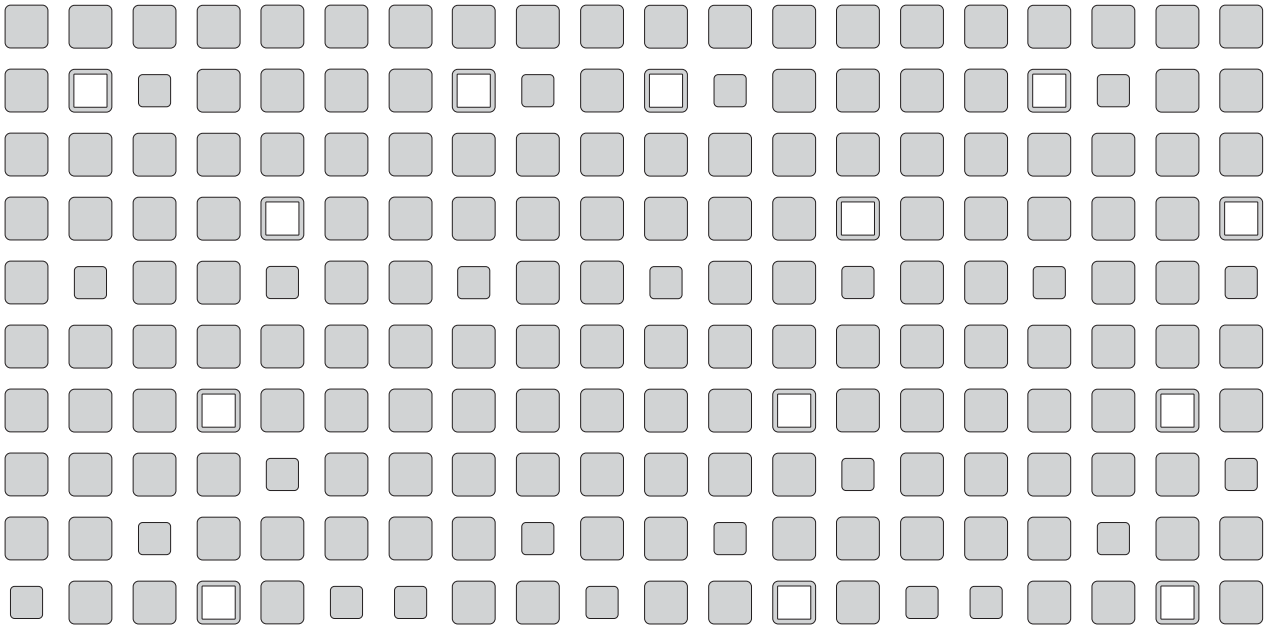
VERSION 1.4

# Virtual Infrastructure SDK

## Programming Guide



vmware®

**VMware, Inc.**

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

**Please note that you will 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 20060602 Version 1.4 Item: VC-ENG-Q206-231

# Table of Contents

CHAPTER 1

# Introducing the Programming Guide

The purpose of this programming guide is to help you develop a client application that you can use to connect to the VMware VirtualCenter Web Service. 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.

We discuss the following topics in this programming guide:

- VMware SDK Key Concepts on page 21

  This chapter introduces you to the key concepts you should know before building a client application.

- VMware SDK Management Concepts on page 37

  This chapter describes the different management tasks you can perform on both hosts and virtual machines with the VMware SDK product.

- Developing Client Applications on page 45

  This chapter provides an overview of the steps you should consider when building your client application.

- Core Client Programming Concepts for Java Programmers on page 59

  This chapter summarizes best practices in building a client application in Java.

- Advanced Client Concepts for Java Programmers on page 87

  This chapter provides additional programming concepts that build upon the basic concepts described in the preceding chapter.

- Core Client Concepts for Perl Programmers on page 133

  This chapter summarizes best practices in building a client application in SOAP::LITE for Perl.

- Advanced Client Concepts for Perl Programmers on page 173

  This chapter provides additional programming concepts that build upon the basic concepts described in the preceding chapter.

- Sample Applications on page 241

  This chapter provides examples of client applications that perform particular tasks. It includes a description of how the sample applications were conceived and designed.

- Client Development Environments on page 267

  This chapter describes the different development environments you can use to develop your client application.

- Troubleshooting on page 275

  This chapter describes some troubleshooting tips if you have difficulties with the VMware VirtualCenter Web Service.

# Using This Programming Guide

This *Virtual Infrastructure SDK Programming Guide* is a companion book to the *Virtual Infrastructure SDK Reference Guide*.

The *Virtual Infrastructure SDK Programming Guide* includes a detailed description of VMware Virtual Infrastructure SDK (VMware SDK) concepts and how clients interact with the VMware VirtualCenter Web Service. We discuss how to create a simple client application, then discuss basic and advanced programming concepts to help you build your client application. Finally, we discuss the different developer environments you may use, followed by a description of the sample and reference applications supplied in the VMware SDK package.

The *Virtual Infrastructure SDK Reference Guide* contains a description of the data models, or logical structure of the VMware VirtualCenter Web Service. This is followed by a list of the datatypes present in `vma.wsdl`. Finally, this guide contains a comprehensive list of all the Web service operations, including its input message, its output message, and any Faults.

**Note:** Be sure to read this programming guide before using the VMware SDK. The programming guide contains information that helps you understand this product, and enhances your use of the VMware SDK.

## Intended Audience

This programming guide is written for programmers that 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, VMware ESX Server and other VMware products.

# Related VMware Products

**Note:** In this release, the VMware SDK supports VMware VirtualCenter 1.2, ESX Server 2.0.1 2.1.x, and 2.5, and GSX Server 3.1.

## VMware VirtualCenter

VMware VirtualCenter is a software solution for deploying and managing virtual machines across multiple server machines running ESX Server and GSX Server™ hosts. It enables customers to manage a distributed, heterogeneous computing environment as a single pool of computing resources. VirtualCenter includes a new technology called VMotion™ that enables the seamless migration of running virtual machines across hosts with no service interruption.

VirtualCenter provides virtual infrastructure services and functionality in five areas:

- Centralized management of large virtual machine environments through a single user interface
- Automated virtual machine provisioning
- Virtual machine performance and workload analysis
- Virtual machine migration, including VMotion
- Secure access control that integrates into existing Microsoft® Windows® user management systems

## VMware ESX Server

VMware ESX Server is virtual infrastructure software for partitioning, consolidating and managing systems in mission-critical environments. ESX Server and VMware Virtual Infrastructure Nodes provide a highly scalable virtual machine platform with advanced resource management capabilities, which can be managed by VMware VirtualCenter.

## VMware GSX Server

VMware GSX Server is an enterprise-class virtual infrastructure software for x86-based servers. It is used for server consolidation, disaster recovery and streamlining software development processes. Tight integration with VMware VirtualCenter enables GSX Server to deliver exceptional manageability and scalability.

**Note:** VMotion is not supported with GSX Server in this release.

# VMware SDK Architecture

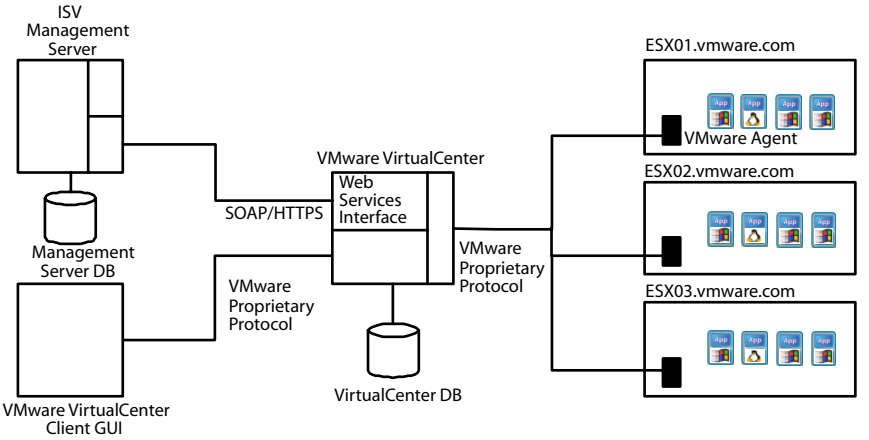
The VMware SDK comprises two components.

- VMware VirtualCenter Web Service — Includes the file, `vma.exe` (starts the Web service manually), `vmaConfig.xml` (Web service configuration file) and other associated files.
- VMware SDK package — SDK documentation, including the sample code and a `README` file.

## VMware VirtualCenter Web Service

The VMware VirtualCenter Web Service is installed as part of the VMware VirtualCenter installation. Refer to the VirtualCenter documentation for complete information on installing and verifying the installation of this Web service.

The VMware VirtualCenter Web Service architecture is displayed in the following diagram:




The VMware VirtualCenter Web Service provides a Web services interface to ISV Management Servers. These servers may also receive information directly from ISV Agents that exist on the virtual machines, using their own proprietary protocol.

The Web service manages a collection of objects that represent tasks, schedules, events, performance collectors, virtual machines, Farms, Farm groups, virtual machine groups, and hosts. The objects are organized in a hierarchy, and are identified by using the full path name in the hierarchy. A path is like the full path name of a file (for example, `/vcenter/farm1/host2`) and an object is described by an XML document that is the value of the object. We provide more details of paths, objects, and related terms in the next chapter.

### vma.exe Program

You can use the `vma.exe` command-line program to start the Web service manually. Execute this program on the command line. Type `C:\Program Files\VMware\VMware VirtualCenter \vma.exe -help` for a list of the options for this command. Refer to the section titled "Changing VMware Web Service Options After Installation" in the *VMware VirtualCenter User's Manual* for complete information about this program.

### vmaConfig.xml File

The `vmaConfig.xml` file configures the VMware VirtualCenter Web Service. Here is an example of this file:

```
<vma>
  <service>
    <wsdl>vma.wsdl</wsdl>
    <eventlog rollover="true" level="info" file="vma" console="true" />
    <sslport>8443</sslport>
    <externalSchemas>
      <schema>autoprep-types.xsd</schema>
    </externalSchemas>
    <sslCert>C:\Documents and Settings\All Users\Application
      Data\VMware\VMware VirtualCenter\VMA\server.pem</sslCert>
    <sslCAChain>C:\Documents and Settings\All Users\Application
      Data\VMware\VMware VirtualCenter\VMA\root.pem</sslCAChain>
  </service>
  <subjects>
    <subject>
      <implementation>VCenter 1.1</implementation>
      <path>/vcenter</path>
      <hostname>VC-Hostname</hostname>
      <port>905</port>
      <eventlog level="info" />
      <ssl>true</ssl>
      <preload>true</preload>
      <index>
        <defaultFarm>Default Farm</defaultFarm>
      </index>
    </subject>
  </subjects>
</vma>
```

This file comprises three sections:

- `service`
- `externalSchemas`
- `subjects`

***service —*** The `service` section includes information on how to configure the Web service. You can configure the elements in this section by editing `vmaConfig.xml` (eventlog element) and running the `vma` command (all other elements in this section). For an example of using the `vma` command, see the example following this table. For a complete list of `vma` options, refer to the section titled "Changing VMware Web Service Options After Installation" in the *VMware VirtualCenter User's Manual*.

| Element | Description |
|---|---|
| eventlog | Specifies event logging. You may update the values for this element by editing `vmaConfig.xml`.<br>• rollover — Specifies whether or not the event log is a rollover log. If "true" (the default value), then the event log comprises rollover log files. There are 10 event log files, by default. When each log file reaches its maximum size, then logging starts in a new log file. For example, if `vma-0.txt` reaches 1000KB, then new logging information is written to `vma-1.txt`. When the 10th log file reaches 1000KB, then `vma-0.txt` is overwritten with the new information.<br>• level — Level of information contained in the basic event log file. The default value is "info". However, you may change the log level to "verbose" for additional information.<br>• file — Name of the event log file. The default name is "vma".<br>• console — Applicable when the Web service (vma.exe) is running on a command line in a console (or terminal) window. Acceptable values are "true" or "false". If "true", then the event log is printed to the console window as well as the event log file.<br>• fsizeLimit — Maximum log file size, in KB, before the log rolls over to the next file. The default size is 1000KB.<br>• fcountLimit — Maximum number of files in the event rollover logs. The default number is 10. For example, if the log file name is `vma`, then the first log file is `vma-0.txt`, the second log file is `vma-1.txt`, and so on. |
| sslport | The SSL port number at which the Web service listens. Clients should use the HTTPS protocol to send requests to this port. You may update the value for this element by running the `vma` command with the `-VMAport` option. |
| sslCert | Certificate file in Privacy Enhanced Mail (PEM) format. You may update the value for this element by running the `vma` command with the `-sslCert` option. |
| sslCAChain | Certificate CA chain file in PEM format. You may update the value for this element by running the `vma` command with the `-sslCAChain` option. |
| sslPrivateKey | (Encrypted and stored in the Windows registry.) Private key file in PEM format. You may update the value for this element by running the `vma` command with the `-sslPrivateKey` option. |

| Element | Description |
|---|---|
| sslPassphrase | Passphrase for the private key. It is used only once to decrypt the Private key. The decrypted private key is stored in the Windows Registry with the Windows encryption method. You may update the value for this element by running the `vma` command with the `-sslPassphrase` option. |

For example, if you want to update the SSL security information, then complete the following:

1. Stop the Web service. Refer to section titled "Changing VMware Web Service Options After Installation" in the *VMware VirtualCenter User's Manual*.
2. Execute the `vma` command with the `-update` option on the command line:

   ```
   cd C:\Documents and Settings\All Users\Application
   Data\VMware\VMware\VirtualCenter\VMA
   ```

   ```
   C:\Program Files\VMware\VMware VirtualCenter \vma.exe -update
   -sslCert server.pem -sslCAChain root.pem -sslPrivateKey server.pem
   -sslPassphrase password
   ```
3. Restart the Web service.

***externalSchemas —*** The `externalSchemas` section lists all `.xsd` files that must be publicly accessible along with `vma.wsdl`. `autoprep-types.xsd` is the customization schema used to customize the guest operating system in a virtual machine. See Customizing a Virtual Machine on page 100.

**Note:** Do not edit the `externalSchemas` section.

***subjects —*** The Web service is configured to interface with multiple sources of information and display the information from each source in a client view. Each configuration is called a subject. In this release, only a single subject is supported. The current subject is a VirtualCenter implementation.

**Note:** Do not edit the `subjects` section unless asked to do so by Support.

| Element | Description |
|---|---|
| implementation | Name of the subject; in this case, VirtualCenter. |
| path | Path to the contents of the subject; in this case, the `/vcenter` directory. |
| hostname | Host name where VirtualCenter is running. |
| port | Port number at which VirtualCenter listens; by default, this is 905. Do not change this port number unless you have already changed the default port number for VirtualCenter. |
| eventlog level | Specifies the amount of detail in the advanced debugging event log file. The default value is "info". |

| Element | Description |
|---|---|
| ssl | Specifies whether or not SSL should be used for communicating between the Web service and VirtualCenter. If true, SSL is used. If false, a non-SSL port is used. |
| preload | By setting this element to true, the VirtualCenter information is loaded immediately upon startup of the Web service, rather than upon the first request from a Web service client. |
| defaultFarm | Name of a Farm in the `/vcenter` hierarchy. The default value for this element is "Default Farm" that corresponds to the `/vcenter/Default Farm` directory. |

## VMware SDK Package

Obtain the VMware SDK package on the Web at *www.vmware.com/support/developer/vc-sdk*. This page includes information on what is contained in the package and the instructions for downloading this package.

## Web Service Standards and the VMware SDK

- VMware SDK strives to conform to the Web Services Interoperability organization (WS-I) Basic Profile 1.0. The WS-I organization has a wide industry following with 127 members (see *www.ws-i.org*.

The mission of WS-I is to promote interoperability of Web services across platforms, operating systems, and programming languages. By conforming to the WS-I Basic Profile, we ensure that the VMware SDK can be used from within the widest range of development environments.

A WS-I profile is a prescription for the use of a set of existing standards at specific versions. For example, the WS-I Basic Profile 1.0 covers XML Schema 1.0, SOAP 1.1, WSDL 1.1, and UDDI 2.0. In addition to prescribing these standards, the profile also narrows these standards by specifying best practices, and excluding optional or loosely specified features.

The WS-I basic profile is expressed as a set of assertions about the format, and sometimes the interpretation of SOAP messages and the WSDL document. The verification of most of these assertions can be done automatically by examining artifacts (SOAP messages and the WSDL file) exchanged between a Web service client and server. The WS-I organization has published tools that do this automatic verification. VMware, Inc. uses these tools on a regular basis to test the conformance of the VMware SDK to the WS-I Basic Profile 1.0.

In addition to WS-I, VMware, Inc. also actively monitors the evolution of other Web Services standards that may be relevant to its domain.

- DMTF — *www.dmtf.org*
- W3C — *www.w3.org*

# Technical Support Resources

Refer to the following for additional information.

- VMware SDK — *www.vmware.com/support/developer/vc-sdk*
- VMware ESX Server — *www.vmware.com/products/server/esx_features.html*
- VMware GSX Server — *www.vmware.com/products/server/gsx_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*
- Perl — *www.cpan.org*

CHAPTER 2

# VMware SDK Key Concepts

To use the VMware VirtualCenter Web Service efficiently, you first need to understand the terminology associated with this product. In this section, we'll describe VMware VirtualCenter terminology and its association with the VMware SDK.

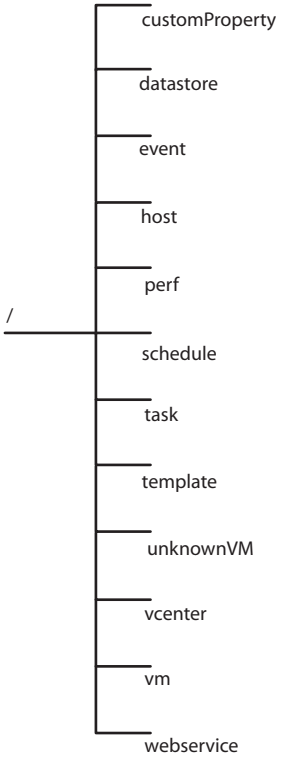
| VirtualCenter Term | SDK Term | Possible Paths to the Object |
|---|---|---|
| Server Farm | Container | `/vcenter`, represents the highest level in the VirtualCenter hierarchy (Server Farm). It is a logical group that acts as a Container. All farm groups, Farms, hosts, virtual machine groups, and virtual machines belong to the `/vcenter` hierarchy. |
| Farm Group | Container | A Farm group is also a logical group that acts as a Container. It belongs to the /vcenter hierarchy, and contains other Farm groups and Farms. For example:<br>• `/vcenter/<Farm_Group>`<br>• `/vcenter/<Farm_Group>/<Farm_Group>` |

| VirtualCenter Term | SDK Term | Possible Paths to the Object |
|---|---|---|
| Farm | Farm | A Farm in the VMware SDK represents the same entity in VirtualCenter. It also belongs to the `/vcenter` hierarchy, and contains hosts, virtual machines, and virtual machine groups. For example:<br>• `/vcenter/<Farm_Group>/<Farm>`<br>• `/vcenter/<Farm>` |
| Host | Host | A host in the VMware SDK represents the same entity in VirtualCenter. It also belongs to the `/vcenter` hierarchy, and is a member of a Farm.<br>The SDK also has a special view, `/host`, that contains all hosts. When performing an operation on a host, you can use either the `/host` hierarchy, or the `/vcenter` hierarchy. For example:<br>• `/vcenter/<Farm_Group>/<Farm>/<host>`<br>• `/vcenter/<Farm>/<host>`<br>• `/host/<host>` |
| Virtual Machine Group | Virtual Machine Group | A virtual machine group in the VMware SDK represents the same entity in VirtualCenter. It also belongs to the `/vcenter` hierarchy, and contains virtual machines and virtual machine groups. For example:<br>• `/vcenter/<Farm_Group>/<Farm>/ <Virtual_Machine_Group>`<br>• `/vcenter/<Farm>/<Virtual_Machine_Group>` |
| Virtual machine | Virtual machine | A virtual machine in the VMware SDK represents the same entity in VirtualCenter. It also belongs to the `/vcenter` hierarchy, and is a member of a virtual machine group.<br>Like `/host`, the SDK also has a special view, `/vm`, that contains all virtual machines. When performing an operation on a virtual machine, you can specify a path in either the `/vm` hierarchy, or the `/vcenter` hierarchy. For example:<br>• `/vcenter/<Farm_Group>/<Farm>/ <Virtual_Machine_Group>/<Virtual_machine>`<br>• `/vcenter/<Farm_Group>/<Farm>/<Virtual_machine>`<br>• `/vcenter/<Farm>/<Virtual_Machine_Group>/ <Virtual_Machine>`<br>• `/vm/<Virtual_Machine>` |

# Path Hierarchy

The VMware VirtualCenter Web Service exports the following predefined paths.

**Note:** Do not attempt to rename any of these predefined paths. If you do so, the Web service may stop abruptly.



The following table describes these predefined paths:

| Path | Description |
|---|---|
| customProperty | Lists the names of all defined customProperties. This list is also available by using the customPropertyDef handle in the HostInfo or VirtualMachineInfo datatypes. This is an array of type Property where:<br>• key is the index of the custom property.<br>• val is the name of the custom property.<br>Clients may update the existing values by using the PutUpdates operation and can add new properties only through the use of VirtualCenter. |
| /datastore | Lists the datastores that are accessible. A datastore indicates a storage location where a virtual machine may place its virtual disk(s) and associated files. The Web service exports the path `/datastore` that identifies an object of type `DatastoreInfoList`, containing information about all the datastores known by the Web service. |
| /event | List of all events and event declarations. |
| /host | The Web service provides a flat name space for all the host machines under management in `/host`. A host is referenced both in the `/host` and in the `/vcenter` directories. Each name in the `/host` directory is represented as a string that is a Fully Qualified Domain Name (FQDN); for example, `esx1.vmware.com`.<br>`/host` enables client programs to reference a host machine without knowing the exact location of the host in the `/vcenter` hierarchy. |
| /perf | Lists all objects that collect performance statistics at various interval rates. |
| /schedule | Lists tasks that are scheduled for the future. Each task runs at its scheduled time. |
| /task | Lists tasks that are currently running or recently finished. |
| /template | Lists all of the templates that an Administrator may make available to clients for cloning operations. |
| /unknownVM | Lists any running virtual machines that have a duplicate UUID. As soon as the virtual machine is powered off, VirtualCenter assigns it a UUID and the Web service removes this virtual machine from the `/unknownVM` directory and places it in the `/vm` directory. |
| /vcenter | Reflects the hierarchy that is present in VirtualCenter. See the VirtualCenter documentation for additional information on Farm groups, Farms, and Virtual machine groups. |

| Path | Description |
|---|---|
| /vm | The Web service provides a flat name space for all the virtual machines under management in `/vm`. Virtual machines are referenced both in the `/vm` and in the `/vcenter` hierarchies. Each name in `/vm` is presented as a string that is a hexadecimal (HEX) representation of the UUID (Universally Unique Identifier); for example, `f81d4fae-7dec-11d0-a765-00a0c91e6bf6`.<br>`/vm` enables client programs to reference a virtual machine without knowing exactly where the virtual machine is located in the `/vcenter` hierarchy. |
| /webservice | Provides information about the VMware VirtualCenter Web Service. |

## Escaped Characters in the VMware VirtualCenter Web Service

Clients need to escape the `%` (percent) character (unless it is being used to start an escape sequence) and the / (slash) character (unless it is being used as a path separator) in the path parameter to the ResolvePath operation and in the name parameter to the Create and Rename operations. For example, clients must send the `%` character as `%25`.

Similarly, the `%` and / characters are escaped in the name parameter in an Item datatype or the path parameter in a Host or VirtualMachine datatype.

**Note:** Clients can choose to escape any additional characters in these parameters.

# Understanding VMware SDK Terminology

The concepts defined in this section are used throughout this guide and the *Virtual Infrastructure SDK Reference Guide*. Familiarize yourself with this terminology to enhance your use of the VMware SDK.

## Object, Item, Path, Handle, and vHandle

This section includes descriptions for the different types of objects and Items (that describe child objects) as well as paths (that identify objects), and handles and vHandles to an object.

### Object

The VMware VirtualCenter Web Service interface provides operations to manage a collection of objects. This is a complete list of objects managed by the Web service:

- Container
- Farm
- VirtualMachineGroup
- Host
- VirtualMachine
- Task
- TaskSchedule
- EventDeclList
- EventCollection
- PerfFilter
- PerfCollection
- Template

See the GetContents operation in the *Virtual Infrastructure SDK Reference Guide* to see where in the hierarchy the various objects appear.

An object has a type associated with it and is described by an XML document, that is the value of the object. If you use the file system analogy, Container, Farm, and VirtualMachineGroup are comparable to directories in a file system while all the other objects are comparable to individual files in a file system.

Nodes of the XML document that represent the object are also referred to as "interior nodes" of the object. For example, the "hardware" node of a VirtualMachine object is considered an interior node of the virtual machine object.

**Item**

An Item is a special element of the Container, Farm and VirtualMachineGroup objects. It provides information about a child object contained in the parent object. An Item has a key, a name, type, and access control list (ACL) associated with it. The path of the parent object may be concatenated with the name of an Item, to form a new path. This new path points to an object whose type is defined by the Item.

The key value is the handle to the object identified by this new path. Clients may pass the handle (key value) as a parameter to the GetContents operation. A ResolvePath operation on the new path returns the same value as the key value, as long as the object identified by the path still exists and is the same object.

For example, executing the GetContents operation on the handle for `/vcenter` returns an object of type Container, that contains an array of Items. Suppose that one of the Items in the array has the name `farm1`. The path `/vcenter/farm1` constitutes a new path and the value for `Item.key` is the handle to this new path.

**Container**

A Container is an object that comprises an array of Items. See the GetContents section in the *Virtual Infrastructure SDK Reference Guide* to see the list of possible nodes (in the hierarchy exported by the Web service) that can be of type Container and also the type of Items that can be in a Container.

For example, Containers represent VirtualCenter Farm groups in `/vcenter`. We also use Containers in the `/host` and `/vm` hierarchies (containing hosts and virtual machines, respectively).

**Farm**

A Farm is an object that comprises an array of Items. We use it to represent VMware VirtualCenter Farms and it appears as nodes in the `/vcenter` hierarchy. Farms may contain only Items of type VirtualMachineGroup, VirtualMachine or Host.

**VirtualMachineGroup**

A VirtualMachineGroup is an object that comprises an array of Items. We use it to represent VMware VirtualCenter Virtual machine groups and it appears as nodes in the `/vcenter` hierarchy. VirtualMachineGroups may only contain Items of type VirtualMachineGroup and VirtualMachine.

**Path**

A path is a string that names an object in the Web service hierarchy just like a full path name names a file in a file system. For example, `/vcenter/FarmGroup/Farm/VMGroup/Win2000` is a path that identifies a virtual machine object while `/vcenter/FarmGroup/Farm/esx01.vmware.com` identifies a host object.

Paths that do not name an object are not valid.

Clients traverse the hierarchy by starting with the / directory or with one of the predefined paths exposed by the Web service. (The list of predefined paths is listed in the preceding section.) The ResolvePath operation takes a path as its only input parameter and returns a handle to the object that is referenced by the path. A GetContents operation on the handle returns the object specified by the handle.

VMware VirtualCenter Web Service paths and names follow the syntactic rules for `abs_path` and `path_segments` as described in section 3 of RFC2396 at *www.ietf.org/rfc/rfc2396.txt*. All paths and names that are communicated between clients and the Web service have reserved characters that are escaped.

See Item on page 27 to learn how to generate new paths and traverse the hierarchy.

**Handle**

A handle is a temporary token used by a Web service client to invoke Web service operations that require a reference to an object. An object handle is analogous to a file handle (descriptor) returned by a operating system. This is similar to UNIX, when a file is opened using the "open" system call. Like a file handle, an object handle is a temporary handle that always refers to the same object.

However, unlike files, "opening" a Web service object and holding its handle does not prevent the object from being deleted. In this regard, object handles are more similar to socket handles than to file handles. The object handle remains unchanged, even if the object is renamed or is moved to a different part of the path hierarchy. A handle is a useful, path-independent way to reference an object.

Handles always provide a unique reference to a given object. There can never be more than one valid handle that refers to the same object in a single session. You may compare handles for equality and check if they denote the same underlying object.

Handles are valid only for the login session. They expire once the session expires. Clients should not store handles and expect to use them at a later date if the session has expired.

**vHandle**

Objects may change over a period of time; for example, users can add a new disk to a virtual machine, the name of a FarmGroup may change, and so on. A vHandle is an object handle that refers to the specific memory state of an object at a certain point in time, and has a version number associated with it. The version number determines the specific memory state.

Versioned handles allow a client application to invoke a GetUpdates operation and retrieve updates that are more recent (newer) than the version number of the versioned handle. Similarly, clients can pass a versioned handle to the PutUpdates operation to perform version checking in

the Web service when updating object values. However, clients should not try to interpret vHandles.

Because clients provide vHandles as arguments to the GetUpdates operation, the Web service doesn't need to send all of the data constituting an object whenever the object changes. Instead, the Web service can send changes to the object, relative to the client-specific vHandle specified in the GetUpdates operation. Two vHandles match if they refer to the exact same version of the same object.

Handles and vHandles may become invalid at any time. For example, this can happen if the underlying object is deleted by another client, or if the Web service is restarted. A client must keep a mapping of the object's path to handles and vHandles. If a Web service operation returns a FaultKind of BadVersion, then the client must try to get a fresh vHandle by using the ResolvePath operation.

## Commonly Used VMware SDK Operations

This section gives a brief overview of three of the most common operations.

### GetContents

A client executes the GetContents operation, passing in a handle as a parameter, in order to retrieve an XML document that describes the object identified by the handle. The XML document is the value of the object. For example, executing the GetContents operation on the handle for `/vm/<UUID>` returns the handle, a vHandle, and an XML document describing the virtual machine object.

### GetUpdates

A client executes the GetUpdates operation, passing in one or more vHandles as a parameter, in order to retrieve the change(s) in the object identified by the vHandle. Only the change(s), or the "delta" is returned as a `diff` of the current XML document. The change(s) reflect the difference between the object as is currently maintained by the Web service, compared with the original XML document initially received by the client as part of the GetContents or GetUpdates operation that returned the specified vHandle.

### PutUpdates

A client executes the PutUpdates operation, passing in a vHandle or a handle and an XML document `diff` (showing only changes) to the object referred to by the vHandle. Clients should not pass in the complete XML document describing the updated object. By passing in only the `diff` (the changes), the client minimizes the amount of data that it must send to the Web service.

Clients can use a handle instead of a vHandle when the client wants to update an object value, regardless of any other updates that might have occurred concurrently. Clients should use a

vHandle when it wishes to update an object handle, but only if the object has not been changed since the version represented by the vHandle.

# Client-Web Service Interactions

The primary interaction between the client and the Web service involves obtaining handles to objects identified by paths, getting XML documents that describe these objects, and then managing these objects by receiving updates when their values change, by making changes to the objects, by committing these changes to the Web service, or by invoking operations on these objects.

By managing these objects, the clients can implement various kinds of application functionality based on VMware virtual machine technology.

- Obtaining the values of objects

  An object is described by an XML document that constitutes the value of the object. Clients either have the handle to the object or get the handle by using the ResolvePath operation. Clients then obtain the values of an object by issuing the GetContents call to the Web service passing the handle as a parameter.

- Obtaining updates for an object

  After a client has the value for an object by using the GetContents operation, it may ask for updates or changes to that object by using the GetUpdates operation. A vHandle to that object is passed as a parameter to the GetUpdates operation. The updates are returned as change objects that describe only the changes to the object, thus avoiding the need to return the entire new object to the client.

- Sending updates of values

  Clients can decide to make changes to objects and then send these changes, as change objects to the Web service, by issuing the PutUpdates operation. The change objects describe only the changes to the object, thus avoiding the need to send the entire object to the Web service. The vHandle or handle to the object is passed as a parameter to the PutUpdates operation.

There are some interesting points to remember about the preceding operations.

- Only the latest updates are returned to the client from a GetUpdates call

  All intermediate updates are lost. For example, if one application changes the memory size of a virtual machine from 1GB to 2GB, and then immediately changes it back to 1GB, it is possible that another application may never realize that this change occurred.

- Versioning

  For change objects to have a meaning, the Web service must have the vHandle of the object on the client. For this reason, all calls to GetContents returns a vHandle for each object. All calls to GetUpdates are required to send a vHandle (not a path or handle), as a parameter that returns a fresh vHandle along with the updates. All calls to PutUpdates are required to send either a handle or a vHandle.

  The client is free to discard vHandles, but it then has to call GetContents each time it wants the latest value of the object, rather than just requesting the changes through GetUpdates. This action has the effect of increasing network traffic.

# Virtual Machine Identification

All virtual machines are listed in the `/vm` directory and have a universally unique identifier (UUID) that specifies the virtual machine during its lifetime, regardless of the host where the virtual machine runs. The UUID does not depend upon the host machine that the virtual machine runs on. The UUID is a 128-bit number represented in hexadecimal, with hyphens as recommended by ISO-11578; for example, `f81d4fae-7dec-11d0-a765-00a0c91e6bf6`.

All virtual machines created by the Web services API or by VirtualCenter are always assigned a UUID. However, virtual machines created by the VMware Management Interface do not have a UUID.

The VirtualCenter product handles this case in the following manner, depending on whether or not VirtualCenter was managing the host when the management interface created the virtual machine.

- Case 1 — VirtualCenter is managing the host:
  - When the management interface creates a virtual machine, the virtual machine is discovered immediately.
  - If the virtual machine is in the powered-off state, then it is assigned a UUID. However, if the virtual machine is powered on or suspended, then it is assigned a UUID when it is next powered off.
- Case 2 — VirtualCenter was not managing the host when the management interface created the virtual machine:
  - When the virtual machine is brought under management of VirtualCenter, the virtual machine is immediately discovered.
  - If the virtual machine is in the powered-off state, then it is assigned a UUID. However, if the virtual machine is powered on or suspended, then it is assigned a UUID when it is next powered off.

A running virtual machine with a duplicate UUID is placed in a special directory, `/unknownVM`. For more information on this situation, refer to the EnableHost operation in the *Virtual Infrastructure SDK Reference Guide*.

# Host Identification

Many Virtual Computing operations take a host as a parameter. The client programs may specify a host by using any one of the following:

- Fully Qualified Domain Name (FQDN); for example, `esx1.vmware.com`
- IP address; for example, 172.10.20.24
- Network node name; for example, `esx1`

Ideally, client programs will use a FQDN to specify a host. If a node name is used, the Web service attempts to look up the node name and convert it into a FQDN. If this succeeds, the Web service returns the FQDN, as part of the GetContents operation and other requests. If the lookup does not return the FQDN, then the node name is used without any changes. However, the Web service continues to accept client requests that use just the node name.

# Session Management

The Web service maintains session state for a client by using a special token in the HTTP header to identify the session. A session token is an identifier given to a Web service client upon login. This identifier is used by the client to invoke operations over multiple connections since with every connection, the identifier must be passed in the HTTP header, over any connection.

Session tokens can be passed across multiple connections to the Web service. A session token expires after a period of inactivity, and may expire after a certain period of time, even if it is actively being used.

By default, there is a SSL connection from the Web service client to the Web service. Although the default SSL port is set to 8443, you had an option to override this port during installation.

If you want to use the non-SSL port 8080 instead, add the following entry to the `vmaConfig.xml` file, located in `C:\Documents And Settings\All Users\Application Data\VMware\VMware VirtualCenter\VMA`.

1. Open `vmaConfig.xml` and look for the `<service>` element.
2. Add the 8080 port number.

   ```
   <port>8080</port>
   ```

**Note:** You must restart the Web service for any changes to `vmaConfig.xml` to take effect.

# Security Model

The VMware VirtualCenter Web Service provides a Login call for clients to authenticate themselves and obtain access privileges. Other Web service operations may only be called after a successful call to Login. The client and Web service use the HTTPS protocol to secure all communication.

The Web service does not manage or expose the directory of users and groups. Instead, it relies on a user directory provided by the host platform.

