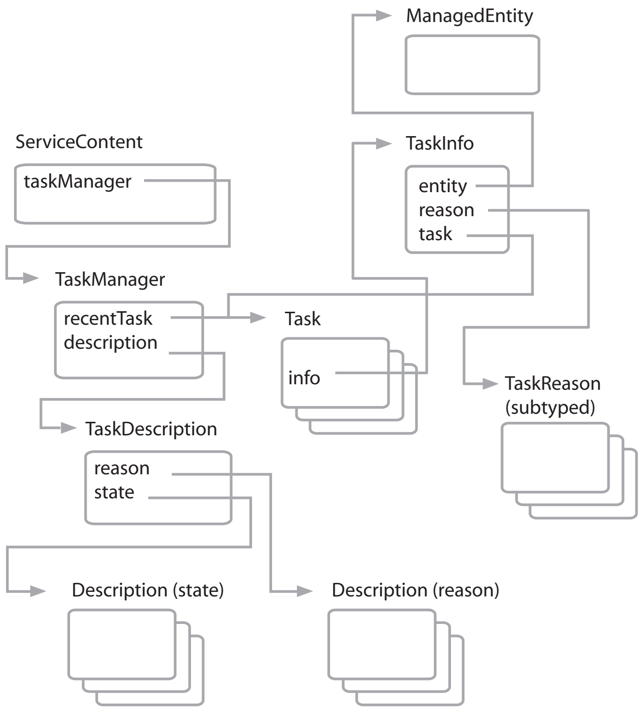
The preceding illustration shows the TaskManager and Task data types exported by the Web service. Task objects are used to track operations that do not complete immediately, such as:

- Shutting down a virtual machine.
- Migrating a virtual machine.
- Sending an email message as a result of an alarm. See Alarms on page 60.

Clients may start a task and check the status. Tasks may be automatically started based on a client operation request if the task is expected to take a long time. Alternatively, clients may also schedule tasks. See Scheduled Tasks on page 59.

The TaskManager is accessed from the ServiceInstance. The TaskManager contains references to the most recent tasks. The TaskManager also manages task descriptions, in a user-readable form that can be localized.

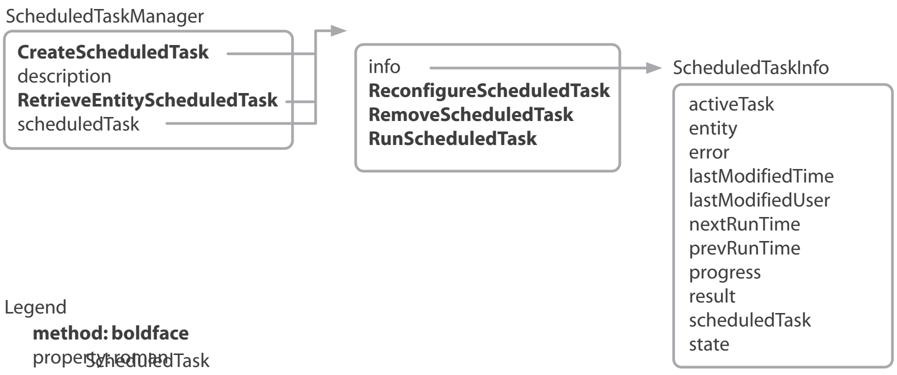
The following illustration shows the connections between several objects related to tasks.



All task history is kept in a database. You can view older tasks by creating a task history collector to select the desired tasks from the database. The specification for creating the task filter can select tasks during a specific time range, belonging to a particular user, or in a particular state. You can also limit the selection to specific alarms, entities, or scheduled tasks.

Task objects keep status information such as the task's start time, current progress (as percentage complete), state (such as queued or running), and completion status. All this information is preserved in the historical record of the task. However, the managed object for the task is needed only while the task is still active and referenced by the TaskManager. Thus, the Task managed object is detached from its TaskInfo data object, so the Task object can be discarded when it's no longer needed.

Scheduled Tasks



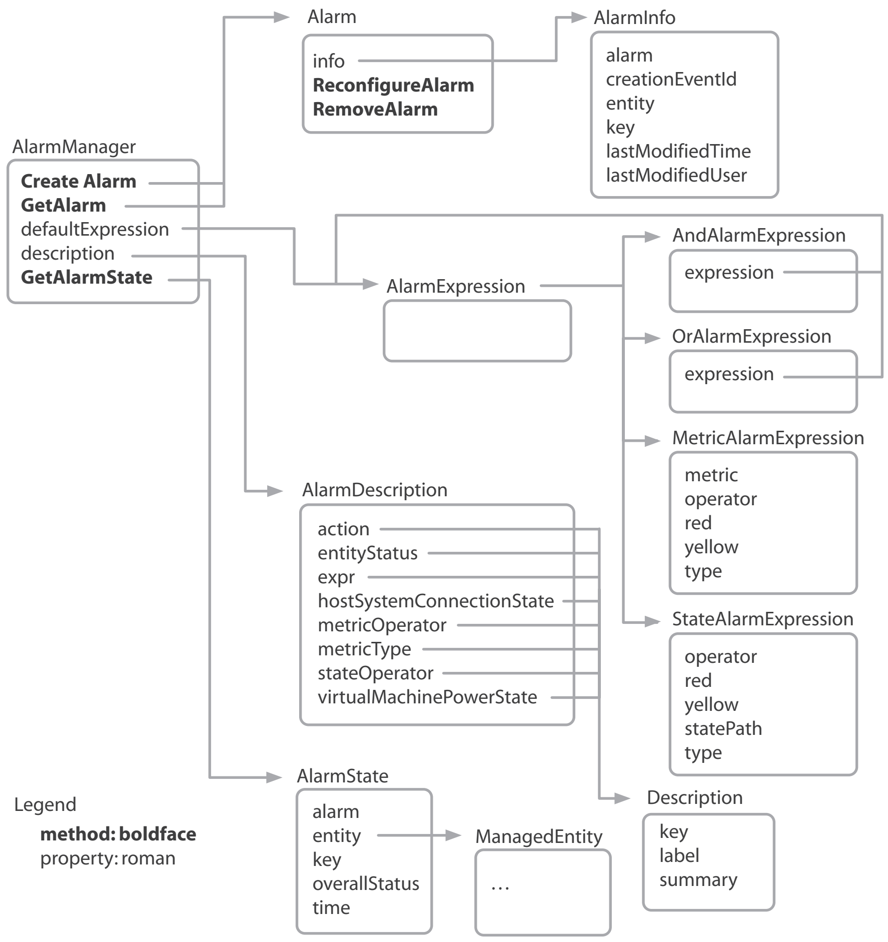
The preceding illustration shows the ScheduledTaskManager and ScheduledTask data types exported by the Web service. Scheduled tasks are used to configure the VirtualCenter server to invoke an SDK operation at a future time.

The time schedules available for scheduled tasks are:

- After starting up the VirtualCenter server.
- At a specified time.
- At hourly, daily, weekly, or monthly intervals.

The run status and progress of a scheduled task are tracked in the associated ScheduledTaskInfo object. While the scheduled task is running, a reference to a Task object is included. Task objects are explained at TaskManager on page 57.

Alarms



The preceding illustration shows the AlarmManager and Alarm object types exported by the Virtual Infrastructure Web Service. Alarms allow you to trigger actions conditionally. The AlarmExpression data object type provides a way to specify complex conditions for triggering alarms.

The actions that can be triggered by alarms are:

- Invoking an operation with the SDK.
- Running a shell script on the VirtualCenter server.
- Sending an email message.
- Sending an SNMP trap.