All Web service paths that identify objects have an access control list (ACL) that defines the users and groups that have the right to perform operations on the identified objects. Each entry in the list is of the form <user/group,rights>, where user/group specifies either a user or a group, and rights is a set of {Browse, Interact, Configure, Administer} flags.

- **Browse** rights allow a user or group to get the value, but not change the value of an object; for example, the ability to discover and monitor a virtual machine. We grant Browse permission to all users and groups, to all top level directories listed in `/` other than `/vcenter`.
- **Interact** rights allow a user or group to perform operations that change the state of a virtual machine, or connect and disconnect removable devices. These rights are analogous to *Execute* rights in file systems; for example, powering on or powering off a virtual machine.
- **Configure** rights allow a user or group to create virtual machine, change resource management settings and add or remove virtual hardware. These rights are analogous to the ability to write to a file; for example, add a new virtual disk or create a new virtual machine.
- **Administer** rights allow a user or group to change the permissions for an object.

Access privileges are inherited by nested paths. For example, if user A has Interact rights for the path `/vcenter,` then this user also automatically gets Interact rights for the path, `/vcenter/farm1`. Note that nested paths can increase access privileges, but cannot decrease them. In this example, the path `/vcenter/farm1` can increase the inherited privileges by setting the rights explicitly to Configure.

**Note:** ACL checking is on by default.

You can turn this off by adding the following entry to the `vmaConfig.xml` file, located in `C:\Documents And Settings\All Users\Application Data\VMware\VMware VirtualCenter\VMA`.

1. Open `vmaConfig.xml` and look for the `<subject>` element.
2. (Optional) Add the following and change the authorization to `false`.

```
<authorizationEnable>false</authorizationEnable>
```

**Note:** You must restart the Web service for any changes to `vmaConfig.xml` to take effect.

# Datastores

A datastore is a storage location for virtual machines. All the files constituting the virtual machine configuration and disks are located in a datastore. On an ESX Server host, each VMFS volume is represented as a datastore. The name of the datastore is the VMFS label.

A datastore path is a generalization of a file path that allows locating virtual machines and virtual disks inside a datastore. It is used as the dataLocator parameter in the VirtualDiskInfo datatype when clients create virtual disks. Similarly, a datastore path is also used as the file parameter in the VirtualMachineSpec datatype, and as the dataLocator parameter in the MoveVM and MigrateVM operations.

The datastore path syntax is:

```
datastore_path ::= <datastore> <filepath>
datastore  ::= empty
               '[' <name> ']'
```

Note the following:

- Datastore paths always use the / separator. (The \ separator is converted on Windows systems).
- The `[<host>//]` notation is used to denote the special non-VMFS location used to store the `.vmx` configuration file and log files of a virtual machine.
- When creating a virtual machine or virtual disk, clients can specify just the datastore. VirtualCenter automatically assigns the file path, based on a standard file-naming convention.
- If the datastore portion is empty, `'[]'`, then the file is not located on a datastore, and the Web service shows the host-local path. (This typically only happens for a virtual machine that has an invalid configuration.)

Here are a few examples to illustrate the datastore name format:

```
[QA-SAN] X.vmdk         # The X.vmdk disk on the shared QA-SAN
[AB14-DS] home/abc/Y.vmdk  # Disk Y.vmdk on home/abc relative to AB14-DS shared datastore
[QA-SAN]                # Root directory of the QA-SAN shared datastore
[]/path/to/x.vmdk        # A host local path
[]g:/path/to/x.vmdk      # Another host local path
[aarhus//] x/x.vmx      # A VMX file stored in the defaultVmDir on an ESX Server host
x.vmdk                  # Disk x.vmdk relative to the virtual machine's configuration file
[] x.vmdk               # Disk x.vmdk relative to the virtual machine's configuration file
[] []                   # A relative filename []
```

CHAPTER 3

# VMware SDK Management Concepts

This chapter describes the following topics:

# Managing Hosts and Virtual Machines

VMware SDK enables you to monitor and manage your hosts and virtual machines.

## Managing Hosts

You can use the VMware SDK to accomplish the following:

- Shut down or restart a host.
- Enable or disable a host for virtual machine operations — When the host is enabled, a client can perform virtual machine operations on the host. When the host is disabled, a client cannot monitor or manage any of the virtual machines on the host through VirtualCenter or the VMware SDK.
- Obtain performance data for the host — CPU, memory, disk I/O, and network utilization.
- Obtain the status of a host.

## Managing Virtual Machines

You can use the VMware SDK to accomplish the following:

- Perform power operations on virtual machines — Power on, power off, suspend, or resume virtual machines.
- Create virtual machines — Create a new virtual machine and specify its name.
- Configure virtual machines — Configure the virtual machine's virtual CPU, memory, disk, network, and hardware.
- Obtain performance data for the virtual machine host — Obtain virtual CPU, memory, disk, and network utilization.
- Monitor virtual machine events — Monitor addition or removal of virtual hardware components, virtual machine creation or deletion, or changes in a virtual machine's power state.
- VMotion™ — Migrate a running virtual machine to a specific host (ESX Server only).
- Deploy templates — Deploy a virtual machine from a template.
- Create templates — Create a template from a virtual machine.

# Life Cycle Operations

These operations are used in the life cycle of hosts and virtual machines.

- Create
- Rename
- Delete

Clients can manipulate the path hierarchy by operating on Containers, Farms and VirtualMachineGroups. Clients may also use the Create operation to bring hosts under management of the Web service and to create new virtual machines.

For more information on these operations, see Creating and Deleting Objects on page 94 and Renaming an Object on page 106.

# Events

The VMware VirtualCenter Web Service records and keeps track of important events (up to 1000 events) that occur in VMware VirtualCenter. Clients can collect events on hosts or on virtual machines, such as changes in power operations or device status (connected, disconnected, or busy). In addition, clients can receive updates when alarms occur; for example, when memory or CPU usage, or virtual machine heartbeat is either above or below normal.

There are five types of events:

- alert — Indicates a problem that requires attention (for example, out of memory errors, hardware failures, and so on)
- error — Indicates an erroneous condition (for example, a system error)
- warning — Indicates a potential problem (for example, CPU utilization is at 95 percent)
- info — Indicates an event occurred; however, no action is required
- user — Indicates a user-event (for example, a user suspended a virtual machine)

Each event comprises two parts: a declaration (the type of event), and the actual event (an event log). For more information on events, see Monitoring Events on page 110.

# Scheduled Tasks

Clients may use the Create operation to create a new task that is scheduled to run at some point in the future. Clients can only schedule the following operations:

- Power operations — StopVM, StartVM, and ResetVM.
- PutUpdates — Changing the resource settings of a virtual machine.
- MigrateVM — Migrates a virtual machine from one host, to another host, without any changes to the location of its virtual disk(s).
- MoveVM — Moves a virtual machine's virtual disk(s) to a different location. Clients may (optionally) also use this operation to move the virtual machine to a different host.
- CloneVM — Creates a new virtual machine by using as its source, an existing virtual machine or a template.
- CreateTemplate — Creates a new template from an existing virtual machine.

You may schedule a task as a single task that runs only once, hourly, daily, weekly, or monthly.

Each task is in one of the following categories:

- scheduled
- starting
- running
- completed
- failed
- killed

We have two terms associated with a task:

- Schedule — Specifies when the task runs.
- Event — A task always generates an event when the task is complete. The task may also generate additional events as it runs.

For more information on tasks, see Task Scheduling and Monitoring on page 114.

# Performance Monitoring

Clients can collect performance data on hosts or on virtual machines, including CPU and memory utilization, network and disk performance data, and floppy and CD-ROM drive performance, and so on. You can specify the frequency of updates (polling period), the number of samples in each update, and the performance data of interest to the client. In addition, you can retain a history for the performance data.

Clients obtain performance statistics through a performance collector. A performance collector, also known as a perf collector, is an object that collects a certain set of statistics at a specified interval frequency.

Use the GetContents and GetUpdates operations to obtain current performance statistics, for a performance monitoring/graphing tool, or similar application. Similarly, use the QueryPerfData operation to obtain historical performance statistics.

For more information on performance monitoring, see Collecting Performance Data on page 119.

# Migrating and Moving Virtual Machines

The VMware SDK differentiates between migrating a virtual machine and moving a virtual machine.

- MigrateVM operation — The MigrateVM operation migrates a virtual machine (its `.vmx` configuration file) from one host to another host, without moving its virtual disk file(s). In this release, the virtual machine must be in the poweredOn state for the MigrateVM operation to succeed.

  **Note:** The MigrateVM operation is not supported for GSX Server in this release.

- MoveVM operation — The MoveVM operation moves a virtual machine's disk file(s) to a different location. It also optionally moves the virtual machine's configuration file (`.vmx`).

## Migrating a Virtual Machine

To migrate a virtual machine, clients must know the handle to the virtual machine that is being moved, and the handle to the destination host. If the dataLocator parameter, describing the location of the virtual machine configuration file on the destination host, is used, then the operation places the `.vmx` file where requested. Otherwise, the system determines the location of the configuration file.

Clients can determine if the migration is successful by monitoring the ViewContents of the task performing the migration or by monitoring the virtual machine state (the detail or host fields). If the virtual machine has been successfully migrated, then the host field should contain the target host handle. For more information, see Migrating a Virtual Machine on page 107.

## Moving a Virtual Machine

To move a virtual machine's disk(s), the virtual machine must be powered off and the client must know the handle to the virtual machine. If the host parameter, providing the handle to the destination host, is added, then both the virtual machine (its `.vmx` configuration file) and its virtual disk(s) are moved.

For example, virtual machine A belongs to host A, that has its VMFS volumes on disk 1. You are planning to move virtual machine A to host B, that has its VMFS volumes on disk 2. If you decide to move virtual machine A's virtual disk(s) to disk 2, then you must also move virtual machine A to host B. If you do not move virtual machine A to host B, it has no access to its virtual disk(s). (Virtual machine A on host A only has access to disk1, and not disk 2.)

Similar to the MigrateVM operation, the client can optionally specify the destination for the virtual disk(s) and if moved, the virtual machine configuration file. If the client does not specify the destination, then the system determines the destination.

**Note:** In this release, the host parameter is required and the dataLocator parameter is ignored.

For more information, see Moving a Virtual Machine's Virtual Disks on page 108.

# Provisioning a Virtual Machine

By provisioning a virtual machine, we are referring to the process of creating a functioning virtual machine by assigning resources such as CPU, memory, and virtual hardware, and then deploying a system image. Traditionally, users need to install the guest operating system and applications manually. However, by using the VirtualCenter user interface or the VMware SDK, users can now create a new virtual machine by cloning an existing one, or by deploying a new virtual machine from a template.

The CloneVM operation creates a new virtual machine by using as its source, an existing virtual machine or a template. Clients can choose to customize a cloned virtual machine by specifying the right parameters. Clients can configure the guest operating system as part of the CloneVM operation by passing in this customization from generated stub classes.

Typically, you provision a new virtual machine from an existing virtual machine that is similar to the new one you are creating. For example, you have a virtual machine A that has Windows 2000 as its operating system, plus a database application. You are interested in creating virtual machine B, also with the Windows 2000 operating system, and the same database application installed in virtual machine A, and an additional backup application.

To create virtual machine B, you have two choices:

- Use the Create operation to create a new virtual machine, then install Windows 2000 and the two applications manually.
- Use the CloneVM operation to copy and customize, resulting in virtual machine B. By using the CloneVM operation, you don't need to perform a manual installation of Windows 2000 and the database application in the new virtual machine.

Clients may also use the CreateTemplate operation to create a template out of an existing virtual machine. (A template is a golden image of a virtual machine, that can include an installed guest operating system and a set of applications). Clients can use this template as a starting point for the creation of new virtual machines by using the CloneVM operation.

For more information, see Cloning a Virtual Machine on page 100.

CHAPTER 4

# Developing Client Applications

This chapter contains the following sections:

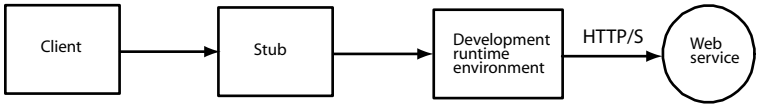


**Note:** The sample code in this chapter, and the following chapters, represents a small portion of code and is designed only to show the basic logic and programming constructs needed to use the VMware SDK effectively. A complete listing of samples is contained in the in `\SDK\WebService\samples` directory.

# Connecting to the VMware VirtualCenter Web Service

In order to connect to the VMware VirtualCenter Web Service, a client application needs to know how to send and receive data from the Web service. In particular, the client needs to know about the Web service's data types, its parameters, return types, location, and transmission details of the VMware VirtualCenter Web Service. The Web Services Description Language (WSDL) is an XML-based language that describes these Web service interface details. The WDSL also describes how the interface is tied to a transport protocol (in this case, secure HTTP) and encoding (in this case, SOAP). From the WSDL, you can generate a stub.

At a high level, the client uses the VMware VirtualCenter Web Service to connect to the Web service as shown in the following diagram.




1. The client invokes a method on a stub (a proxy object) to perform a task. The stub acts as a proxy for the Web service.
2. The development runtime environment (see Client Development Environments on page 267) converts the invocation into a SOAP message. This SOAP message is then transmitted over HTTP/S to the Web service.

In general, you need to complete the following steps to create a Web service client. Perform these steps, or a variation of these steps, depending upon the development environment you select.

1. Obtain the interface details of the VMware VirtualCenter Web Service in `vma.wsdl`.
2. Generate a stub to invoke `vma.wsdl`.
3. Write the client application. See Core Client Programming Concepts for Java Programmers on page 59.
4. Run the client application.

# Reviewing the Web Services Description Language

This section is only intended to provide a brief overview of WSDL concepts. If you need more detailed information, refer to the Web sites listed in Technical Support Resources on page 20.

In general, a WDSL file includes these elements:

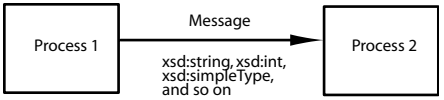
- Datatypes of the Web service interface, comprising its parameters and return types
- Messages, that group data type variables together
- `<portType>` elements, that group incoming, outgoing, and fault messages into logical operations
- Bindings, that describe how a `<portType>` element is mapped to a specific transport protocol
- Services, that describe the connection information for a specific binding

## <types> Element

The `<types>` element allows you to define the data types that are required by the Web service when exchanging messages. Usually, this element also includes a schema element that defines various datatypes.

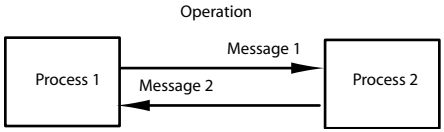
## <message> Element

The `<message>` element describes the logical content of a message that is communicated between two processes. This logical content comprises logical part(s), defined by the `<part>` element. Each `<part>` element includes both name and type attributes, that specify the message part name and its datatype.




## <portType> Element

The `<portType>` element groups `<message>` elements into logical operations that a process can execute. Each operation can contain `<input>, <output>` and `<fault>` elements, corresponding to incoming, outgoing, and fault messages. The `<input>` element comprises the requirements for a client transmission request to the Web service. Similarly, the `<output>` element declares the contents of the response from the Web service. The `<fault>` element describes exceptions that have occurred as the Web service responds to the client's request.




## <binding> Element

The `<binding>` element in the WSDL describes a supported protocol. Similar to the `<portType>` element, the `<binding>` element includes supported operations, as well as the `input>, <output>` and `<fault>` elements, for each operation. The bindings specify the protocol that is being used, the data transport method, and the location of the Web service.

The `<binding>` element includes both a name attribute and a type attribute that references the `<portType>` element. The name attribute provides a unique name for this binding. The type attribute identifies the port type that it binds. (The port type is defined in the `<portType>` element.

## <service> Element

The `<service>` element defines a service (a group of related ports), that are supported by the Web service. There is one port element for each of the supported transmission protocols.

Each `<service>` element identifies the related ports by using the `<port>` child element to identify each port. The `<port>` element has name and binding attributes and one child element, the `<soap:address>` element that provides the address of the Web service's SOAP request handler.

# Selecting a Development Environment

A Web service development environment facilitates the building of Web service servers and clients. For more information on selecting a development environment, see Client Development Environments on page 267.

**Note:** The examples in this chapter are written in Java and Perl. The Java samples are based on a specific developer environment, the Websphere Software Developer Kit (WSDK) for Web services. You may use a different developer environment and language to build your client program. Adjust the examples accordingly for your developer environment.

# Generating the Stub Files

This section shows you how to use the WSDK to perform a set of one-time operations to prepare yourself to build client applications.

**Note:** The following is written in Java and is intended solely as an example.

1. Include the following `.jar` files from the WSDK directory, `appserver/lib` to your CLASSPATH variable:

   ```
   webservices.jar
   commons-logging-api.jar
   j2ee.jar
   qname.jar
   ```

2. Generate the stub files by typing:

   ```
   WSDL2Client -NStoPkg urn:vma1=com.vmware.vma -project . vma.wsdl
   ```

   The first argument to `WSDL2Client` (`-NStoPkg urn:vma1=com.vmware.vma`) specifies a namespace translation from the Web service namespace to the Java namespace. Essentially, all the stub code classes are generated into the package, `com.vmware.vma`.

   The second argument (`-project .`) specifies where the stub code is generated (into the current directory). Specifically, the stubs are generated into the directory `./client-side/com/vmware/vma`.

   The third argument (`vma.wsdl`) is the name of the WSDL file we provide. We assume this file is also present in the current directory, otherwise you should type the complete path to this file.

   If this command works correctly, you should see a large number of Java files in the directory, `./client-side/com/vmware/vma`.

3. Add the directory `./client-side` to your CLASSPATH variable and compile (using `javac`) all the files in the directory `./client-side/com/vmware/vma`.

   We are now ready to build our clients.

# Communicating Securely

All communication between the client program and the Web service occurs over SSL. During the establishment of the SSL connection, the Web service submits its digital certificate to the client. The client program must have loaded the root certificate in order to authenticate the Web service.

## Enabling Java Client SSL Connections

Complete the following setup before communicating with the Web service.

1. Create a directory to store the SSL certificates; for example, the `C:\sslCert` directory.
2. Copy the root certificate to the directory created in the previous step. In a terminal window, type the following at the command line. If you saved the SSL certificates in another directory, substitute that directory location for the `C:\sslCert` directory.

   ```
   copy "C:\Documents And Settings\All Users\Application
   Data\VMware\VMware VirtualCenter\VMA\root.pem" " C:\sslCert"
   ```

3. Import the root certificate. JDK 1.4.1 supplies a tool called keytool, that you may use to import the root certificate. For example:

```
keytool.exe -import -storepass <password> -keystore C:\sslCert\client.keystore
-alias rootCert -file C:\sslCert\root.pem
```

4. The keytool prompts you, `Trust the certificate?` Type yes.
5. The following Java sample code snippets for the IBM Websphere and Apache Axis environments illustrate how to specify the location of the key material for the TrustManager. This overrides the default `cacerts` file in the (Java Runtime Environment) JRE `lib` directory.

### IBM Websphere

```
if (args.length > 3)
{
   // Specify the location of where to find key material for the TrustManager.
   // This overrides the default cacerts file in the JRE lib directory.
   System.setProperty("javax.net.ssl.trustStore", args[3]);
}
System.setProperty("java.protocol.handler.pkgs", "com.ibm.net.ssl.www.protocol");
```

### Apache Axis

```
if (args.length > 3)
{
   // Specify the location of where to find key material for the TrustManager.
   // This overrides the default cacerts file in the JRE lib directory.
   System.setProperty("javax.net.ssl.trustStore", args[3]);
}
```

# Creating a Simple Client

We are now ready to create our first SDK client. This client requests the list of all virtual machines known to the Web service server by making a call to GetContents with `/vm` as the argument. The client code is shown below. This Java code is also present as part of the sample code shipped with the SDK.

First, this program connects to the Web service server and makes the connection persistent (to maintain sessions). Then the client logs into the Web service server, followed by a call to the Web service operation, ResolvePath. This returns a handle for the `/vm` Container. The client then passes this handle to the GetContents operation to retrieve the contents of the `/vm` Container.

The Container has an array of Items. Each Item contains the UUIDs of each virtual machine. Finally, there is the loop that prints out all the virtual machine UUIDs. Finally, log out of the Web service server.

## Simple Client Program in Java

```
package com.vmware.sample.Simple;

import java.net.*;
import com.vmware.vma.*;
import javax.xml.rpc.*;

public class SimpleClient
{
    /**
     * This main program takes 3 arguments - the web service URL, the
     * user name and password.  It then connects to the Web service server at
     * the URL using the provided user name and password.  Finally it
     * does a GetContents on /vm and prints out the UUIDs of the VMs.
     */
    public static void main(String[] args)
        throws Exception
    {
        if (args.length != 3)
        {
            System.out.println("Usage: java " +
                "com.vmware.sample.Simple.SimpleClient " +
                "<webserviceurl> <userid> <password>");
            System.exit(0);
        }

        // Initialize the connection to the server with a persistent
        // connection.
```

```
        VmaService vmaservice = new VmaServiceLocator();
        VmaPortType serviceConnection = new VmaBindingStub(new URL(args[0]), vmaservice);
        ((Stub)vmaPort)._setProperty(Stub.SESSION_MAINTAIN_PROPERTY,
            Boolean.TRUE);

        // Logs in to the server.

        serviceConnection.login(args[1], args[2]);

        // Get handle for /vm
        String handle = serviceConnection.resolvePath("/vm");

        // Now calls GetContents on /vm.
        ViewContents vc = serviceConnection.getContents(handle);

        // Print the uuid's of all virtual machines.

        Container vmlist = (Container)(vc.getBody());
        Item[] vms = vmlist.getItem();
        if (vms == null)
        {
            System.out.println("There are no VMs");
        }
        else
        {
            System.out.println("The uuid's of the VMs are:");
            for (int i = 0; i < vms.length; i++)
            {
                System.out.println("    " + vms[i].getName());
                // vms[i].getKey() is the handle for this VM
            }
        }

        // Now log out.
        serviceConnection.logout();
    }

}
```

## Simple Client Program in Perl

You can find the complete sample program at `/SDK/WebService/samples/perl/SimpleClient.pl`.

```
#!/usr/bin/perl -w

use HTTP::Cookies;
```

```
use SOAP::Lite;
use Tie::IxHash;


sub Usage
{
    print "\nUsage:perl SimpleClient.pl <webserviceurl> <userid>";
    print " <password> [<wsdl>]\n";
    print "\nNOTE: Special characters like \$ should be escaped.\n";
}


sub PrintUUID
{
    my $result = shift;

      #
      # Extract the list of names of the VMs from the result
      # and display
      #

      my @names = $result->valueof('//name');

      foreach my $name (@names){
            print "\n$name";
      }

    return;
}


sub FaultHandler
{
    my($soapErr, $resultErr) = @_;

    die ref $resultErr ?
        "\nFault: ".$resultErr->faultdetail->{'FaultInfo'}->{'kind'}."\n".
      $resultErr->faultdetail->{'FaultInfo'}->{'info'} :
       "\nFault: ".$soapErr->transport->status, "\n";
}


######################################################################
#
# Main program --
#
######################################################################
```

```
if ($#ARGV+1 < 3 || $#ARGV+1 > 4){
    Usage();
    exit;
}

my ($webserviceURL, $userID, $password, $vmaWSDL) =@ARGV;


if ( !defined($vmaWSDL) ){
    $vmaWSDL = $webserviceURL."?wsdl";
}

#
# Create SOAP::Lite object by specifying location of vma.wsdl and the URL
# of the web service. Session information is automatically maintained by
# enabling cookies. A  fault handler is also defined to handle SOAP
# faults as well as transport errors.
#

my $service =
  SOAP::Lite
    -> service($vmaWSDL)
    -> proxy($webserviceURL,
          cookie_jar => HTTP::Cookies->new(ignore_discard => 1))
    -> on_fault(sub{FaultHandler(@_);});


my ($method, @params);

#
# Setup & call the Login method
#

$method = SOAP::Data->name('Login')
    ->attr({xmlns => 'urn:vma1'});

@params = (
            SOAP::Data->name(userName => $userID),
            SOAP::Data->name(password => $password)
           );

$service->call($method => @params)->result;


#
# Setup & call the ResolvePath method
```

```
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(path => '/vm'));

$handle = $service->call($method => @params)->result;


#
# Setup and call the GetContents method
#

$method = SOAP::Data->name('GetContents')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(handle => $handle));


my $result = $service->call($method => @params);

print "\nUUIDs of the Virtual Machines are:\n";

PrintUUID($result);


#
# Setup & call the Logout method
#


$method = SOAP::Data->name('Logout')
    ->attr({xmlns => 'urn:vma1'});

$service->call($method);
```

# Compiling the Java Client Application

To compile the client, add the directory containing `SimpleClient.java` to the CLASSPATH variable, and then compile `SimpleClient.java` using `javac`.

# Running the Client Application

This client application returns a list of all the virtual machines. If there are no virtual machines, then use the VMware VirtualCenter interface to create some.

Be sure the Web service server is running. Refer to the VirtualCenter documentation for instructions on how to start this server.

For the Java sample, type:

```
java SimpleClient <webserviceurl> <username> <password>
```

For the Perl sample, type:

```
perl SimpleClient.pl <WebService_URL> <username> <password> <wsdl>
```

where `<WebServer_URL>` is the VirtualCenter URL, `<username>` is the user ID for logging into VirtualCenter, `<password>` is the corresponding password for `<username>`, and `<wsdl>` is an optional parameter that specifies the location of the `vma.wsdl` file.

The client should print out a list similar to the following:

```
The uuid’s of the VMs are:
    564d08e5-6253-7e43-7740-774b3dc0cfd2
    564d3040-b31a-cb26-6a7d-989550c5bd7a
```

If you do not see a listing of virtual machines, then see Troubleshooting on page 275.

CHAPTER 5

# Core Client Programming Concepts for Java Programmers

This chapter shows you how to write a good client application. The previous chapter introduced a very simple application that performed a GetContents operation.

**Note:** Similar to the previous chapter, the examples in this chapter are written in Java, and are based on a specific developer environment, the Websphere Software Developer Kit (WSDK) for Web services. See IBM Websphere Software Developer Kit on page 269.

If you are interested in Perl samples, see Core Client Concepts for Perl Programmers on page 133.

**Note:** You may use a different developer environment and language to build your client program. Adjust the examples accordingly for your developer environment.

This chapter covers the following topics:

# Logging into the Web Service

The following sample illustrates how to log into the Web service using the WSDK.

```
// Initialize connection and login to the Web service.
String userId = "Administrator";
String password = "password";
java.net.URL serviceURL = new java.net.URL("http://<hostname>:8080");
VmaService vmaservice = new VmaServiceLocator();
VmaPortType serviceConnection = new VmaBindingStub(serviceURL, vmaservice);
((Stub)vmaPort)._setProperty(Stub.SESSION_MAINTAIN_PROPERTY, Boolean.TRUE);

// Logs in to the server.
serviceConnection.login(userId, password);
```

# Permissions

The client must have the right set of permissions on the object being updated or retrieved in order to invoke the different operations. For example, the client must have Browse rights for the GetContents operation, Interact rights for the GetUpdates operation, Configure rights for the PutUpdates operation, and Administer rights for the ChangePermissions operation. Refer to the *Virtual Infrastructure SDK Reference Guide* for the required permission for each operation. Also, see Changing Permissions on page 127 for an example of changing the permission of an object.

# Getting Basic Information about an Object

Client applications can use the GetInfo operation to retrieve information about an object. This operation takes one argument, the handle of the object of interest. The return value is a ViewInfo object that contains the handle of the parent Container of this object, the name of the object, the type of the object, and the list of permissions on the object.

The following code snippet demonstrates how to invoke the GetInfo operation and retrieve the result.

```
String handle = serviceConnection.resolvePath("/vcenter/MyFarmGroup/Farm/myVM");
ViewInfo info = serviceConnection.getInfo(handle);
String parentHandle = info.getParent();
String name = info.getName();
String type = info.getType();
Permission[] perms = info.getPerm();
if (perms == null) {
    // no perms
} else {
    for (int i = 0; i < perms.length; i++) {
        String rights = perms[i].getRights();
        String userName = perms[i].getKey();
    }
}
```

# Object Inventory

You can write a client application to obtain a list of hosts or virtual machines available to you. Once you obtain these lists, you can request for information about a specific host or virtual machine. Then, you can request updates when they occur.

Clients use the GetContents operation to return information about the following objects, provided you have Browse access to these objects.

- Containers (Server Farms and Farm groups)
- Farm
- Host
- Virtual machine group
- Virtual machines

## Using GetContents to Obtain Information About Hosts and Virtual Machines

For example, if you wanted to obtain a list of all the hosts, you can issue a GetContents request on `/host`. Similarly, if you wanted to obtain a list of all virtual machines, for all hosts, you can issue a GetContents request on `/vm`, as illustrated in Creating a Simple Client on page 52.

For example, if you have Browse rights on two virtual machines, identified by UUIDs 000111 and 000112, then the XML document returned from the GetContents request for `/vm` is:

```
<handle>vma-0000-0000-0006</handle>
<vHandle>vma-0000-0000-0006@fbc7b64959000002</vHandle>
<body xsi:type="Container">
    <item xsi:type="Item">
          <key>vma-vm-00000000001</key>
          <name>000111 </name>
          <type>VirtualMachine</type>
    </item>
    <item xsi:type="Item">
          <key>vma-vm-00000000001</key>
          <name>000112</name>
          <type>VirtualMachine</type>
    </item>
</body>
```

## Using GetContents to Obtain Information About Individual Hosts and Virtual Machines

Once you obtain the list of hosts (virtual machines), you can use the GetContents operation to obtain detailed information about an individual host machine or virtual machine. For example, by issuing a GetContents request on a specific host, you can see information about its hardware, status, resource allocations, networking devices, and so on, as indicated by the Host Machine data model in the *Virtual Infrastructure SDK Reference Guide*.

Similarly, by issuing a GetContents request on a specific virtual machine, you can see information about its virtual hardware, status, resource allocations, virtual networking, guest operating system, and so on, as indicated by the Virtual Machine data model in the *Virtual Infrastructure SDK Reference Guide*.

The following sample illustrates how to use the GetContents operation for a host view. Clients can obtain the host handle by calling the ResolvePath operation on either its nested path under the `/vcenter` hierarchy or by using its FQDN (Fully qualified domain name) under `/host`. The example below uses the host's nested path in the `/vcenter` hierarchy.

```
// Get Contents for /vcenter/New Farm Group/New Farm/<Fully qualified hostname>
// Alternate path for host could be /host/<mytesthost.mydomain.com>
String path = "/vcenter/<New Farm Group>/<New Farm>/<Fully qualified hostname>"
ViewContents vc = null;
try {
   // serviceConnection is com.vmware.vma.VmaPortType
   String hostHandle = serviceConnection.resolvePath(path);
   vc = serviceConnection.getContents(hostHandle);
}
catch (Exception ex) {
   System.out.println("Got Exception calling getContents : " +
   ex.getMessage());
   ex.printStackTrace(System.out);
   throw ex;
}

//This can be used in a subsequent call to getUpdates()
String vHandle = vc.getVHandle();

// The xml document representation describing this host object
Object bean = vc.getBody();
```

# The Basic Data Synchronization Loop

In the previous chapter, we developed an example to perform the GetContents operation on the `/vm` view. (See Creating a Simple Client on page 52.) The operation returns a data structure containing the list of the available virtual machines. This list contains handles that refer to the XML documents corresponding to each virtual machine.

We'll start with building the data synchronization loop for a single virtual machine; for example, `/vm/564d08e5-6253-7e43-7740-774b3dc0cfd2`. The Java code below replaces the portion of the code between the calls to `login` and `logout` in Creating a Simple Client on page 52.

For now, let us start with a simple first version that simply calls GetContents repeatedly:

```
com.vmware.vma.VirtualMachine clientData;
com.vmware.vma.ViewContents vc;
String handle = serviceConnection.resolvePath("/vm/564d08e5-6253-7e43-7740-774b3dc0cfd2");
while (true) {
   vc = serviceConnection.getContents(handle);
   clientData = (VirtualMachine) (vc.getBody());
   Thread.sleep(10000);
}
```

This code polls the Web service server every 10 seconds to obtain a fresh copy of the data corresponding to a specific virtual machine. The preceding code may work for a simple application, but it will not scale to a system that is managing a large number of virtual machines and other entities, due to the large amount of network traffic it generates.

# Versions and Handles

In this section, we provide a brief description of versions, handles, and vHandles. For complete information on handles, version numbers, paths, and the GetContents and GetUpdates operations, see Understanding VMware SDK Terminology on page 26.

A handle is a temporary token, used by a Web service client, to invoke Web service operations that require a reference to an object. An object handle is somewhat analogous to a file handle (descriptor) returned by a operating system similar to UNIX, when a file is opened using the "open" system call. Like a file handle, an object handle is a temporary handle that always refers to the same object.

A vHandle is a versioned handle that acts as a reference to the specific memory state of an object at a certain point in time. That is, a vHandle is an object handle that has a version number associated with it. The version number determines the specific memory state. Each version identifier corresponds to a different point in time.

The path resembles a full path name of a file, which identifies an object in the Web service hierarchy. Its value is an XML document in the Web services world. This value is different in other worlds. (For example, in the Java world, the value is a Java object derived from the Web service document.)

A client executes the GetContents operation, passing in a handle (to an object) as a parameter and retrieving an XML document that describes the object identified by the handle. The XML document is the value of the object and is associated with a handle and a vHandle that are returned by the GetContents operation.

A client executes the GetUpdates operation, passing in a vHandle as a parameter and retrieving the change(s) in the object identified by the vHandle. Only the change(s), or the "delta" is returned as a `diff` of the current XML document that describes the object as currently maintained by the Web service, compared with the original XML document as identified by the vHandle that was passed to the GetUpdates operation. The GetUpdates operation also returns an updated vHandle, identifying the latest version of the object that was returned to the client.

For additional information on handles, version numbers, paths, and the GetContents and GetUpdates operations, refer to the *Virtual Infrastructure SDK Reference Guide*.

# Calling the GetUpdates Operation

In order to keep data on the client side up-to-date with data on the server side, you can combine the use of the GetContents operation with GetUpdates.

- The GetContents operation provides all the information for a particular object at the time of the request.
- The GetUpdates operation returns incremental changes to the data, that can be patched onto the existing data on the client. Theoretically, one can call GetContents repeatedly, but this causes excessive network traffic.

For example, you have issued a GetContents request on a specific virtual machine. The response is a list of the information specified by the Virtual Machine data model. Someone, with Interact rights on this virtual machine, changes the memory shares. If your client then issues a GetUpdates request, you can see the updated memory shares and any other changes that have occurred since the original GetContents request.

The first refinement is to replace the repeated calls to GetContents with calls to GetUpdates. By using GetUpdates instead, the server only returns the changes within the VirtualMachine data structure, and not the entire new version of the VirtualMachine object.

```
ViewContents vc = serviceConnection.resolvePath("/vm/564d5a05-29a7-b09b-d576-9cb8a719d940");
VirtualMachine clientData = (VirtualMachine) (vc.getBody());
boolean wait = true;
while (true)
{
   VHandleList vHandleList = new VHandleList();
   vHandleList.setVHandle(new String[] { vc.getVHandle() });
   UpdateList updateList = serviceConnection.getUpdates(vHandleList, wait);
   Update [] updates = updateList.getUpdate();
   // Since we've requested updates only on 1 vHandle, updates.length will be 1
   for (int i = 0; i < updates.length; i++)
   {
      String handle = updates[i].getHandle();
      Change[] changes = updates[i].getChange();
      for (int j = 0; j < changes.length; j++)
      {
          //… apply change to client data …
          processChange(handle, changes[j]);
      }
      vc.setVHandle(updates[i].getVHandle());
   }
}
```

Note the following comments regarding the preceding code:

- We are not handling any exceptions; these are covered later.
- We have not shown how to apply changes to the client data; this is covered in the next section.
- The call to GetUpdates blocks indefinitely, until there is some change that causes the server to send back a response. In some circumstances, particularly in the presence of intermediaries (such as proxies), the client and server can get disconnected with the client remaining blocked on the GetUpdates call. A lower layer disconnection mechanism, such as TCP timeout, is required to handle this case.

  If the second parameter to the GetUpdates call is set to false, then the call doesn't block indefinitely. The call returns immediately with any available updates. (This is useful if the client needs to check for any immediate changes.) Most client applications use the blocking version of the GetUpdates call to receive updates from the Web service.
- When a client calls GetContents, it receives the value of the object identified by the handle, along with a vHandle. Then, when the client makes calls to GetUpdates, it sends the vHandle to the Web service. The Web service returns a set of changes made to the object identified by the vHandle, and an updated vHandle, identifying the latest version of the object. See Versions and Handles on page 67.

  In this example, we are working with one particular view. Therefore, the parameter to GetUpdates is a list containing a single vHandle object. However, clients can also send a list of multiple vHandles to GetUpdates.
- GetUpdates returns a list of Update objects, with one object per vHandle.

  In this case, the return value of GetUpdates can either contain an empty list (if no changes occurred at the time of return) or a list with a single Update object. The list cannot contain more than one Update object because the parameter to GetUpdates contained only a single vHandle.

# Applying Changes to the Client Data

In the following sections, we expand on the following piece of code left out of the previous section:

```
...apply change to clientData...;
```

This change is presented as an object of type Change. Change objects are quite complex, as there are many kinds of changes and we attempt to provide changes in increments as small as possible. We first present a high-level overview of the Change object, and then cover the various kinds of changes on a case-by-case basis.

## The Change Object

Each field of the Change object is described briefly in this section.

### op Field

The primary field of the Change object is `op`, obtained by calling `getOp()`. This field describes the kind of change.

The kind of change is one of the following:

- `ins` — Describes the insertion of new data at the location specified by the target field.
- `del` — Describes the deletion of existing data at the location specified by the target field.
- `repl` — Describes the replacement of existing data at the location specified by the target field. You can consider a `repl` operation as a combination of a `del` operation, followed by an `ins` operation.
- `edit` — Describes a change to existing data. Portions of the data can be changed, while leaving other portions unchanged. This edit change applies only to primitives and objects of type xsd:string and xsd:dateTime.
- `move` — Describes a change where the Web service indicates a move of an object from one (source) Container to a (destination) Container. This is similar to the cut and paste operation. In the cut and paste operation, two move changes are received for the object. The parent of the object that is cut receives one move change and the recipient of the paste operation receives the second move change.

### target Field

The `target` field, obtained by calling `getTarget()`, describes the location where the change applies within the View object The target is specified as a list of identifiers separated by slashes, that indicate how to navigate within the data structure to reach the desired field. For more information on the data structure, see the data models in the *Virtual Infrastructure SDK Reference Guide*.

There are two broad categories of targets that clients can use to communicate changes by using the Change datatype: leaf values and composites.

- Leaf value — An example of a leaf value is the value "My Great Virtual Machine" for the "name" field of the VirtualMachineInfo datatype. Leaf values correspond to text nodes within an XML document. This includes xsd:string, xsd:int, xsd:long and other similar primitive datatypes.
- Composites — A composite or aggregate is an entire object such as an object of type Item, Host, or VirtualMachine, or an interior node of such an object such as the VirtualNetworkAdapter inside the VirtualMachine datatype.

For example, if the target is `hardware/memory/sizeMb` (within a VirtualMachine object), then the corresponding field within the client data structure is obtained by calling:

```
clientData.getHardware().getMemory().getSizeMb()
```

and this field is set to a new value by calling:

```
clientData.getHardware().getMemory().setSizeMb(newValue)
```

To perform these operations on the specified path, we need to use the reflection capabilities of Java. The library classes `java.beans.Expression` and `java.beans.Statement` greatly facilitate performing these operations. These classes are available only from Java Development Kit (JDK) version 1.4. For earlier JDK versions, you need to code the functionality within these library classes.

The target field may also contain array indexing; for example, `hardware/disk[1]/controllerId`. If the location of the change is an array element, then the target ends with an array index; for example, `hardware/disk[2]`.

Refer to keyed arrays by using the key of the array element, instead of the array index; for example, `hardware/net/adapter["_nic001"]` or `hardware/net/adapter["_nic001"]/name`. The string enclosed by the quotes within the square brackets is the key of the array element; for example, `_nic001` is the key in the preceding example.

For more information on arrays, see Indexed and Key-based Arrays on page 77.

#### val Field

The Change object field `val`, obtained by calling `getVal()`, specifies the new leaf value or composite of the field specified in the target field. In some cases (such as `del` changes), this field is not relevant and you can ignore it.

#### inserted, deleted and editPos Fields

There are three more fields in the Change object:

- `inserted` — For `ins` and `repl` changes, this field specifies the number of characters that are inserted. For `move` changes, this identifies the paste operation with `inserted = 1.`

- `deleted` — For `repl` and `del` changes, this field specifies the number of characters that are deleted. For `move` changes, this identifies the cut operation with `deleted = 1`.
- `editPos` — This field is valid only for `edit` changes of string values. It specifies the location in the string where the edit starts.

## Processing the Various Kinds of Change

The following sections explain the various kinds of changes and how to process them:


- Insert Change on page 73
- Delete Change on page 74
- Replace Change on page 74
- Edit Change on page 74
- Move Change on page 76


### Insert, Delete, or Replace (ins, del, or repl) Change Operations

Clients can send these change operations for composites, for both arrays and non-arrays. The usage of Change fields for `ins`, `del`, or `repl` is:

- `target` — Composite that is being inserted, deleted, or replaced. An example is `hardware/net/adapter["#_nic0"]` where the network adapter with key `"#_nic0"` is either being inserted, deleted, or replaced.
- `editPos` — Not Applicable.
- `deleted` — Number of composite nodes being deleted.
- `inserted` — Number of composite nodes being inserted.
- `val` — Contains the new composite or set of composites being inserted or replaced. When the change operation is delete, this field is NULL.

As an example, a single replace change can replace one `#_nic0` adapter with two adapters `#_nic1` and `#_nic2`. In this case, the changeOp is `repl`, the `target` is `hardware/net/adapter["#_nic0"]`, `deleted` is `1`, `inserted` is `2`, and `val` contains an array of the two new adapters `#_nic1` and `#_nic2`. Note that the keys are assigned by the Web service.

- When the array is an indexed array, the target contains the index of the array element before or at the location where the operation occurs. For example, if `target` is `vm[1]`, `deleted` is `0`, `inserted` is `2`, and `val` contains two new `vm` array elements, then the new array elements are inserted before the existing `vm[1]`. The first of these new elements becomes `vm[1]`, the second new element becomes `vm[2]` and the old `vm[1]` is now `vm[3]`. In this same example, if `deleted` is also set to `1`, then the old `vm[1]` gets deleted and the two new elements are inserted in its place as `vm[1]` and `vm[2]`.

### Insert Change

The insert change may be a change within an object, or it can be an insertion of a new top-level object. For example, if the client is querying for updates on an top-level Container such as `/vm`, then the client gets notified about new virtual machines. Similarly, querying for updates on `/vcenter`, the Web service can notify the client of new Farms, Farm groups, virtual machines, virtual machine groups, hosts, and so on.

The code snippet below illustrates how to process these changes.

```
String target = change.getTarget();
if (target.startsWith("item[\""))
    …insert new top level object…
else
    …insert into existing object…
```

For top-level object insertions, the Web service sends the handle of the new object. The client must call getContents on this handle to obtain the contents of this new object.

```
String target = change.getTarget();
Item item = (Item) change.getVal();
String handleBeingInserted= item.getKey();
// call getContents() on this handle now
```

Insertions into existing objects can either be insertions of new objects or insertions of new array elements.

We use reflection to access the object being changed, as identified in the change target. By using reflection, we get the field Type of the object being changed and use this to determine if it's a change to an array.

```
Object body = viewContents.getBody();
Object toBeChanged = SampleUtil.findObject(body, change.getTarget());
ChangeData changeData = SampleUtil.createChangeData(toBeChanged, change.getTarget());

if (changeData.getFieldType().isArray())
    SampleUtil.insertArrayEntries(changeData, change.getInserted(), change);
else
    changeData.setFieldValue(change.getVal());
```

The `findObject()`, `getFieldType()`, and `setFieldValue()` methods used in the preceding code fragment are straightforward uses of Java's reflection mechanism. We include complete code for these methods, along with the complete code for the Discovery example in the samples directory downloaded in the SDK package.

### Delete Change

A delete change is analogous to the insert change described previously. Delete changes can either be deletions of top-level objects or deletions within existing objects.

```
String target = change.getTarget();
if (target.startsWith("item[\""))
    int startIndex = "item[\"".length();
    String handleBeingDeleted = target.substring(startIndex,target.length() - 2);
    …delete top level object…
else
    …delete from existing object…
```

Delete changes from existing objects can be either deletions of whole objects, or deletions of array elements.

```
        if (changeData.getFieldType().isArray())
            SampleUtil.deleteArrayEntries(changeData, change.getDeleted());
        else
            changeData.setFieldValue(null);
```

### Replace Change

The replace change is a combination of the insert and delete changes and is processed as such by the Web service.

```
    public void processReplace(String handle, Change change)
        throws Exception
    {
        processDelete(handle, change);
        processInsert(handle, change);
    }
```

### Edit Change

The client may send an edit change operation only when changing leaf values. Similarly the Web service will send the edit change operation only when changing leaf values. We can further categorize leaf values as string values or non-string values.

***String Values —*** These are values of type xsd:string. The following fields of Change are relevant when string values are edited. The usage of these fields is identical for the GetUpdates and PutUpdates operations.

- `target` — Interior node whose value is being edited. An example is `Item["xxx"]/name` when the value for the interior 'name' node is being edited for an Item whose handle is `"xxx"`.

- `editPos` — For deletions, this specifies the starting position of the substring being deleted. For insertions, this is the character position before which a substring is inserted. If this field is NULL, it must be interpreted as 0 (zero).
- `deleted` — Number of characters being deleted starting at editPos. If this field is NULL, it must be interpreted as 0 (zero).
- `inserted` — Number of characters being inserted before the editPos character. If this field is NULL, it must be interpreted as 0 (zero).
- `val` — this contains the substring to be inserted if the 'inserted' field is non-zero.

If an entire string is being replaced by using the edit operation, then `editPos` is `0` (zero), `deleted` is the length of the old string, `inserted` is the length of the new string, and `val` contains the new string. This is a common case where the edit operation is used to replace an entire string.

***Non-string values —*** These are leaf values not of type xsd:string. The edit operation for such values always replaces the existing value with a new value. The client can ignore the `editPos`, `inserted`, and `deleted` fields of Change. Similarly, when doing a PutUpdates operation, the Web service ignores these fields for non-string values. The usage of Change fields for edit for non-string values is:

- `target` — Interior node whose value is being edited. An example is `hardware/cpu/controls/shares` where the value for the shares node is being replaced for a virtual machine.
- `editPos` — Not Applicable.
- `deleted` — Not Applicable.
- `inserted` — Not Applicable.
- `val` — Contains the new value that replaces the old value on the target.

The following code fragment illustrates how to apply changes to string values:

```
Class fieldClass = changeData.getFieldType();
if (fieldClass.equals(String.class))
{
    int editIndex = 0;
    if (change.getEditPos() != null)
        editIndex = change.getEditPos().intValue();

    // Retrieve the string value that is currently set in the object
    // Edit the string and set the new string into the object
    String string = (String) changeData.getFieldValue();
    int numInserted = change.getInserted().intValue();
```

```
        int numDeleted = change.getDeleted().intValue();
        StringBuffer sb = new StringBuffer(string);
        sb.delete(editIndex, editIndex + numDeleted);
        sb.insert(editIndex, change.getVal());
        changeData.setFieldValue(sb.toString());
    }
```

The following code fragment illustrates how to apply edit changes to other non-String objects.

```
if (fieldClass.isPrimitive() || fieldClass.equals(Calendar.class))
    changeData.setFieldValue(change.getVal());
```

**Move Change**

The Web service uses this change operation to indicate a move of an object from one Container to another. This operation can never be used from a client to the Web service in a PutUpdates call. To move objects, clients must instead use the Rename operation.

The cut operation is identified as a move operation with `change.deleted = 1`, and the corresponding paste operation is received from the Web service along with the move change and `change.inserted = 1`.

The value in the change object hasn't changed as part of the move operation and is therefore considered uninteresting from the move operation's perspective.

```
int numInserted = change.getInserted().intValue();
int numDeleted = change.getDeleted().intValue();

String target = change.getTarget();
int startIndex = "item[\"".length();
String handleBeingMoved = target.substring(startIndex, target.length() - 2);

if (numInserted == 0 && numDeleted == 1)
{
    // Cut operation
    …adjust the client data structure
}
else if (numInserted == 1 && numDeleted == 0)
{
    // Paste operation
    …adjust the client data structure
}
```

# Indexed and Key-based Arrays

Some elements in the VMware VirtualCenter Web Service data models (described in the *Virtual Infrastructure SDK Reference Guide*) can occur multiple times (for example, multiple NICs). In these cases, these fields have the `minoccurs` attribute set to 0 and the `maxoccurs` attribute set to `unbounded`. When the `maxoccurs` attribute is not `1`, the field is represented in the Java bindings as an array. The Web service data structures have two categories of arrays; indexed arrays and key-based arrays. Indexed arrays are only used for arrays of basic types; for example, `/host/vm[]` and `host/info/datastore[]`. All other arrays are key-based arrays.

Indexed arrays are accessed by an index (the usual manner of accessing arrays). Key-based arrays are accessed by keys that are strings. The component type of the array must have a string field called `key`, in order to be a key-based array. The value of this field is unique across all the components of an array, and is the key of the array component. See Key-based Arrays on page 78.

## Indexed Arrays

Some examples of indexed arrays are:

```
host/vm[1]
```

The target in the Change object for indexed arrays always ends with the name of the array field, followed by the index where the change is to occur (`...arrayName[index]`).

The kind of change is one of the following:

- `ins` — The `inserted` field specifies the number of array elements that are being inserted at the location specified by the target. You must ignore the `editPos` and `deleted` fields. The `val` field is an array component if `inserted` is 1, otherwise the `val` field is an array with `inserted` elements.
- `del` — The `deleted` field specifies the number of array elements that are to be deleted starting at the location specified by the target. You must ignore the `editPos`, `inserted`, and `val` fields.
- `repl` — The `repl` field is a combination of a `del` operation followed by an `ins` operation. You must ignore the `editPos` field. All the other fields have the same meaning as previously specified.

The following code fragment illustrates how indexed arrays are handled:

```
if (changeData.getFieldType().isArray()) {
    // snippet from insertArrayEntries() referred to above
   int numInserted = numInsertedInt.intValue();
   Object array = changeData.getFieldValue();
   Class arrayFieldType = changeData.getFieldType().getComponentType();
```

```
   int newLength = currentLength + numInserted; // use numDeleted incase of delete op
   Object newArray = Array.newInstance(arrayFieldType, newLength);
   if (changeData.isIndexedArray()) {
      int index = changeData.getArrayIndex();
      // ... insert or delete Entries into newArray depending on the change Op
   }
   // Set this new array into the bean
   changeData.setFieldValue(newArray);
}
```

## Key-based Arrays

Key-based arrays are very similar to indexed arrays except the key is a string, and not an index. Some examples of key-based arrays are:

```
hardware/net/adapter["_nic001"]
```

The only difference between key-based arrays and indexed arrays in the `Change` objects is that each array component must be deleted individually, because there is no ordering concept for key-based array components. Therefore, you must ignore the `deleted` field in the `Change` object.

You can insert multiple components into a key-based array at one time. Array keys are generated by the Web service, and are not specifiable by the programmer when the component is inserted for the first time. Each component in a single array will have a unique key.

The following code fragment illustrates how to create a new key-based array element; in this case, VirtualNetworkAdapter.

```
VirtualNetworkAdapter adapter = new VirtualNetworkAdapter();
adapter.setMode(VirtualNetworkMode.monitor);
adapter.setNetwork("Internal Network");
// fill in other fields for adapter...

Change chng = new Change();
chng.setOp(ChangeOp.ins);

// Note : do not specify array key.
// this is generated, and not user setable
chng.setTarget("/hardware/net/adapter");

chng.setVal(adapter);
chng.setInserted(new Integer(1));

ChangeReq upd = new ChangeReq();
upd.setChange(new Change[] { chng });
upd.setHandle(vhandle);
```

The following code fragment illustrates how to handle key-based arrays:

```
if (changeData.getFieldType().isArray()) {
   Object array = changeData.getFieldValue();
   Class arrayFieldType = changeData.getFieldType().getComponentType();
  //determine new Length by looking at the deleted/inserted field based on the changeOp
   Object newArray = Array.newInstance(arrayFieldType, newLength);
   if (changeData.isKeyedArray()) {
      if insert change (insert or repl with inserted > 0)
      {
            if (numInserted == 1)
                // append the new value at the end
                Array.set(newArray, currentLength, val);
            else
                // append new elements to array
               System.arraycopy(val, 0, newArray, currentLength, numInserted);
      }
      else if delete change or repl with deleted > 0
      {
           // Loop thru each element in the existing array looking for key match
            // If key didnt match, insert the array element into new array
            int curIndex = 0;
            for (int i = 0; i < currentLength; i++)
            {
                String key = getArrayElementKey(array,i);
                if (!key.equals(changeData.getArrayKey()))
                {
                    // Key didnt match - copy this element to new array
                    Array.set(newArray, curIndex, element);
                    curIndex++;
                }
               // else key matched - ignore this since this is getting deleted
            }
        }
        // Set this new array into the bean
        changeData.setFieldValue(newArray);
    }
```

# Calling the PutUpdates Operation

The client uses the PutUpdates operation to perform updates to the Web service not performed by the other API operations in the VMware SDK. The argument to PutUpdates is a ChangeReqList that contains an array of ChangeReqs. Each ChangeReq has a handle, and a list of Change objects describing various changes that are being applied to the object being identified by the handle. The format of the Change object is exactly as it is in the GetUpdates operation.

The purpose of the PutUpdates operation is to make the changes, specified in the Change objects, to the Web service. There are two forms of PutUpdates: "last one wins" and "versioned".

- Last one wins (unversioned PutUpdates call) — Changes are applied to the Web service object referred to by the handle. The order in which the changes from multiple clients are applied is unspecified and it is possible for another client's changes to override this client's changes and vice versa.
- Versioned — Rather than providing a handle, clients give a versioned handle, or vHandle, to the PutUpdates operation. In this case, the operation is performed only if the Web service's version of the object matches that of the client. Otherwise a version mismatch occurs and the call fails.

The value returned by the PutUpdates operation is exactly the same as the value returned by the GetUpdates operation for the changes on the vHandles passed to PutUpdates. It is as if the updates have been made and the client has called a GetUpdates operation with the same set of vHandles. If an unversioned PutUpdates call is made (on handles), then no updates are returned in the response.

**Note:** It is possible to obtain a change list from the PutUpdates operation that is greater than the your list of changes, if another client is also making changes to the same object at the same time.

## Using the PutUpdates Operation to Update the Memory Setting for a Virtual Machine

The following code fragment illustrates the use of PutUpdates. It shows how a client can update the memory setting for a virtual machine. Because the sizeMb field is an integer, we use an edit operation in the change object.

1. Create the change object.

```
String target = "hardware/memory/sizeMb";
Integer newSize = getMemorySize();

Change change = new Change();
change.setOp(ChangeOp.edit);
change.setTarget(target);
change.setVal(newSize);
```

```
Change [] changes = new Change[] { change };
```

2. By using the ResolvePath and GetContents operations, obtain the handle for the virtual machine to which the change is being applied.

```
String handle = serviceConnection.resolvePath(viewPath);
ViewContents viewContents = serviceConnection.getContents(handle);
```

3. Call the PutUpdates operation with this change.

```
ChangeReqList changeList = new ChangeReqList();
ChangeReq changeReq = new ChangeReq();
// For un-versioned putUpdates, send the un-versioned handle
changeReq.setHandle(handle);

// For versioned putUpdates set the versioned handle here:
// changeReq.setHandle(viewContents.getVHandle());

changeReq.setChange(changes);
ChangeReq[] changeReqs = new ChangeReq[] { changeReq };
changeList.setReq(changeReqs);
UpdateList updateList = serviceConnection.putUpdates(changeList);
```

## Using the PutUpdates Operation to Make Changes to Array Elements

To insert elements into an array, the client should not specify the index or key (keyed arrays) in the change target. For example, to insert a new network adapter into a virtual machine, the change target is `hardware/net/adapter`, without any square brackets.

To delete an array item, specify the index of the array element. Alternately, if the client is updating a keyed array, then specify the key for the element (that will be deleted) in the change target. Clients can use standard reflection APIs to look inside the object and retrieve all the keys of an array in order to determine which element to delete.

For example, to delete a network adapter with key `nic001`, the change object is:

```
String target = "hardware/net/adapter["nic001"]";
Change change = new Change();
change.setOp(ChangeOp.del);
change.setTarget(target);
change.setVal(null);
Change [] changes = new Change[] { change };
```

Read `/SDK/SDK-README.html`. It has a link to the complete code listing for the PutUpdates operation.

## Using the PutUpdates Operation to Specify a CustomProperty

A customProperty is a user-defined property created through VirtualCenter. It is an array of type Property where:

- key is the index of the custom property.
- val is the name of the custom property.

Clients may update the existing values by using the PutUpdates operation and can add new properties through the use of VirtualCenter.

The following code fragment illustrates specifying a value of a customProperty for a host or a virtual machine:

```
Specifying a value of a Custom Property for a host :
// Create change object and set target as "info/customProperty"
// The val for this must be specified as a Property object as below
change ch = new Change();
ch.setTarget("info/customProperty");

// set other change object fields

// The key is the same as the key of the custom property name
// in the Property array in /customProperty
Property p = new Property();
p.setKey("#key From Property.getKey() for single custom property");
p.setVal("a new value for the property for host or VM");

ch.setVal(p);

// this change object may now be used to PutUpdate a value for a host or VM
```

# Running the Sample Code

The sample code discussed in this document is included with this distribution. Read `/SDK/SDK-README.html` for a link to the instructions on how to run the sample code. See Client Development Environments on page 267 for complete information on how to set up your client development environment.

# Handling Exceptions in the Data Synchronization Loop

Exceptions can occur when any Web service method is called. These exceptions can occur during initialization (such as calling the `VmaServiceLocator` constructor), or during calls to Web service operations (such as ResolvePath, GetContents and GetUpdates).

One class of exceptions that requires handling is the standard Java exceptions that arise due to remote communication. This class includes some of the exceptions in the packages, `java.net` and `java.rmi`. There is nothing special regarding the handling of these exceptions for the Web service. Instead, it is simply how the client application recovers from the exception situations.

The other class are exceptions that show up as the Java exception, `com.vmware.vma.FaultInfo`.

For more information, see Exception Handling and Faults on page 129.

# Testing

To test your client applications, complete the following.

1. If VirtualCenter is not running, then start this application.
2. Start the Web service.
3. Run your test programs.
    a. Make changes in the VirtualCenter application and see if your client application responds appropriately.
    b. Make changes using your client application and see if VirtualCenter reflects your changes.

**Note:** You can also test the sample code that is provided with this distribution in a similar manner.

For more information on testing your client applications, see Troubleshooting on page 275.

# Complete Code Listing

You can find the complete code for this simple application in your VMware SDK package.

CHAPTER 6

# Advanced Client Concepts for Java Programmers

This chapter provides examples of client applications that you can create, to perform the following tasks. For complete information on the syntax for these client applications, refer to the *Virtual Infrastructure SDK Reference Guide*.

**Note:** The examples in this chapter are written in Java, and are based on a specific developer environment, the Websphere Software Developer Kit (WSDK) for Web services. If you are interested in Perl samples, see Advanced Client Concepts for Perl Programmers on page 173. You may use a different developer environment and language to build your client program. Adjust the examples accordingly for your developer environment.

# Virtual Machine Power Operations

Client applications can perform the following power operations:

- Power on (start) or resume a virtual machine
- Power off (stop) or suspend a virtual machine
- Reset a virtual machine

**Note:** For all these applications, the client must have Interact rights on the Server Farm, Farm group, Farm, and host that contains the virtual machine. If the client has only Browse rights, then the client can view information about these objects (GetContents and GetUpdates requests), but cannot issue any power operations.

If the client is listening for updates on this virtual machine and the operation succeeds, then the client will see updates to the VirtualMachine state.

In addition, a task is initiated on the Web service for each of these operations. The XML document describing that task is returned to the client. The client can monitor the progress of this task and check for errors, status, and so on. For sample code on how to monitor the progress of a task, see Task Scheduling and Monitoring on page 114.

## Starting or Resuming a Virtual Machine

The StartVM operation initiates the process of starting a virtual machine or resuming a suspended a virtual machine. If you have already configured a script or any other application to run during the power-on (resume) operation, the script (or other application) will run.

It takes one argument, the handle of the virtual machine that is to be started. The return value is the handle to the task created to start this virtual machine. If the virtual machine that is to be started is currently suspended, then this virtual machine is resumed.

The following code fragment illustrates how clients call the StartVM operation, then how the client can monitor the resulting task to determine when the operation completes.

```
String vmPath = "/vm/564d5a05-29a7-b09b-d576-9cb8a719d940)";
String vmHandle = serviceConnection.resolvePath(vmPath);
ViewContents task = serviceConnection.startVM(vmHandle);
```

## Stopping or Suspending a Virtual Machine

The StopVM operation initiates the process of stopping or suspending a powered-on virtual machine.

If you have already configured a script or any other application to run during the power-off (suspend) operation, the script (or other application) will run. It takes three arguments: the handle of the virtual machine that is to be stopped, whether the virtual machine should be suspended or

powered off, and whether or not stopping the virtual machine is a "soft "power off operation. The return value is the handle to the task that represents the task created to power off this virtual machine.

The following code fragment illustrates how clients call the StopVM operation, then how the client can monitor the resulting task to determine when the operation completes.

```
String vmPath = "/vm/564d5a05-29a7-b09b-d576-9cb8a719d940)";
String vmHandle = serviceConnection.resolvePath(vmPath);
ViewContents task = null;

// Determine what operation to perform
if (operation.equals("stop"))
{
   boolean suspend = false;
   boolean gracefulShutdown = true;
   task = serviceConnection.stopVM(vmHandle, suspend, gracefulShutdown);
}
else if (operation.equals("suspend"))
{
   boolean suspend = true;
   boolean enterStandbyMode = true;
   task = serviceConnection.stopVM(vmHandle, suspend, enterStandbyMode);
}
```

## Boolean Flags in the VirtualMachineTools Datatype

The VirtualMachineTools datatype includes four Boolean flags that clients may use to determine whether or not scripts execute in the guest operating system when a virtual machine's power state changes through the StartVM or StopVM operations.

- afterPowerOn — Flag determines whether or not scripts should run after the virtual machine is powered on. If this boolean is set to true, then custom startup scripts (if there are any) run on the guest operating system after the virtual machine powers on.
- afterResume — Flag determines whether or not scripts should run after the virtual machine is resumed. If this boolean is set to true, then custom startup scripts (if there are any) run on the guest operating system after the virtual machine resumes.
- beforeSuspend — Flag determines whether or not scripts should run before the virtual machine is suspended. If this boolean is set to true, then custom startup scripts (if there are any) run on the guest operating system before the virtual machine is suspended, regardless of whether the soft flag is specified during the StopVM operation.
- beforePowerOff — Flag determines whether or not scripts should run before the virtual machine is powered off. If this boolean is set to true, then custom startup scripts (if there are

any) run on the guest operating system before the virtual machine powers off, regardless of whether the soft flag is specified during the StopVM operation.

**Note:** If one of these Boolean flags is set, then the scripts will run, regardless of the soft flag setting in the StopVM operation.

## Resetting a Virtual Machine

The ResetVM operation initiates the process of resetting a virtual machine, which also resets the virtual hardware. (A reset operation is equivalent to pushing the Reset button on a physical machine.)

The ResetVM operation first attempts to shut down the guest operating system before resetting the virtual machine. This is similar to selecting Restart on a Windows operating system, where Windows gracefully shuts down, then starts up again. If you have already configured a script or any other application to run during the reset operation (during the shutdown or startup of the guest operating system), the script (or other application) will run.

If this attempt fails, then the ResetVM operation looks into the virtual machine's configuration file. By default, a virtual machine's configuration file setting for a reset is a "hard" reset where the virtual machine immediately powers off, regardless of what is occurring in the guest operating system. This is similar to pressing and holding the power button on a physical machine until it powers off, then restarting the physical machine.

The ResetVM operation takes one argument, the handle of the virtual machine that is to be reset. The return value is the handle to the task that represents the task created to reset this virtual machine.

The following code fragment illustrates how clients call the ResetVM operation, then how the client can monitor the resulting task to determine when the operation completes.

```
String vmPath = "/vm/564d5a05-29a7-b09b-d576-9cb8a719d940)";
String vmHandle = serviceConnection.resolvePath(vmPath);
ViewContents task = serviceConnection.resetVM(vmHandle);
```

# Host Operations

Clients can enable (connect) or disable (disconnect) hosts. Clients can also shut down a host provided the host is ESX Server 2.1 or higher. The client must have Configure rights on the Farm that contains the host, in order for the operations to succeed. These operations return an empty response.

## Enabling a Host

Clients may enable a host in the Disabled state by using the EnableHost operation. When a host is "created", and a user name and password are supplied during the host creation, then the host is automatically enabled for virtual machine operations. In the Enabled state, clients can perform virtual machine operations and discover new virtual machines. The EnableHost operation takes one mandatory argument, the handle to the host that will be enabled. There are two optional arguments: the user name and password that VirtualCenter uses to connect to the host specified by the handle.

Upon success, an empty response message is returned.

```
String handle = serviceConnection.resolvePath("/host/myhost.mydomain.com");
String userName = getUserName(); // optional parameter, null if not supplied
String password = getPassword(); // optional parameter, null if not supplied
serviceConnection.enableHost(handle, userName, password);
```

## Disabling a Host

Clients may disable the host from virtual machine operations by using the DisableHost operation. By using the Create operation without supplying the user name and password, clients may add a host to the Web service inventory in the Disabled state. When a host is disabled, clients are unable to perform any virtual machine operations.

The DisableHost operation takes one argument, the handle to the host that will be disabled. Upon success, an empty response message is returned.

**Note:** The DisableHost operation does not remove the host name from the Farm. Clients must remove the host by using the Delete operation. These two operations are separate, so that disabled hosts can continue to be managed without having to remove them.

```
String handle = serviceConnection.resolvePath("/host/myhost.mydomain.com");
serviceConnection.disableHost(handle);
```

## Stopping or Restarting a Host

The StopHost operation permits the client to shut down or restart the host. It takes four arguments: the handle of the host that is to be stopped, whether the host should be gracefully shut down (soft) or immediately powered off (hard), whether or not to restart the host, and a reason string.

```
String reason = "User initiated action";
String handle = serviceConnection.resolvePath("/host/myhost.mydomain.com");
if (operation.equals("shutdown"))
{
    boolean suspend = false;
    boolean restart = false;
    serviceConnection.stopHost(handle, suspend, restart, reason);
 }
 else if (operation.equals("restart"))
 {
    boolean suspend = false;
    boolean restart = true;
    serviceConnection.stopHost(handle, suspend, restart, reason);
 }
```

# Creating and Deleting Objects

Clients can create or delete top-level objects on the Web service server by using the Create and Delete operations, respectively. The client must have Configure rights on the Container of the new object being created or deleted.

## Creating an Object

The Create operation takes three mandatory arguments: the handle of the Container for the new object, the name of the new object, and the type of the new object. There is also a fourth optional argument, that is an initial XML document providing additional information to create the object. Upon success, the handle to the newly created object is returned.

Clients may create the following objects with this Create operation:

- VirtualMachine
- Host
- Container, Farm, or VirtualMachineGroup
- TaskSchedule
- PerfCollector
- Event collector

The sample code in this section illustrates how to create new hosts, Farms, Containers and VirtualMachineGroups, but not virtual machines, tasks, or performance collectors. For more information on creating these objects see:

- Creating and Configuring a Virtual Machine on page 96 for information on creating a virtual machine.
- Task Scheduling and Monitoring on page 114 for information on how to create new scheduled tasks.
- Collecting Performance Data on page 119 for information on how to create new performance collectors.

If the client is creating a host and the parent Container is the `/host` handle, then the new host is placed in the default Farm in the `/vcenter` hierarchy, `/vcenter/Default Farm`. The initial value of the host object is specified by a HostSpec, as shown in the following sample.

```
if (type.equals("Container"))
   initVal = (Object) new com.vmware.vma.Container();
else if (type.equals("Farm"))
   initVal = (Object) new com.vmware.vma.Farm();
else if (type.equals("VirtualMachineGroup"))
   initVal = (Object) new com.vmware.vma.VirtualMachineGroup();
```

```
else if (type.equals("Host"))
{
   HostSpec hostSpec = new com.vmware.vma.HostSpec();
   // Prompt user for username & password for this host
   String userName = getUserName();
   String password = getPassword();
   Integer port = getPort();

   hostSpec.setPort(port);
   hostSpec.setUserName(userName);
   hostSpec.setPassword(password);
   initVal = hostSpec;
}
```

## Deleting an Object

The following sample illustrates how to delete an object, such as a Container, Farm, VirtualMachineGroup, host, or a virtual machine. The Delete operation takes one argument, the handle of the object that will be deleted. When deleting a virtual machine, the Delete operation removes the virtual machine's configuration file and any other associated files, including the virtual disk file.

Container, Farm and VirtualMachineGroup objects are not required to be empty before they can be deleted.

**Note:** When a client deletes a top-level object (that contains other objects), then a delete change is only shown for the top-level object. For example, if a client deletes a Farm, then the client sees a delete change on the Farm, but not on any hosts in the Farm, or any virtual machines on the hosts.

Similarly, a virtual machine must be powered off (stopped) or suspended, and cannot be migrating, for the Delete operation to succeed. If a client wants to delete a virtual machine but not its virtual disk, then the client must do the following steps:

1. Stop the virtual machine
2. Using the PutUpdates operation, disconnect the virtual disk(s) of the virtual machine
3. Delete the virtual machine

The client must have Configure rights for both the object being deleted and the Container that has the object. Upon success, an empty response message is returned. The following sample illustrates how to delete an object.

```
String path = "/vm/564d5a05-29a7-b09b-d576-9cb8a719d940";
String handle = serviceConnection.resolvePath(path);
serviceConnection.delete(handle);
```

# Creating and Configuring a Virtual Machine

In this section, we describe first how to create a virtual machine, then how to configure it by adding virtual disks.

## Creating a Virtual Machine

The following sample illustrates how to create a virtual machine. A virtual machine must always reside in a Farm or VirtualMachineGroup in the `/vcenter` hierarchy. If such an array is not identified for the virtual machine, then the operation places the newly created virtual machine in a special Farm `/vcenter/Default Farm`.

The client must have Configure rights on the Container where the virtual machine is being created. VirutalMachineSpec object specifies the initial value of a new virtual machine. The virtual machine's hardware configuration must be specified in this initial value.

The following sample illustrates how to initialize the VirutalMachineSpec object and invoke the Create operation. See the full code listing included with this distribution in the `/SDK/WebService/samples` directory for complete information on how to create a virtual machine's hardware.

```
VirtualMachineSpec vmSpec = new VirtualMachineSpec();
VirtualHardware hardware = createHardware();
vmSpec.setHost(hostHandle);
vmSpec.setGuestOS(guestOS);
vmSpec.setHardware(hardware);

String hostHandle = serviceConnection.resolvePath(/host/<fully qualified hostname>);
VirtualMachineSpec vmSpec = createVMSpec(guestOS, hostHandle);
String parentHandle = serviceConnection.resolvePath(parentPath);
String vmHandle = serviceConnection.create(parentHandle, name,
        "VirtualMachine", vmSpec);
```

## Adding a Virtual Disk to a Virtual Machine

Now that we've created a virtual machine, we can add a disk to it. The CreateVirtualDisk operation requests the creation of a virtual disk for a virtual machine. It takes two arguments: the handle specifying the target virtual machine and the XML document (DiskInfo) specifying the properties of the new virtual disk.

**Note:** If a client attempts to create a virtual disk that already exists, the operation returns a successful task. It does not fail and does not return a fault. However, another virtual disk is not created.