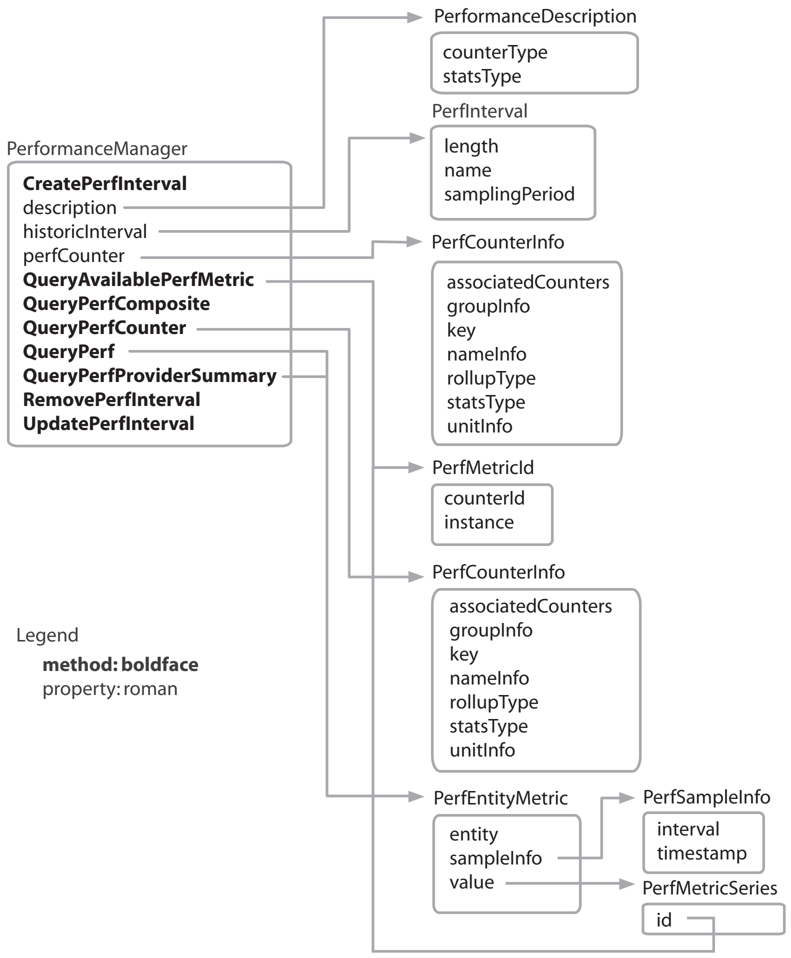
The conditions that can trigger alarms include:

- Power state of a virtual machine.
- Network connection state of a host.
- Resource usage metrics that exceed a defined limit.

Alarm expressions allow you to define conditions to monitor, and to combine conditions using boolean logic. This flexibility allows you to configure VirtualCenter to monitor your datacenter in whatever way you choose.

When an alarm triggers, it produces an event for the event history database. The action triggered by the alarm may also leave an event history.

PerformanceManager



The preceding illustration shows the PerformanceManager managed object type exported by the Virtual Infrastructure Web Service. The PerformanceManager allows you to query raw statistics or metadata about performance counters, and to configure statistics settings.

There are two basic ways to query statistics:

- QueryPerf accesses metrics for a list of managed entities. Use this when you want to monitor specific entities that provide performance data.
- QueryPerfComposite accesses metrics or a single managed entity and all of its child entities. Use this when you are interested in a group of related entities, such as all virtual machines that belong to a given resource pool.

When you invoke QueryPerf or QueryPerfComposite, you can choose a PerfInterval to suit your reporting need. PerfIntervals are specified by their sampling periods, called "interval IDs." Or you can choose the performance metric provider's raw sampling period, called its "refresh rate," for data within the past hour only. When you use the provider's refresh rate, you access an implied PerfInterval with a fixed sample period, and a fixed retention length of one hour.

Data older than one hour are consolidated into the shortest defined PerfInterval and stored in the historical database. Historical performance data is kept according to the defined PerfIntervals. When the retention length expires, older data is summarized into the next larger PerfInterval. For more on PerfIntervals, see Configuring PerformanceManager Settings on page 100.

Performance counters are available for a wide variety of system performance characteristics, and in a number of different forms. The rest of this section summarizes the capabilities of the SDK performance counters.

Objects of statistics include:

- CPU usage, with breakdown into system processes, wait time, ready time, and so on.
- Memory usage, with breakdown into active, shared, heap, swap, and so on.
- Network usage, with breakdown into statistics for transmitted and received packets.
- Disk usage, with breakdown into reads and writes.
- System uptime and heartbeat statistics.

Performance measurements can be collected from the following managed objects:

- Hosts
- Virtual machines
- Resource pools

Statistics types are of three kinds:

- Absolute value — A raw measurement of some quantity.
- Delta — A measurement of the difference between successive raw measurements.
- Rate — A measurement of rate as a delta value normalized to account for differing statistics intervals.

Counter types:

- Average
- Minimum
- Maximum
- Latest
- Summation

For more information on using the PerformanceManager, refer to Using the PerformanceManager on page 99.

CHAPTER 5

Glossary

Access Control List
An access control list is a set of <group, rights> pairs that defines the access rights for an object.

Alarm
An entity that monitors one or more properties of a virtual machine, such as CPU load. Alarms use green, red, and yellow color coding to issue notifications as directed by the configurable alarm definition.

Allocated disk
A type of virtual disk in which all disk space for the virtual machine is allocated at the time the disk is created. This is the default type of virtual disk created by VirtualCenter.

Apache Axis
Apache Axis is a more modular, more flexible, and higher-performing SOAP implementation designed around a streaming model and is a successor to Apache SOAP 2.0.

API
Application programming interface. A specified set of functions that allows one to access a module or service programmatically.

Authorization Role
A set of privileges grouped for convenient identification under names such as "Administrator".

Child
A managed entity grouped by a Folder object or other managed entity.

Clone
The process of making a copy of a virtual machine. This process includes the option to customize the guest operating system of the new virtual machine. When a clone is created, VirtualCenter provides an option to customize the guest operating system of that virtual machine. Clones can be stored on any host within the same farm as the original virtual machine.

Cluster compute resource
An extended ComputeResource that represents a cluster of hosts available for backing virtual machines.

ComputeResource
A managed object that represents either a standalone host or a cluster of hosts available for backing virtual machines.

Configuration
See Virtual machine configuration file.

Customization
The process of customizing a guest operating system in a virtual machine as it is being deployed from a template or cloned from another existing virtual machine. Customization options include changing the new virtual machine identification and network information.

Datacenter
An entity used to group virtualmachine and host entities.

Data object
A composite object that is passed by value between the client and the Web service. A data object has properties associated with it but does not have any operations of its own. See also ManagedObject on page 68.

Datastore
The storage location for the virtual machine files. This may be a physical disk, a RAID, a SAN, or a partition on any of these.

Event
An action that is of interest to VirtualCenter. Each event triggers an event message. Event messages are archived in the VirtualCenter database.

Farm
A structure (in VirtualCenter 1) under which hosts and their associated virtual machines are added to the VirtualCenter server.

Fault
A data object containing information about an exceptional condition encountered by an operation.

Folder
A managed entity used to group other managed entities. The contents of a group are child entities with respect to the Folder object. See Child on page 66.

A Folder can contain different child entities depending on its location in the ManagedEntity inventory. The childType property of the Folder determines the managed entities that can be grouped within the Folder object at any particular level. See the Folder managed object in the *VMware Infrastructure SDK Reference Guide* for more information.

Group
A group is a set of users and groups.

Guest operating system
An operating system that runs inside a virtual machine.

Host
The physical computer on which the virtual machines managed by VirtualCenter are installed.

Host agent
Installed on a virtual machine host, it performs actions on behalf of a remote client.

IDL
Interface definition language. A human-readable syntax used to specify an API. The IDL may be compiled into stubs on a client machine. See Stub on page 70.

Inventory
A hierarchical structure used by the VirtualCenter server or the host agent to organize managed entities.

JAX-RPC
JAX-RPC (or Java API for XML-based RPC) is an API that builds Web services and clients that use remote procedure calls (RPC) and XML. Remote procedure calls and responses are transmitted as SOAP messages (XML files) over HTTP (the Web).

ManagedEntity

A managed object that is present in the inventory. See Inventory on page 67.

ManagedObject

A composite object that resides on a server and is passed between the client and the Web service only by reference. A managed object has operations associated with it, but might or might not have properties. See also Data object on page 66.

Message

A message is a data element that is used by an operation to carry data. It lists the data types exchanged between the Web service and the client.

Microsoft SOAP Toolkit

The Microsoft Simple Object Access Protocol (SOAP) Toolkit 2.0 comprises a client-side component, a server-side component and other components that construct, transmit, read, and process SOAP messages. This toolkit also provides additional tools that simplify application development.

Migration

Moving a virtual machine between hosts. Unless VMotion is used, the virtual machine must be powered off when you migrate it. See VMotion on page 72.

Nonpersistent mode

If you configure a virtual disk as an independent disk in nonpersistent mode, all disk writes issued by software running inside a virtual machine with a disk in nonpersistent mode appear to be written to disk but are in fact discarded after the virtual machine is powered off. As a result, a virtual disk or physical disk in independent-nonpersistent mode is not modified by activity in the virtual machine.

See also Persistent mode on page 68.

Operation

A function performed for a client by the Web service.

Permission

A data object comprising an authorization role, a user or group name, and a managed entity reference. Allows a specified user to access the entity with any of the privileges pertaining to the role. See also Authorization Role on page 66.

Persistent mode

If you configure a virtual disk as an independent disk in persistent mode, all disk writes issued by software running inside a virtual machine are immediately and permanently written to the virtual disk in persistent mode. As a result, a virtual disk or physical disk in independent-persistent mode behaves like a conventional disk drive on a physical computer.

See also Nonpersistent mode on page 68.

Privilege
Authorization to perform a specific action or set of actions on a managed entity or group of managed entities. Privileges are grouped together under an authorization role. See also Authorization Role on page 66.

Property
An attribute of a managed object or data object. A property may be a nested data object or a managed object reference.