The client must have Configure rights on the virtual machine where the disk is being created, in order for this operation to succeed.

Once the operation has been initiated, the Web service initiates a task for the CreateVirtualDisk operation. The XML document describing that task is returned to the client. The client can monitor the progress of this task and check for errors, status, and so on.

See Task Scheduling and Monitoring on page 114 for sample code on how to monitor the progress of a task.

```
VirtualDiskInfo diskInfo = new VirtualDiskInfo();
diskInfo.setControllerType(DiskControllerType.scsi);
diskInfo.setDeviceNumber(1);
diskInfo.setControllerId(0);
diskInfo.setAdapterType(DiskAdapterType.busLogic);

DiskType diskType = new DiskType();
diskType.setDiskKind(DiskKind.file);
DiskFileInfo diskFileInfo = new DiskFileInfo();
diskFileInfo.setSize(size);
diskFileInfo.setPreAllocated(new Boolean(false));
diskFileInfo.setFlat(new Boolean(false));
diskType.setDiskFileInfo(diskFileInfo);

diskInfo.setDiskType(diskType);
diskInfo.setMode(VirtualDiskMode.persistent);
diskInfo.setRemovable(null);

String vmHandle = serviceConnection.resolvePath("/vm/5038d66d-ed8b-2d32-8fdb-4a5e090715f7");
serviceConnection.createVirtualDisk(vmHandle, diskInfo);
```

# Responding to Virtual Machine Questions

A running virtual machine can generate a question that requires input from a user before the virtual machine can proceed. For example, if a virtual machine with an undoable disk is powered off, the virtual machine asks the user whether or not to discard the changes to the virtual disk.

The SDK provides a mechanism for users to handle this case programmatically. The question, generated by the virtual machine, appears in the virtual machine's state as a msgWaiting object. Clients will see this object when they are listening for updates on the virtual machine (through the GetUpdates operation), or if they call the GetContents operation on the virtual machine (which is blocked on the question).

The msgWaiting object has the following fields:

- msg — Question posed by the virtual machine.
- id — Internal server ID for the message.
- choice — List of possible answers, presented as an array of key-value pairs.
- defaultChoiceIndex — Default choice for this question, if there is one. This is the integer index of the default choice in the choices array. If this value does not exist, then the defaultChoice is 0 (zero).

The client can respond to this question by using the AnswerVM operation. The AnswerVM operation takes 3 arguments: the handle of the virtual machine that is blocked (and waiting for an answer), the choice (a key-value pair), and the ID of the message that requires a response.

The following sample code illustrates how to invoke the AnswerVM operation.

```
for (;;) {
    .. getUpdates on the vHandle of the virtualMachine
    .. applyUpdates to the VirtualMachine object
    // check to see if there is a message waiting
    MsgWaiting msgWaiting = vm.getState().getMsgWaiting();
    if (msgWaiting != null && msgWaiting.getMsg() != null) {
        MsgWaiting msgWaiting = vm.getState().getMsgWaiting();
        String question = msgWaiting.getMsg();
        KeyedValue[] choices = msgWaiting.getChoice();
        for (int i = 0; i < choices.length; i++) {
         System.out.println(choices[i].getKey() + " " + choices[i].getVal() + " [" + i + "]");
        }
        int defaultChoice = 0;
        if (msgWaiting.getDefaultChoiceIndex() != null) {
            defaultChoice = msgWaiting.getDefaultChoiceIndex().intValue();
        }
```

```
        ... prompt user with question and choices. Get user's selection

        serviceConnection.answerVM(handle, choices[index], msgWaiting.getId());
        break;
    }
}
```

# Cloning a Virtual Machine

The CloneVM operation creates a new virtual machine by using as its source, an existing virtual machine or a template. It takes five mandatory arguments:

- Handle to the (source) virtual machine or template that will be cloned.
- Handle to the VirtualMachineGroup or Farm in which the cloned virtual machine will reside (the parent handle).
- Handle to the host where the new virtual machine will reside.
- Name of the newly cloned virtual machine.
- Location on the destination host where the cloned virtual machine's configuration files and virtual disks will reside.

There are two optional arguments:

- customization of the guest operating system for the newly cloned virtual machine. See the next section for additional details.
- flag that determines whether or not the newly cloned virtual machine will automatically power on once the cloning operation is complete.

Virtual machines must be powered off or suspended in order for the CloneVM operation to succeed. If the source is a template located on a host's datastore, then the newly cloned virtual machine must also reside on the same host where the datastore is located. Alternately, if hosts are on the same SAN and share the same datastore, then the CloneVM operation (from a template) may be done across these hosts.

If the source is a virtual machine, then the client must have Configure rights on that virtual machine and on the Farm of its host. If the newly cloned virtual machine will reside on a different host, then the client must have Configure rights on the destination host of this new virtual machine.

Once the operation has been initiated, the request returns a task handle to the client. The client may monitor the task for the progress of the operation.

## Customizing a Virtual Machine

The schema for the customization specification has been incorporated into `vma.wsdl`. When a client generates stub files, a class (structure) is created that represents the customization parameters.

Clients can customize both Windows and Linux guest operating systems. For more information on customizing a guest operating system, refer to the section titled "Preparing for Guest Customization" in the *VMware VirtualCenter User's Manual*.

### Customizing a Windows Guest Operating System

If you plan to customize a Windows guest operating system, then you must first install the Microsoft Sysprep tools package on the VirtualCenter server machine. Follow the procedure in the section titled "Preparing for Guest Customization" in the *VMware VirtualCenter User's Manual*.

For example, clients can customize the following:

- Registration information — User's full name and organization.
- Computer name — Computer or host name, used for identifying this virtual machine on a network.
- Administrator password — Password for the Administrative user.
- Timezone — Time zone for the virtual machine.
- AutoLogon — Enables the virtual machine to log on automatically to the Administrator account the first time the machine boots.
- Product ID — Product ID (license key) for the guest operating system
- Network settings — DHCP or static IP address.

The following is a customization code example for a Windows guest operating system:

```
public static Autoprep getCustomizationSettings()
{
   Autoprep autoprep = new Autoprep();

   Sysprep sysprep = new Sysprep();
   autoprep.setSysprep(sysprep);

   GuiUnattended guiUnattended = new GuiUnattended();
   sysprep.setGuiUnattended(guiUnattended);

   Password password = new Password();
   password.setPlainText(true);
   password.setValue("admin_password");
   guiUnattended.setAdminPassword(password);
   guiUnattended.setTimeZone("004");
   guiUnattended.setAutoLogon(Boolean.TRUE);
   guiUnattended.setAutoLogonCount(new BigInteger("1000"));

   UserData userdata = new UserData();
   sysprep.setUserData(userdata);

   userdata.setFullName("John Doe");
   userdata.setOrgName("ACME Corp.");
   userdata.setComputerName("John Doe Win2000");
```

```
    Identification id = new Identification();
    sysprep.setIdentification(id);
    id.setJoinWorkgroup("Marketing workgroup");


    Adapters adapters = new Adapters();
    Adapter[] tmp = new Adapter[1];
    tmp[0] = new Adapter();
    tmp[0].setMACAddress("MAC00");
    tmp[0].setUseDHCP(Boolean.TRUE);
    tmp[0].setDNSFromDHCP(Boolean.TRUE);
    tmp[0].setNetBIOS(NetBIOSEnum.EnableNetBIOS);
    adapters.setAdapter(tmp);

    autoprep.setAdapters(adapters);

    return autoprep;
}
```

**Customizing a Linux Guest Operating System**

If you plan to customize a Linux guest operating system, then you must first install the VMware Open Source components on the VirtualCenter server machine. Click on the download link for Open Sources at *www.vmware.com/download* and follow the procedure in the section titled "Preparing for Guest Customization" in the *VMware VirtualCenter User's Manual*.

For example, clients can customize the following:

- Computer name — Computer or host name, used for identifying this virtual machine on a network.
- Network settings — DHCP or static IP address.

```
public static Autoprep getCustomizationSettings()
{
    Autoprep autoPrep = new Autoprep();

    Options options = new Options();
    autoPrep.setOptions(options);

    LinuxGlobal linuxGlobal = new LinuxGlobal();
    linuxGlobal.setComputerName("LinuxCustomVM");
    linuxGlobal.setDomain("ACMEcorp.com");
    autoPrep.setLinuxGlobal(linuxGlobal);


    Adapters adapters = new Adapters();
```

```
    Adapter[] adapt = new Adapter[1];

    adapt[0] = new Adapter();
    adapt[0].setMACAddress("MAC00");
    adapt[0].setUseDHCP(Boolean.TRUE);
    adapt[0].setDNSFromDHCP(Boolean.TRUE);
    adapt[0].setNetBIOS(NetBIOSEnum.EnableNetBIOS);
    adapters.setAdapter(adapt);

    autoPrep.setAdapters(adapters);
    return autoPrep;
}
```

#### Calling the CloneVM Operation

Before calling the CloneVM operation, the client first needs to instantiate an object of type Autoprep, and set the various parameters in this object. Then the client can pass this object as an additional argument to the CloneVM operation.

**Caution:** Be sure that the Microsoft Sysprep tools (Windows guest operating system) or the VMware Open Source components (Linux guest operating system) is installed in the VirtualCenter server machine before starting the CloneVM operation. Otherwise, the CloneVM operation will fail at the end of this operation.

## CloneVM Sample

This sample illustrates the CloneVM operation, but does not include customization. If customization were included, then `null`, in the sixth argument, is replaced with the passing in of the autoprep object.

```
String parentHandle = serviceConnection.resolvePath("/vcenter/MyFarmGroup/
Farm/MyVMGroup");

// Now clone the VM
ViewContents task =
    serviceConnection.cloneVM(
        srcHandle, // handle to a template or a VM
        parentHandle,
        hostHandle, // host where this new VM will reside
        "myNewClonedVM",
        datastore,
        null,
        Boolean.TRUE);
```

# Creating a Template

Clients can specify a handle to a template as the first argument of the CloneVM operation. Clients can create this template by using the CreateTemplate operation. The CreateTemplate operation takes two arguments, the handle of the source virtual machine being used to create the template, and an XML document describing a TemplateSpec.

The TemplateSpec specifies attributes of the new template such as its name, the datastore that contains the template's configuration file, the location of the virtual disk(s) for this template, and a user description for this template. If no datastore is specified in the XML document, then the configuration file and virtual disks for this template are placed in a local template upload directory on VirtualCenter.

The client must have Configure rights on the source virtual machine.

A task is initiated on the Web service for the CreateTemplate operation. The XML document describing that task is returned to the client. The client can monitor the progress of this task and check for errors, status and so on.

See Task Scheduling and Monitoring on page 114 for sample code on how to monitor the progress of a task.

```
/** vmPath is the path of the VM that the template is being created for */
String srcVMHandle = serviceConnection.resolvePath(vmPath);
TemplateSpec templateSpec = new TemplateSpec();
String name = "my new template";
String description = "Test template";
templateSpec.setName("VM1-template");
templateSpec.setDatastore(datastores[0]);
templateSpec.setDescription("Test template from VM1");
ViewContents task = serviceConnection.createTemplate(srcVMHandle, templateSpec);
```

## Specifying a Datastore

Clients must specify a datastore in TemplateSpec. Clients can obtain the datastore from the Host object, once the host for a virtual machine has been determined. Alternatively, clients can obtain a list of all the available datastores by calling a ResolvePath, then a GetContents operation on the `/datastore` path. The following sample code illustrates how to correlate the datastore information obtained from the Host object with the datastore information from the `/datastore` path.

```
/** Obtaining a datastore from a host */
ViewContents hostView = serviceConnection.getContents(hostHandle);
```

```
Host host = (Host) hostView.getBody();
// get the names of the datastores on this host
String[] datastores = host.getInfo().getDatastore();

// get all datastores
String datastoreRootHandle = serviceConnection.resolvePath("/datastore");
ViewContents dsView = serviceConnection.getContents(datastoreRootHandle);
DatastoreInfoList dsList = (DatastoreInfoList) dsView.getBody();
DatastoreInfo[] ds = dsList.getDatastore();

// The key in the DatastoreInfo object is the name of the datastore
// ds[0].getKey() -- name of the datastore
// ds[0].getCapacityMB() - capacity in MB of this datastore
// ds[0].getFreeSpaceMB() - available space in MB on this datastore
```

# Renaming an Object

The Rename operation requests a change to the name of an existing object and optionally moves it in the `/vcenter` hierarchy. Clients can rename Containers, Farms, virtual machine groups and virtual machines. However, clients cannot use the Rename operation to migrate a virtual machine and move it from one host to another.

**Note:** You can only move a virtual machine or VirtualMachineGroup to a new VirtualMachineGroup. Similarly, you can only move a Farm or a Farm group (Container) to a new Farm group. You cannot move a VirtualMachineGroup to a new Farm and you cannot move objects across Farms.

The Rename operation takes two mandatory arguments: the handle of the existing object that is to be renamed and the new name of the object. If a name change is not desired, then pass in the current name of the object. It has one optional argument, the new destination for the object being moved. If this parameter is not specified, then the object is not moved, but is simply renamed in its current location.

**Note:** The destination object, pointed to by destHandle, must be capable of holding the type of object specified by the handle parameter or an error is returned.

The client must have Browse and Configure rights for both the current and new Container for the object. Upon success, an empty response message is returned.

The following sample illustrates renaming an object.

```
String sourcePath;
String destPath;
String renameTo;  // This can be null if the name of the object isnt being changed

String srcHandle = serviceConnection.resolvePath(sourcePath);
String destHandle = serviceConnection.resolvePath(destPath);
serviceConnection.rename(srcHandle, destHandle, renameTo);
```

# Moving Virtual Machines

VMware SDK provides two operations to move virtual machines across hosts, MigrateVM and MoveVM. Clients should use MigrateVM when the virtual machine is being moved to a new host without moving its virtual disk(s). This operation is applicable when both the original host and the destination host share a SAN. By contrast, clients should use the MoveVM operation to move a virtual machine's disk to the destination host.

A task is initiated on the Web service for the MigrateVM and MoveVM operations. The XML document describing that task is returned to the client. The client can monitor the progress of this task and check for errors, status, and so on. See Task Scheduling and Monitoring on page 114 for sample code on how to monitor the progress of a task.

If the virtual machine is being moved and the client is monitoring it for updates, then the client will receive updates on the VirtualMachineState (on the host and the details fields). The host field in the VirtualMachineState is updated with the destination host name once the migration successfully completes.

The following sample illustrates moving a virtual machine's disks:

```
String vmPath = "/vm/564d5a05-29a7-b09b-d576-9cb8a719d940";
String hostPath = "/host/myhost.mydomain.com";
String vmHandle = serviceConnection.resolvePath(vmPath);
String hostHandle = serviceConnection.resolvePath(host);
String diskKey = null;
String dataLocator = null;

VirtualDiskDestination diskToMove = new VirtualDiskDestination();
diskToMove.setKey(diskKey);
diskToMove.setDataLocator("[" + dataStoreName + "]");
VirtualDiskDestination[] disks = new VirtualDiskDestination[] { diskToMove };

ViewContents task = serviceConnection.moveVM(vmHandle, hostHandle,
       dataLocator, disks);

// monitor this task handle for completion/errors
monitorTask(task, serviceConnection);
```

## Migrating a Virtual Machine

The MigrateVM operation starts the process of migrating a virtual machine to a specific host, without moving the virtual disk(s). In this release, the virtual machine must be in the poweredOn state. This operation never moves the virtual machine's disks from its current location.

**Note:** The MigrateVM operation is not supported for a GSX Server host.

This operation takes two mandatory arguments: the handle to the virtual machine that will be migrated, and the handle to the destination (target) host. There are two optional arguments: priority (determines whether or not resources are preallocated before migration starts) and the path describing the location of the virtual machine configuration file.

The client must have Configure rights on the virtual machine and on the Farm.

The request returns once the MigrateVM operation has been initiated by returning a task handle back to the client. You can determine whether the migration is successful, by separately monitoring the task that is performing the migration or by monitoring the virtual machine state (specifically the detail or host fields in VirtualMachineState). If the virtual machine has been successfully migrated, then the host field should contain the target host handle.

The following sample illustrates how to migrate a virtual machine:

```
String vmPath = "/vm/564d5a05-29a7-b09b-d576-9cb8a719d940";
String hostPath = "/host/myhost.mydomain.com";
String vmHandle = serviceConnection.resolvePath(vmPath);
String hostHandle = serviceConnection.resolvePath(host);

com.vmware.vma.Level priority = null;
String dataLocator = null;
ViewContents task = serviceConnection.migrateVM(vmHandle, hostHandle,
        priority, dataLocator);

// monitor this task handle for completion/errors
monitorTask(task, serviceConnection);
```

## Moving a Virtual Machine's Virtual Disks

The MoveVM operation moves a virtual machine's virtual disk(s) to a different location. You may (optionally) also use this operation to move the virtual machine to a different host. You must perform this operation on a virtual machine while it is in the poweredOff state.

The MoveVM operation takes one mandatory argument, the handle to the virtual machine that will be enabled. There are three optional arguments: the handle to the destination host (if both the virtual machine and its virtual disk(s) are moved), the dataLocator parameter (path) describing the location of the virtual machine configuration file, and the destination for all disks in the virtual machine.

The client must have Configure rights on the virtual machine and on the Farm of its host. If you are also planning to move this virtual machine to a different host, then the client must also have Configure rights on the target (destination) host.

The request returns once the operation has been initiated by returning a task handle back to the client. The client may monitor the task for progress of the operation.

# Monitoring Events

The client can obtain events information, as specified by the Event data model, described in the *Virtual Infrastructure SDK Reference Guide*.

Clients can collect events on hosts or on virtual machines, including informational, warning, or error messages such as changes in power operations or device status (connected, disconnected, or busy). In addition, clients can receive updates when alarms occur; for example, when memory or CPU usage, or virtual machine heartbeat is either above or below normal.

Each event comprises two parts: a declaration (the type of event), and the actual event (an event log).

Clients can access all information related to events by calling GetContents on the `/event` handle. This returns a Container with two Items:

- Handle for all event declarations under `/event/decls`
- Handle for all events under `/event/all`

## Event Declarations

The view, `/event/decls`, represents all the event declarations. This view is an EventDeclList object, that contains a single array field called "decl". Each entry is this array is a separate EventDecl (event declaration). By calling GetContents on `/event/decls`, the client can obtain all the known event declarations in the system. The event declarations are a set of pre-defined event types that do not get updated. Therefore, clients do not need to call GetUpdates on the handle for `/event/decls`.

Each event declaration has the following attributes:

- key — String that is the ID of this event declaration.
- kind — Type of event, that is one of the following: "alert", "error", "warning", "info", or "user".
- msgFmt — Array of format strings that describes how the event message is rendered; for example, "Task%0 created on %1".
- schedule — (Optional) Handle of the schedule (if any) that caused this event.

## Event Logs

The view, `/event/all`, represents all the available events. This view is an EventCollector Container that has 2 Items: `/event/all/filter`, which is an EventFilter and `/event/all/events`, which is of type EventCollection.

The EventFilter object describes how all the events listed under EventCollection are grouped together. The EventCollection contains a single array field called `log`. Each entry in this array is a separate Event object. Each Event object describes a unique event that occurred in the system.

In the case of `/event/all`, no filtering is done and all events are listed under `/event/all/events`; the filter under `/event/all/filter` is empty. EventFilter objects have the following attributes, indicating the attribute used to group the events:

- parent — (Optional) ID of the parent event that generated the events in this filter. (If this attribute is set, then the events in this event collector are grouped by the generating parent event.)
- schedule — (Optional) ID of the scheduled task that generated this event. (If this attribute is set, then the events in this event collector are all the events generated by this task schedule.)
- kind — String indicating the event kind. Events of the same kind are grouped together.
- startTime — Collects events that occurred after this start time.
- end time — Collects events that occurred before this end time.
- farm — (Optional) Collects events that occurred under this Farm.
- vm — (Optional) Collects events that occurred on this virtual machine.
- host — (Optional) Collects events that occurred on this host.
- declId — (Optional) Collects events that have this declaration ID.
- totalEvents — (Optional) Collects this number of events in the event collector, giving priority to more recent events.

Event objects have the following attributes:

- key — String that is the ID of the event.
- decl — Declaration ID corresponding to this event.
- arg — Array of name-value pairs that holds the value for the parameters in the message format string. Each entry in this array corresponds to a matching entry in the argType[ ] array in this event's declaration.
- parent — String that is the event ID of the parent event that caused this event.
- timestamp— Date and time when this event occurred.
- userDesc — (Optional) String that is the user description of this event.
- vm — (Optional) Handle of the virtual machine that owns this event.
- host — (Optional) Handle of the host that owns this event.
- farm — (Optional) Handle of the Farm that owns this event.

Clients that are interested in gathering event information should first call GetContents on the handle for `/event/decl` and `/event/all/events`. The declaration data that is received

should be cached by the client. The client can then call GetUpdates on `/event/all/events` to be notified of events, as they occur.

The following sample illustrates how clients can monitor events as they occur.

```
String handle = serviceConnection.resolvePath("/event/all/events");
 ViewContents vc = serviceConnection.getContents(handle);
 EventCollection eventList = (EventCollection)(vc.getBody());
 Event[] eventItems = eventList.getLog();
 if (eventItems == null)
 {
     System.out.println("There are no Events");
     return;
 }
 for (int i = 0; i < eventItems.length; i++)
 {
     ...process event...
 }
```

## Creating an Event Collector

Clients can create an event collector to filter events on the following attributes:

- kind — Filters by the type of event; for example, alert, info, warning, and so on.
- declId — Filters an event specified by its VirtualCenter event declaration ID
- startTime — Filters any events that occurred before the specified start time.
- endTime — Filters any events that occurred after the specified end time.
- parent event — Filters any events with the specified parent event.
- schedule — Filters any events that are caused by the specified schedule
- host, vm, or farm — Filters for events associated with the specified host, virtual machine, or Farm.

Once it has been created, this event collector is only accessible by the handle returned by the Create operation. Clients cannot view this event collector in the /event view hierarchy.

For example, a client may create an event collector if it is interested in only "error" events from a particular host or virtual machine. The newly created event collector only shows these events.

The following sample code illustrates the creation of a filtered event collector:

```
EventFilter eventFil = new EventFilter();
String srcHandle = serviceConnection.resolvePath(srcpath);
eventFil.setHost(srcHandle);
eventFil.setKind(EventKind.error);
```

```
String eventParent = serviceConnection.resolvePath("/event");
EventCollector eventCol = new EventCollector();
eventCol.setFilter(eventFil);
String eventColHandle =
    serviceConnection.create(eventParent, "Errors", "EventCollector", eventCol);
```

The following code sample shows how to extract the list of events from this event collector:

```
ViewContents eventColView = serviceConnection.getContents(eventColHandle);
Container c = (Container) eventColView.getBody();
Item[] items = c.getItem();
String eventCollectionHandle = null;
for (int i = 0; i < items.length; i++) {
    if (items[i].getType().equals("EventCollection")) {
        eventCollectionHandle = items[i].getKey();
        break;
    }
}

ViewContents eventListView =
serviceConnection.getContents(eventCollectionHandle);
EventCollection eventList = (EventCollection) eventListView.getBody();

// display the events here. Monitor the EventCollection for new events...
// See code sample for full listing on how to do this.
```

# Task Scheduling and Monitoring

Clients create tasks on the Web service in order to execute API operations.

## Active Tasks and Scheduled Tasks

There are two types of tasks: active tasks and scheduled tasks. Active tasks are tasks that are currently running on the Web service. Scheduled tasks are tasks that will be run on a pre-defined, and possibly a recurring, schedule. When a scheduled task runs, a new active task is created in the Web service.

Clients can obtain information about all currently active tasks by calling GetContents on the `/task` handle. Similarly, clients can obtain information about all scheduled tasks that currently exist on the Web service, by calling GetContents on the `/schedule` handle.

The Web service can create tasks in response to certain long-running operations. The Web service initiates such operations as a task. The Web service returns control to the client without waiting for the task or operation to complete. The response contains information about the task that was created on the Web service for that operation. Some examples of long running tasks include StartVM, StopVM, ResetVM, MigrateVM, and so on. Clients can access information about the task to determine its status and its progress as described in the next few sections.

Clients can also create scheduled tasks directly on the Web service. Clients identify the operation, then specify a schedule for running the task. The Web service uses this schedule to determine when to run the task. See Creating New Scheduled Tasks on page 116.

## Monitoring Tasks

Clients can monitor for new tasks as they occur by calling GetUpdates on the `/task` handle. Clients can receive notifications on new scheduled tasks as they are created on the Web service by calling GetUpdates on the `/schedule` handle.

The GetUpdates operation on the `/schedule` or the `/task` handle returns a Container of Items. Each Item has a handle to a Task object. Each Task object has the attributes listed in the following table.

| Task Object Attribute | Description |
|---|---|
| operationName | String identifying the operation that this task is running. |
| cause | String identifying the agent that caused this task's creation. |
| schedule | String identifying the handle for the scheduled task that created this task, if applicable. |

| Task Object Attribute | Description |
|---|---|
| entity | String identifying the handle for the object that this task is operating on. |
| eventCollector | Handle of the EventCollector that contains all events associated with this task. |
| currentState | TaskRunState, an object that indicates this task's current status (running, failed, completed, and so on). |
| percentCompleted | Float indicating the percentage of the task that has completed. |
| normalReturn | Object that contains the return value from the operation that the task executed. |
| faultReturn | FaultInfo object that captures any errors or exceptions encountered by the operation that this task was executing. |
| allowCancel | Boolean indicating if this running task can be cancelled |
| queueTime | Date and time this task was created. |

The following sample illustrates how the client can stay informed of all new tasks in the system.

```
// Replace /task with /schedule to get all scheduled tasks in the system
String handle = serviceConnection.resolvePath("/task");
ViewContents vc = serviceConnection.getContents(handle);
tasks.setTaskRoot(vc);
Container taskList = (Container)(vc.getBody());
Item[] taskItems = taskList.getItem();
if (taskItems == null)
{
    System.out.println("There are no Active tasks");
}
else
{
    for (int i = 0; i < taskItems.length; i++)
    {
        // get the details on the task
       ViewContents taskVC = serviceConnection.getContents(taskItems[i].getKey());
        Task task = (Task) taskVC.getBody();
    }
}
```

Clients can monitor a task for its progress by calling GetUpdates on the handle for the task of interest. The client can look at the task's current run state and percentCompleted values to chart a task's progress and determine when it completes. The following sample illustrates this concept.

```
// taskHandles is a VHandleList of all the vHandles of the tasks
```

```
// the client wants to get notified of updates on
UpdateList updateList = serviceConnection.getUpdates(taskHandles, true);
Update[] updates = updateList.getUpdate();

for (int i = 0; i < updates.length; i++)
{
    Change[] changes = updates[i].getChange();
    for (int j = 0; j < changes.length; j++)
    {
         Change change = changes[i];
         String target = change.getTarget();
         if (!target.startsWith("item[\""))
         {
             // updates to the /task or /schedule container
             // add new task on if ins op
             // delete completed task if del op
             processTasks(changes);
             // update the client vhandle for /task or /schedule container
         }
         else
         {
             // update to an existing task. Apply updates to task
             applyUpdates(task, change);
             // update the task's VHandle
             // display task's updated progress (runState, percentCompleted)
         }
    }
}
```

## Creating New Scheduled Tasks

Clients can create new scheduled tasks on the Web service. The following operations can be scheduled as a task:

- Power operations — StopVM, StartVM, and ResetVM.
- PutUpdates — changing the resource settings of a virtual machine. Clients must pass a handle (and not a vHandle) for a scheduled PutUpdates operation.
- MigrateVM
- MoveVM
- CloneVM
- CreateTemplate

To create a task schedule, clients need four parameters, as specified by the TaskScheduleSpec datatype:

- name — Name of the task schedule.
- operationName — Name of the scheduled API operation.
- parameter — Parameters for the operation. If the operation requires no parameters, then this field is not required.
- recurrence — Recurrence of this task schedule.

Once a task schedule is created, there are additional parameters that define the task schedule. Refer to the description for the TaskSchedule datatype in the *Virtual Infrastructure SDK Reference Guide*.

The following sample code illustrates creating a scheduled task that performs the power-on operation.

```
KeyedValue [] args = getArgs(operationName, vmHandle);
Object recurrence = getRecurrence();
TaskScheduleSpec task = new TaskScheduleSpec();
task.setName("powerOnVMTask");
task.setOperationName("StartVM");
task.setParameters(args);
task.setRecurrence(recurrence);


// Get the handle for /schedule
String parentHandle = serviceConnection.resolvePath("/schedule");

// Create this task on the server
String taskHandle = serviceConnection.create(parentHandle, name,
"TaskSchedule", taskSchedule);
return taskHandle; // handle of newly created task
```

In the preceding sample, the args array is an array of KeyedValue objects that represent the input parameters for the operation.

The operation name specified in the TaskScheduleSpec datatype must match exactly the operation name specified in the WSDL; for example, when setting up a task to clone a virtual machine, the operation name should be CloneVM. Similarly, the parameter names should exactly match the parameter names for that operation as specified in the WSDL. The following sample code illustrates the parameters for the StartVM operation.

```
KeyedValue[] args = new KeyedValue[2];
```

```
args = new KeyedValue[1];
args[0] = new KeyedValue();
args[0].setKey("vm");
args[0].setVal(vmHandle);
```

The recurrence for a scheduled task is an object as specified in the preceding table. Each object has a set of parameters that specify when to run the task. For example, clients can specify a weekly task:

```
WeeklyTask weekly = new WeeklyTask();
// What time of the day this task should fire
weekly.setHours(6);
weekly.setMinutes(5);
// Run this task every xx weeks
weekly.setInterval(2);
// On which days of the week should this fire
Weekday weekdays [] = new Weekday[2];
weekdays[0] = Weekday.thursday;
weekdays[1] = Weekday.sunday;
weekly.setDayOfWeek(weekdays);

task.setRecurrence(weekly);
```

## Running a Scheduled Task

Clients can run a scheduled task on the Web service by calling the RunTask operation. This operation takes one argument, the handle of the task. The client must have Interact rights for the specified task.

Upon success, an empty response message is returned.

```
String taskHandle; // handle of task to run
serviceConnection.runTask(taskHandle);
```

## Ending a Task

Clients can stop a running task, or cancel a task that has not yet been started on the Web service by calling the EndTask operation. This operation takes one argument, the handle of the task. The client must have Interact rights for the specified task.

Upon success, an empty response message is returned.

```
String taskHandle; // handle of task to cancel
serviceConnection.endTask(taskHandle);
```

# Collecting Performance Data

The client can obtain performance data, as specified by the Performance Metric data model, described in the *Virtual Infrastructure SDK Reference Guide*. Clients can collect performance data on hosts or on virtual machines, including CPU and memory utilization, network and disk performance data, and floppy and CD-ROM drive performance, and so on.

Clients obtain performance statistics through a performance collector. A performance collector, also known as a perf collector, is an object that collects a certain set of statistics at a specified interval frequency.

The view, `/perf`, is a container for all the performance collectors. There are two types of performance collectors:

- VirtualCenter perf collectors — VirtualCenter perf collectors specify a sampling interval and are visible in the VMware VirtualCenter application.
- Filtered perf collector — Filtered perf collectors (children) are filtered from an existing VirtualCenter perf collector (parent) and are not visible in the VirtualCenter application.

Clients can create both types of performance collectors by using the Create operation.

- A new VirtualCenter perf collector with a different sampling interval from any other VirtualCenter perf collectors.
- A perf collector that filters the statistics of an existing VirtualCenter perf collector.

## VirtualCenter Perf Collector

VirtualCenter performance collectors are named according to their sampling interval. They have only a sampling interval and no default filter. There are four default VirtualCenter perf collectors:

- Five minutes — `/perf/0000000300`. Historical samples are retained for a day.
- One hour — `/perf/0000003600`. Historical samples are retained for a week.
- Six hours — `/perf/0000021600`. Historical samples are retained for a month.
- One day — `/perf/0000086400`. Historical samples are retained for a year.

Clients can also create a VirtualCenter perf collector by using the Create operation. When doing so, specify only the name and the sampling interval (initial parameter comprising XML document of type PerfCollector). See Creating a VirtualCenter Perf Collector on page 122.

**Note:** When creating a VirtualCenter perf collector, the user-friendly name does not appear in the `/perf` directory. Instead, the newly created VirtualCenter perf collector appears as its sampling interval. However, the user-friendly name appears in the VirtualCenter application.

For example, if you create a new VirtualCenter perf Collector called "Every Minute" with a sampling interval of 60 seconds, it appears in the VirtualCenter application as "Every Minute" but appears as its sampling interval, `/perf/0000000060`.

Because VirtualCenter perf collectors only have a sampling interval, they collect statistics on all virtual machines and all hosts. Because updating the contents of these VirtualCenter performance collectors for each sampling interval would generate much communication traffic; by default, the contents of VirtualCenter performance collectors do not keep current data.

If you want the contents of VirtualCenter perf collectors to be up-to-date, then change the default behavior by setting the `periodicPerfRefreshEnable` config variable to TRUE in the Web service configuration file (`vmaConfig.xml`). If this variable is set to TRUE, then the contents of all VirtualCenter perf collectors are also kept up-to-date.

## Filtered Perf Collectors

Filtered perf collectors are "children" derived from a VirtualCenter perf collector. Each VirtualCenter perf collector can have multiple "children" filtered perf collectors. However, a filtered per collector cannot be used to create additional "children" filtered perf collectors.

As discussed in the previous section, VirtualCenter perf collectors have only a sampling interval, and therefore, collect statistics on all virtual machines and all hosts. Since it is impractical to collect all this performance data, clients should create "filtered perf collectors" to monitor the current values of particular performance statistics. Unlike VirtualCenter perf collectors, the contents of filtered perf collectors are kept up-to-date.

Filtered perf collectors typically specify a source (host or virtual machine) and the performance statistics of interest. For example, a default VirtualCenter perf collector has a sampling interval of five minutes. By filtering this VirtualCenter perf collector, clients can create a filtered perf collector collecting performance statistics every five minutes on "host A", another that collects memory statistics every five minutes on "John's virtual machine", and so on.

Clients also create a filtered perf collector by using the Create operation. See Creating a Filtered Perf Collector on page 122. When doing so, specify the name and the filter (initial parameter comprising XML document of type PerfCollector). This XML document *must include* the handle to the parent VirtualCenter perf collector. The filter should also include the source (host or virtual machine being queried), the samples, and the performance statistics of interest.

**Note:** Do not specify the sampling interval. The filtered perf collector has the same sampling interval as its parent VirtualCenter perf collector. The WSDL stubs insert a value of zero (`0`) that is ignored by the Web service.

**Note:** If clients leave the spec field uninitialized (or NULL), then all performance statistics from the source are returned.

**Note:** Unlike VirtualCenter perf collectors, the user-friendly name selected for the filtered perf collector appears in the `/perf` directory, but does not appear in the VirtualCenter application.

## Comparing VirtualCenter and Filtered Perf Collectors

The following table summarizes the differences between VirtualCenter and filtered perf collectors.

| Characteristic | VirtualCenter Perf Collector | Filtered Perf Collector |
|---|---|---|
| What does it specify? | Specifies a unique sampling interval. Does not specify any other filter. | Specifies the source (of the performance statistics) and defines the performance statistics (CPU, memory, and so on).<br>Uses the same sampling interval as its "parent" VirtualCenter perf collector. |
| Can be created? | Yes. Use the Create operation and specify a name, sampling interval, and total number of samples. | Yes. Use the Create operation and specify a name, a parent VirtualCenter perf collector, and a filter. |
| Can be deleted? | Yes. Use the Delete operation. | Yes. Use the Delete operation. |
| Life expectancy. | Persists until deleted. | Persists until deleted or the Web service is stopped. When the Web service is restarted, filtered perf collectors no longer exist. |
| Filters an existing perf collector? | No. VirtualCenter perf collectors have only a name and a sampling interval. | Yes. Filtered perf collectors filter an existing VirtualCenter perf collector. |
| Gets current performance statistics? | No, unless the `periodicPerfRefreshEnable` config variable is set to TRUE. This is set to FALSE by default. | Yes. Filtered perf collectors can get current statistics through the GetContents and GetUpdates operations. |
| Used to create filtered perf collectors? | Yes. Clients can create filtered perf collectors from a VirtualCenter perf collector. | No. Clients cannot "filter" a filtered perf collector. |
| Query historical performance statistics? | Yes. Use the QueryPerfData and QueryPerfData2 operations. Specify the handle of the perf collector and optionally, a filter. | Yes. Use the QueryPerfData and QueryPerfData2 operations. Specify the handle of the perf collector. Do not specify a filter. |

## Performance Metric Data Model

A diagram of the Performance Metric data model is included in the *Virtual Infrastructure SDK Reference Guide*. Here, we provide a brief summary of this model.