PropertyCollector
A managed object used to control the reporting of managed object properties. The primary means of monitoring status on host machines.

Resource pool
A division of computing resources used to manage allocations between virtual machines. Represented by the ResourcePool managed object.

Resume
Return a virtual machine to operation from its suspended state. When you resume a suspended virtual machine, all applications are in the same state they were when the virtual machine was suspended.

See also Suspend on page 70.

Role
See Authorization Role on page 66.

Scheduled task
A VirtualCenter activity that is configured to occur at designated times. Represented by the ScheduledTask managed object.

SDK
Software developer kit. It comprises some or all of: an API, an IDL, client stubs, sample code, and manuals.

Service console
The command line interface for an ESX Server system. The service console lets administrators configure the ESX Server system. You can open the service console directly on an ESX Server

system. If the ESX Server system's configuration allows Telnet or SSH connections, you can also connect remotely to the service console.

Service instance
The managed entity at the root of the inventory. Clients must access the service instance to begin a session.

Snapshot
A snapshot preserves the virtual machine just as it was when you took the snapshot — the state of the data on all the virtual machine's disks and whether the virtual machine was powered on, powered off, or suspended. SDK lets you take a snapshot of a virtual machine at any time and revert to that snapshot at any time. You can take a snapshot when a virtual machine is powered on, powered off, or suspended.

SOAP
Simple Object Access Protocol (SOAP) is an XML-based communication protocol and encoding format for inter-application communication in a decentralized, distributed environment. It specifies a standard way to encode parameters and return values in XML, and standard ways to pass them over common network protocols like HTTP (Web) and SMTP (email). SOAP provides an open methodology for application-to-application communication (Web services).

SOAP::LITE
SOAP::Lite is an open source collection of Perl modules that provides a simple and lightweight interface to SOAP.

Stub
A procedure that implements the client side of a remote procedure call. The client calls the stub to perform a task, and the stub then transmits parameters over the network to the server and returns the results to the client.

Suspend
Save the current state of a running virtual machine. To return a suspended virtual machine to operation, use the resume feature.

See also Resume.

To return a suspended virtual machine to operation, use the resume feature.s

Task
A managed object representing the state of a long-running operation.

TCP
Transmission Control Protocol. A reliable transfer protocol between two endpoints on a network. TCP is built on top of IP.

Template
A master image of a virtual machine. This typically includes a specified operating system and a configuration that provides virtual counterparts to hardware components. Optionally, a template can include an installed guest operating system and a set of applications. Templates are used by VirtualCenter to create new virtual machines.

UUID
Universally Unique Identifier (ID). This is a 128-bit number represented in hexadecimal (HEX) format when passed as a string; for example, `f81d4fae-7dec-11d0-a765-00a0c91e6bf6`.

User
A user is a principal known to the system.

VirtualCenter
VMware VirtualCenter is a software solution for deploying and managing virtual machines across the data center.

VirtualCenter agent
Installed on each virtual machine host, it coordinates the actions received from the VirtualCenter server.

VirtualCenter database
A persistent storage area for maintaining status of each virtual machine and user managed in the VirtualCenter environment. This is located on the same machine as the VirtualCenter server.

VirtualCenter server
A service that acts as a central administrator for VMware servers connected on a network. This service directs actions upon the virtual machines and the virtual machine hosts. VirtualCenter server is the central working core of VirtualCenter.

Virtual disk
A virtual disk is a file or set of files that appear as a physical disk drive to a guest operating system. These files can be on the host machine or on a remote file system.

Virtual machine
A virtualized x86 PC environment in which a guest operating system and associated application software can run. Multiple virtual machines can operate on the same host system concurrently.

Virtual machine configuration
The specification of what virtual devices (such as disks and memory) are present in a virtual machine and how they are mapped to host files and devices.

Virtual machine configuration file
A file containing a virtual machine configuration. It is created when you create the virtual machine. It is used by SDK to identify and run a specific virtual machine.

VMFS
See VMware ESX Server file system on page 72.

VMODL
The interface definition language used in the VMware Infrastructure SDK.

VMotion
The feature that lets you move a virtual machine from one ESX Server system to another without interrupting service. VMotion requires licensing on both the source and target hosts. VMotion is activated by the VirtualCenter agent. The VirtualCenter server centrally coordinates all VMotion activities.

VMware DRS
VMware DRS is a feature that intelligently and continuously balances virtual machine workloads across your ESX Server hosts. VMware DRS detects when virtual machine activity saturates an ESX Server host and triggers automated VMotion live migrations, moving running virtual machines to other ESX Server nodes so that all resource commitments are met.

VMware ESX Server file system
A file system that is optimized for storing virtual machines. Also referred to as VMFS.

One VMFS partition is supported per SCSI storage device or SAN. Each version of ESX Server uses a corresponding version of VMFS. For example, VMFS3 was introduced with ESX Server 3.

VMware HA
VMware HA is a feature that detects failed virtual machines and automatically restarts them on alternate ESX Server hosts. VMware HA selects a failover host that can honor the virtual machine's resource allocations so that service level guarantees remain intact.

VMware Infrastructure
A system of hosts, agents, and clients that communicate to deploy and operate virtual machines. The total VMware solution to managing a data center. See Host on page 67, Host agent on page 67, VirtualCenter on page 71.

VMware Tools
A suite of utilities and drivers that enhances the performance and functionality of your guest operating system. Key features of VMware Tools include some or all of the following, depending on your guest operating system: an SVGA driver, a mouse driver, the VMware guest operating system service, the VMware Tools control panel, time synchronization with the host, VMware Tools scripts, and connecting and disconnecting devices while the virtual machine is running.

Web service
A programming interface based on SOAP and WSDL.

WSDL
Web Services Description Language, an XML-based language used to describe a Web service's capabilities and to provide a way for individuals and businesses to access those services electronically.

XML
Extensible Markup Language, a text-based markup language that is especially designed for Web documents.

APPENDIX A

PropertyCollector Tutorial

This appendix contains a tutorial on using the PropertyCollector. The PropertyCollector is central to retrieving and monitoring the contents of objects managed by the Web service. The PropertyCollector provides a great deal of power and flexibility in specifying the properties returned by the Web service.

At the same time, it can be difficult at first to grasp the correct use of the PropertyCollector. This tutorial provides some help in understanding the structures that work together to specify how properties are to be collected.

This tutorial contains the following sections:

Using the PropertyCollector

How you choose to use the PropertyCollector depends on your needs. The operations available to fetch property values are:

- RetrieveProperties on page 76
- CheckForUpdates on page 76
- WaitForUpdates on page 76

All the operations require the client to specify a property filter to select the objects and properties to be returned to the client. For information on how to specify property filters, see PropertyFilter on page 81.

RetrieveProperties

The simplest use of the PropertyCollector is to invoke the RetrieveProperties operation. This operation requires a PropertyFilterSpec object, but not a complete PropertyFilter object. Using RetrieveProperties is like creating a temporary filter that lasts only for the duration of the operation. See PropertyFilter on page 81 for details on how to use PropertyFilterSpec.

Often, a single use of RetrieveProperties is not enough. If the values of the properties change frequently, and the client needs to refresh the information, the changes need to be transmitted to the client continually.

CheckForUpdates

You can invoke RetrieveProperties repeatedly to get the most recent values of the desired properties, but this is the least efficient way to get updated information. Rather than fetch all the properties each time, you can reduce network traffic by invoking CheckForUpdates or WaitForUpdates. These operations are designed to return only the updated values, using network bandwidth more efficiently.

CheckForUpdates can be invoked periodically to poll for updates to the values of properties already stored at the client. This is similar to invoking RetrieveProperties periodically, except that CheckForUpdates returns only the properties that have changed since the last transmission, whereas RetrieveProperties retrieves all the properties each time, regardless of whether they changed. If you invoke CheckForUpdates when there are no changes since the last invocation, CheckForUpdates returns an empty change list.

WaitForUpdates

The most efficient use of network resources is the WaitForUpdates operation, which implements a form of change notification rather than the polling approach of CheckForUpdates. Rather than returning an empty change list, WaitForUpdates does not complete until an update happens. Th

WaitForUpdates operation blocks a processing thread in the client, but can be canceled from another thread with the CancelWaitForUpdates operation.

To use the CheckForUpdates or WaitForUpdates operations, you must first create a PropertyFilter object with the CreateFilter operation. The PropertyFilter object is created from a PropertyFilterSpec object, which specifies the objects of interest and the properties to be collected from them. Refer to PropertyFilter on page 81 for details on creating a PropertyFilterSpec object.

Nested Properties and Property Paths

When you specify a property with a property path string in the form `a.b`, you are specifying a property named `a` which is a reference to a complex data type (either a managed object or a data object) containing a property named b. The property b is called a nested property.

Properties can nest to several levels, such as `a.b.c`. In this example, property `b` is a complex data type containing a property named `c`. The property `c` might be a simple type, or it might be another complex type. If the final property in the property path string is a complex type, you get back an instance of the data type of the last property mentioned. For example, by specifying `a.b.c`, if `c` is aninteger, then you get an integer. If `c` is a data object type, then you get an instance of that type. If `c` is a ManagedOjbectReference, you get an instance of a ManagedObjectReference.

If the final property in the property path string is a complex type, and you request notification of property updates to that property, you get back changes to any of its constituent properties. For instance, specifying `a.b.c` results in notification of changes to `a.b.c.d` or `a.b.c.e` or any other nested property at that level. You can control whether these change notifications are sent individually using the partialUpdates flag, which is described at The partialUpdates Flag on page 95.

Key-Based Arrays

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

Indexed arrays are accessed in the usual manner with an index integer. They are used for arrays of data types where the position in the array does not change. For example, the roleList property of the AuthorizationManager managed object is an array of autorization roles. If a new role is added to the array, the position of the other elements does not change.

Key-based arrays refer to properties with either of the following characteristics:

- Its type is an array and the base type of the array has a key property.
- Its type is an array of managed object references.

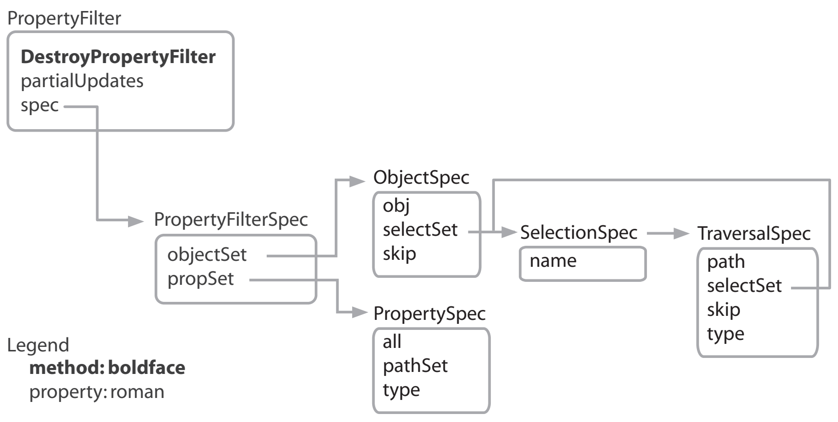
The contents of a key-based array property are accessed by the value of either the key property or, in the case of a managed object reference, its value property. The value of these fields is unique across all the components of an array. The key-based array is used for information whose position might change in the array. For example, the latestPage property of the TaskCollector managed object represents the items in the viewable latest page. As new items are added to the collector, they are appended at the end of the page. The oldest item is removed from the collector whenever there are more items in the collector than allowed. Because the position of the elements in the array might change, it is more efficient to use the key, a unique, unchanging value, to access an element. Usually these values are string values, but they can be integer values, as in the case of the key property of an Event array.

Returned Properties

Properties are returned to the client in one of two forms, depending on which operation was used to request the properties.

- If the client used RetrieveProperties to request the properties, the properties are returned in a list of ObjectContent data objects. Each ObjectContent contains a reference to the object whose properties are returned and a list of name-value pairs, one for each property of the object. Names of nested properties have the same form as a property path name; for instance, if a client requests a property of a VirtualMachine object named `runtime.powerState`, the property may be returned with name=`runtime.powerState` and value=`poweredOn`.
- If the client used CheckForUpdates or WaitForUpdates to request the properties, the properties are returned in a list of PropertyFilterUpdate objects. Each object represents a set of changes to properties that satisfy a PropertyFilter defined by the client. The changes for each filter are organized in a list of object references and change types, along with property name and value. As with the RetrieveProperties return values, names of nested properties have the same form as a property path name.

PropertyFilter



The correct specification of property filters is key to the operation of the PropertyCollector. This section shows the structure of property filters and discusses how to use construct multi-level filters for various property collection needs. The following topics are covered:

- ObjectSpec on page 82
- PropertySpec on page 83
- Traversal and Recursion on page 83
- Object Selection Without TraversalSpec Objects on page 84
- Object Selection With a Single TraversalSpec Object on page 84
- Traversing a Fixed Number of Levels on page 86
- Using Recursion in TraversalSpec Objects on page 90
- The partialUpdates Flag on page 95

Property filters allow a client to retrieve a subset of the properties of a managed object. Besides minimizing network traffic, this can simplify processing for the client. Only the properties of interest need be requested and processed by the client.

Property filters are also the means to specify the objects from which properties are to be retrieved. A client can specify either a single object or a set of related objects.

A client can create any number of property filters. Each filter defines both a set of objects and a set of properties. When the client requests property updates, the PropertyCollector draws from the union of all the client’s property filters. A change in any of the selected properties specified by any of the filters results in a change set entry returned to the client.

Specifying the objects and their properties is done with a PropertyFilterSpec object. The PropertyFilterSpec contains two parts:

1. A specification for the set of objects from which to retrieve properties. This is done with the ObjectSet property of the filter’s PropertyFilterSpec. See ObjectSpec on page 82.
2. A specification for the set of properties to retrieve from the selected objects. This is done with the PropSet property of the filter’s PropertyFilterSpec. See PropertySpec on page 83.

Part of the power of the property filter is the ability to guide the selection of a set of objects in dynamic ways. You can specify in detail how the server should traverse the inventory tree to the desired managed objects, without the need for the client to see the contents of intervening objects. This feature is explained at Traversal and Recursion on page 83.

ObjectSpec

An ObjectSpec object identifies the managed objects whose properties are returned to the client. The ObjectSpec can be used to specify a single managed object or a set of related managed objects reached by traversing the inventory tree.

The ObjectSet property of the PropertyFilterSpec object comprises a list of ObjectSpec objects, where each ObjectSpec can select one or more managed objects of interest to the client. Each ObjectSpec works independently to choose a subset of managed objects, which is then joined with the subsets specified by other ObjectSpecs in the same property filter to form the set of managed objects from which returned properties are retrieved.

The simplest use of an ObjectSpec is to set the obj property to a managed object reference, and set the skip property to `false`, omitting the selectSet property. The result is to select only the single managed object from which to retrieve properties.

To select a set of related managed objects, you need to use the selectSet property of the ObjectSpec. This causes VirtualCenter or the host agent to traverse the inventory tree, starting with the managed object specified in the obj property. Each managed object visited during the traversal may be selected for property retrieval, or may be used to continue the traversal.

The selectSet property comprises a list of TraversalSpec objects, each of which can specify a path to another managed object and another TraversalSpec object to continue the traversal. TraversalSpecs can be nested as needed to identify a specific path in the inventory tree. For a discussion on how to traverse the inventory, see Traversal and Recursion on page 83.

PropertySpec

The PropSet property of the PropertyFilterSpec object comprises a list of PropertySpec objects. A PropertySpec object specifies properties to be returned from the set of managed objects selected by the ObjectSpec part of the property filter.

Each PropertySpec object can specify a set of properties for a different object type. This gives you the ability to select a number of managed objects of different types (using the ObjectSet property of the PropertyFilterSpec) and retrieve properties that are specific to the different managed object types. For each type of managed object from which a property is desired, the client must create a corresponding PropertySpec object within the property filter.

The type field of the PropertySpec object is set to the type name for the desired type of managed object. See the description of the PropertySpec object in the *VMware Infrastructure SDK Reference Guide* for type property values.

The properties desired from the managed object are specified in one of three ways:

- If you want all the properties of the managed object, set the "all" property of the PropertySpec object to `true` and leave the pathSet property empty.
- If you want a reference to the managed object, rather than its properties, set the "all" property of the PropertySpec object to `false` and leave the pathSet property empty.
- If you want a specific property, or some of the properties, of the managed object, omit the "all" property and set the pathSet property to contain the names of the properties you want.

The pathSet is a list of property path names. The form of a property path name depends on whether the property is a nested property.

- The property path name of a nested property is dot-separated — for example, `PropertyFilterSpec.PropSet.type`.
- The property path name of a simple property contains only the property name — for example, `PartialUpdates`.

Traversal and Recursion

Traversing the managed object inventory tree generally requires a property filter containing nested TraversalSpec objects. This section explains in detail how to nest TraversalSpec objects to traverse paths of known configuration, how to specify a choice of possible paths, and how to use recursion to traverse nested path configurations.

The properties of the TraversalSpec data type have the following functions:

- type — The type of object to which the TraversalSpec applies. If the tree you are traversing contains an object that extends this type, the PropertyCollector applies the base type's TraversalSpec.

- path — The name of a property containing a reference to another object; the traversal proceeds to the other object by this reference.
- skip — A boolean flag that determines whether the referenced object should be included in the set of objects monitored by the PropertyCollector, or merely used as a step in traversing a tree of objects. This field is relevant only if the PropertyFilterSpec contains a PropertySpec object keyed to the type of the referenced object. Where the PropertySpecnormally causes the PropertyCollector to report properties of the referenced object, a "true" value in this field overrides the PropertySpec. Usually, it's safe to omit this field.
- selectSet — An array of SelectionSpec objects representing the next step of the traversal; each may be another TraversalSpec object or may simply name a TraversalSpec object defined elsewhere.