A PerfCollector is a collector of performance statistics. Each Perf Collector includes a filter (specifying the sample interval and the performance statistics that are collected) and stats, referring to the actual performance statistics that are collected (a PerfCollection).

Each PerfCollection has an array of statistics from various sources, either hosts or virtual machines. Each PerfSource has an array of samples for that source. Each PerfSample has an array of statistics for the sample from that source. Each stat is a PerfStat, and can have a type specified by PerfStatType, that describes the type of performance statistic: cpu, net, disk, floppy, and so on.

The actual statistics are in the `data` field of the PerfStat, and comprise many different datatypes: CPUPerf, MemoryPerf, NetPerf, and so on.

## Creating a VirtualCenter Perf Collector

Clients use the Create operation to create a new VirtualCenter perf collector, as shown in the following sample.

```
// Creates a New Virtual Center Perf Collector.
// serviceConnection is a reference to vmaBindingStub

String pcName = "MyPerfCollector";

String parentHandle = serviceConnection.resolvePath("/perf");
PerfCollector p = new PerfCollector();
PerfFilter f = new PerfFilter();
f.setInterval(180);
f.setSamples(new Integer(2));
pc.setFilter(f);

String handle =
    serviceConnection.create(parentHandle, pcName, "PerfCollector", pc);
```

## Creating a Filtered Perf Collector

This sample illustrates creating a filtered perf collector, from the parent VirtualCenter perf collector created in the preceding section.

```
// Creates a Filtered Perf Collector for a Host.
// serviceConnection is a reference to vmaBindingStub

String src1path = "/vcenter/Farm/yourhostnamehere";
String filteredColName = "MyFilteredCollector";

PerfFilter f = new PerfFilter();
f.setSpec(new PerfSourceType[2]);
```

```
f.setHandle(serviceConnection.resolvePath("/perf/0000000300"));

int samples = 1;
f.setSamples(new Integer(samples));
p.setFilter(f);

PerfSourceType s = new PerfSourceType();
f.setSpec(0, s);
s.setKey(serviceConnection.resolvePath(src1path));
s.setSample(new PerfType[1]);
// Add more sources here as needed...

PerfType t = new PerfType();
s.setSample(0, t);
t.setType(PerfStatType.cpu);
t.setDevice("0");

//Add more stats here as needed.
String perfParentHandle = serviceConnection.resolvePath("/perf");

String handle =
    serviceConnection.create(perfParentHandle, filteredColName,
                             "PerfCollector", p);

ViewContents pcContents = serviceConnection.getContents(handle);
// Now call getUpdates on the stats object of this collector...
```

## Collecting Current Performance Data

We then call the GetContents operation on the newly created filtered perfCollector (by calling GetContents on the perfCollector handle returned from the Create operation).

This operation returns a perfCollector Container that contains two Items, the PerfFilter and PerfCollection (statistics) datatypes. Each Item contains a key, that is the handle to the Item. To get the latest statistics, clients should use the handle to the PerfCollection object and first call the GetContents operation, then the GetUpdates operation on this PerfCollection object.

The following sample illustrates this concept.

```
ViewContents pcContents =
serviceConnection.getContents(filteredperfcollectorhandle);

Container filtPerfCollector = (Container)pcContents.body;

// get the filter item and stats collection item.
Item filterItem = filtPerfCollector.item[0];
```

```
Item statsItem = filtPerfCollector.item[1];
if (statsItem.type.equals("PerfFilter")) {
   statsItem = filtPerfCollector.item[0];
   filterItem = filtPerfCollector.item[1];
}

// call getcontents with these item's handles to get
// the PerfFilter and PerfCollection Objects.

ViewContents filterContents = serviceConnection.getContents(filterItem.key);
ViewContents statsContents = serviceConnection.getContents(statsItem.key);

PerfCollection stats = statsContent.body;
// use this PerfCollection to get statistics for.
```

The following sample code illustrates how to extract the appropriate statistic from the PerfCollection object:

```
    // serviceConnection is a reference to vmaBindingStub
    // Assumes you have already got the PerfCollection Object.
    // stats is of type PerfCollection
    PerfSource[] source = stats.getSource();
    // Sources from where statistics has been pulled (hosts/VMs)
    for (int i = 0; i < source.length; i++) {
        ViewContents vc = serviceConnection.getContents(source[i].getKey());
        if (vc.getBody() instanceof Host) {
            Host host = (Host) vc.getBody();
            String hostName = host.getInfo().getHostname();
            System.out.println("Stats for host:" + hostName);
            PerfSample[] sample = source[i].getSample();
            for (int j = 0; j < sample.length; j++) {
                PerfStat[] statsSample = sample[j].getStat();
                for (int k = 0; k < statsSample.length; k++) {
                    PerfStat stat = statsSample[k];
                    PerfStatType statType = stat.getType();
                    // The app should allow the user to pick
                    // the device to show data for.
                    // Here we are showing CPU data only
                    if (statType == PerfStatType.cpu) {
                        CPUPerf data = (CPUPerf) stat.getData();
                        // display the CPUPerf data
                    }
                }
            }
        }
    }
```

## Collecting Historical Data

Similarly, clients can use the QueryPerfData and QueryPerfData2 operations on both VirtualCenter and filtered performance collectors to obtain historical performance data, for any specified time period. The time period may be completely in the past, or it may be specified to extend up to, and including, the most recent update.

When using the QueryPerfData and QueryPerfData2 operations, clients must specify the handle to the perf collector. Specify the filter parameter only when querying a VirtualCenter perf collector.

**Note:** Do not use the filter parameter only when querying a filtered perf collector (as it is already "filtered"). If the filter parameter is specified for a filtered perf collector, the operation returns an error.

**Note:** If you see output similar to `/vpx/vm/#000512` or `/vpx/host/#000a376` for the PerfSourceType "key" field in the PerfCollection object, then the specified virtual machine or host has been deleted.

The following sample illustrates how a client calls the QueryPerfData operation.

```
/*
 * specify the source for which to collect stats for
 * Here we specify the 1st CPU of the host specified in
 * the command line parameters.
 */

// serviceConnection is a reference to vmaBindingStub

String hostPath = "/vcenter/Farm/yourhostnamehere";
String fromDateTime = "07/10/2004 1:00, PDT";

PerfFilter sourceFilter = new PerfFilter();
sourceFilter.setSpec(new PerfSourceType[1]);
PerfSourceType s = new PerfSourceType();
sourceFilter.setSpec(0, s);
s.setKey(serviceConnection.resolvePath(hostPath));
s.setSample(new PerfType[1]);

PerfType t = new PerfType();
s.setSample(0, t);
t.setType(PerfStatType.cpu);
t.setDevice("0");

DateFormat df = new SimpleDateFormat("MM/dd/yyyy HH:mm, z");
Date chgDate = df.parse(fromDateTime);
Calendar calDate = Calendar.getInstance();
calDate.setTime(chgDate);
```

```
PerfCollector col =
    serviceConnection.queryPerfData(handle, calDate, 1, sourceFilter);
```

Clients call the QueryPerfData2 operation exactly as they call the QueryPerfData operation. However, the QueryPerfData2 operation returns the new CPUPerf2, MemoryPerf2, and VirtualMachineMemoryPerf2 datatypes whereas the QueryPerfData operation returns the CPUPerf, MemoryPerf, and VirtualMachineMemoryPerf datatypes. These new datatypes contain additional statistics for ESX Server hosts and virtual machines on ESX Server.

**Note:** Clients can only obtain the statistics from the CPUPerf2, MemoryPerf2, and VirtualMachineMemoryPerf2 datatypes through the QueryPerfData2 operation. If you are already calling the GetUpdates operation on the PerfCollection object, then be sure to call the QueryPerfData2 operation to obtain these extra statistics.

This next sample shows how to display the statistics from CPUPerf2, returned after calling the QueryPerfData2 operation.

```
CPUPerf2 data = (CPUPerf2) stat.getData();
Integer pcpu = data.getPcpu();
Long system = data.getSystem();
Long ready = data.getReady();
Long wait = data.getWait();
Long used = data.getUsed();

if (used != null) {
    System.out.println("CPU used: " + used);
}
if (system != null) {
    System.out.println("CPU System time: " + system);
}
if (ready != null) {
    System.out.println("CPU Ready time: " + ready);
}
if (wait != null) {
    System.out.println("CPU Wait time: " + wait);
}
if (pcpu != null) {
    System.out.println("CPU PCPU: " + pcpu);
}
```

# Changing Permissions

Clients can view and change permissions on any object by using the VMware SDK. To view the current permissions, clients either call the GetInfo operation on the object or call the GetContents operation on the parent Container of the object. Each Item that is returned in the Container's Item list contains the permissions associated with that Item.

The permissions are returned in an array of Permission objects. Each permission object has two String fields: the key field (identifies the user with the assigned permissions) and the rights field (indicates the permissions granted to the user).

Once the current permissions are known, clients can modify these permissions by using the ChangePermissions operation. The ChangePermissions operation takes two arguments: the vHandle (versioned identifier of the object) and a PermissionList (encapsulates the new Permissions array).

By using the ChangePermissions operation, clients can add new permissions, or delete or modify existing permissions. To add new permissions, clients should add a new Permission entry to the Permissions array, then send the new Permissions array to the ChangePermissions operation.

Similarly, clients can delete permissions by removing the desired entries from the Permissions array.

Clients can modify permissions by modifying the permission object in the Permissions array, then sending the updated Permissions array to the ChangePermissions operation.

**Note:** The new PermissionList that is specified in the ChangePermissions operation replaces all the existing permissions on the object. If you don't want to delete any existing entries, then the client must also send back all the existing entries to the ChangePermissions operation.

The following code sample illustrates how to invoke the ChangePermissions operation.

```
String handle =
    serviceConnection.resolvePath(
        "/vcenter/MyFarmGroup/Farm/myVirtualMachine");
ViewContents vc = serviceConnection.getContents(handle);
ViewInfo info = serviceConnection.getInfo(handle);
Permission[] perms = info.getPerm();

...edit permissions here (add / delete / modify perms array)
PermissionList permList = new PermissionList();
permList.setPerm(perms);
serviceConnection.changePermissions(vHandle, permList);
```

# Taking a Snapshot of a Virtual Machine

The following code sample illustrates using the new snapshot operations. The first, SnapshotVM, takes a "snapshot" (picture) of a virtual machine at a particular point in time. The second, RevertVM, discards any existing snapshot and returns the virtual machine to its state preceding the snapshot. The third, ConsolidateVM, commits the changes contained in the snapshot, to the base virtual disk(s).

```
String vmHandle = serviceConnection.resolvePath(args[3]);
if (args[4].equals("snapshot")) {
    task = serviceConnection.snapshotVM(vmHandle);
} else if (args[4].equals("revert")) {
    task = serviceConnection.revertVM(vmHandle);
} else if (args[4].equals("consolidate")) {
    task = serviceConnection.consolidateVM(vmHandle);

// Monitor task here...
```

# Exception Handling and Faults

The Web service can encounter an exception or error (also referred to as a fault) while executing an operation. We previously described how these exceptions can occur (Handling Exceptions in the Data Synchronization Loop on page 84). Details about these errors is encapsulated in the FaultInfo object. This object extends the `java.lang.Exception` class and contains specific details on the exception that occurred. The FaultInfo object has three attributes:

- FaultKind — Kind of fault occurred
- Code associated with the fault, classified like the HTTP error codes
- Value that contains additional information on the fault, if applicable.

The following table lists the various faults and the values that are present in the FaultInfo object, if applicable.

| FaultKind | Description | FaultInfo.info | FaultInfo.val |
|---|---|---|---|
| AlreadyExists | | String, with the remainder of the path where the exception occurred. | Not applicable. |
| BadName | | String, with the remainder of the path where the exception occurred. | Not applicable. |
| NotDirectory | | String, with the remainder of the path where the exception occurred. | Not applicable. |
| NotFound | | String, with the remainder of the path where the exception occurred. | Not applicable. |
| PermissionDenied | | String, with the remainder of the path where the exception occurred. | Not applicable. |
| HandleFault | | | HandleFaultList with at least one HandleFaultInfo containing specific data. |
| NotApplicable | | Handle that is the incorrect type; for example, a non-virtual machine handle specified to the StartVM operation. | |

| FaultKind | Description | FaultInfo.info | FaultInfo.val |
|---|---|---|---|
| ChangeConflict | Not yet thrown. | String with target expression that failed. | |
| NoResources | Not yet thrown. | | |
| BadState | Not yet thrown. | String describing the state of the virtual machine when the exception occurred. | |
| Interrupted | | No additional data is provided. | No additional data is provided. |
| BadRequest | | Text from input that caused the error. | |

The following sample shows how to handle some of the faults listed in the preceding table. Refer to the *Virtual Infrastructure SDK Reference Guide* to see the faults thrown by each API operation.

```
try
{
    // Initialize connection to Web service and login
    // some VMA operation invoked here e.g. resolvePath/getContents/StartVM etc.
    // Log out
}
catch (FaultInfo faultInfo)
{
    handleExceptions(faultInfo);
}
catch (Exception ex)
{
    // some other exception
    ex.printStackTrace(System.out)
}
```

The following sample illustrates retrieving the fault information from the Exception (FaultInfo) object:

```
FaultKind faultKind = faultInfo.getKind();
System.out.println("Got Fault:" + faultKind.toString());
if (faultKind.equals(FaultKind.AlreadyExists))
{
    // thrown from Create, CreateVirtualDisk
    System.out.println("The target name is already defined at path:" +
        faultInfo.getInfo());
}
else if (faultKind.equals(FaultKind.HandleFault))
{
    HandleFaultList handleFaultList = (HandleFaultList) faultInfo.getVal();
    processHandleFault(handleFaultList);
```

```
} // other exceptions handled similarly
```

The following sample illustrates processing the HandleFault exception:

```
HandleFaultInfo [] handles = handleList.getInfo();
for (int i = 0; i < handles.length; i++)
{
    HandleFaultKind faultKind = handles[i].getKind();
    if (faultKind.equals(HandleFaultKind.ObsoleteHandle))
    {
        // delete handle from client's data structure
    }
    else if (faultKind.equals(HandleFaultKind.BadHandle))
    {
        System.out.println("Not a recognized handle:" + handles[i].getHandle());
    }
    else if (faultKind.equals(HandleFaultKind.BadVersion))
    {
        System.out.println("Bad versioned handle. Versioned handle :" +
            "not recognized:" + handles[i].getHandle());
    }
    else if (faultKind.equals(HandleFaultKind.Unauthorized))
    {
        System.out.println("Unauthorized access. Client does not have access " +
            " to perform the operation on:" + handles[i].getHandle());
    }
}
```

CHAPTER 7

# Core Client Concepts for Perl Programmers

This chapter shows you how to write a good client application in Perl using SOAP::LITE. See SOAP::LITE for Perl on page 274.

The two previous chapters contained the same information, written in Java. You may use a different developer environment and language to build your client program. Adjust the examples accordingly for your developer environment.

**Note:** All the code samples in this chapter, are located in their entirety in `\SDK\WebService\samples\perl\sampleapp`.

This chapter covers the following topics:

# Using SOAP::LITE with VMware SDK

SOAP::Lite is an open source collection of Perl modules that provides a simple and lightweight interface to SOAP and is currently the standard for designing Perl-based Web service applications. SOAP::Lite is limited in its support for WSDL schemas. By default, it supports the RPC/Literal schema whereas our `vma.wsdl` uses the Document/Literal schema.

To get around this issue, all SOAP messages are slightly modified to adhere to the Doc/Literal schema before transmission. We build a wrapper around all SDK method calls, specifying the namespace and providing a mapping between argument names and their respective values. With this modification, SOAP::Lite works with our Web service and can support the full complement of SDK operations.

For example:

```
$soap->Login('Administrator','foobar');

We use:

my $method = SOAP::Data->name('Login')
              ->attr({xmlns => 'urn:vma1'});

my @params = (
               SOAP::Data->name(userName => 'Administrator'),
               SOAP::Data->name(password => 'foobar')
              );

$service->call($method => @params)->result;
```

The syntax for running the Perl sample applications is:

```
perl <ScriptName>.pl <WebService_URL> <username> <password> <other parameters>
```

where `<WebService_URL>` is the VMware VirtualCenter Web Service URL, `<username>` is the user name used to log into VMware VirtualCenter, and `<password>` is the password you would for the VirtualCenter user. `<other_parameters>` represents any other parameters that you may specify, when running a specific operation.

# Creating a SOAP::LITE Object

The following code sample demonstrates how to create a SOAP::Lite object by specifying the URL of the VMware VirtualCenter Web Service. The SOAP::Lite object is configured to maintain cookies and also includes a custom deserializer (SdkDeserializer) for parsing internal VMA datatypes. It also includes a fault handler for handling SOAP faults and transport errors. This SOAP::Lite object is then used to call all SDK methods.

```
my $webserviceURL = 'http://localhost:8080';  # URL of web service

my $vmaWSDL = $webserviceURL.'?wsdl';         # Location of vma.wsdl

my $service =

  SOAP::Lite

    -> service( $vmaWSDL )

    -> proxy($webserviceURL,
         cookie_jar => HTTP::Cookies->new(ignore_discard => 1), timeout => 0)

    -> deserializer(SdkDeserializer->new)

    -> on_fault(sub{FaultHandler(@_);});
```

## SOAP::Lite Deserializer

SOAP::Lite lacks support for handling WSDL-specified custom datatypes. While complex types are handled by returning a nested structure of hashes, this is a problem for simple types that appear as leaf elements in a SOAP message. When such simple types are encountered, SOAP::Lite fails to recognize the type and throws a fault. To avoid problem, we override the `typecast()` method in the SOAP::Deserializer package by defining the following custom deserializer.

```
######################################################################
#
# SdkDeserializer --
#
#    SdkDeserializer is a subclass of the default SOAP::Deserializer
#    and overrides the typecast() method of the deserializer to
#    handle VMA types.
#
```

```
# Results:
#    Returns the value for custom types defined under the urn:vma1
#    namespace.
#
# Side effects:
#       None.
#
#########################################################################


BEGIN {
    package SdkDeserializer;
    @SdkDeserializer::ISA = 'SOAP::Deserializer';

    #
    # Overriding typecast() method of SOAP::Deserializer to return
    # value of VMA types
    #

    sub typecast
    {
         my $self = shift;
         my ($value, $name, $attrs, $children, $type) = @_;

         if ( $type ){
            if( rindex ( $type,"urn:vma1") != -1 ){
                   return $value;
            }
         }
         return undef;
    }
}
```

# Logging into the Web Service

The following sample illustrates how to log into the Web service using the WSDK.

```
my $method = SOAP::Data->name('Login')
            ->attr({xmlns => 'urn:vma1'});

my @params = (
               SOAP::Data->name(userName => 'Administrator'),
               SOAP::Data->name(password => 'MyPasword')
             );

$service->call($method => @params)->result;
```

# Permissions

The client must have the right set of permissions on the object being updated or retrieved in order to invoke the different operations. For example, the client must have Browse rights for the GetContents operation, Interact rights for the GetUpdates operation, Configure rights for the PutUpdates operation, and Administer rights for the ChangePermissions operation. Refer to the *Virtual Infrastructure SDK Reference Guide* for the required permission for each operation. Also, see Changing Permissions on page 127 for an example of changing the permission of an object.

# Retrieving the Handle for an Object

Many of the Perl samples in these next two chapters include a ResolvePath operation. This operation takes a path as its input parameter and returns a handle to the object referenced by the path.This handle is then used by other SDK operations.

```
#
# Setup & call the ResolvePath method
#

my $method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

my @params = (SOAP::Data->name(path => '/vm'));

my $handle = $service->call($method => @params)->result;
```

# Getting Basic Information about an Object

Client applications can use the GetInfo operation to retrieve information about an object. This operation takes one argument, the handle of the object of interest. The return value is a ViewInfo object that contains the handle of the parent Container of this object, the name of the object, the type of the object, and the list of permissions on the object.

The following sample code demonstrates how to invoke the GetInfo operation and retrieve the result.

```
#
# Setup and call the GetInfo method
#

my $method = SOAP::Data->name('GetInfo')
    ->attr({xmlns => 'urn:vma1'});

my @params = (SOAP::Data->name(handle => $handle));

my $result = $service->call($method => @params)->result;


#
# Display information
#

print "\nParent : $result->{parent}";
print "\nName : $result->{name}";
print "\nType : $result->{type}";
```

# Object Inventory

You can write a client application to obtain a list of hosts or virtual machines available to you. Once you obtain these lists, you can request for information about a specific host or virtual machine. Then, you can request updates when they occur.

Clients use the GetContents operation to return information about the following objects, provided you have Browse access to these objects.

- Containers (Server Farms and Farm groups)
- Farm
- Host
- Virtual machine group
- Virtual machines

## Handling Complex Objects in SOAP::Lite

SOAP::Lite deals with complex objects by parsing them into a nested structure of hashes and returning a hash reference. As a result, when there are arrays of complex objects returned (for example, arrays of Item or Change objects), then the array elements are overwritten in the hash and only the last Item in the array can be retrieved. To prevent this, we avoid calling the default `result` method to retrieve SOAP responses and instead retrieve the top-level envelope object. This object can be manually traversed and parsed to retrieve the individual array elements as defined in the `PrintContents` method.

### Tie::IxHash

In SOAP::Lite, the concept of constructing complex objects to pass as parameters for methods, follows the same principles as retrieving these objects; we construct a hash to represent the object. To ensure ordering of elements in the hash, we use the Tie::IxHash module. The syntax for its usage is:

```
tie %$<hash-name>, "Tie::IxHash";
```

This binding must be repeated for each nested level in the hash, when ordering must be maintained. When an object contains nested arrays, it needs to be handled differently as illustrated in StartVM Operation Parameters on page 220, but in most cases, an ordered hash is sufficient to represent complex objects.

## Using GetContents to Obtain Information About Hosts and Virtual Machines

For example, if you wanted to obtain a list of all the hosts, you can issue a GetContents request on `/host`. Similarly, if you wanted to obtain a list of all virtual machines, for all hosts, you can issue a GetContents request on `/vm`, as illustrated in Simple Client Program in Perl on page 53.

For example, if you have Browse rights on two virtual machines, then the output returned from the GetContents request for `/vm` is:

```
handle   vma-domain-00000000001
vHandle   vma-domain-00000000001@fe11fe5439000
body
   item
      key   vma-host-00000000001
      name   user-dev.vmware.com
      type   Host
   item
      key   vma-vmgroup-00000000002
      name   Discovered VMs
      type   VirtualMachineGroup
   item
      key   vma-vm-00000000027
      name   clonetest3
      type   VirtualMachine
      perm
         key   user
         rights   Interact
```

## Using GetContents to Obtain Information About Individual Hosts and Virtual Machines

Once you obtain the list of hosts (virtual machines), you can use the GetContents operation to obtain detailed information about an individual host machine or virtual machine. For example, by issuing a GetContents request on a specific host, you can see information about its hardware, status, resource allocations, networking devices, and so on, as indicated by the Host Machine data model in the *Virtual Infrastructure SDK Reference Guide*.

Similarly, by issuing a GetContents request on a specific virtual machine, you can see information about its virtual hardware, status, resource allocations, virtual networking, guest operating system, and so on, as indicated by the Virtual Machine data model in the *Virtual Infrastructure SDK Reference Guide*.

The following sample illustrates how to use the GetContents operation for a host view. Clients can obtain the host handle by calling the ResolvePath operation on either its nested path under the `/vcenter` hierarchy or by using its FQDN (Fully qualified domain name) under `/host`. The example below uses the host's nested path in the `/vcenter` hierarchy.

```
#
# Get Contents for /vcenter/New Farm Group/New Farm/<Fully qualified hostname>
# Alternate path for host could be /host/<mytesthost.mydomain.com>
#

$path = '/vcenter/<New Farm Group>/<New Farm>/<Fully qualified hostname>';

#
# Setup & call the ResolvePath method
#

my $method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

my @params = (SOAP::Data->name(path => $path));

my $handle = $service->call($method => @params)->result;


#
# Setup and call the GetContents method.
#

$method = SOAP::Data->name('GetContents')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(handle => $handle));

#
# Avoid calling result() method on SOAP response to prevent
# overwriting of arrays
#

my $result = $service->call($method => @params);

PrintContents($result);


######################################################################
#
```

```
# PrintContents --
#
#      Display subroutine that accepts a reference to an envelope
#      object and recursively parses the structure to display the
#      contents.
#
# Results:
#      Displays contents of an envelope object.
#
# Side effects:
#       None.
#
########################################################################

sub PrintContents
{
     my ( $result, $path, $tab ) = @_;

     #
     # Define the path for the outermost object of interest
     #

     if (!defined $match ) {
          $match = "//returnval/*";
     }

     #
     # Set initial offset to zero
     #

if (!defined $tab){
     $tab = 0;
}

     #
     # Iterate through each element in the current object
     # and check to see if it contains a text value or a
     # child object
     #

     my $i =0;
     foreach my $element ($result->dataof( $match )) {
          $i++;
          print "\n". "   " x $tab.$element->name."   ";

          #
          # Element contains child object. Adjust path for the
```

```
            # new object and recursively call subroutine to parse
            # the child object
            #

            if ( ref $element->value ) {
                 $newMatch = $match;
                 chop ( $newMatch );
                 $newMatch = $newMatch."[$i]/*";
                 PrintContents ( $result, $newMatch, $tab+1 );
            } else {

                 #
                 # Element contains text value
                 #

                 print $element->value;
            }
       }
}
```

This results in sample output similar to the following:

```
handle vma-host-00000000008
vHandle vma-host-00000000008@fe96a2e1c4000003
body
   info
      uuid 44 45 4c 4c 47 00 10 34-80 53 b4 c0 4f 36 34 31
      hostname myhost.mydomain.com
      port 902
      system
         name VMware ESX Server
         type esx-server
         version 2.5.0
         build build-9746
     datastore [Local]
     datastore [Shared]
     network
         key Adapter0 Network
     network
         key Internal Network
     configLimits
         maxAudio 1
         maxFloppy 2
         maxNet 4
         maxParallel 3
         maxSerial 4
```

```
maxUSBController 1
ideLimits
   maxControllers 2
   maxDevicesPerController 2
scsiLimits
   maxControllers 4
   maxDevicesPerController 16
guestOS
   key other
   label Other
   supported false
   maxCpu 1
   defaultDiskSize 4294967296
   defaultColorDepth 16
   defaultDiskControllerType ide
   defaultSCSIAdapterType lsiLogic
   memory
      minHostMb 1
      minMb 32
      maxMb 1390
      defaultMb 256
guestOS
   key netware6
   label NetWare 6
   supported true
   maxCpu 1
   defaultDiskSize 4294967296
   defaultColorDepth 8
   defaultDiskControllerType scsi
   defaultSCSIAdapterType lsiLogic
   memory
      minHostMb 1
      minMb 512
      maxMb 1390
      defaultMb 512
guestOS
   key linux
   label Linux
   supported true
   maxCpu 1
   defaultDiskSize 4294967296
   defaultColorDepth 16
   defaultDiskControllerType scsi
   defaultSCSIAdapterType lsiLogic
   memory
      minHostMb 1
      minMb 32
```

```
            maxMb 1390
            defaultMb 256
      guestOS
         key winNetEnterprise
         label Windows Server 2003 Enterprise Edition
         supported true
         maxCpu 1
         defaultDiskSize 4294967296
         defaultColorDepth 8
         defaultDiskControllerType scsi
         defaultSCSIAdapterType lsiLogic
         memory
            minHostMb 1
            minMb 128
            maxMb 1390
            defaultMb 384
      guestOS
         key win2000AdvServ
         label Windows 2000 Advanced Server
         supported true
         maxCpu 1
         defaultDiskSize 4294967296
         defaultColorDepth 8
         defaultDiskControllerType scsi
         defaultSCSIAdapterType lsiLogic
         memory
            minHostMb 1
            minMb 128
            maxMb 1390
            defaultMb 384
      vmLimit
            key #1
            cpuCount 1
            maxMb 1672
  migrationEnabled false
  migrationInfo
  customPropertyDef vma-global-conf
hardware
  cpu
     key #00000001
     description Intel(R) Xeon(TM) CPU 2.80GHz
     family 15
     features -1075053569
     id 1
     mhz 2784
     model 2
     stepping 9
```

```
        type 0
        vendor intel
     cpu
        key #00000000
        description Intel(R) Xeon(TM) CPU 2.80GHz
        family 15
        features -1075053569
        id 0
        mhz 2784
        model 2
        stepping 9
        type 0
        vendor intel
     memory
        sizeMb 2034
   state
     status enabled
     detail connected
     connected true
     bootTime 1969-12-31T16:00:00-08:00
     eventCollector vma-0000-0000-000a
vm vma-vm-00000000061
vm vma-vm-00000000062
.
.
.vm vma-vm-00000000066
path /vcenter/farm/myhost.mydomain.com
```

# The Basic Data Synchronization Loop

In Developing Client Applications on page 45, we developed a Java example to perform the GetContents operation on the `/vm` view. (See Creating a Simple Client on page 52.) The operation returns a data structure containing the list of the available virtual machines. This list contains handles that refer to the XML documents corresponding to each virtual machine.

We'll start with building the data synchronization loop for a single virtual machine; for example, `/vm/564d08e5-6253-7e43-7740-774b3dc0cfd2`.

For now, let us start with a simple first version that simply calls GetContents repeatedly:

```
#
# Setup & call the ResolvePath method
#

my $method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

my @params = (SOAP::Data->name(path => '/vm/564d08e5-6253-7e43-7740-774b3dc0cfd2'));

my $handle = $service->call($method => @params)->result;


#
# Setup and call the GetContents method repeatedly
#

$method = SOAP::Data->name('GetContents')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(handle => $handle));

while (1)  {

        $result = $service->call($method => @params);
        PrintContents($result);
        sleep(10);
}
```

This code polls the Web service server every 10 seconds to obtain a fresh copy of the data corresponding to a specific virtual machine. The preceding code may work for a simple application, but it will not scale to a system that is managing a large number of virtual machines and other entities, due to the large amount of network traffic it generates.

# Versions and Handles

In this section, we provide a brief description of versions, handles, and vHandles. For complete information on handles, version numbers, paths, and the GetContents and GetUpdates operations, see Understanding VMware SDK Terminology on page 26.

A handle is a temporary token, used by a Web service client, to invoke Web service operations that require a reference to an object. An object handle is somewhat analogous to a file handle (descriptor) returned by a operating system similar to UNIX, when a file is opened using the "open" system call. Like a file handle, an object handle is a temporary handle that always refers to the same object.

A vHandle is a versioned handle that acts as a reference to the specific memory state of an object at a certain point in time. That is, a vHandle is an object handle that has a version number associated with it. The version number determines the specific memory state. Each version identifier corresponds to a different point in time.

The path resembles a full path name of a file, which identifies an object in the Web service hierarchy. Its value is an XML document in the Web services world. This value is different in other worlds.

A client executes the GetContents operation, passing in a handle (to an object) as a parameter and retrieving an XML document that describes the object identified by the handle. The XML document is the value of the object and is associated with a handle and a vHandle that are returned by the GetContents operation.

A client executes the GetUpdates operation, passing in a vHandle as a parameter and retrieving the change(s) in the object identified by the vHandle. Only the change(s), or the "delta" is returned as a `diff` of the current XML document that describes the object as currently maintained by the Web service, compared with the original XML document as identified by the vHandle that was passed to the GetUpdates operation. The GetUpdates operation also returns an updated vHandle, identifying the latest version of the object that was returned to the client.

For additional information on handles, version numbers, paths, and the GetContents and GetUpdates operations, refer to the *Virtual Infrastructure SDK Reference Guide*.

# Calling the GetUpdates Operation

In order to keep data on the client side up-to-date with data on the server side, you can combine the use of the GetContents operation with GetUpdates.

- The GetContents operation provides all the information for a particular object at the time of the request.
- The GetUpdates operation returns incremental changes to the data, that can be patched onto the existing data on the client. Theoretically, one can call GetContents repeatedly, but this causes excessive network traffic.

For example, you have issued a GetContents request on a specific virtual machine. The response is a list of the information specified by the Virtual Machine data model. Someone, with Interact rights on this virtual machine, changes the memory shares. If your client then issues a GetUpdates request, you can see the updated memory shares and any other changes that have occurred since the original GetContents request.

The first refinement is to replace the repeated calls to GetContents with calls to GetUpdates. By using GetUpdates instead, the server only returns the changes within the VirtualMachine data structure, and not the entire new version of the VirtualMachine object.

```
#
# Setup & call the ResolvePath method
#

my $method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

my @params = (SOAP::Data->name(path => '/vm/564d08e5-6253-7e43-7740-774b3dc0cfd2'));

my $handle = $service->call($method => @params)->result;

#
# Setup and call the GetContents method to get the vHandle
#

$method = SOAP::Data->name('GetContents')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(handle => $handle));

#
# OK to call result() here since we only need the vHandle of the object
#
```

```
my $result = $service->call($method => @params)->result;
my $vHandle = $result->{ vHandle };

#
# Setup the call to the GetUpdates method
#

$method = SOAP::Data->name('GetUpdates')
       ->attr({xmlns => 'urn:vma1'});

while (1)  {

        #
        # Update the vHandle list
        #

        my $vHandleList->{ 'vHandle' } = $vHandle;

        @params = (
                    SOAP::Data->name(vHandleList => $vHandleList),
                    SOAP::Data->type('xsd:boolean')->name(wait => 'true')
                   );

        #
        # Call GetUpdates
        #

        $result = $service->call($method => @params);
        PrintContents($result);

        #
        # Obtain new vHandle
        #

        $vHandle = $result->result>{ 'update' }->{ 'vHandle' };
}
```

Note the following comments regarding the preceding code:

- Exceptions are handled by a top-level fault handler, described later.
- We are not processing any updates; this is covered in the next section.
- The call to GetUpdates blocks indefinitely, until there is some change that causes the server to send back a response. In some circumstances, particularly in the presence of intermediaries (such as proxies), the client and server can get disconnected with the client

remaining blocked on the GetUpdates call. A lower layer disconnection mechanism, such as TCP timeout, is required to handle this case.

The timeout period can also be set in the SOAP::Lite object by using the `proxy()` method at the time of object creation as shown in the following code sample:

```
my $service =
  SOAP::Lite
    -> service($vmaWSDL)
    -> proxy($webserviceURL,
         cookie_jar => HTTP::Cookies->new(ignore_discard => 1), timeout => 0)
    -> deserializer(SdkDeserializer->new)
    -> on_fault(sub{FaultHandler(@_);});
```

A timeout value of `0` (zero) implies no timeout from the client side. Therefore, a client that is blocked on a GetUpdates call will continue to wait indefinitely until the call returns. Alternatively, this value can be set to a positive integer to indicate a timeout period, in seconds.

- When a client calls GetContents, it receives the value of the object identified by the handle, along with a vHandle. Then, when the client makes calls to GetUpdates, it sends the vHandle to the Web service. The Web service returns a set of changes made to the object identified by the vHandle, and an updated vHandle, identifying the latest version of the object. See Versions and Handles on page 151.

  In this example, we are working with one particular view. Therefore, the parameter to GetUpdates is a list containing a single vHandle object. However, clients can also send a list of multiple vHandles to GetUpdates.

- GetUpdates returns a list of Update objects, with one object per vHandle.

  In this case, the return value of GetUpdates can either contain an empty list (if no changes occurred at the time of return) or a list with a single Update object. The list cannot contain more than one Update object because the parameter to GetUpdates contained only a single vHandle.

# Applying Changes to the Client Data

Each change is presented as an object of type Change. Change objects are quite complex, as there are many kinds of changes and we attempt to provide changes in increments as small as possible. We first present a high-level overview of the Change object, and then cover the various kinds of changes on a case-by-case basis. If state is being maintained on the client side, then these changes can be applied to the client-side objects on an incremental basis.

## The Change Object

Each field of the Change object is described briefly in this section.

### op Field

The primary field of the Change object is `op`, obtained by calling `$change->{Op}`, where `$change` represents a Change object. This field describes the kind of change.

The kind of change is one of the following:

- `ins` — Describes the insertion of new data at the location specified by the target field.
- `del` — Describes the deletion of existing data at the location specified by the target field.
- `repl` — Describes the replacement of existing data at the location specified by the target field. You can consider a `repl` operation as a combination of a `del` operation, followed by an `ins` operation.
- `edit` — Describes a change to existing data. Portions of the data can be changed, while leaving other portions unchanged. This edit change applies only to primitives and objects of type xsd:string and xsd:dateTime.
- `move` — Describes a change where the Web service indicates a move of an object from one (source) Container to a (destination) Container. This is similar to the cut and paste operation. In the cut and paste operation, two move changes are received for the object. The parent of the object that is cut receives one move change and the recipient of the paste operation receives the second move change.

### target Field

The `target` field is obtained by the reference to `$change->{target}`, where `$change` represents a Change object. It describes the location where the change applies within the View object The target is specified as a list of identifiers separated by slashes, that indicate how to navigate within the data structure to reach the desired field. For more information on the data structure, see the data models in the *Virtual Infrastructure SDK Reference Guide*.

There are two broad categories of targets that clients can use to communicate changes by using the Change datatype: leaf values and composites.

- Leaf value — An example of a leaf value is the value "My Great Virtual Machine" for the "name" field of the VirtualMachineInfo datatype. Leaf values correspond to text nodes within an XML document. This includes xsd:string, xsd:int, xsd:long and other similar primitive datatypes.
- Composites — A composite or aggregate is an entire object such as an object of type Item, Host, or VirtualMachine, or an interior node of such an object such as the VirtualNetworkAdapter inside the VirtualMachine datatype.

For example, if the target is `hardware/memory/sizeMb` (within a VirtualMachine object) and the virtual machine contents have been stored in `$vmContents`, then the corresponding field within the client data structure is obtained by calling:

```
$vmContents->{body}->{hardware}->{memory}->{sizeMb}
```

Alternatively, client-side classes and objects can be defined to maintain state information. If these objects are being maintained, each Change object must be processed and the change that is defined, must be applied to the client object.

Although we do not demonstrate the use of these objects in the sample applications, the next section discusses how these changes can be applied to objects.

**val Field**

The Change object field `val`, obtained by the reference to `$change->{Val}`, specifies the new leaf value or composite of the field specified in the target field. In some cases (such as `del` changes), this field is not relevant and you can ignore it.

**inserted, deleted and editPos Fields**

There are three more fields in the Change object:

- `inserted` — For `ins` and `repl` changes, this field specifies the number of characters that are inserted. For `move` changes, this identifies the paste operation with `inserted = 1.`
- `deleted` — For `repl` and `del` changes, this field specifies the number of characters that are deleted. For `move` changes, this identifies the cut operation with `deleted = 1`.
- `editPos` — This field is valid only for `edit` changes of string values. It specifies the location in the string where the edit starts.

## Processing the Various Kinds of Change

The following sections explain the various kinds of changes and how to process them:


- Insert Change on page 157
- Delete Change on page 158
- Replace Change on page 160
- Edit Change on page 160

• Move Change on page 161

**Insert, Delete, or Replace (ins, del, or repl) Change Operations**

Clients can send these change operations for composites, for both arrays and non-arrays. The usage of Change fields for `ins`, `del`, or `repl` is:

- `target` — Composite that is being inserted, deleted, or replaced. An example is `hardware/net/adapter["#_nic0"]` where the network adapter with key `"#_nic0"` is either being inserted, deleted, or replaced.
- `editPos` — Not Applicable.
- `deleted` — Number of composite nodes being deleted.
- `inserted` — Number of composite nodes being inserted.
- `val` — Contains the new composite or set of composites being inserted or replaced. When the change operation is delete, this field is NULL.

As an example, a single replace change can replace one `#_nic0` adapter with two adapters `#_nic1` and `#_nic2`. In this case, the changeOp is `repl`, the `target` is `hardware/net/adapter["#_nic0"]`, `deleted` is `1`, `inserted` is `2`, and `val` contains an array of the two new adapters `#_nic1` and `#_nic2`. Note that the keys are assigned by the Web service.

- When the array is an indexed array, the target contains the index of the array element before or at the location where the operation occurs. For example, if `target` is `vm[1]`, `deleted` is `0`, `inserted` is `2`, and `val` contains two new `vm` array elements, then the new array elements are inserted before the existing `vm[1]`. The first of these new elements becomes `vm[1]`, the second new element becomes `vm[2]` and the old `vm[1]` is now `vm[3]`. In this same example, if `deleted` is also set to `1`, then the old `vm[1]` gets deleted and the two new elements are inserted in its place as `vm[1]` and `vm[2]`.

**Insert Change**

The insert change may be a change within an object, or it can be an insertion of a new top-level object. For example, if the client is querying for updates on an top-level Container such as `/vm`, then the client gets notified about new virtual machines. Similarly, querying for updates on `/vcenter`, the Web service can notify the client of new Farms, Farm groups, virtual machines, virtual machine groups, hosts, and so on.

The following sample code illustrates how to process these changes.

```
# Call to GetUpdates
my $result = $service->call($method => @params);

# Extract the list of changes
@changes = $somObj->valueof('//change');
```

```
# Process each change
foreach $change ( @changes ) {

        $op = $change->{ op };
        $target = $change->{ target };
        $val = $change->{ val };

        # insert operation
        if ( $op eq "ins" ) {
                 if ( $target =~ /^item/ ) {
                         ...insert new top level object ....
                 } else {
                         ...insert into existing object ...
                 }
        }
}
```

For top-level object insertions, the Web service sends the handle of the new object. The client must call getContents on this handle to obtain the contents of this new object.

```
if ( $op eq "ins" ) {
      if ( $target =~ /^item/ ) {

         # Extract the handle of the new object

         @handle_split = split ( /\"/, $target );
         $new_handle = $handle_split[1];

         # Call GetContents on the new object

         $method = SOAP::Data->name('GetContents')
               ->attr({xmlns => 'urn:vma1'});
         @params = ( SOAP::Data->name(handle => $new_handle));
         $result = $service->call($method => @params)->result;

         ....obtain vHandle of new object and add it to list of current vHandles
      }
}
```

### Delete Change

A delete change is analogous to the insert change described previously. Delete changes can either be deletions of top-level objects or deletions within existing objects.

```
# Call to GetUpdates
my $result = $service->call($method => @params);

# Extract the list of changes
@changes = $somObj->valueof('//change');

# Process each change
foreach $change ( @changes ) {

        $op = $change->{ op };
        $target = $change->{ target };
        $val = $change->{ val };

        # insert operation
         if ( $op eq "del" ) {
               if ( $target =~ /^item/ ) {
                     ...delete existing object
               } else {
                     ...delete from existing object ...
               }
         }
}
```

For top-level object deletions, the Web service sends the handle of the deleted object. The client must remove the vHandle of this object from the list of currently watched vHandles for the GetUpdates operation.

```
if ( $op eq "del" ) {
         if ( $target =~ /^item/ ) {

              # Extract the handle of the deleted object

              @handle_split = split ( /\"/, $target );
              $del_handle = $handle_split[1];

              # Delete the vHandle from the current list
              for(my $counter = 0; $counter <= $#vHandleList; $counter++ ) {

                   if ( $vHandleList[$counter] =~ /$del_handle/ ) {
                        splice ( @vHandleList, $counter, 1 );
                        last;
                   }
              }
         }
}
```

### Replace Change

The replace change is a combination of the insert and delete changes and is processed as such by the Web service.

```
sub processReplace
    {
        my ( $service, $change ) = @_;
        processDelete( $service, $change);
        processInsert( $service, $change);
    }
```

### Edit Change

The client may send an edit change operation only when changing leaf values. Similarly the Web service will send the edit change operation only when changing leaf values. We can further categorize leaf values as string values or non-string values.

***String Values —*** These are values of type xsd:string. The following fields of Change are relevant when string values are edited. The usage of these fields is identical for the GetUpdates and PutUpdates operations.

- `target` — Interior node whose value is being edited. An example is `Item["xxx"]/name` when the value for the interior 'name' node is being edited for an Item whose handle is `"xxx"`.
- `editPos` — For deletions, this specifies the starting position of the substring being deleted. For insertions, this is the character position before which a substring is inserted. If this field is NULL, it must be interpreted as 0 (zero).
- `deleted` — Number of characters being deleted starting at editPos. If this field is NULL, it must be interpreted as 0 (zero).
- `inserted` — Number of characters being inserted before the editPos character. If this field is NULL, it must be interpreted as 0 (zero).
- `val` — this contains the substring to be inserted if the 'inserted' field is non-zero.

If an entire string is being replaced by using the edit operation, then `editPos` is `0` (zero), `deleted` is the length of the old string, `inserted` is the length of the new string, and `val` contains the new string. This is a common case where the edit operation is used to replace an entire string.

***Non-string values —*** These are leaf values not of type xsd:string. The edit operation for such values always replaces the existing value with a new value. The client can ignore the `editPos`, `inserted`, and `deleted` fields of Change. Similarly, when doing a PutUpdates operation, the

Web service ignores these fields for non-string values. The usage of Change fields for edit for non-string values is:

- `target` — Interior node whose value is being edited. An example is `hardware/cpu/controls/shares` where the value for the shares node is being replaced for a virtual machine.
- `editPos` — Not Applicable.
- `deleted` — Not Applicable.
- `inserted` — Not Applicable.
- `val` — Contains the new value that replaces the old value on the target.

The following code fragment illustrates how to apply an edit change to the name of an object.

```
if ( $op eq "edit" && $target =~ /$name/ ){

      #
      # Renaming of a vcenter object - update hash
      #

      @handle_split = split ( /\"/, $target );
      $handle = $handle_split[1];

      # $handleList is a hash maintaining a handle=>name mapping for all objects
      if ( ! ref $val ){
            $handleList->{ $handle } = $val;
      }
}
```

**Move Change**

The Web service uses this change operation to indicate a move of an object from one Container to another. This operation can never be used from a client to the Web service in a PutUpdates call. To move objects, clients must instead use the Rename operation.

The cut operation is identified as a move operation with `$change->{deleted} = 1`, and the corresponding paste operation is received from the Web service along with the move change and `$change->{inserted} = 1`.

The value in the change object hasn't changed as part of the move operation and is therefore considered uninteresting from the move operation's perspective.

```
if ( $op eq "move" ) {
               if ( $target =~ /^item/ ) {
```

```
                    $numInserted = $change->{inserted};
                    $numDeleted = $change->{deleted};

                    # Extract the handle of the moved object


                    @handle_split = split ( /\"/, $target );
                    $moved_handle = $handle_split[1];

                    if ( $numInserted == 0 && $numDeleted == 1 ) {
                    // Cut operation
                    ...adjust the client data structure
                    } elsif ( $numInserted == 1 && $numDeleted == 0 ) {
                              // Paste operation
                    ...adjust the client data structure
                    }

                }
}
```

# Indexed and Key-based Arrays

Some elements in the VMware VirtualCenter Web Service data models (described in the *Virtual Infrastructure SDK Reference Guide*) can occur multiple times (for example, multiple NICs). In these cases, these fields have the `minoccurs` attribute set to 0 and the `maxoccurs` attribute set to `unbounded`. When the `maxoccurs` attribute is not `1`, the field is treated as an array, provided that it is accessed directly by using the `valueof()` or `dataof()` methods from the envelope object. The Web service data structures have two categories of arrays: indexed arrays and key-based arrays. Indexed arrays are only used for arrays of basic types; for example, `/host/vm[]` and `host/info/datastore[]`. All other arrays are key-based arrays.

Indexed arrays are accessed by an index (the usual manner of accessing arrays). Key-based arrays are accessed by keys that are strings. The component type of the array must have a string field called `key`, in order to be a key-based array. The value of this field is unique across all the components of an array, and is the key of the array component. See Key-based Arrays on page 163.

## Indexed Arrays

Some examples of indexed arrays are:

```
host/vm[1]
```

The target in the Change object for indexed arrays always ends with the name of the array field, followed by the index where the change is to occur (`...arrayName[index]`).

The kind of change is one of the following:

- `ins` — The `inserted` field specifies the number of array elements that are being inserted at the location specified by the target. You must ignore the `editPos` and `deleted` fields. The `val` field is an array component if `inserted` is 1, otherwise the `val` field is an array with `inserted` elements.
- `del` — The `deleted` field specifies the number of array elements that are to be deleted starting at the location specified by the target. You must ignore the `editPos`, `inserted`, and `val` fields.
- `repl` — The `repl` field is a combination of a `del` operation followed by an `ins` operation. You must ignore the `editPos` field. All the other fields have the same meaning as previously specified.

## Key-based Arrays

Key-based arrays are very similar to indexed arrays except the key is a string, and not an index. Some examples of key-based arrays are:

```
hardware/net/adapter["_nic001"]
```

The only difference between key-based arrays and indexed arrays in the Change objects is that each array component must be deleted individually, because there is no ordering concept for key-based array components. Therefore, you must ignore the `deleted` field in the Change object.

You can insert multiple components into a key-based array at one time. Array keys are generated by the Web service, and are not specifiable by the programmer when the component is inserted for the first time. Each component in a single array will have a unique key.

The following code fragment illustrates how to create a new key-based array element; in this case, VirtualNetworkAdapter.

```
// create the adapter structure
my %val =  SOAP::Data ->type('VirtualNetworkAdapter');
tie %$val, "Tie::IxHash";
$val->{ 'mode' } = 'monitor';
$val->{ 'network' } = 'Internal Network';

// setup the change to do a PutUpdates
$changeRef->{ 'req' }->{ 'handle' } = $handle;
$changeRef->{ 'req' }->{ 'change' }->{ 'op' } = 'ins';
// Note : do not specify array key.
// this is generated, and not user setable
$changeRef->{ 'req' }->{ 'change' }
          ->{ 'target' } = 'hardware/net/adapter';
$changeRef->{ 'req' }->{ 'change' }->{ 'deleted' } = 0;
$changeRef->{ 'req' }->{ 'change' }->{ 'inserted' } = 1;

$changeRef->{ 'req' }->{ 'change' }->{ 'val' } = $val;
```

## Determining the Array Category

The following sample code illustrates how to determine if the Change object is referring to an indexed array or a key-based array.

```
#
# Extract the list of changes from a GetUpdates response
#
@changes = $result->valueof('//change');

foreach my $change ( @changes ) {

        my $target =  $change->{ target } ;

        #
        #Extract the index portion of the target
```

```
        #

        my @components = split ( /\[/ , $target );
        my @indices = split ( /\]/, $components[1] );

        # Check if index is numeric

        if ( $indices[0] =~ /\d/ ) {
                ... Indexed Array
        } else {
                ...Key-based Array
        }
}
```

# Calling the PutUpdates Operation

The client uses the PutUpdates operation to perform updates to the Web service not performed by the other API operations in the VMware SDK. The argument to PutUpdates is a ChangeReqList that contains an array of ChangeReqs. Each ChangeReq has a handle, and a list of Change objects describing various changes that are being applied to the object being identified by the handle. The format of the Change object is exactly as it is in the GetUpdates operation.

The purpose of the PutUpdates operation is to make the changes, specified in the Change objects, to the Web service. There are two forms of PutUpdates: "last one wins" and "versioned".

- Last one wins (unversioned PutUpdates call) — Changes are applied to the Web service object referred to by the handle. The order in which the changes from multiple clients are applied is unspecified and it is possible for another client's changes to override this client's changes and vice versa.
- Versioned — Rather than providing a handle, clients give a versioned handle, or vHandle, to the PutUpdates operation. In this case, the operation is performed only if the Web service's version of the object matches that of the client. Otherwise a version mismatch occurs and the call fails.

The value returned by the PutUpdates operation is exactly the same as the value returned by the GetUpdates operation for the changes on the vHandles passed to PutUpdates. It is as if the updates have been made and the client has called a GetUpdates operation with the same set of vHandles. If an unversioned PutUpdates call is made (on handles), then no updates are returned in the response.

**Note:** It is possible to obtain a change list from the PutUpdates operation that is greater than the your list of changes, if another client is also making changes to the same object at the same time.

## Using the PutUpdates Operation to Update the Memory Setting for a Virtual Machine

The following code fragment illustrates the use of PutUpdates. It shows how a client can update the memory setting for a virtual machine. Because the sizeMb field is an integer, we use an edit operation in the change object.

1. By using the ResolvePath operation, obtain the handle for the virtual machine to which the change is being applied.

```
$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(path => $path ));
```

```
$handle = $service->call($method => @params)->result;
```

2. Create the change object.

```
my %changeRef = ();

#
# Use Tie::IxHash package to ensure ordering in the hash
#

tie %{$changeRef->{ 'req' }}, "Tie::IxHash";
tie %{$changeRef->{ 'req' }->{ 'change' }}, "Tie::IxHash";

#
# Construct change object to edit memory
#

$changeRef->{ 'req' }->{ 'handle' } = $handle;
$changeRef->{ 'req' }->{ 'change' }->{ 'op' } = 'edit';
$changeRef->{ 'req' }->{ 'change' }
          ->{ 'target' } = 'hardware/memory/sizeMb';
$changeRef->{ 'req' }->{ 'change' }->{ 'val' } = $newMemorySize;
```

3. Call the PutUpdates operation with this change.

```
my $method = SOAP::Data->name('PutUpdates')
     ->attr({xmlns => 'urn:vma1'});

my @params = (SOAP::Data->name(req => $changeRef));

my $result = $service->call($method => @params)->result;
```

## Using the PutUpdates Operation to Make Changes to Array Elements

To insert elements into an array, the client should not specify the index or key (keyed arrays) in the change target. For example, to insert a new network adapter into a virtual machine, the change target is `hardware/net/adapter`, without any square brackets.

To delete an array item, specify the index of the array element. Alternately, if the client is updating a keyed array, then specify the key for the element (that will be deleted) in the change target. Clients can use standard reflection APIs to look inside the object and retrieve all the keys of an array in order to determine which element to delete.

For example, to delete a network adapter with key `nic001`, the change object is:

```
#
# Construct change object to delete a network adapter
#

%changeRef = ();

# The Tie::IxHash module is used to ensure ordering in a hash
tie %$changeRef, "Tie::IxHash";
$changeRef->{ 'req' }->{ 'handle' } = $handle;
$changeRef->{ 'req' }->{ 'change' }->{ 'op' } = 'del';
$changeRef->{ 'req' }->{ 'change' }->{ 'target' }
= "hardware/net/adapter[\"".$adapter->{ key }."\"]";
$changeRef->{ 'req' }->{ 'change' }->{ 'deleted' } = 1;
$changeRef->{ 'req' }->{ 'change' }->{ 'inserted' } = 0;

#
# Call PutUpdates
#

$method = SOAP::Data->name('PutUpdates')
     ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(req => $changeRef));
$service->call($method => @params);
print "\nOperation Successful \n";
```

Read `/SDK/SDK-README.html`. It has a link to the complete code listing for the PutUpdates operation.

# Running the Sample Code

The sample code discussed in this document is included with this distribution. Read `/SDK/SDK-README.html` for a link to the instructions on how to run the sample code. See Client Development Environments on page 267 for complete information on how to set up your client development environment.

# Fault Handling in SOAP::Lite

Transport errors and SOAP::Faults returned by the Web service can be handled either at the top-level SOAP::Lite object or individually for each SOAP response. The following method describes a generic fault handler that traps both kinds of faults and displays an error message.

```
######################################################################
#
# FaultHandler --
#
#      Subroutine that handles SOAP faults as well as transport
#      errors.
#
# Results:
#      Displays fault information and terminates the program.
#
# Side effects:
#       None.
#
######################################################################


sub FaultHandler
{
    my($transportFault, $soapFault) = @_;

    die ref $soapFault ?
        "\nFault: ".$soapFault->faultdetail->{'FaultInfo'}->{'kind'}."\n".
            $soapFault->faultdetail->{'FaultInfo'}->{'info'} :
              "\nFault: ".$transportFault->transport->status, "\n";
}
```

By linking the preceding method to the SOAP::Lite object, it is automatically called when a fault occurs:

```
my $service =
      SOAP::Lite
         -> service($vmaWSDL)
         -> proxy($webserviceURL,
             cookie_jar => HTTP::Cookies->new(ignore_discard => 1), timeout => 0)
         -> deserializer(SdkDeserializer->new)
         -> on_fault( sub
                        {
                                 FaultHandler(@_);
                        }
                        );
```

# Testing

To test your client applications, complete the following.

1. Be sure that SOAP::Lite and other required modules have been installed properly.
2. If VirtualCenter is not running, then start this application.
3. Start the Web service.
4. Run your test programs.
    a. Make changes in the VirtualCenter application and see if your client application responds appropriately.
    b. Make changes using your client application and see if VirtualCenter reflects your changes.

**Note:** You can also test the sample code that is provided with this distribution in a similar manner.

For more information on testing your client applications, see Troubleshooting on page 275.

# Complete Code Listing

You can find the complete code for this simple application in your VMware SDK package.

CHAPTER 8

# Advanced Client Concepts for Perl Programmers

This chapter provides examples of client applications that you can create, to perform the following tasks. For complete information on the syntax for these client applications, refer to the *Virtual Infrastructure SDK Reference Guide*.

**Note:** The examples in this chapter are written in Perl using SOAP::LITE. If you are interested in Java samples, see Advanced Client Concepts for Java Programmers on page 87. All the code samples in this chapter, are located in their entirety in `\SDK\WebService\samples\perl\sampleapp`.

**Note:** You may use a different developer environment and language to build your client program. Adjust the examples accordingly for your developer environment.

# Virtual Machine Power Operations

Client applications can perform the following power operations:

- Power on (start) or resume a virtual machine
- Power off (stop) or suspend a virtual machine
- Reset a virtual machine

**Note:** For all these applications, the client must have Interact rights on the Server Farm, Farm group, Farm, and host that contains the virtual machine. If the client has only Browse rights, then the client can view information about these objects (GetContents and GetUpdates requests), but cannot issue any power operations.

If the client is listening for updates on this virtual machine and the operation succeeds, then the client will see updates to the VirtualMachine state.

In addition, a task is initiated on the Web service for each of these operations. The XML document describing that task is returned to the client. The client can monitor the progress of this task and check for errors, status, and so on. For sample code on how to monitor the progress of a task, see Task Scheduling and Monitoring on page 214.

## Starting or Resuming a Virtual Machine

The StartVM operation initiates the process of starting a virtual machine or resuming a suspended a virtual machine. If you have already configured a script or any other application to run during the power-on (resume) operation, the script (or other application) will run.

It takes one argument, the handle of the virtual machine that is to be started. The return value is the handle to the task created to start this virtual machine. If the virtual machine that is to be started is currently suspended, then this virtual machine is resumed.

The following code fragment illustrates how clients call the StartVM operation, then how the client can monitor the resulting task to determine when the operation completes.

```
#
# Setup & call the ResolvePath method to obtain a handle for the VM
#

my $method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

my @params = (
               SOAP::Data->name(path => '/vm/564d5a05-29a7-b09b-d576-9cb8a719d940')
              );
```

```
my $handle = $service->call($method => @params)->result;

#
# Call to StartVM
#

$method = SOAP::Data->name('StartVM')
            ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(vm => $handle));


my $task = $service->call($method => @params)->result;


#
# Task handle returned - monitor the returned task to check
# progress
#

MonitorTask ($service, $task);
```

## Stopping or Suspending a Virtual Machine

The StopVM operation initiates the process of stopping or suspending a powered-on virtual machine.

If you have already configured a script or any other application to run during the power-off (suspend) operation, the script (or other application) will run. It takes three arguments: the handle of the virtual machine that is to be stopped, whether the virtual machine should be suspended or powered off, and whether or not stopping the virtual machine is a "soft "power off operation. The return value is the handle to the task that represents the task created to power off this virtual machine.

The following code fragment illustrates how clients call the StopVM operation, then how the client can monitor the resulting task to determine when the operation completes.

```
if ($op eq 'stop'){

    $method = SOAP::Data->name('StopVM')
      ->attr({xmlns => 'urn:vma1'});
    @params = (
               SOAP::Data->name(vm => $handle),
               SOAP::Data->name(suspend => 0),
               SOAP::Data->name(soft => 1)
              );
```

```
}elsif ($op eq 'suspend'){

    $method = SOAP::Data->name('StopVM')
      ->attr({xmlns => 'urn:vma1'});

    @params = (
               SOAP::Data->name(vm => $handle),
               SOAP::Data->name(suspend => 1),
               SOAP::Data->name(soft => 1)
              );
}

my $task = $service->call($method => @params)->result;

#
# Task handle returned - monitor the returned task to check
# progress
#

MonitorTask ($service, $task);
```

## Boolean Flags in the VirtualMachineTools Datatype

The VirtualMachineTools datatype includes four Boolean flags that clients may use to determine whether or not scripts execute in the guest operating system when a virtual machine's power state changes through the StartVM or StopVM operations.

- afterPowerOn — Flag determines whether or not scripts should run after the virtual machine is powered on. If this boolean is set to true, then custom startup scripts (if there are any) run on the guest operating system after the virtual machine powers on.
- afterResume — Flag determines whether or not scripts should run after the virtual machine is resumed. If this boolean is set to true, then custom startup scripts (if there are any) run on the guest operating system after the virtual machine resumes.
- beforeSuspend — Flag determines whether or not scripts should run before the virtual machine is suspended. If this boolean is set to true, then custom startup scripts (if there are any) run on the guest operating system before the virtual machine is suspended, regardless of whether the soft flag is specified during the StopVM operation.
- beforePowerOff — Flag determines whether or not scripts should run before the virtual machine is powered off. If this boolean is set to true, then custom startup scripts (if there are any) run on the guest operating system before the virtual machine powers off, regardless of whether the soft flag is specified during the StopVM operation.

**Note:** If one of these Boolean flags is set, then the scripts will run, regardless of the soft flag setting in the StopVM operation.

## Resetting a Virtual Machine

The ResetVM operation initiates the process of resetting a virtual machine, which also resets the virtual hardware. (A reset operation is equivalent to pushing the Reset button on a physical machine.)

The ResetVM operation first attempts to shut down the guest operating system before resetting the virtual machine. This is similar to selecting Restart on a Windows operating system, where Windows gracefully shuts down, then starts up again. If you have already configured a script or any other application to run during the reset operation (during the shutdown or startup of the guest operating system), the script (or other application) will run.

If this attempt fails, then the ResetVM operation looks into the virtual machine's configuration file. By default, a virtual machine's configuration file setting for a reset is a "hard" reset where the virtual machine immediately powers off, regardless of what is occurring in the guest operating system. This is similar to pressing and holding the power button on a physical machine until it powers off, then restarting the physical machine.

The ResetVM operation takes one argument, the handle of the virtual machine that is to be reset. The return value is the handle to the task that represents the task created to reset this virtual machine.

The following code fragment illustrates how clients call the ResetVM operation, then how the client can monitor the resulting task to determine when the operation completes.

```
$method = SOAP::Data->name('ResetVM')
            ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(vm => $handle));


my $task = $service->call($method => @params)->result;


#
# Task handle returned - monitor the returned task to check
# progress
#

MonitorTask ($service, $task);
```

# Host Operations

Clients can enable (connect) or disable (disconnect) hosts. Clients can also shut down a host provided the host is ESX Server 2.1 or higher. The client must have Configure rights on the Farm that contains the host, in order for the operations to succeed. These operations return an empty response.

## Enabling a Host

Clients may enable a host in the Disabled state by using the EnableHost operation. When a host is "created", and a user name and password are supplied during the host creation, then the host is automatically enabled for virtual machine operations. In the Enabled state, clients can perform virtual machine operations and discover new virtual machines. The EnableHost operation takes one mandatory argument, the handle to the host that will be enabled. There are two optional arguments: the user name and password that VirtualCenter uses to connect to the host specified by the handle.

Upon success, an empty response message is returned.

```
#
# Obtain host handle
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(path => '/host/myhost.mydomain.com'));

$handle = $service->call($method => @params)->result;

#
# Setup and call EnableHost
#

#optional parameters
my $hostUserName = 'yourusername';
my $hostPassword = 'yourpassword';

$method = SOAP::Data->name('EnableHost')
      ->attr({xmlns => 'urn:vma1'});

@params =  (
             SOAP::Data->type('xsd:string')->name(host => $handle),
             SOAP::Data->type('xsd:string')->name(userName => $hostUserName),
             SOAP::Data->type('xsd:string')->name(password => $hostPassword)
```

```
            );

$service->call($method => @params);
```

## Disabling a Host

Clients may disable the host from virtual machine operations by using the DisableHost operation. By using the Create operation without supplying the user name and password, clients may add a host to the Web service inventory in the Disabled state. When a host is disabled, clients are unable to perform any virtual machine operations.

The DisableHost operation takes one argument, the handle to the host that will be disabled. Upon success, an empty response message is returned.

**Note:** The DisableHost operation does not remove the host name from the Farm. Clients must remove the host by using the Delete operation. These two operations are separate, so that disabled hosts can continue to be managed without having to remove them.

```
#
# Disable Host
#

$method = SOAP::Data->name('DisableHost')
      ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(host => $handle));


$service->call($method => @params);
```

## Stopping or Restarting a Host

The StopHost operation permits the client to shut down or restart the host. It takes four arguments: the handle of the host that is to be stopped, whether the host should be gracefully shut down (soft) or immediately powered off (hard), whether or not to restart the host, and a reason string.

```
#
# Stop Host -- need input as to whether host should be shutdown
#              gracefully or restarted and optionally, the reason for
#              the shutdown.
#

$method = SOAP::Data->name('StopHost')
      ->attr({xmlns => 'urn:vma1'});
```

```
print "\nSoft shutdown ? (y/n): ";
my $soft = <STDIN>;
$soft = ( $soft =~ /y|Y/ )? 1 : 0;

print "\nRestart host ? (y/n): ";
my $restart = <STDIN>;
$restart = ( $restart =~ /y|Y/ )? 1 : 0;

print "\nReason for shutdown : ";
my $reason = <STDIN>;

@params = (
            SOAP::Data->type('xsd:string')->name(host => $handle),
            SOAP::Data->name(soft => $soft),
            SOAP::Data->name(restart => $restart),
            SOAP::Data->type('xsd:string')->name(reason => $reason)
           );

$service->call($method => @params);
```

# Creating and Deleting Objects

Clients can create or delete top-level objects on the Web service server by using the Create and Delete operations, respectively. The client must have Configure rights on the Container of the new object being created or deleted.

## Creating an Object

The Create operation takes three mandatory arguments: the handle of the Container for the new object, the name of the new object, and the type of the new object. There is also a fourth optional argument, that is an initial XML document providing additional information to create the object. Upon success, the handle to the newly created object is returned.

Clients may create the following objects with this Create operation:

- VirtualMachine
- Host
- Container, Farm, or VirtualMachineGroup
- TaskSchedule
- PerfCollector

The sample code in this section illustrates how to create new hosts, Farms, Containers and VirtualMachineGroups, but not virtual machines, tasks, or performance collectors. For more information on creating these objects see:

- Creating and Configuring a Virtual Machine on page 187 for information on creating a virtual machine.
- Task Scheduling and Monitoring on page 214 for information on how to create new scheduled tasks.
- Collecting Performance Data on page 224 for information on how to create new performance collectors.

If the client is creating a host and the parent Container is the `/host` handle, then the new host is placed in the default Farm in the `/vcenter` hierarchy, `/vcenter/Default Farm`. The initial value of the host object is specified by a HostSpec, as shown in the following sample.

```
#
# Call the appropriate subroutine depending on the object
# to be created
#

if ( $type =~ /^(Container|Farm|VirtualMachineGroup)$/ ) {
```

```
        CreateDomain( $service, $handle, $name, $type );

} elsif ( $type =~ /Host/ ) {

        CreateHost( $service, $handle, $name );

}


#########################################################################
#
# CreateHost --
#
#       This subroutine accepts user input to create the Host Spec
#       and creates a Host.
#
# Results:
#       Calls Create to create a Host after obtaining host
#       specifications from the user.
#
# Side effects:
#        None.
#
#########################################################################


sub CreateHost
{
    my ( $service, $handle, $name ) = @_;

    #
    # Create an ordered hash for storing host specifications
    #

    my %hostSpec = ();
    tie %$hostSpec, "Tie::IxHash";

    #
    # Accept user input for port number, username and password
    # for the host
    #

    print "\nEnter port number to connect to : ";
    chomp ( $hostSpec->{ 'port' } = <STDIN> );
    print "\nEnter username for host : ";
    chomp ( $hostSpec->{ 'userName' } = <STDIN> );
```

```
    print "\nEnter password for host : ";
    chomp ( $hostSpec->{ 'password' } = <STDIN> );

    #
    # Setup and call Create
    #

    my $method = SOAP::Data->name('Create')
      ->attr({xmlns => 'urn:vma1'});
    my @params = ( SOAP::Data->name(handle => $handle),
                   SOAP::Data->name(name => $name),
                   SOAP::Data->name(type => 'Host'),
                   SOAP::Data->type('HostSpec')->name(initial => $hostSpec)
                 );

    print "\nHandle for new object : ".$service->call($method => @params)->result;
    return;
}


########################################################################
#
# CreateDomain --
#
#      This subroutine creates a Container, Farm or Virtual
#      machine group.
#
# Results:
#      Calls Create to create a Container, Farm or Virtual Machine
#      Group.
#
# Side effects:
#       None.
#
########################################################################


sub CreateDomain
{
    #
    # Setup and call the Create method
    #

    my ( $service, $handle, $name, $type ) = @_;
    my $method = SOAP::Data->name('Create')
      ->attr({xmlns => 'urn:vma1'});
    my @params = ( SOAP::Data->name(handle => $handle),
```

```
                    SOAP::Data->name(name => $name),
                    SOAP::Data->name(type => $type)
                   );

    print "\nHandle for new object : ".$service->call($method => @params)->result;
    return;
}
```

## Deleting an Object

The following sample illustrates how to delete an object, such as a Container, Farm, VirtualMachineGroup, host, or a virtual machine. The Delete operation takes one argument, the handle of the object that will be deleted. When deleting a virtual machine, the Delete operation removes the virtual machine's configuration file and any other associated files, including the virtual disk file.

Container, Farm and VirtualMachineGroup objects are not required to be empty before they can be deleted.

**Note:** When a client deletes a top-level object (that contains other objects), then a delete change is only shown for the top-level object. For example, if a client deletes a Farm, then the client sees a delete change on the Farm, but not on any hosts in the Farm, or any virtual machines on the hosts.

Similarly, a virtual machine must be powered off (stopped) or suspended, and cannot be migrating, for the Delete operation to succeed. If a client wants to delete a virtual machine but not its virtual disk, then the client must do the following steps:

1. Stop the virtual machine
2. Using the PutUpdates operation, disconnect the virtual disk(s) of the virtual machine
3. Delete the virtual machine

The client must have Configure rights for both the object being deleted and the Container that has the object. Upon success, an empty response message is returned. The following sample illustrates how to delete an object.

```
#
# Setup & call the ResolvePath method to obtain handle of
# object to be deleted
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = ( SOAP::Data
            ->type( 'xsd:string' )
```

```
            ->name(path => '//vm/564d5a05-29a7-b09b-d576-9cb8a719d940')
        );

$handle = $service->call($method => @params)->result;


#
# Setup and call the Delete method.
#

$method = SOAP::Data->name('Delete')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(handle => $handle));

$service->call($method => @params);
```

# Creating and Configuring a Virtual Machine

In this section, we describe first how to create a virtual machine, then how to configure it by adding virtual disks.

## Creating a Virtual Machine

The following sample illustrates how to create a virtual machine. A virtual machine must always reside in a Farm or VirtualMachineGroup in the `/vcenter` hierarchy. If such an array is not identified for the virtual machine, then the operation places the newly created virtual machine in a special Farm `/vcenter/Default Farm`.