This section describes ways to use TraversalSpec objects to traverse parts of a tree whose structure is known in advance. These cases are presented from less complex to more complex.

Object Selection Without TraversalSpec Objects

In the simplest case, you already have a reference to the object whose properties you want to monitor. There is no need to traverse a tree, so there is no need to define a TraversalSpec object at all. Simply omit the selectSet property of the ObjectSpec, or leave it empty. Set the skip property to false, and create a PropertySpec object to select the desired properties.

Object Selection With a Single TraversalSpec Object

A different case requiring only minimal use of the TraversalSpec data type is a situation in which you are interested only in the immediate children of a managed entity for which you already have a reference. Only a single TraversalSpec object is needed. Similarly, if you are interested in collecting properties from the members of a list property of a managed object, you can do that with a single TraversalSpec object.

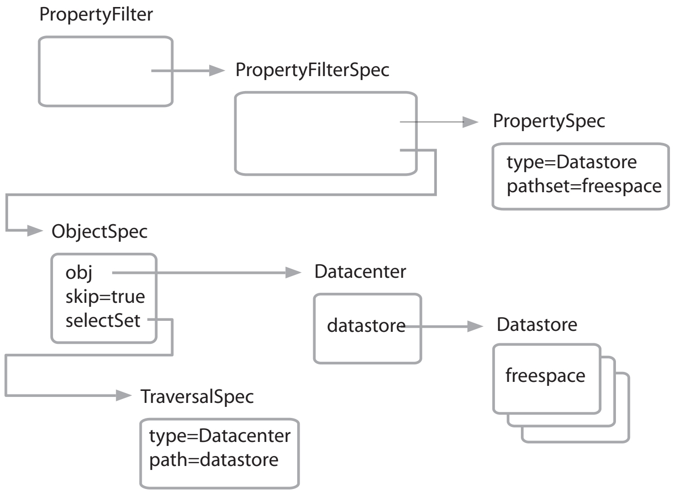
For example, suppose you want to monitor the free space for all datastores within a datacenter. Note that the free space is available as a property of DatastoreSummary, which is a property of Datastore. The Datacenter object contains an array of Datastore objects.

Using the PropertyCollector, you need to create a PropertyFilter object, which contains a PropertyFilterSpec object, which contains a PropertySpec object and an ObjectSpec object. The PropertySpec object needs to specify the Datastore object type, as well as the path to the freespace property from the Datastore object: `summary.freespace`. The resulting PropertySpec object tells the PropertyCollector to collect the value of the "freespace" property from all Datastore objects within the object selection set specified by the ObjectSpec object.

You create the ObjectSpec object as follows:

- Set the obj property to a reference to the Datacenter managed object. For this example, it is assumed that you already have the reference.
- Set the skip field to a true value. You are not interested in collecting the properties of the Datacenter itself.

- Since you know in advance that the object reference you have refers to a Datacenter object, you need only specify how to traverse a Datacenter object to find the associated Datastore objects. This requires setting the selectSet property of the ObjectSpec to an array containing a single TraversalSpec object.
- Set the type of the TraversalSpec object to `Datacenter`. When the PropertyCollector tries to traverse the starting object you specified in the ObjectSpec, it will look for a matching type in the value of the type field to determine how to traverse the Datacenter object.
- The path field of the TraversalSpec object tells the PropertyCollector how to get to the next object in the traversal. Since you want to traverse the array of Datastore objects, you should set the path field to `datastore`.
- The selectSet property of the TraversalSpec object should be an empty array or omitted, because you don't need a second traversal step.



The resulting set of selected objects contains all those Datastore objects reached by traversing the datastore property of the Datacenter object. The selection set is complemented by the PropertySpec, which specifies collecting the "freespace" property from the selected objects.

Traversing a Fixed Number of Levels

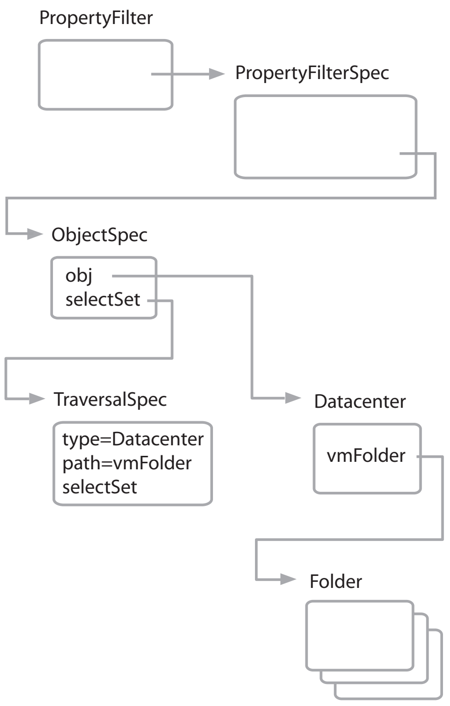
If you know in advance the structure of the hierarchy you want to traverse, you can extend the traversal to as many levels as needed, by nesting TraversalSpec objects. As long as the number of traversal steps is fixed, you don't need recursion; simply extend each TraversalSpec array with another level of TraversalSpec arrays.

For example, suppose you want to collect the UUIDs of all the virtual machines in a datacenter. You have a reference to the datacenter, and you know that your inventory hierarchy has several folders of virtual machines under the datacenter, and that there are no nested folders. Thus, all the VirtualMachine objects are found exactly two levels below the Datacenter objects, and each can be reached by traversing exactly one Folder object.

In this example, set up your ObjectSpec as follows:

- Set the obj property to the reference to the Datacenter object, as the starting point of the traversal.
- You can set the skip property to a "true" value to tell the PropertyCollector not to collect properties from the starting object; or you can rely on the absence of a PropertySpec object with a type property set to `Datacenter`. The PropertyCollector will not collect properties from a Datacenter object unless you also create a PropertySpec object for datacenters.
- Set the selectSet array to contain a single TraversalSpec object. Since you plan to traverse only virtual machine folders below the datacenter, you need only one TraversalSpec object to describe how to make the first step of the traversal.
- In the TraversalSpec object, set the type property to `Datacenter`, to match the type of the first object you want to traverse.
- Set the path property to the name of the property in the Datacenter object that refers to its virtual machine folders: `vmFolder`.
- You can set the skip property of the TraversalSpec to a "true" value, to specify that the PropertyCollector should ignore the properties of the folder, but it is not necessary. The PropertyCollector does not collect properties from a Folder object unless you also create a PropertySpec object for folders.

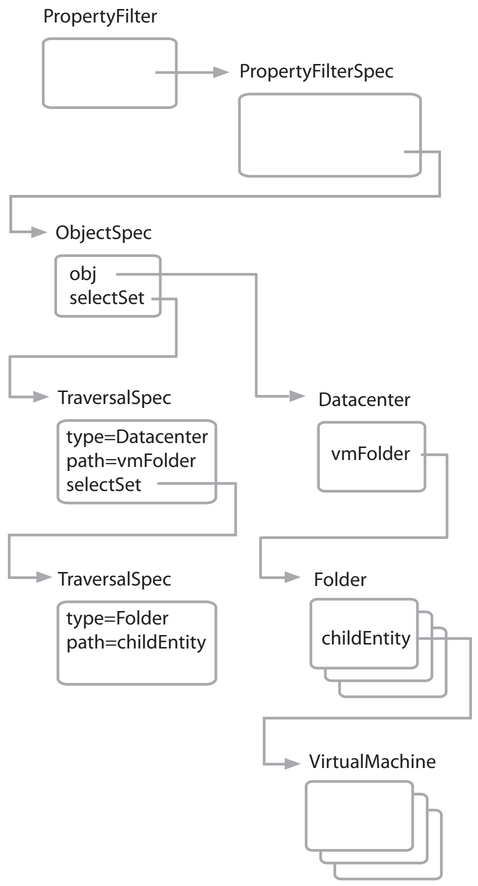
  The following illustration shows the TraversalSpec object needed for the first step of the traversal.



The above actions cause the PropertyCollector to traverse the Datacenter object to access its virtual machine folders; the second step is to traverse the Folder objects to access their virtual machines. To do the second traversal step, follow these steps:

- In the TraversalSpec object you created to traverse the Datacenter object's vmFolder property, add a TraversalSpec object to its selectSet array.
- In this TraversalSpec object, set the type property to `Folder`.
- Set the path property of this TraversalSpec object to `childEntity`. This is the name of the Folder property that refers to the folder's virtual machines.
- Leave the skip property of this TraversalSpec object unset, or set it to a "false" value. If you set it to a "true" value, it will cause the PropertyCollector to ignore any PropertySpec object that applies to VirtualMachine objects.
- The selectSet property of this TraversalSpec object should be an empty array or omitted, because you don't need to traverse beyond the VirtualMachine objects.

The following illustration adds the TraversalSpec object needed to do the second traversal step.

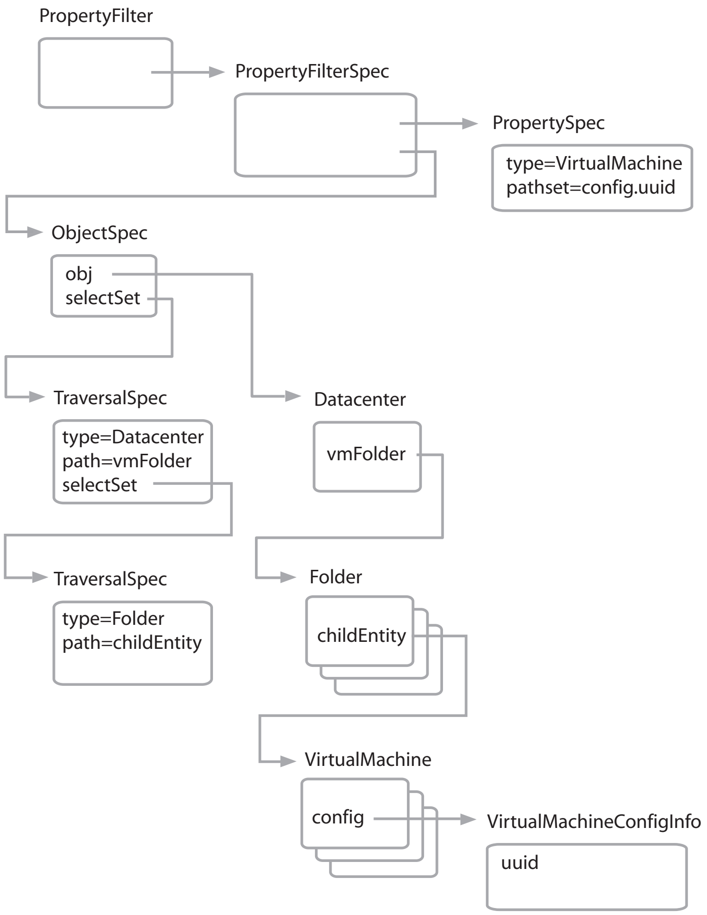


The above steps will cause the PropertyCollector to traverse all the datacenter's virtual machine folders to locate its virtual machines and add them to the object selection set. To collect the desired property from the VirtualMachine objects, use a PropertySpec object, as follows:

- Set the type property to `VirtualMachine`. This causes the PropertyCollector to collect properties from VirtualMachine objects in the selection set.
- Leave the "all" property unset, because you want to collect only a subset of properties from VirtualMachine objects.

- Set the pathSet property to `config.uuid`, because the UUID is found in the uuid property of the config property of the VirtualMachine object.

  The following illustration adds the PropertySpec object needed to collect the UUID properties from the virtual machines located by the traversal steps.



When you create your PropertyFilter from the ObjectSpec and PropertySpec objects described above, the PropertyCollector is prepared to collect all the virtual machine UUIDs on request.

Using Recursion in TraversalSpec Objects

If you need to traverse a tree to an unknown depth, you can use recursive TraversalSpec objects to follow a repeating traversal pattern. Often you can do this with two levels of selectSet arrays linked to your ObjectSpec. The first level, ObjectSpec.selectSet[], contains a list of TraversalSpec objects that is general enough to apply to the starting managed object and any subsequent traversal steps. The second level of selectSet arrays contains a list of SelectionSpecs that tell the PropertyCollector which TraversalSpec object to use for the next step in the traversal.

The first level of selectSet objects contains named TraversalSpecs, with one TraversalSpec for each type of object that you could encounter at any stage of the traversal. The names allow the TraversalSpecs to be referenced at the second level. When control is transferred to a named TraversalSpec, that is called (in this context) recursion.

For example, suppose you want to create a PropertyFilterSpec that contains an ObjectSpec designed to traverse a managed entity hierarchy, starting from any arbitrary object in that hierarchy and selecting all managed entities below and including the starting object. You don't know in advance the type of the starting object, so you need to create an ObjectSpec with a list of TraversalSpec objects, at least one for each type of object you could traverse. The inventory includes these types:

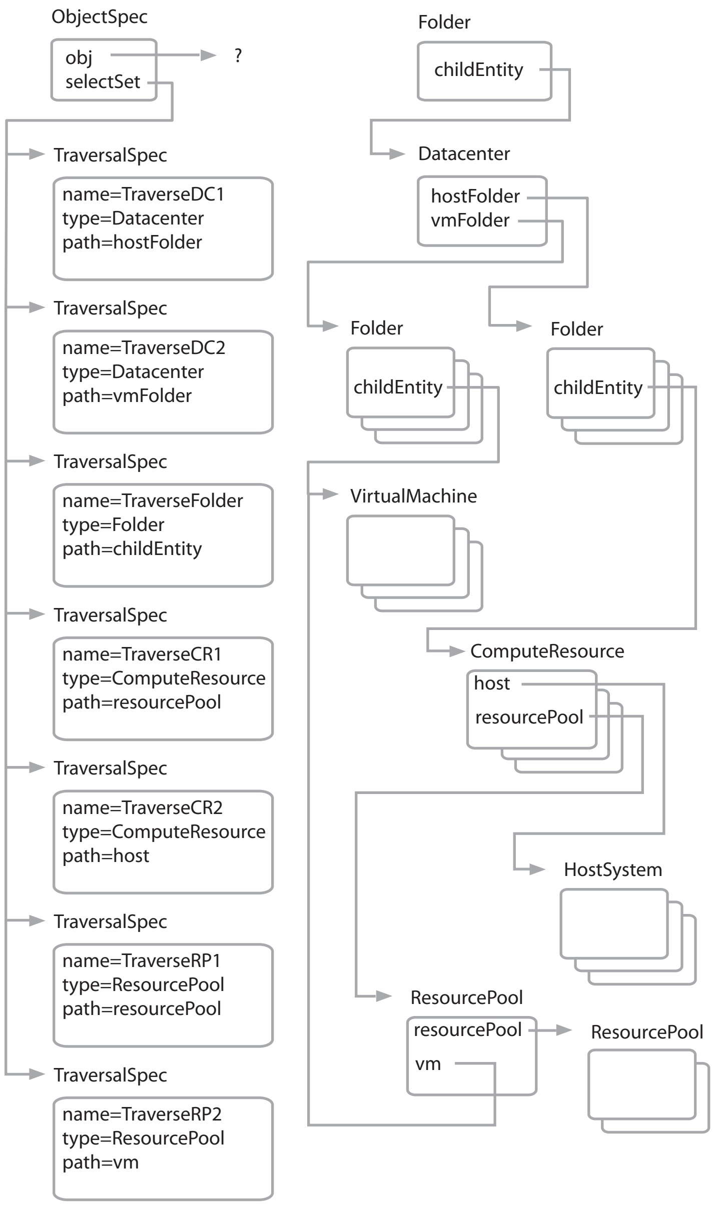
- Folder
- Datacenter
- ComputeResource
- ResourcePool

To traverse any of these entities, perform these actions to create an appropriate ObjectSpec:

- Set the obj property to the starting object.
- Set the skip property to a false value, or omit it.
- Add seven TraversalSpec objects to the selectSet array. Although there are only four types of objects to traverse, you need extra TraversalSpec objects because some objects have two paths to traverse. You can specify only one path in a single TraversalSpec.
  - Another TraversalSpec object has type `Datacenter` and name `TraverseDC1`. Set the path property to `hostFolder`.
  - Another TraversalSpec object has the type `Datacenter` and name `TraverseDC2`. Set the path property to `vmFolder`.
  - One TraversalSpec object has the type `Folder` and name `TraverseFolder`. Set the path property to `childEntity`.