The client must have Configure rights on the Container where the virtual machine is being created. VirutalMachineSpec object specifies the initial value of a new virtual machine. The virtual machine's hardware configuration must be specified in this initial value.

The following sample illustrates how to initialize the VirutalMachineSpec object and invoke the Create operation. See the full code listing included with this distribution in the `/SDK/WebService/samples` directory for complete information on how to create a virtual machine's hardware.

```
#
# Create an ordered hash to store VM specs -- see full code
# listing for more details
#


my %vmSpec = ();
tie %$vmSpec, "Tie::IxHash";

$vmSpec->{ 'host' } = $host_handle;
$vmSpec->{ 'guestOS' } = $guestOS;
$vmSpec->{ 'hardware' } = GetHardwareDetails () ;



#
# Setup & call the ResolvePath method to
# obtain parent handle
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});
```

```
@params = ( SOAP::Data
                ->type('xsd:string')
                ->name(path => '/host/myhost.mydomain.com')
          );

$handle = $service->call($method => @params)->result;


#
# Setup and call the Create method
#

$method = SOAP::Data->name('Create')
        ->attr({xmlns => 'urn:vma1'});

@params = (
            SOAP::Data->name(handle => $handle),
            SOAP::Data->name(name => 'newVM'),
            SOAP::Data->name(type => 'VirtualMachine'),
            SOAP::Data->type('VirtualMachineSpec')->name(initial => $vmSpec)
          );

my $vm_handle = $service->call($method => @params)->result;
```

## Adding a Virtual Disk to a Virtual Machine

Now that we've created a virtual machine, we can add a disk to it. The CreateVirtualDisk operation requests the creation of a virtual disk for a virtual machine. It takes two arguments: the handle specifying the target virtual machine and the XML document (DiskInfo) specifying the properties of the new virtual disk.

**Note:** If a client attempts to create a virtual disk that already exists, the operation returns a successful task. It does not fail and does not return a fault. However, another virtual disk is not created.

The client must have Configure rights on the virtual machine where the disk is being created, in order for this operation to succeed.

Once the operation has been initiated, the Web service initiates a task for the CreateVirtualDisk operation. The XML document describing that task is returned to the client. The client can monitor the progress of this task and check for errors, status, and so on.

See Task Scheduling and Monitoring on page 214 for sample code on how to monitor the progress of a task.

```
#
# Construct ordered hash to store disk info --- refer to full code
# listing for the complete subroutine
#

my %diskInfo = ();

#
# Obtain guest OS supported disk info
#

$defaultDiskSize = $guestOS->{ 'defaultDiskSize' };
$defaultDiskSize = $defaultDiskSize/(1024 * 1024);

#
# Obtain and validate user input for disk size
#

print "\nEnter Disk Size [Default : ".$defaultDiskSize." MB] : ";
chomp ( $diskSize = <STDIN> );
if ( !$diskSize ){
     $diskSize = $defaultDiskSize;
}

#
# Construct Virtual Disk info specs
#

tie %$diskInfo, "Tie::IxHash";
$diskInfo->{ 'key' } = '';
$diskInfo->{ 'name' } = 'Sample Disk Creation';
$diskInfo->{ 'controllerType' } = $guestOS->{ 'defaultDiskControllerType' };
$diskInfo->{ 'controllerId' } = 0;
$diskInfo->{ 'deviceNumber' } = 1;
$diskInfo->{ 'adapterType' } = $guestOS->{ 'defaultSCSIAdapterType' };

tie %{$diskInfo->{ 'diskType'}}, "Tie::IxHash";
$diskInfo->{ 'diskType' }->{ 'diskKind' } = 'file';
$diskInfo->{ 'diskType' }->{ 'diskFileInfo' }->{ 'sizeMb' } = $diskSize;
$diskInfo->{ 'mode' } = 'persistent';

#
# Setup and call CreateVirtualDisk
#

    $method = SOAP::Data->name('CreateVirtualDisk')
        ->attr({xmlns => 'urn:vma1'});
```

```
@params = (
          SOAP::Data->name(vm => $handle),
          SOAP::Data->name(diskInfo => $diskInfo)
         );
$task = $service->call($method => @params)->result;

#
# Task returned -- monitor task
#

MonitorTask ( $service, $task );
```

# Responding to Virtual Machine Questions

A running virtual machine can generate a question that requires input from a user before the virtual machine can proceed. For example, if a virtual machine with an undoable disk is powered off, the virtual machine asks the user whether or not to discard the changes to the virtual disk.

The SDK provides a mechanism for users to handle this case programmatically. The question, generated by the virtual machine, appears in the virtual machine's state as a msgWaiting object. Clients will see this object when they are listening for updates on the virtual machine (through the GetUpdates operation), or if they call the GetContents operation on the virtual machine (which is blocked on the question).

The msgWaiting object has the following fields:

- msg — Question posed by the virtual machine.
- id — Internal server ID for the message.
- choice — List of possible answers, presented as an array of key-value pairs.
- defaultChoiceIndex — Default choice for this question, if there is one. This is the integer index of the default choice in the choices array. If this value does not exist, then the defaultChoice is 0 (zero).

The client can respond to this question by using the AnswerVM operation. The AnswerVM operation takes 3 arguments: the handle of the virtual machine that is blocked (and waiting for an answer), the choice (a key-value pair), and the ID of the message that requires a response.

The following sample code illustrates how to invoke the AnswerVM operation.

```
#
# Call GetContents to obtain initial state of the VM and get
# the vHandle for the VM. Call GetUpdates repeatedly to monitor
# the state of the VM
#

      $method = SOAP::Data->name('GetUpdates')
      ->attr({xmlns => 'urn:vma1'});

      while ( !$questionFound ){

      #
      # Setup and call GetUpdates
      #

      $vHandleList->{ 'vHandle' } = $vHandle;
```

```
@params =  (
            SOAP::Data->name(vHandleList => $vHandleList),
            SOAP::Data->type('xsd:boolean')->name(wait => 'true')
            );

$taskObj = $service->call($method => @params);

@changes = $taskObj->valueof('//change');

#
# Check the list of changes for a pending question
#

$counter = 0;
foreach $change (@changes){

    $target = $change->{ 'target' };
      if ( $target =~ /msgWaiting/ ) {
            $questionFound = 1;
            $questionObj = $change->{ 'val' };

            print "\nQuestion :\n".$questionObj->{ 'msg' }."\n";

            #
            # Extract and display the list of choices
            #

            $taskObj->match('//choice');
            @choices = $taskObj->valueof();

            for ( $i = 0; $i <= $#choices; $i++ ){
                  print "\n[".($i+1)."] ".$choices[$i]->{ 'val' };
            }

            #
            # Accept user input for choice
            #

            print "\n\nEnter a choice [default: 1] : ";
            chomp ( $res = <STDIN> );

            AnswerVM ( $service, $handle, $choices[$res],
                    $questionObj->{ 'id' } );
            last;

      }
      $counter++;
```

```
      }


      #
      # Obtain the latest vHandle
      #


      $vHandle = $taskObj->valueof( '//vHandle' );
      }


######################################################################
#
# AnswerVM --
#
#      Sets up and calls the AnswerVM method with the appropriate
#      parameters.
#
# Results:
#      Calls AnswerVM
#
# Side effects:
#      None.
#
######################################################################


sub AnswerVM
{

    my ( $service, $handle,  $choice, $id ) = @_;

      #
      # Construct an ordered hash for the choice of answer
      #

      tie %$choiceSpec, "Tie::IxHash";
      $choiceSpec->{ 'key' } = $choice->{ 'key' };
      $choiceSpec->{ 'val' } = $choice->{ 'val' };


      #
      # Setup and call AnswerVM
      #
```

```
        my $method = SOAP::Data->name('AnswerVM')
              ->attr({xmlns => 'urn:vma1'});

        @params = (
                    SOAP::Data->name(vm => $handle),
                    SOAP::Data->type('KeyedValue')->name(choice => $choiceSpec),
                    OAP::Data->name(id => $id)
                    );

        $service->call($method => @params);

}
```

# Cloning a Virtual Machine

The CloneVM operation creates a new virtual machine by using as its source, an existing virtual machine or a template. It takes five mandatory arguments:

- Handle to the (source) virtual machine or template that will be cloned.
- Handle to the VirtualMachineGroup or Farm in which the cloned virtual machine will reside (the parent handle).
- Handle to the host where the new virtual machine will reside.
- Name of the newly cloned virtual machine.
- Location on the destination host where the cloned virtual machine's configuration files and virtual disks will reside.

There are two optional arguments:

- customization of the guest operating system for the newly cloned virtual machine. See the next section for additional details.
- flag that determines whether or not the newly cloned virtual machine will automatically power on once the cloning operation is complete.

Virtual machines must be powered off in order for the CloneVM operation to succeed. If the source is a template located on a host's datastore, then the newly cloned virtual machine must also reside on the same host where the datastore is located. However, if hosts are on the same SAN and share the same datastore, then the CloneVM operation (from a template) may be done across these hosts.

If the source is a virtual machine, then the client must have Configure rights on that virtual machine and on the Farm of its host. If the newly cloned virtual machine will reside on a different host, then the client must have Configure rights on the destination host of this new virtual machine.

Once the operation has been initiated, the request returns a task handle to the client. The client may monitor the task for the progress of the operation.

## Customizing a Virtual Machine

The schema for the customization specification has been incorporated into `vma.wsdl`. When a client generates stub files, a class (structure) is created that represents the customization parameters.

Clients can customize both Windows and Linux guest operating systems. For more information on customizing a guest operating system, refer to the section titled "Preparing for Guest Customization" in the *VMware VirtualCenter User's Manual*.

### Customizing a Windows Guest Operating System

If you plan to customize a Windows guest operating system, then you must first install the Microsoft Sysprep tools package on the VirtualCenter server machine. Follow the procedure in the section titled "Preparing for Guest Customization" in the *VMware VirtualCenter User's Manual*.

For example, clients can customize the following:

- Registration information — User's full name and organization.
- Computer name — Computer or host name, used for identifying this virtual machine on a network.
- Administrator password — Password for the Administrative user.
- Timezone — Time zone for the virtual machine.
- AutoLogon — Enables the virtual machine to log on automatically to the Administrator account the first time the machine boots.
- Product ID — Product ID (license key) for the guest operating system
- Network settings — DHCP or static IP address.

For example, here's a customization code example for a Windows guest operating system:

```
sub GetCustomizationSettings
{
   my %custSpec = ();
   tie %$custSpec, "Tie::IxHash";

   tie %{$custSpec->{sysprep}}, "Tie::IxHash";
   tie %{$custSpec->{sysprep}->{GuiUnattended}}, "Tie::IxHash";
   print "\nEnter Windows administrator password : ";
   chomp( $custSpec->{sysprep}->{GuiUnattended}->{AdminPassword} = <STDIN>);
   $custSpec->{sysprep}->{GuiUnattended}->{TimeZone} = '004';
   $custSpec->{sysprep}->{GuiUnattended}->{AutoLogon} = 'true';
   $custSpec->{sysprep}->{GuiUnattended}->{AutoLogonCount} = 1000;

   tie %{$custSpec->{sysprep}->{UserData}}, "Tie::IxHash";
   print "\nEnter user name : ";
   chomp( $custSpec->{sysprep}->{UserData}->{FullName} = <STDIN> );
   print "\nEnter organization name : ";
   chomp( $custSpec->{sysprep}->{UserData}->{OrgName} = <STDIN> );
   print "\nEnter computer name : ";
   chomp( $custSpec->{sysprep}->{UserData}->{ComputerName} = <STDIN> );


   print "\nEnter workgroup name : ";
   chomp( $custSpec->{sysprep}->{Identification}->{JoinWorkgroup} = <STDIN> );
```

```
   tie %{$custSpec->{adapters}->{adapter}}, "Tie::IxHash";
   $custSpec->{adapters}->{adapter}->{MACAddress} = 'MAC00';
   $custSpec->{adapters}->{adapter}->{UseDHCP} = 'true';
   $custSpec->{adapters}->{adapter}->{DNSFromDHCP} = 'true';
   $custSpec->{adapters}->{adapter}->{NetBIOS} = 'EnableNetBIOS';

   return $custSpec;
}
```

### Customizing a Linux Guest Operating System

If you plan to customize a Linux guest operating system, then you must first install the VMware Open Source components on the VirtualCenter server machine. Click on the download link for Open Sources at *www.vmware.com/download* and follow the procedure in the section titled "Preparing for Guest Customization" in the *VMware VirtualCenter User's Manual*.

For example, clients can customize the following:

- Computer name — Computer or host name, used for identifying this virtual machine on a network.
- Network settings — DHCP or static IP address.

```
sub GetCustomizationSettings
{
   my %custSpec = ();
   tie %$custSpec, "Tie::IxHash";

   tie %{$custSpec->{linux-global}}, "Tie::IxHash";
   $custSpec->{linux-global}->{ComputerName} = 'LinuxCustomVM';
   $custSpec->{linux-global}->{Domain} = 'mydomain.com';

   tie %{$custSpec->{linux-global}->{DNS}}, "Tie::IxHash";
   $custSpec->{linux-global}->{DNS}->{PrimaryDNS} = 'xx.xx.xx.x';
   $custSpec->{linux-global}->{DNS}->{SecondaryDNS} = 'xx.xx.xx.x';
   $custSpec->{linux-global}->{DNS}->{DNSSearchPaths}
   ->{DNSSearchPath} = 'mydomain.com';

   tie %{$custSpec->{adapters}->{adapter}}, "Tie::IxHash";
   $custSpec->{adapters}->{adapter}->{MACAddress} = 'MAC00';
   $custSpec->{adapters}->{adapter}->{UseDHCP} = 'true';
   $custSpec->{adapters}->{adapter}->{DNSFromDHCP} = 'true';
   $custSpec->{adapters}->{adapter}->{NetBIOS} = 'EnableNetBIOS';

   return $custSpec;
}
```

### Calling the CloneVM Operation

Before calling the CloneVM operation, the client first needs to instantiate an object of type Autoprep, and set the various parameters in this object. Then the client can pass this object as an additional argument to the CloneVM operation.

**Caution:** Be sure that the Microsoft Sysprep tools (Windows guest operating system) or the VMware Open Source components (Linux guest operating system) is installed in the VirtualCenter server machine before starting the CloneVM operation. Otherwise, the CloneVM operation will fail at the end of this operation.

## CloneVM Sample

This sample illustrates the CloneVM operation:

```
#
# Setup and call the CloneVM method after obtaining the necessary
# handles
#

#
# $srcHandle -- handle to a template or a source VM
# $dstPathHandle -- handle to the parent where the new VM will reside
# $dstHostHandle -- handle to the destination host
# $dataStore -- label of the datastore where the new VM will reside
#


$method = SOAP::Data->name('CloneVM')
    ->attr({xmlns => 'urn:vma1'});

@params = (
           SOAP::Data->name(srcHandle => $srcHandle),
           SOAP::Data->name(parentHandle => $dstPathHandle),
           SOAP::Data->name(destHostHandle => $dstHostHandle),
           SOAP::Data->name(name => 'newVM'),
           SOAP::Data->name(datastore => $dataStore),
           SOAP::Data->type('xsd:boolean' )->name(autopoweron => 1)
          );

$task = $service->call($method => @params)->result;

MonitorTask ( $service, $task );
```

# Creating a Template

Clients can specify a handle to a template as the first argument of the CloneVM operation. Clients can create this template by using the CreateTemplate operation. The CreateTemplate operation takes two arguments, the handle of the source virtual machine being used to create the template, and an XML document describing a TemplateSpec.

The TemplateSpec specifies attributes of the new template such as its name, the datastore that contains the template's configuration file, the location of the virtual disk(s) for this template, and a user description for this template. If no datastore is specified in the XML document, then the configuration file and virtual disks for this template are placed in a local template upload directory on VirtualCenter.

The client must have Configure rights on the source virtual machine.

A task is initiated on the Web service for the CreateTemplate operation. The XML document describing that task is returned to the client. The client can monitor the progress of this task and check for errors, status and so on.

See Task Scheduling and Monitoring on page 214 for sample code on how to monitor the progress of a task.

```
#
# Create TemplateSpec structure
#

my %templateSpec = ();
tie %$templateSpec, "Tie::IxHash";

#
# Call ResolvePath to obtain the handle of source VM
#

$method = SOAP::Data->name('ResolvePath')
   ->attr({xmlns => 'urn:vma1'});
@params =  (
             SOAP::Data
             ->type('xsd:string')
             ->name(path => '/vm/5038d66d-ed8b-2d32-8fdb-4a5e090715f7' )
            );
$srcHandle = $service->call($method => @params)->result;

#
# Populate the TemplateSpec
#
```

```
$templateSpec->{ 'name' } =
  SOAP::Data->type('xsd:string')->name(name => 'newTemplate' );
$templateSpec->{ 'description' } =
  SOAP::Data->type('xsd:string')->name(description => 'none' );
$templateSpec->{ 'datastore' } = $datastore_handle;

#
# Setup and call CreateTemplate
#

$method = SOAP::Data->name('CreateTemplate')
  ->attr({xmlns => 'urn:vma1'});
@params = (
            SOAP::Data->name(srcHandle => $srcHandle),
            SOAP::Data->type('TemplateSpec')
            ->name(info => $templateSpec )
           );
$task = $service->call($method => @params)->result;

#
# Monitor the returned task
#

MonitorTask ( $service, $task );
```

## Specifying a Datastore

Clients must specify a datastore in TemplateSpec. Clients can obtain the datastore from the Host object, once the host for a virtual machine has been determined. Alternatively, clients can obtain a list of all the available datastores by calling a ResolvePath, then a GetContents operation on the `/datastore` path. The following sample code illustrates how to correlate the datastore information obtained from the Host object with the datastore information from the `/datastore` path.

```
#
# Setup and call GetContents on the host
#

$method = SOAP::Data->name('GetContents')
     ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(handle => $handle ));
$result = $service->call($method => @params);

#
# Extract & display the list of datastores available on the host
#
```

```
@dataStores = $result->valueof('//datastore');

if ( $#dataStores+1 == 0 ){
    print "\nNo datastores available. Exiting.\n";
    exit;
}

#
# Call ResolvePath to obtain handle for the container /datastore
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(path => '/datastore' ));
$datastore_handle = $service->call($method => @params)->result;

#
# Obtain capacity information for the available datastores
#

$method = SOAP::Data->name('GetContents')
    ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(handle => $datastore_handle ));
$datastore_contents = $service->call($method => @params);


@dataStoresInfo = $datastore_contents->valueof('//datastore');

#
# Display datastore information
#

$counter = 0;

foreach $dataStore (@dataStores) {

    print "\n\n $dataStore"." [".$counter."] ";

    foreach $dataStoreInfo (@dataStoresInfo) {

        if ( $dataStoreInfo->{ 'key' } eq $dataStore ){
            print "\n\t Capacity : ".$dataStoreInfo->{ 'capacityMB' }." MB";
            print "\n\t Free Space : ".$dataStoreInfo->{ 'freeSpaceMB' }." MB";
        }
    }
    $counter++;
}
```

# Renaming an Object

The Rename operation requests a change to the name of an existing object and optionally moves it in the `/vcenter` hierarchy. Clients can rename Containers, Farms, virtual machine groups and virtual machines. However, clients cannot use the Rename operation to migrate a virtual machine and move it from one host to another.

**Note:** You can only move a virtual machine or VirtualMachineGroup to a new VirtualMachineGroup. Similarly, you can only move a Farm or a Farm group (Container) to a new Farm group. You cannot move a VirtualMachineGroup to a new Farm and you cannot move objects across Farms.

The Rename operation takes two mandatory arguments: the handle of the existing object that is to be renamed and the new name of the object. If a name change is not desired, then pass in the current name of the object. It has one optional argument, the new destination for the object being moved. If this parameter is not specified, then the object is not moved, but is simply renamed in its current location.

**Note:** The destination object, pointed to by destHandle, must be capable of holding the type of object specified by the handle parameter or an error is returned.

The client must have Browse and Configure rights for both the current and new Container for the object. Upon success, an empty response message is returned.

The following sample illustrates renaming an object.

```
#
# Setup & call the ResolvePath method to obtain source handle
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->type('xsd:string')->name(path => $srcPath));

$srcHandle = $service->call($method => @params)->result;


#
# Setup & call the ResolvePath method to obtain destination handle
#

if ( $dstPath ){
      $method = SOAP::Data->name('ResolvePath')
            ->attr({xmlns => 'urn:vma1'});
```

```
      @params = (SOAP::Data->type('xsd:string')->name(path => $dstPath));

      $dstHandle = $service->call($method => @params)->result;
}


#
# Setup and call the Rename method
#

$method = SOAP::Data->name('Rename')
            ->attr({xmlns => 'urn:vma1'});

if ( $dstHandle ){

      @params = (
                  SOAP::Data->name(handle => $srcHandle),
                  SOAP::Data->name(destHandle => $dstHandle),
                  SOAP::Data->name(name => $newName)
                );

} else {

      @params = (
                  SOAP::Data->name(handle => $srcHandle),
                  SOAP::Data->name(name => $newName)
                );
}

$service->call($method => @params)->result;
```

# Moving Virtual Machines

VMware SDK provides two operations to move virtual machines across hosts, MigrateVM and MoveVM. Clients should use MigrateVM when the virtual machine is being moved to a new host without moving its virtual disk(s). This operation is applicable when both the original host and the destination host share a SAN. By contrast, clients should use the MoveVM operation to move a virtual machine's disk to the destination host.

A task is initiated on the Web service for the MigrateVM and MoveVM operations. The XML document describing that task is returned to the client. The client can monitor the progress of this task and check for errors, status, and so on. See Task Scheduling and Monitoring on page 214 for sample code on how to monitor the progress of a task.

If the virtual machine is being moved and the client is monitoring it for updates, then the client will receive updates on the VirtualMachineState (on the host and the details fields). The host field in the VirtualMachineState is updated with the destination host name once the migration successfully completes.

The following sample illustrates moving a virtual machine's disks:

```
#
# Setup & call the ResolvePath to obtain vm handle and host handle
#



$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->type('xsd:string')->name(path => $vmPath));

$vmHandle = $service->call($method => @params)->result;

@params = (SOAP::Data->type('xsd:string')->name(path => '/host/'.$host));

$dstHostHandle = $service->call($method => @params)->result;


#
# Setup and call MoveVM
#

$method = SOAP::Data->name('MoveVM')
    ->attr({xmlns => 'urn:vma1'});
```

```
if ( defined $diskKey ) {

    #
    # DiskKey defined - create parameter of type VirtualDiskDestination
    #

      my %diskInfo = ();
      tie %$diskInfo, "Tie::IxHash";

      $diskInfo->{ 'key' } = $diskKey;
      $diskInfo->{ 'dataLocator' } = '';

      @params = (
                  SOAP::Data->name(vm => $vmHandle),
                  SOAP::Data->name(host => $dstHostHandle),
                  SOAP::Data
                    ->type('VirtualDiskDestination')
                    ->name(disk => $diskInfo)
                );
} else {

      @params = (
                  SOAP::Data->name(vm => $vmHandle),
                  SOAP::Data->name(host => $dstHostHandle),
                );
}



$task = $service->call($method => @params)->result;

#
# Monitor the task to determine progress
#

MonitorTask ( $service, $task );
```

## Migrating a Virtual Machine

The MigrateVM operation starts the process of migrating a virtual machine to a specific host, without moving the virtual disk(s). In this release, the virtual machine must be in the poweredOn state. This operation never moves the virtual machine's disks from its current location.

**Note:** The MigrateVM operation is not supported for a GSX Server host.

This operation takes two mandatory arguments: the handle to the virtual machine that will be migrated, and the handle to the destination (target) host. There are two optional arguments: priority (determines whether or not resources are preallocated before migration starts) and the path describing the location of the virtual machine configuration file.

The client must have Configure rights on the virtual machine and on the Farm.

The request returns once the MigrateVM operation has been initiated by returning a task handle back to the client. You can determine whether the migration is successful, by separately monitoring the task that is performing the migration or by monitoring the virtual machine state (specifically the detail or host fields in VirtualMachineState). If the virtual machine has been successfully migrated, then the host field should contain the target host handle.

The following sample illustrates how to migrate a virtual machine:

```
#
# Setup & call the ResolvePath to obtain vm handle and host handle
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (
            SOAP::Data
              ->type('xsd:string')
              ->name(path => '/vm/564d5a05-29a7-b09b-d576-9cb8a719d940')
          );


$vmHandle = $service->call($method => @params)->result;

@params = (
            SOAP::Data
              ->type('xsd:string')
              ->name(path => '/host/myhost.mydomain.com')
          );


$dstHostHandle = $service->call($method => @params)->result;

#
# Setup and call MigrateVM
#

$method = SOAP::Data->name('MigrateVM')
    ->attr({xmlns => 'urn:vma1'});
```

```
@params = (
             SOAP::Data->name(vm => $vmHandle),
             SOAP::Data->name(host => $dstHostHandle),
           );

$task = $service->call($method => @params)->result;

#
# Monitor the task to determine progress
#

MonitorTask ( $service, $task );
```

## Moving a Virtual Machine's Virtual Disks

The MoveVM operation moves a virtual machine's virtual disk(s) to a different location. You may (optionally) also use this operation to move the virtual machine to a different host. You must perform this operation on a virtual machine while it is in the poweredOff state.

The MoveVM operation takes one mandatory argument, the handle to the virtual machine that will be enabled. There are three optional arguments: the handle to the destination host (if both the virtual machine and its virtual disk(s) are moved), the dataLocator parameter (path) describing the location of the virtual machine configuration file, and the destination for all disks in the virtual machine.

The client must have Configure rights on the virtual machine and on the Farm of its host. If you are also planning to move this virtual machine to a different host, then the client must also have Configure rights on the target (destination) host.

The request returns once the operation has been initiated by returning a task handle back to the client. The client may monitor the task for progress of the operation.

# Monitoring Events

The client can obtain events information, as specified by the Event Log data models, described in the *Virtual Infrastructure SDK Reference Guide*.

Clients can collect events on hosts or on virtual machines, including informational, warning, or error messages such as changes in power operations or device status (connected, disconnected, or busy). In addition, clients can receive updates when alarms occur; for example, when memory or CPU usage, or virtual machine heartbeat is either above or below normal.

Each event comprises two parts: a declaration (the type of event), and the actual event (an event log).

Clients can access all information related to events by calling GetContents on the `/event` handle. This returns a Container with two Items:

- Handle for all event declarations under `/event/decls`
- Handle for all events under `/event/all`

## Event Declarations

The view, `/event/decls`, represents all the event declarations. This view is an EventDeclList object, that contains a single array field called "decl". Each entry is this array is a separate EventDecl (event declaration). By calling GetContents on `/event/decls`, the client can obtain all the known event declarations in the system. The event declarations are a set of pre-defined event types that do not get updated. Therefore, clients do not need to call GetUpdates on the handle for `/event/decls`.

Each event declaration has the following attributes:

- key — String that is the ID of this event declaration.
- kind — Type of event, that is one of the following: "alert", "error", "warning", "info", or "user".
- msgFmt — Array of format strings that describes how the event message is rendered; for example, "Task%0 created on %1".
- schedule — (Optional) Handle of the schedule (if any) that caused this event.

## Event Logs

The view, `/event/all`, represents all the available events. This view is an EventCollector Container that has 2 Items: `/event/all/filter`, which is an EventFilter and `/event/all/events`, which is of type EventCollection.

The EventFilter object describes how all the events listed under EventCollection are grouped together. The EventCollection contains a single array field called `log`. Each entry in this array is a separate Event object. Each Event object describes a unique event that occurred in the system.

In the case of `/event/all`, no filtering is done and all events are listed under `/event/all/events`; the filter under `/event/all/filter` is empty. EventFilter objects have the following attributes, indicating the attribute used to group the events:

- parent — (Optional) ID of the parent event that generated the events in this filter. (If this attribute is set, then the events in this event collector are grouped by the generating parent event.)
- schedule — (Optional) ID of the scheduled task that generated this event. (If this attribute is set, then the events in this event collector are all the events generated by this task schedule.)
- kind — String indicating the event kind. Events of the same kind are grouped together.
- startTime — Collects events that occurred after this start time.
- end time — Collects events that occurred before this end time.
- farm — (Optional) Collects events that occurred under this Farm.
- vm — (Optional) Collects events that occurred on this virtual machine.
- host — (Optional) Collects events that occurred on this host.
- declId — (Optional) Collects events that have this declaration ID.
- totalEvents — (Optional) Collects this number of events in the event collector, giving priority to more recent events.

Event objects have the following attributes:

- key — String that is the ID of the event.
- decl — Declaration ID corresponding to this event.
- arg — Array of name-value pairs that holds the value for the parameters in the message format string. Each entry in this array corresponds to a matching entry in the argType[ ] array in this event's declaration.
- parent — String that is the event ID of the parent event that caused this event.
- timestamp— Date and time when this event occurred.
- userDesc — (Optional) String that is the user description of this event.
- vm — (Optional) Handle of the virtual machine that owns this event.
- host — (Optional) Handle of the host that owns this event.
- farm — (Optional) Handle of the Farm that owns this event.

Clients that are interested in gathering event information should first call GetContents on the handle for `/event/decl` and `/event/all/events`. The declaration data that is received

should be cached by the client. The client can then call GetUpdates on `/event/all/events` to be notified of events, as they occur.

The following sample illustrates how clients can monitor events as they occur.

```
#
#  Setup and call ResolvePath to obtain handle
#

$method = SOAP::Data->name('ResolvePath')
                    ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(path => "/event/all/events"));
$eventHandle = $service->call($method => @params)->result;

#
#  Setup and call GetContents to get a list of all the events
#

$method = SOAP::Data->name('GetContents')
                     ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(handle => $eventHandle));
$result = $service->call($method => @params);

#
#  Parse the list of events by traversing each element in the
#  body of GetContentsResponse
#

if ( !defined $result->valueof('//event') ){
       print "\n There are no Events\n";
} else {
       my $i =0;
       foreach my $obj ($result->dataof( "//body/*" )) {
               $i++;
               foreach my $event ( $result->dataof( "//body/[$i]/*" )) {

               ...process the event...

               }

       }
}
```

## Creating an Event Collector

Clients can create an event collector to filter events on the following attributes:

- kind — Filters by the type of event; for example, alert, info, warning, and so on.
- declId — Filters an event specified by its VirtualCenter event declaration ID
- startTime — Filters any events that occurred before the specified start time.
- endTime — Filters any events that occurred after the specified end time.
- parent event — Filters any events with the specified parent event.
- schedule — Filters any events that are caused by the specified schedule
- host, vm, or farm — Filters for events associated with the specified host, virtual machine, or Farm.

Once it has been created, this event collector is only accessible by the handle returned by the Create operation. Clients cannot view this event collector in the /event view hierarchy.

For example, a client may create an event collector if it is interested in only "error" events from a particular host or virtual machine. The newly created event collector only shows these events.

The following sample code illustrates the creation of a filtered event collector:

```
#
# Call ResolvePath to obtain top-level /event handle
# and handle for the host
#

$method = SOAP::Data->name('ResolvePath')
                    ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(path => "/event"));
my $eventHandle = $service->call($method => @params)->result;

@params = (SOAP::Data->name(path => '/host/myhost.mydomain.com'));
my $hostHandle = $service->call($method => @params)->result;

#
# Construct event collector spec to filter the events returned
# The following event collector retrieves error events associated with a
# particular host
#

my %eventSpec = ();
tie %{$eventSpec->{filter}}, "Tie::IxHash";

$eventSpec->{filter}->{kind} = 'error';
```

```
$eventSpec->{filter}->{startTime} = '';
$eventSpec->{filter}->{endTime} = '';
$eventSpec->{filter}->{host} = $hostHandle;

#
# Setup and call Create to create the EventCollector
#

$method = SOAP::Data->name('Create')
                    ->attr({xmlns => 'urn:vma1'});

@params =     (
              SOAP::Data->name(handle => $eventHandle),
              SOAP::Data->name(name => 'HostErrors'),
              SOAP::Data->name(type => 'EventCollector'),
              SOAP::Data->type('EventCollector')
                        ->name(initial => $eventSpec)
              );

my $handle = $service->call($method => @params)->result;
```

The following code sample shows how to extract the list of events from this event collector:

```
#
# Call GetContents on the newly created EventCollector
#

$method = SOAP::Data->name('GetContents')
        ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(handle => $handle));
$result = $service->call($method => @params);

#
# Each event collector contains 2 items - an event
# filter and an Event Collection object. Obtain the
# handle for the Event Collection object.
#

my @items = $result->valueof('//item');

foreach my $item ( @items ) {
        if ( $item->{type} eq 'EventCollection' ) {
                $eventHandle = $item->{key};
                last;
        }
}
```

```
#
# Obtain the contents of the Event Collection object
#

@params = (SOAP::Data->name(handle => $eventHandle));
$result = $service->call($method => @params);

#
# Display the events
#

PrintEvents( $result);

# See code sample for full listing on how to do this
```

# Task Scheduling and Monitoring

Clients create tasks on the Web service in order to execute API operations.

## Active Tasks and Scheduled Tasks

There are two types of tasks: active tasks and scheduled tasks. Active tasks are tasks that are currently running on the Web service. Scheduled tasks are tasks that will be run on a pre-defined, and possibly a recurring, schedule. When a scheduled task runs, a new active task is created in the Web service.

Clients can obtain information about all currently active tasks by calling GetContents on the `/task` handle. Similarly, clients can obtain information about all scheduled tasks that currently exist on the Web service, by calling GetContents on the `/schedule` handle.

The Web service can create tasks in response to certain long-running operations. The Web service initiates such operations as a task. The Web service returns control to the client without waiting for the task or operation to complete. The response contains information about the task that was created on the Web service for that operation. Some examples of long running tasks include StartVM, StopVM, ResetVM, MigrateVM, and so on. Clients can access information about the task to determine its status and its progress as described in the next few sections.

Clients can also create scheduled tasks directly on the Web service. Clients identify the operation, then specify a schedule for running the task. The Web service uses this schedule to determine when to run the task. See Creating New Scheduled Tasks on page 218.

## Monitoring Tasks

Clients can monitor for new tasks as they occur by calling GetUpdates on the `/task` handle. Clients can receive notifications on new scheduled tasks as they are created on the Web service by calling GetUpdates on the `/schedule` handle.