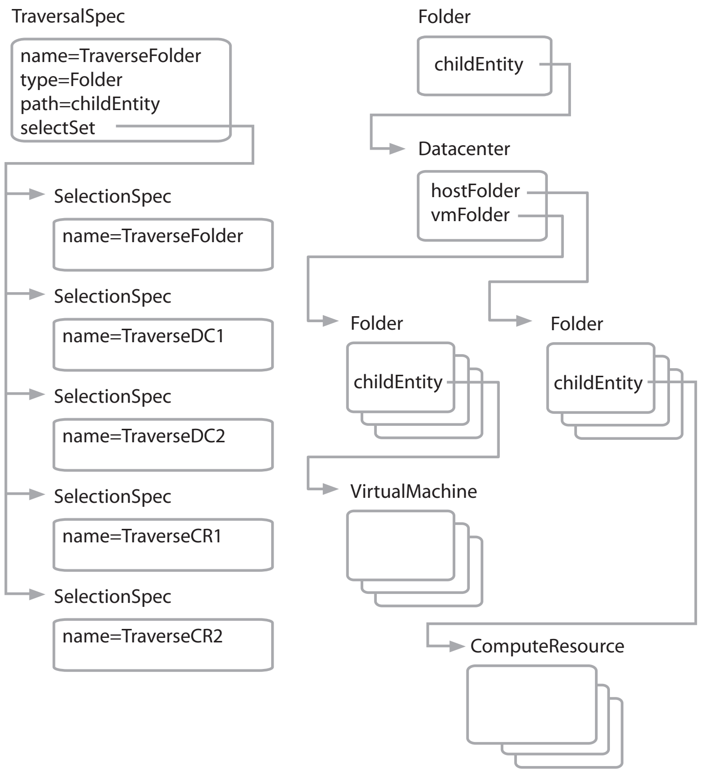
- Another TraversalSpec object has the type `ComputeResource` and name `TraverseCR1`. Set the path property to `resourcePool`.
- Another TraversalSpec object has the type `ComputeResource` and name `TraverseCR2`. Set the path property to `host`.
- Another TraversalSpec object has the type `ResourcePool` and name `TraverseRP1`. Set the path property to `resourcePool`.
- Another TraversalSpec object has the type `ResourcePool` and name `TraverseRP2`. Set the path property to `vm`.

The following illustration shows the TraversalSpec objects needed for the first step of the traversal. The illustration includes a diagram of the objects to be traversed in the inventory hierarchy.


ObjectSpec
obj
selectSet
?
TraversalSpec
name=TraverseDC1
type=Datacenter
path=hostFolder
TraversalSpec
name=TraverseDC2
type=Datacenter
path=vmFolder
TraversalSpec
name=TraverseFolder
type=Folder
path=childEntity
TraversalSpec
name=TraverseCR1
type=ComputeResource
path=resourcePool
TraversalSpec
name=TraverseCR2
type=ComputeResource
path=host
TraversalSpec
name=TraverseRP1
type=ResourcePool
path=resourcePool
TraversalSpec
name=TraverseRP2
type=ResourcePool
path=vm
Folder
childEntity
Datacenter
hostFolder
vmFolder
Folder
childEntity
Folder
childEntity
VirtualMachine
ComputeResource
host
resourcePool
HostSystem
ResourcePool
resourcePool
vm
ResourcePool

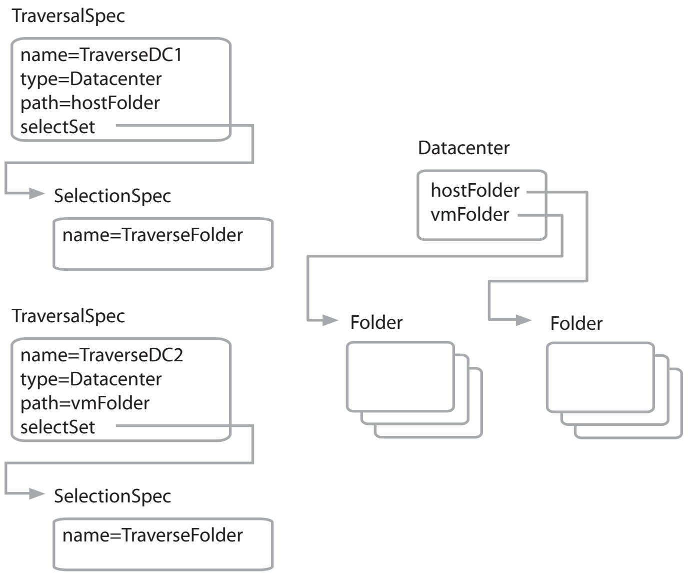
- Each TraversalSpec object needs a selectSet array, containing SelectionSpec objects — not TraversalSpec objects — that simply name the types of objects that could be encountered at the next traversal step. The PropertyCollector uses the names to identify the next TraversalSpec object to apply. Thus, the following selectSet arrays should be created:
  - Because a Folder can be followed by another Folder, a Datacenter, or a ComputeResource, TraverseFolder needs five SelectionSpec objects. Their names are `TraverseFolder`, `TraverseDC1`, `TraverseDC2`, `TraverseCR1`, and `TraverseCR2`. Each SelectionSpec object invokes the name of a TraversalSpec object defined elsewhere in the PropertyFilter.

The following illustration shows these five SelectionSpec objects. The other six TraversalSpec objects shown in the previous illustration are omitted here because of space limitations.



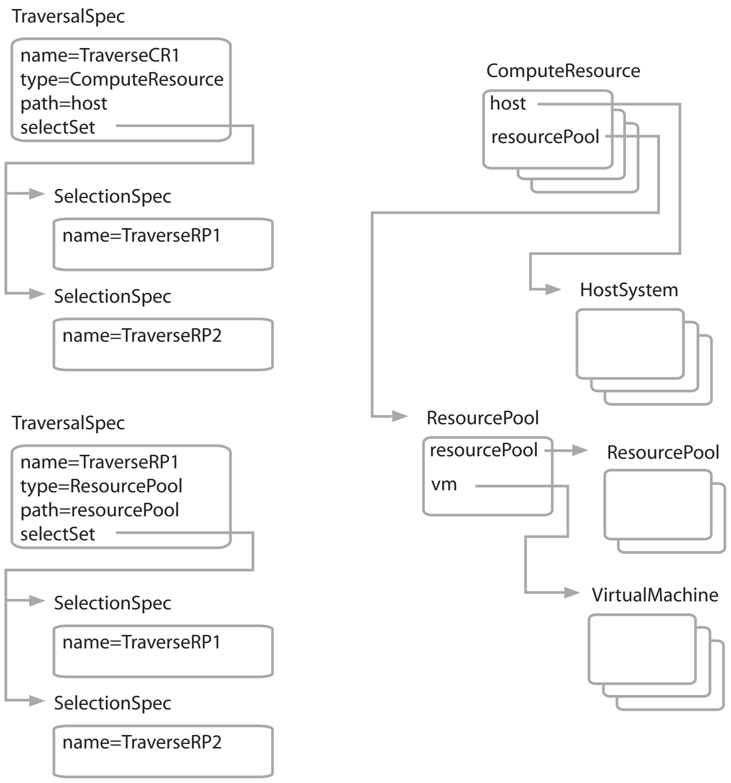
- A Datacenter object can be followed only by a folder object, so TraverseDC1 needs an array of one SelectionSpec object, named `TraverseFolder`.
- Similarly, TraverseDC2 needs an array of one SelectionSpec object, named `TraverseFolder`.

The following illustration shows the SelectionSpec objects for TraverseDC1 and TraverseDC2.



- A ComputeResource object can be followed either by a ResourcePool or a HostSystem. There is no need to traverse a HostSystem, but there are two different ResourcePool TraversalSpecs to cover, so TraverseCR1 needs an array of two SelectionSpec objects, named `TraverseRP1` and `TraverseRP2`.
- TraverseCR2 can lead only to a HostSystem object, so there is no need for it to have a selectSet array.
- TraverseRP1 can lead only to another ResourcePool, but there are two paths out of ResourcePool, so it needs an array of two SelectionSpec objects, named `TraverseRP1` and `TraverseRP2`.
- TraverseRP2 can lead only to VirtualMachine objects, so there is no need for it to have a selectSet array.

The following illustration shows the SelectionSpec objects needed for the TraverseCR1 and TraverseRP1 TraversalSpec objects.



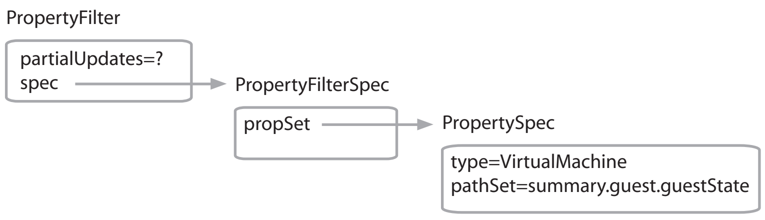
The above procedure results in a selection set that includes all managed entities below and including the starting managed object. After creating one or more PropertySpec objects, you can use the PropertyCollector to collect properties from all the selected managed entities — for instance, the name, configuration status, and a reference to each managed entity.

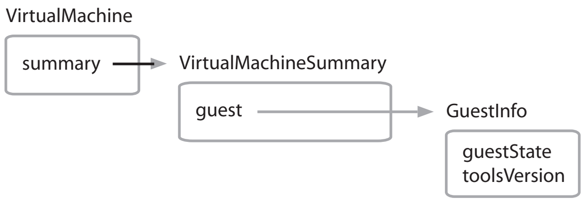
The partialUpdates Flag

The partialUpdates flag offers another way to minimize network traffic. In cases where a requested property is a data object type rather than a built-in type, the requested property contains properties of its own. These are nested properties, but the client may not have specified them in the PropertyFilterSpec. The partialUpdates flag allows the user to retrieve only the nested property that changed, rather than all the properties of the data object.

For example, suppose a client creates a property filter that specifies the `summary.guest.guestState` property of a virtual machine. When the client invokes

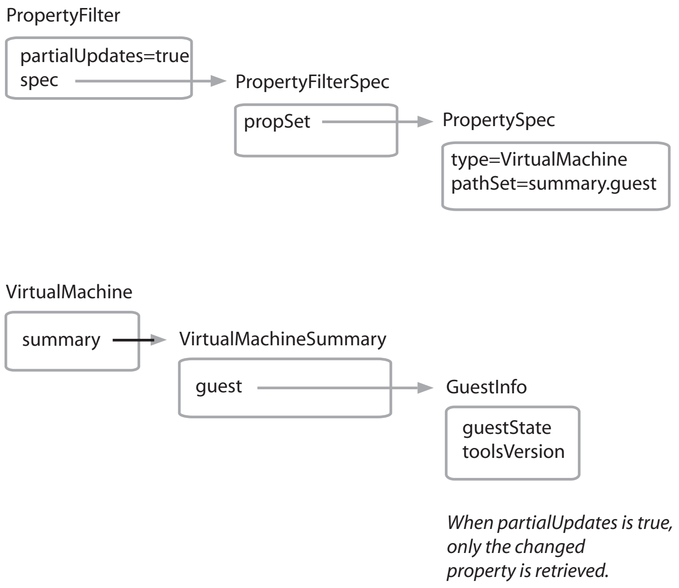
CheckForUpdates or WaitForUpdates, a change to the `guestState` property is returned in a change list. However, a change to the `toolsVersion` property is ignored, because it is not specified in the property filter. The objects are shown in the following illustration.





Now suppose the client uses a filter that specifies the `summary.guest` property of a virtual machine. If the `toolsVersion` property changes, that falls into the range of the new filter because toolsVersion is a nested property of `VirtualMachineSummary.GuestInfo.` A change to a data object is by definition a change to one of its nested properties, since a data object has no single value of its own.

When the change set is returned to the client, it could include a copy of the entire data object called `summary.guest`, or it could return only the nested property (`toolsVersion`) that changed. The choice is determined by the partialUpdates flag. If the flag is set to true, only the nested property is returned; if the flag is set to false, the whole property specified in the property filter is returned. The situation is shown in the following illustration.


PropertyFilter
partialUpdates=true
spec
PropertyFilterSpec
propSet
PropertySpec
type=VirtualMachine
pathSet=summary.guest
VirtualMachine
summary
VirtualMachineSummary
guest
GuestInfo
guestState
toolsVersion
When partialUpdates is true, only the changed property is retrieved.

APPENDIX B

Using the PerformanceManager

This appendix contains additional information on using the PerformanceManager. The topics covered are:

Configuring PerformanceManager Settings

A PerfInterval object has a name, a lifetime (how long it's retained in the database), and a sampling period. The default performance intervals are:

- Sample period: five minutes,
  retention length: one day.
- Sample period: one hour,
  retention length: one week.
- Sample period: six hours,
  retention length: 30 days.
- Sample period: one day,
  retention length: one year.

A client can configure custom performance intervals, using these operations:

- CreatePerfInterval — Adds a new interval to the list.
- RemovePerfIntervalRemovePerfInterval — Removes an interval from the list.
- UpdatePerfInterval — Modifies an interval in the list.

Restrictions on PerfInterval Configuration

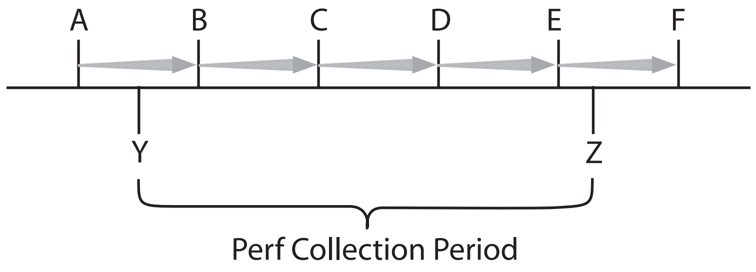
UpdatePerfInterval can be used to modify either a default PerfInterval or a custom PerfInterval. VirtualCenter supports PerfInterval configuration; the host agent does not. The following restrictions apply:

- The minimum sample period is twenty seconds.
- PerfInterval sample periods are measured in seconds. The sample period length is the key used to identify a PerfInterval for update or removal. As a consequence, the only way to change the sample period of an existing PerfInterval is to delete the PerfInterval and create a new one with the desired sample period.
- When you update or add a PerfInterval, the resulting set of intervals must follow the rule that each sample period (except the smallest) is an integral multiple of the previous sample period. Thus, you can set sample periods of one hour, two hours, and four hours, but you may not set intervals of one hour, two hours, and three hours.
- PerfInterval retention lengths are also measured in seconds. The retention length of a PerfInterval must be a multiple of the sample period.

- The retention length for each successive interval must be greater than the retention length of the previous interval (the interval with the next shorter sample period). However, retention lengths do not need to be integral multiples of smaller retention lengths.
- If you modify PerfIntervals, the change is global. The new intervals affect all users at once.

Interpreting Statistics

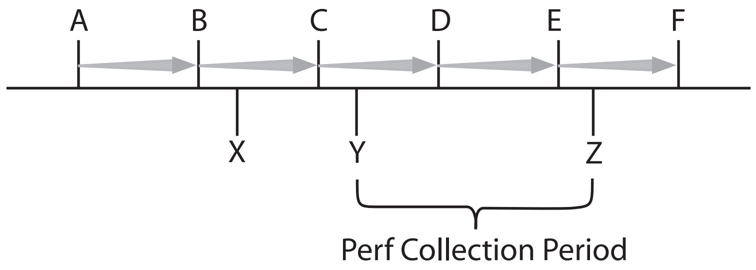
When you request counters for a given block of time, what you get back is the counters that finished accumulating during that time interval. Each counter accumulates data for the interval that went before it. Depending on the length of the interval, some of the data accumulated in a counter could represent a time before the beginning of the interval for which you want to collect statistics.



Referring to the illustration above, if you collect statistics for the period from time Y to time Z you get the counters that accumulate up to B, C, D, and E. The counter accumulating to time B represents a PerfInterval from A to B — part of which lies outside the intended collection period. Also, time Z falls during the PerfInterval from E to F, but you do not get information from any time after E; that information is stored in the counter accumulating to time F, which falls outside the collection period. As a result, when you specify the collection period from Y to Z, what you actually get is a collection period from A to E.

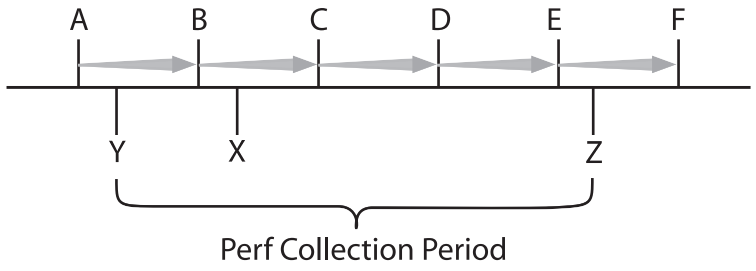
Since you have no control over the exact time when an interval starts, you have no way to exactly synchronize an interval with the collection time you choose. If you need to collect statistics relative to an external event, you may have to adjust your collection time to make sure that the external event is either excluded or included in the block of time for which you request statistics. For example:

- If you want to collect statistics that represent a time when a virtual machine is running but not when it starts up, you should first note the time when the virtual machine started, then add the length of the longest PerfInterval you want to examine, and then request statistics for a time period beginning after that point. This ensures that the virtual machine is running and stable before the first PerfInterval begins.



Referring to the illustration above, assume an event happens at time X which you expect to have a significant effect on the information you want to collect. To make sure the event at X is excluded from the data, add the length of the PerfInterval to time X to get the beginning collection time, Y. This results in an actual collection time beginning at C.

- If you want to collect statistics that capture an event such as a virtual machine starting, you should note the time when the virtual machine was started, subtract the length of the longest PerfInterval you want to examine, and then request statistics for a time period beginning at that point. This ensures that the first counter represents a PerfInterval that began before the virtual machine started.



Referring to the illustration above, assume an event happens at time X which you expect to have a significant effect on the information you want to collect. To make sure the event at X is included in the data, subtract the length of the PerfInterval from time X to get the beginning collection time, Y. This results in an actual collection time beginning at A.

APPENDIX C

Revision History

The following table lists the revision history for the *VMware Infrastructure SDK Getting Started Guide*.

Date	Description
June 15, 2006	Initial publication of the *VMware Infrastructure SDK Getting Started Guide*.

Index