The GetUpdates operation on the `/schedule` or the `/task` handle returns a Container of Items. Each Item has a handle to a Task object. Each Task object has the attributes listed in the following table.

| Task Object Attribute | Description |
|---|---|
| operationName | String identifying the operation that this task is running. |
| cause | String identifying the agent that caused this task's creation. |
| schedule | String identifying the handle for the scheduled task that created this task, if applicable. |

| Task Object Attribute | Description |
|---|---|
| entity | String identifying the handle for the object that this task is operating on. |
| eventCollector | Handle of the EventCollector that contains all events associated with this task. |
| currentState | TaskRunState, an object that indicates this task's current status (running, failed, completed, and so on). |
| percentCompleted | Float indicating the percentage of the task that has completed. |
| normalReturn | Object that contains the return value from the operation that the task executed. |
| faultReturn | FaultInfo object that captures any errors or exceptions encountered by the operation that this task was executing. |
| allowCancel | Boolean indicating if this running task can be cancelled |
| queueTime | Date and time this task was created. |

The following sample illustrates how the client can stay informed of all new tasks in the system.

```
#
# Setup & call the ResolvePath method
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(path => '/task'));

$handle = $service->call($method => @params)->result;


#
# Setup and call the GetContents method.
#



$method = SOAP::Data->name('GetContents')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(handle => $handle));

$result = $service->call($method => @params);
```

```
# Extract the list of tasks

my @items = $result->valueof('//item');

if ( $#items < 0 ) {
      print "\nThere are no Active tasks\n";
} else {

      foreach my $item ( @items ) {

      print "\n Key : ". $item->{key};
      print "\n Name : ". $item->{name};
      }
}
```

Clients can monitor a task for its progress by calling GetUpdates on the handle for the task of interest. The client can look at the task's current run state and percentCompleted values to chart a task's progress and determine when it completes. The following sample illustrates this concept.

```
sub MonitorTask
{
    my ( $taskState, $vHandle, $method, @params, @changes, $vHandleList,
$taskObj );
    my $service = shift;
    my $taskRef = shift;

    $taskState = $taskRef->{ 'body' }->{ 'currentState' };
    $vHandle = $taskRef->{ 'vHandle' };

    #
    # Call GetUpdates repeatedly to monitor the status of the Task
    #

    $method = SOAP::Data->name('GetUpdates')
      ->attr({xmlns => 'urn:vma1'});

    print "\nTask Status : $taskState \n";

    while ( $taskState ne 'completed' && $taskState ne 'failed'){

      #
      # Setup and call GetUpdates
      #

      $vHandleList->{ 'vHandle' } = $vHandle;
```

```
@params = (
            SOAP::Data->name(vHandleList => $vHandleList),
            SOAP::Data->type('xsd:boolean')->name(wait => 'true')
           );

$task = $service->call($method => @params);
@changes = $task->valueof('//change');

#
# Update Task status information using the list of changes
#

foreach $change (@changes){

    $target = $change->{ 'target' };
    $taskState = $change->{ 'val' };

    #
    # Task failed -- fault returned
    #

    if ( $target eq 'faultReturn' ){
      $taskState = 'failed';
      print "\n$change->{ 'val' }->{ 'kind' } : ";
      print "$change->{ 'val' }->{ 'info' }";
      last;
    }

    if ( $target eq 'percentCompleted' ){
      if ( $taskState == 100){
          $taskState = 'completed';
      }elsif ( $taskState == 0 ){
          $taskState = 'failed';
      }else {
          $taskState = 'running';
      }
    }

    print "\nTask Status : $taskState \n";
}

#
# Obtain the latest vHandle
#

$vHandle = $task->valueof('//vHandle');
```

```
    }

    return;
}
```

## Creating New Scheduled Tasks

Clients can create new scheduled tasks on the Web service. The following operations can be scheduled as a task:

- Power operations — StopVM, StartVM, and ResetVM.
- PutUpdates — changing the resource settings of a virtual machine. Clients must pass a handle (and not a vHandle) for a scheduled PutUpdates operation.
- MigrateVM
- MoveVM
- CloneVM
- CreateTemplate

To create a task schedule, clients need four parameters, as specified by the TaskScheduleSpec datatype:

- name — Name of the task schedule.
- operationName — Name of the scheduled API operation.
- parameter — Parameters for the operation. If the operation requires no parameters, then this field is not required.
- recurrence — Recurrence of this task schedule.

Once a task schedule is created, there are additional parameters that define the task schedule. Refer to the description for the TaskSchedule datatype in the *Virtual Infrastructure SDK Reference Guide*.

The following sample code illustrates creating a scheduled task that performs the power-on operation.

**Creating a Scheduled Task**

```
#
# Create a TaskSpec hash
#

my %taskSpec = SOAP::Data
                      ->type('TaskScheduleSpec');
```

```
tie %$taskSpec, "Tie::IxHash";

$taskSpec->{ 'name' } = 'newTask';
$taskSpec->{ 'operationName' } = 'StartVM';

#
# Set the parameter for the task
#

tie %{$taskSpec->{ 'parameter' }}, "Tie::IxHash";
$taskSpec->{ 'parameter' }->{ 'key' } = 'vm';
$taskSpec->{ 'parameter' }->{ 'val' } = $vmhandle;

#
# Set task recurrence
#

my %task = ();
tie %$task, "Tie::IxHash";

#
# One Time Task
#

$task->{ 'runTime' } = undef;
$taskType = SOAP::Data
                ->type('OneTimeTask' )
                ->name(recurrence => $task);


$taskSpec->{ 'recurrence' } = $taskType;

#
# Obtain the handle for /schedule
#

$method = SOAP::Data->name('ResolvePath')
                ->attr({xmlns => 'urn:vma1'});
@params = ( SOAP::Data->name(path => '/schedule'));

$handle = $service->call($method => @params)->result;

#
# Create the task schedule
#

$method = SOAP::Data->name('Create')
```

```
                ->attr({xmlns => 'urn:vma1'});

@params = (
           SOAP::Data->name(handle => $handle),
           SOAP::Data->name(name => $name),
           SOAP::Data->name(type => 'TaskSchedule'),
           SOAP::Data->type('TaskScheduleSpec')->name(initial => $taskSpec)
           );

my $taskHandle = $service->call($method => @params)->result;
```

In the preceding sample, parameters for the operation are specified using KeyedValue objects named `parameter`.

The operation name specified in the TaskScheduleSpec datatype must match exactly the operation name specified in the WSDL; for example, when setting up a task to clone a virtual machine, the operation name should be CloneVM. Similarly, the parameter names should exactly match the parameter names for that operation as specified in the WSDL. The following sample code illustrates the parameters for the StartVM operation.

**StartVM Operation Parameters**

```
#
# Obtain the handle for the VM for which the task is scheduled
#

$method = SOAP::Data->name('ResolvePath')
                ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(path => $vmPath));

$vmhandle = $service->call($method => @params)->result;

#
# Specify the parameter(s) for the task
#

tie %{$taskSpec->{ 'parameter' }}, "Tie::IxHash";
$taskSpec->{ 'parameter' }->{ 'key' } = 'vm';
$taskSpec->{ 'parameter' }->{ 'val' } = $vmhandle;
```

When multiple parameters are involved, such as in the StopVM operation, the specification must be constructed more carefully so as to avoid overwriting of duplicate keys in a hash. This is demonstrated in the following sample:

```
#
# Create the task schedule
#

$method = SOAP::Data->name('Create')
     ->attr({xmlns => 'urn:vma1'});

@params =
     (
     SOAP::Data->name(handle => $handle),
     SOAP::Data->name(name => $name),
     SOAP::Data->name(type => 'TaskSchedule'),
     SOAP::Data->type('TaskScheduleSpec')->name(initial =>
          \SOAP::Data->value(
               SOAP::Data->name( name => $name),
               SOAP::Data->name( operationName => 'StopVM'),
               SOAP::Data->name( parameter =>
                    \SOAP::Data->value (
                         SOAP::Data->name( key => 'vm'),
                         SOAP::Data->name( val => $vmhandle) )),
               SOAP::Data->name( parameter =>
                    \SOAP::Data->value (
                         SOAP::Data->name( key => 'suspend'),
                         SOAP::Data->name( val => 'true') )),
               SOAP::Data->name( parameter =>
                    \SOAP::Data->value (
                         SOAP::Data->name( key => 'soft'),
                         SOAP::Data->name( val => 'true') )),
               SOAP::Data->name( recurrence => $taskType ) ))
     );


my $taskHandle = $service->call($method => @params)->result;
```

The recurrence for a scheduled task is an object as specified in the preceding table. Each object has a set of parameters that specify when to run the task. For example, clients can specify a weekly task:

```
my %task = ();
tie %$task, "Tie::IxHash";

#
# Weekly Task
#

# What time of the day this task should fire
$task->{ 'hours' } = 6;
```

```
$task->{ 'minutes' } = 5;

#on which day(s) of the week should this fire
$task->{ 'dayOfWeek' } = 'sunday';

# Run this task every xx weeks
$task->{ 'interval' } = 2;

$taskType = SOAP::Data
            ->type('WeeklyTask' )
            ->name(recurrence => $task);

$taskSpec->{ 'recurrence' } = $taskType;
```

## Running a Scheduled Task

Clients can run a scheduled task on the Web service by calling the RunTask operation. This operation takes one argument, the handle of the task. The client must have Interact rights for the specified task.

Upon success, an empty response message is returned.

```
#
# Run the scheduled task
#

$method = SOAP::Data->name('RunTask')
            ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(task => $taskHandle));
```

## Ending a Task

Clients can stop a running task, or cancel a task that has not yet been started on the Web service by calling the EndTask operation. This operation takes one argument, the handle of the task. The client must have Interact rights for the specified task.

Upon success, an empty response message is returned.

```
#
# End the scheduled task
#

$method = SOAP::Data->name('EndTask')
            ->attr({xmlns => 'urn:vma1'});
```

```
@params = (SOAP::Data->name(task => $taskHandle));

$service->call($method => @params);
```

# Collecting Performance Data

The client can obtain performance data, as specified by the Performance Metric data model, described in the *Virtual Infrastructure SDK Reference Guide*. Clients can collect performance data on hosts or on virtual machines, including CPU and memory utilization, network and disk performance data, and floppy and CD-ROM drive performance, and so on.

Clients obtain performance statistics through a performance collector. A performance collector, also known as a perf collector, is an object that collects a certain set of statistics at a specified interval frequency.

The view, `/perf`, is a container for all the performance collectors. There are two types of performance collectors:

- VirtualCenter perf collectors — VirtualCenter perf collectors specify a sampling interval and are visible in the VMware VirtualCenter application.
- Filtered perf collector — Filtered perf collectors (children) are filtered from an existing VirtualCenter perf collector (parent) and are not visible in the VirtualCenter application.

Clients can create both types of performance collectors by using the Create operation.

- A new VirtualCenter perf collector with a different sampling interval from any other VirtualCenter perf collectors.
- A perf collector that filters the statistics of an existing VirtualCenter perf collector.

## VirtualCenter Perf Collector

VirtualCenter performance collectors are named according to their sampling interval. They have only a sampling interval and no default filter. There are four default VirtualCenter perf collectors:

- Five minutes — `/perf/0000000300`. Historical samples are retained for a day.
- One hour — `/perf/0000003600`. Historical samples are retained for a week.
- Six hours — `/perf/0000021600`. Historical samples are retained for a month.
- One day — `/perf/0000086400`. Historical samples are retained for a year.

Clients can also create a VirtualCenter perf collector by using the Create operation. When doing so, specify only the name and the sampling interval (initial parameter comprising XML document of type PerfCollector). See Creating a VirtualCenter Perf Collector on page 227.

**Note:** When creating a VirtualCenter perf collector, the user-friendly name does not appear in the `/perf` directory. Instead, the newly created VirtualCenter perf collector appears as its sampling interval. However, the user-friendly name appears in the VirtualCenter application.

For example, if you create a new VirtualCenter perf Collector called "Every Minute" with a sampling interval of 60 seconds, it appears in the VirtualCenter application as "Every Minute" but appears as its sampling interval, `/perf/0000000060`.

Because VirtualCenter perf collectors only have a sampling interval, they collect statistics on all virtual machines and all hosts. Because updating the contents of these VirtualCenter performance collectors for each sampling interval would generate much communication traffic; by default, the contents of VirtualCenter performance collectors do not keep current data.

If you want the contents of VirtualCenter perf collectors to be up-to-date, then change the default behavior by setting the `periodicPerfRefreshEnable` config variable to TRUE in the Web service configuration file (`vmaConfig.xml`). If this variable is set to TRUE, then the contents of all VirtualCenter perf collectors are also kept up-to-date.

## Filtered Perf Collectors

Filtered perf collectors are "children" derived from a VirtualCenter perf collector. Each VirtualCenter perf collector can have multiple "children" filtered perf collectors. However, a filtered per collector cannot be used to create additional "children" filtered perf collectors.

As discussed in the previous section, VirtualCenter perf collectors have only a sampling interval, and therefore, collect statistics on all virtual machines and all hosts. Since it is impractical to collect all this performance data, clients should create "filtered perf collectors" to monitor the current values of particular performance statistics. Unlike VirtualCenter perf collectors, the contents of filtered perf collectors are kept up-to-date.

Filtered perf collectors typically specify a source (host or virtual machine) and the performance statistics of interest. For example, a default VirtualCenter perf collector has a sampling interval of five minutes. By filtering this VirtualCenter perf collector, clients can create a filtered perf collector collecting performance statistics every five minutes on "host A", another that collects memory statistics every five minutes on "John's virtual machine", and so on.

Clients also create a filtered perf collector by using the Create operation. When doing so, specify the name and the filter (initial parameter comprising XML document of type PerfCollector). This XML document *must include* the handle to the parent VirtualCenter perf collector. The filter should also include the source (host or virtual machine being queried), the samples, and the performance statistics of interest. See Creating a Filtered Perf Collector on page 228.

**Note:** Do not specify the sampling interval. The filtered perf collector has the same sampling interval as its parent VirtualCenter perf collector. The WSDL stubs insert a value of zero (`0`) that is ignored by the Web service.

**Note:** If clients leave the spec field uninitialized (or NULL), then all performance statistics from the source are returned.

**Note:** Unlike VirtualCenter perf collectors, the user-friendly name selected for the filtered perf collector appears in the `/perf` directory, but does not appear in the VirtualCenter application.

## Comparing VirtualCenter and Filtered Perf Collectors

The following table summarizes the differences between VirtualCenter and filtered perf collectors.

| Characteristic | VirtualCenter Perf Collector | Filtered Perf Collector |
|---|---|---|
| What does it specify? | Specifies a unique sampling interval. Does not specify any other filter. | Specifies the source (of the performance statistics) and defines the performance statistics (CPU, memory, and so on).<br>Uses the same sampling interval as its "parent" VirtualCenter perf collector. |
| Can be created? | Yes. Use the Create operation and specify a name, sampling interval, and total number of samples. | Yes. Use the Create operation and specify a name, a parent VirtualCenter perf collector, and a filter. |
| Can be deleted? | Yes. Use the Delete operation. | Yes. Use the Delete operation. |
| Life expectancy. | Persists until deleted. | Persists until deleted or the Web service is stopped. When the Web service is restarted, filtered perf collectors no longer exist. |
| Filters an existing perf collector? | No. VirtualCenter perf collectors have only a name and a sampling interval. | Yes. Filtered perf collectors filter an existing VirtualCenter perf collector. |
| Gets current performance statistics? | No, unless the `periodicPerfRefreshEnable` config variable is set to TRUE. This is set to FALSE by default. | Yes. Filtered perf collectors can get current statistics through the GetContents and GetUpdates operations. |
| Used to create filtered perf collectors? | Yes. Clients can create filtered perf collectors from a VirtualCenter perf collector. | No. Clients cannot "filter" a filtered perf collector. |
| Query historical performance statistics? | Yes. Use the QueryPerfData and QueryPerfData2 operations. Specify the handle of the perf collector and optionally, a filter. | Yes. Use the QueryPerfData and QueryPerfData2 operations. Specify the handle of the perf collector. Do not specify a filter. |

## Performance Metric Data Model

A diagram of the Performance Metric data model is included in the *Virtual Infrastructure SDK Reference Guide*. Here, we provide a brief summary of this model.

A PerfCollector is a collector of performance statistics. Each Perf Collector includes a filter (specifying the sample interval and the performance statistics that are collected) and stats, referring to the actual performance statistics that are collected (a PerfCollection).

Each PerfCollection has an array of statistics from various sources, either hosts or virtual machines. Each PerfSource has an array of samples for that source. Each PerfSample has an array of statistics for the sample from that source. Each stat is a PerfStat, and can have a type specified by PerfStatType, that describes the type of performance statistic: cpu, net, disk, floppy, and so on.

The actual statistics are in the `data` field of the PerfStat, and comprise many different datatypes: CPUPerf, MemoryPerf, NetPerf, and so on.

## Creating a VirtualCenter Perf Collector

Clients use the Create operation to create a new VirtualCenter perf collector, as shown in the following sample.

```
{
    my ( $service, $handle, $name ) = @_;

    #
    # Create an ordered hash for the PerfCollector
    #

    my %perfSpec = ();
    tie %$perfSpec, "Tie::IxHash";
    tie %{$perfSpec->{ 'filter' }}, "Tie::IxHash";

    #
    # Accepting input for Perf Collector
    # NOTE: Not all specifications are included for input.
        # Refer documentation for details on more Perf Collector specs.
    #

    Print "\nInterval (in seconds) : ";
    chomp ( $perfSpec->{ 'filter' }->{ 'interval' } = <STDIN> );
    $perfSpec->{ 'filter' }->{ 'name' } = $name;
    Print "\nNumber of samples returned [optional] : ";
    chomp ( $perfSpec->{ 'filter' }->{ 'samples' } = <STDIN> );

    #
    # Setup and call Create
```

```
    #

    my $method = SOAP::Data->name('Create')
        ->attr({xmlns => 'urn:vma1'});

    my @params = ( SOAP::Data->name(handle => $handle),
                   SOAP::Data->name(name => $name),
                   SOAP::Data->name(type => 'PerfCollector'),
                   SOAP::Data->type('PerfCollector')->name(initial => $perfSpec));

    print "\nHandle for new object : ".
                $service->call($method => @params)->result;
    Print "\n\nOperation Successful \n";
    return;
}
```

## Creating a Filtered Perf Collector

Clients use the Create operation to create a new filtered perf collector, as shown in the following sample.

```
{
    my ($service, $handle, $name) = @_;

    #
    # Create an ordered hash for the PerfCollector
    #

    my %perfSpec = ();
    tie %$perfSpec, "Tie::IxHash";
    tie %{$perfSpec->{'filter'}}, "Tie::IxHash";

    #
    # Call ResolvePath to obtain handle to the default VC Perf Collector.
    #
    $method = SOAP::Data->name('ResolvePath')
                 ->attr({xmlns => 'urn:vma1'});
    @params = (SOAP::Data->name(path => "/perf/0000000300"));
    my $pcHandle = $service->call($method => @params)->result;


    #
    # Call ResolvePath to obtain handle for the host
    #
    Print "\nHost Path : ";
    my $host_path;
```

```
chomp($host_path = <STDIN>);

$method = SOAP::Data->name('ResolvePath')
                  ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->type('xsd:string')->name(path => $host_path ));
my $hostHandle = $service->call($method => @params)->result;

#
# Fills in filter spec to retrieve cpu0 info for host.
#
my %filterSpec = ();
tie %$filterSpec, "Tie::IxHash";
$filterSpec->{interval} = 0;
tie %{$filterSpec->{spec}}, "Tie::IxHash";
$filterSpec->{spec}->{key} = $hostHandle;
tie %{$filterSpec->{spec}->{sample}}, "Tie::IxHash";
$filterSpec->{spec}->{sample}->{key} = '';
$filterSpec->{spec}->{sample}->{type} = 'cpu';
$filterSpec->{spec}->{sample}->{device} = 0;

#
# Accepting input for Perf Collector
# NOTE: Not all specifications are included for input.
# Refer documentation for details on more Perf Collector specs.
#

$perfSpec->{'filter'}->{'spec'} = $filterSpec;
$perfSpec->{'filter'}->{'handle'} = $pcHandle;
Print "\nNumber of samples returned : ";
chomp ($perfSpec->{'filter'}->{'samples'} = <STDIN>);

#
# Setup and call Create
#

my $method = SOAP::Data->name('Create')
    ->attr({xmlns => 'urn:vma1'});

my @params = ( SOAP::Data->name(handle => $handle),
               SOAP::Data->name(name => $name),
               SOAP::Data->name(type => 'PerfCollector'),
              SOAP::Data->type('PerfCollector')->name(initial => $perfSpec));

print "\nHandle for new object : ".
            $service->call($method => @params)->result;
Print "\n\nOperation Successful \n";
return;
```

```
}
```

## Collecting Current Performance Data

Clients can access current performance statistics by initially doing a GetContents operation on the performance collector Container (specified by `/perf/<interval>`) and its sub-items.

This operation returns a perfCollector Container that contains two Items, the PerfFilter and PerfCollection (statistics) datatypes. Each Item contains a key, that is the handle to the Item.To get the latest statistics, clients should call the GetUpdates operation on the PerfCollection object that holds the statistics. The following sample illustrates this concept.

```
#
# Call ResolvePath to obtain the handle for the
# parent Perf collector /perf/0000000300
#

$method = SOAP::Data->name('ResolvePath')
      ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(path => "/perf/0000000300"));

my $parentHandle = $service->call($method => @params)->result;

#
# Call ResolvePath to obtain handle for the host for which
# the stats are needed
#

@params = (SOAP::Data->name(path => "/host/myhost.mydomain.com"));

my $hostHandle = $service->call($method => @params)->result;

#
# Create spec for filter
#

my %filterSpec = ();
tie %{$filterSpec->{filter}}, "Tie::IxHash";

$filterSpec->{filter}->{interval} = 0;
tie %{$filterSpec->{filter}->{spec}}, "Tie::IxHash";
$filterSpec->{filter}->{spec}->{key} = $hostHandle;
tie %{$filterSpec->{filter}->{spec}->{sample}}, "Tie::IxHash";
$filterSpec->{filter}->{spec}->{sample}->{key} = '';
$filterSpec->{filter}->{spec}->{sample}->{type} = 'cpu';
$filterSpec->{filter}->{spec}->{sample}->{device} = 0;
```

```
$filterSpec->{filter}->{handle} = $parentHandle;
$filterSpec->{filter}->{samples} = 2;

#
# Call ResolvePath to obtain a handle to /perf
#

$method = SOAP::Data->name('ResolvePath')
            ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(path => "/perf"));
my $perfHandle = $service->call($method => @params)->result;



#
# Setup and call Create
#

my $method = SOAP::Data->name('Create')
            ->attr({xmlns => 'urn:vma1'});

my @params = (
               SOAP::Data->name(handle => $perfHandle),
               SOAP::Data->name(name => 'newPerf'),
               SOAP::Data->name(type => 'PerfCollector'),
               SOAP::Data->type('PerfCollector')->name(initial => $filterSpec)
             );

my $handle = $service->call($method => @params)->result;

#
# Get the perf collector details using GetContents
#

$method = SOAP::Data->name('GetContents')
                  ->attr({xmlns => 'urn:vma1'});
@params = (SOAP::Data->name(handle => $handle));

my $perfContents = $service->call($method => @params);

#
# Now call GetUpdates on the stats object of this collector ...
#
```

The following sample code illustrates how to extract the appropriate statistic from the PerfCollection object:

```
sub PrintStats
{

    my ( $result, $match ) = @_;

    #
    # Define the path for the outermost object of interest
    #

    if (!defined $match ) {
            $match = "//returnval/*";
    }

    #
    # Iterate through each element in the current object
    # and check to see if it contains a text value or a
    # child object
    #

    my $i =0;
    foreach my $element ($result->dataof( $match )) {
        $i++;

        #
        # Extract key value and display object handle
        #

        if ( $element->name eq 'key' ) {
                print  "\n\nSamples for ".$element->value ." :\n";
                next;
        }

        #
        # Check the type of stat and display if type corresponds to CPU
        #

        if ( $element->name eq 'stat') {
            if ($element->value->{type} =~ /cpu/ ) {
               print "\nKey : ".$element->value->{key};
               if ( exists $element->value->{data}->{used} ) {
                   print "\nUsed : ".$element->value->{data}->{used}." milliseconds\n";
               }
            }
```

```
            next;
        }

        #
        # Element contains child object. Adjust path for the
        # new object and recursively call subroutine to parse
        # the child object
        #

        if ( ref $element->value ) {
            $newMatch = $match;
            chop ( $newMatch );
            $newMatch = $newMatch."[$i]/*";
            PrintStats ( $result, $newMatch );
        }

    }
}
```

## Collecting Historical Data

Similarly, clients can use the QueryPerfData and QueryPerfData2 operations on both VirtualCenter and filtered performance collectors to obtain historical performance data, for any specified time period. The time period may be completely in the past, or it may be specified to extend up to, and including, the most recent update.

When using the QueryPerfData and QueryPerfData2 operations, clients must specify the handle to the perf collector. Specify the filter parameter only when querying a VirtualCenter perf collector.

**Note:** Do not use the filter parameter only when querying a filtered perf collector (as it is already "filtered"). If the filter parameter is specified for a filtered perf collector, the operation returns an error.

**Note:** If you see output similar to `/vpx/vm/#000512` or `/vpx/host/#000a376` for the PerfSourceType "key" field in the PerfCollection object, then the specified virtual machine or host has been deleted.

The following sample illustrates how a client calls the QueryPerfData operation.

```
#
# Call ResolvePath to obtain the handle for the
# default Perf collector /perf/0000000300
#

$method = SOAP::Data->name('ResolvePath')
      ->attr({xmlns => 'urn:vma1'});
```

```
@params = (SOAP::Data->name(path => "/perf/0000000300"));

my $handle = $service->call($method => @params)->result;

#
# Call ResolvePath to obtain handle for the host for which
# the stats are needed
#

@params = (SOAP::Data->name(path => "/host/myhost.mydomain.com"));

my $hostHandle = $service->call($method => @params)->result;

#
# Create spec for filter
#

my %filterSpec = ();
tie %$filterSpec, "Tie::IxHash";


$filterSpec->{interval} = 0;
tie %{$filterSpec->{spec}}, "Tie::IxHash";
$filterSpec->{spec}->{key} = $hostHandle;
tie %{$filterSpec->{spec}->{sample}}, "Tie::IxHash";
$filterSpec->{spec}->{sample}->{key} = '';
$filterSpec->{spec}->{sample}->{type} = 'cpu';
$filterSpec->{spec}->{sample}->{device} = 0;

#
# Format local time to obtain start time for the samples
#

my ($sec,$min,$hour,$mday,$mon,$year) = localtime(time);

my $startTime = sprintf "%4d-%02d-%02dT%02d:%02d:%02d",
            $year+1900,$mon+1,$mday,$hour,$min,$sec;


#
# Setup and call QueryPerfData
#

$method = SOAP::Data->name('QueryPerfData')
      ->attr({xmlns => 'urn:vma1'});

@params = (
```

```
            SOAP::Data->name(handle => $handle),
            SOAP::Data->(name(startTime => $startTime),
            SOAP::Data->name(samples => 1),
            SOAP::Data->type('PerfFilter')->name(filter => $filterSpec)
           );

my $result = $service->call($method => @params);

PrintStats ( $result );
```

Clients call the QueryPerfData2 operation exactly as they call the QueryPerfData operation. However, the QueryPerfData2 operation returns the new CPUPerf2, MemoryPerf2, and VirtualMachineMemoryPerf2 datatypes whereas the QueryPerfData operation returns the CPUPerf, MemoryPerf, and VirtualMachineMemoryPerf datatypes. These new datatypes contain additional statistics for ESX Server hosts and virtual machines on ESX Server.

Clients can only obtain the statistics from the CPUPerf2, MemoryPerf2, and VirtualMachineMemoryPerf2 datatypes through the QueryPerfData2 operation. If you are already calling the GetUpdates operation on the PerfCollection object, then be sure to call the QueryPerfData2 operation to obtain these extra statistics.

This next sample shows how to display the statistics from CPUPerf2, returned after calling the QueryPerfData2 operation.

```
sub PrintStats
{
     my $result = shift;

     #
     # Display sample start time
     #

     my $startTime = $result->valueof('//startTime');
     print "\n\nSample start time : $startTime" ;

     my @stats = $result->valueof('//stat');

     #
     # Iterate through each CPU stat object in the current sample and
     # print the key and used time
     #

     print "\n\nCPU stats for the host :\n";

     foreach my $stat ( @stats ) {
```

```
            print "\nKey: ".$stat->{key};
            if ( exists $stat->{data}->{used} ) {
                 print "\nUsed time : ".$stat->{data}->{used}."
                      milliseconds\n";
            }
            if ( exists $stat->{data}->{system} ) {
                 print "\nSystem time : ".$stat->{data}->{system}."
                      milliseconds\n";
            }
            if ( exists $stat->{data}->{ready} ) {
                 print "\nReady time : ".$stat->{data}->{ready}."
                      milliseconds\n";
            }
            if ( exists $stat->{data}->{wait} ) {
                 print "\nWait time : ".$stat->{data}->{wait}."
                      milliseconds\n";
            }
            if ( exists $stat->{data}->{pcpu} ) {
                 print "\nPCPU : ".$stat->{data}->{pcpu};
            }

       }
}
```

# Changing Permissions

Clients can view and change permissions on any object by using the VMware SDK. To view the current permissions, clients either call the GetInfo operation on the object or call the GetContents operation on the parent Container of the object. Each Item that is returned in the Container's Item list contains the permissions associated with that Item.

The permissions are returned in an array of Permission objects. Each permission object has two String fields: the key field (identifies the user with the assigned permissions) and the rights field (indicates the permissions granted to the user).

Once the current permissions are known, clients can modify these permissions by using the ChangePermissions operation. The ChangePermissions operation takes two arguments: the vHandle (versioned identifier of the object) and a PermissionList (encapsulates the new Permissions array).

By using the ChangePermissions operation, clients can add new permissions, or delete or modify existing permissions. To add new permissions, clients should add a new Permission entry to the Permissions array, then send the new Permissions array to the ChangePermissions operation.

Similarly, clients can delete permissions by removing the desired entries from the Permissions array.

Clients can modify permissions by modifying the permission object in the Permissions array, then sending the updated Permissions array to the ChangePermissions operation.

**Note:** The new PermissionList that is specified in the ChangePermissions operation replaces all the existing permissions on the object. If you don't want to delete any existing entries, then the client must also send back all the existing entries to the ChangePermissions operation.

The following code sample illustrates how to invoke the ChangePermissions operation.

```
#
# Setup & call the ResolvePath to obtain handle of the object
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->type('xsd:string')->name(path => $path));

$handle = $service->call($method => @params)->result;

#
# Call GetInfo to obtain the list of current permissions on the object
#
```

```
$method = SOAP::Data->name('GetInfo')
                ->attr({xmlns => 'urn:vma1'});
@params = ( SOAP::Data->name( handle => $handle));

my $infoObj = $service->call($method => @params);

@perms = $infoObj->valueof('//perm');

... edit permissions here ( add / delete / modify perms array )

#
# Setup and Call ChangePermissions
#

$permList = SOAP::Data->name(perm => @perms);
$soapPermList->{ 'permissions' } = $permList;

$method = SOAP::Data->name('ChangePermissions')
                ->attr({xmlns => 'urn:vma1'});

@params = ( SOAP::Data->name( vHandle => $result->{ 'vHandle' }),
                              SOAP::Data
                              ->type('PermissionList')
                              ->name( permissions => $soapPermList )
          );

$service->call($method => @params);
```

# Taking a Snapshot of a Virtual Machine

The following code sample illustrates using the new snapshot operations. The first, SnapshotVM, takes a "snapshot" (picture) of a virtual machine at a particular point in time. The second, RevertVM, discards any existing snapshot and returns the virtual machine to its state preceding the snapshot. The third, ConsolidateVM, commits the changes contained in the snapshot, to the base virtual disk(s).

```
#
# Setup & call the ResolvePath method
#

$method = SOAP::Data->name('ResolvePath')
    ->attr({xmlns => 'urn:vma1'});

@params = (SOAP::Data->name(path => $vmPath));

my $handle = $service->call($method => @params)->result;


#
# Depending on choice of snapshot operation, call the appropriate
# method
#

if ( $op eq 'snap' ){
      $method = SOAP::Data->name('SnapshotVM')
            ->attr({xmlns => 'urn:vma1'});
} elsif ( $op eq 'revert' ){
      $method = SOAP::Data->name('RevertVM')
            ->attr({xmlns => 'urn:vma1'});
} else {
      $method = SOAP::Data->name('ConsolidateVM')
            ->attr({xmlns => 'urn:vma1'});
}


@params = (SOAP::Data->name(vm => $handle));

my $task = $service->call($method => @params)->result;

MonitorTask ( $service, $task );
```

CHAPTER 9

# Sample Applications

This chapter describes the sample applications created with the VMware SDK. It also discusses the proxy layer architecture used by two of the samples.

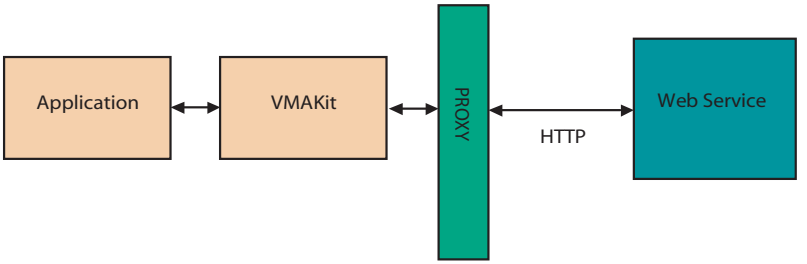
These sample applications, their source code, and installation instructions are contained in the VMware SDK package. Read `SDK\SDK-README.html` for links to these sample applications.

In addition, read `SDK\WebService\samples\README`. This file contains detailed information about each of the sample applications, and supplements the information contained in this *Virtual Infrastructure SDK Programming Guide*.

The Web-based monitoring and management application and the inventory and virtual machine provisioning applications are Java applications, while the Alerts and VMA Viewer applications are C# applications developed with Microsoft Visual Studio® .NET. The SimpleListing, VMPowerOps, and TestOps applications are Visual Basic (VB) applications, also developed with Visual Studio .NET.

# Proxy Layer Abstraction

The Web application (Web-based Monitoring and Management Application on page 246) and the inventory application (Inventory and Virtual Machine Provisioning Application on page 250) use a proxy layer architecture that abstracts the Web service layer.




## VMAKit Public Interface

This VMAKit class abstracts and encapsulates all access to the datatypes and operations exposed by the VMware SDK. It acts as a client-side business abstraction that shields the presentation-tier clients from possible volatility in the implementation of the underlying service API, thereby potentially reducing the number of changes that must be made to the presentation-tier client code, when the underlying service API implementation changes.

It's primary purpose is to provide the presentation-tier clients with data aggregated and computed from several different sources, using the VmaProxy APIs. Clients only need to focus on presenting the information provided by the VMAKit, instead of data aggregation and computation (done by the VMAProxy object). That is, clients are unaware of the underlying VMware SDK; they are only aware of the VMAProxy objects.

## VmaProxy Object

The VmaProxy object abstracts the underlying data source access implementation to enable transparent access to the data source (Web service). This transparency allows the VmaProxy to migrate to different implementations, without affecting the clients or the business components.

The VMware SDK describes a virtual machine through the various datatypes and operations in the Web service. The proxy layer takes all this information and creates a VmaProxy object for the virtual machine. Applications can then directly access this VmaProxy object without interfacing with the underlying datatypes and operations. The underlying datatypes and operations in the SDK are transparent to the application.

The VmaProxy maintains a data cache (data is cached upon request) that is kept up-to-date by the update-listening mechanism provided by the VMware SDK. The caching mechanism is used to

reduce the network overhead that can result due to marshalling and unmarshalling of large sized data and to eliminate the complexities involved in caching the data for business components (see Threading Model Used in the Reference Application Service (Proxy) Layer on page 244). Business objects can also delegate edit, virtual computing, or power operations to the VmaProxy object by using the simpler APIs exposed by it.

**Threading Model Used in the Reference Application Service (Proxy) Layer**

The proxy layer creates a thread solely for listening for updates from the Web service. This GetUpdates thread is created as part of the login operation. This thread blocks until the client application registers objects of interest to the client by using the registerObservable( ) API. (The client wants to receive update notifications on these registered objects).

The other thread running in the proxy layer is the client application thread, which invokes the various operations on the Web service such as StartVM, PutUpdates, and so on. Once an item is available in the proxy's object table, the GetUpdates thread invokes the GetUpdates operation in blocking mode, waiting for updates from the Web service.

As new objects are registered with the proxy, they are added into the proxy's object table as objects on which the proxy should listen for updates. However, if the GetUpdates thread is currently blocked performing a GetUpdates operation, then the newly registered objects won't receive updates until the current outstanding GetUpdates request returns.

In order to get around this blocking, CancelGetUpdates is called when a new object is added. This CancelGetUpdates operation immediately unblocks the pending GetUpdates operation. The GetUpdates thread now repeats the blocking operation and listens for updates on all the registered objects in the proxy.

The CancelGetUpdates operation is also called when the application exits, in order to shut down the blocked GetUpdates thread cleanly.

As updates are received, they are applied to the client-side objects. However, the changes need to be synchronized with the PutUpdates (client) thread in the event the client is also making changes to the same object at the same time. The algorithm used to accomplish this is as follows:

GetUpdates Thread logic:

```
Lock (Proxy Object table);
VHandleList =  all vHandles in proxy object table that are currently NOT being
edited by the client
Unlock(proxy object table)

Call getUpdates(Blocking mode);
Lock (proxy object table)
For each object for which an update was received:  {
       If (object is currently being modified by the client thread) {
```

```
                // Ignore this update from VMA
        } else {
                apply update to object
        }
}
unlock (proxy object table);
```

PutUpdates Thread Logic:

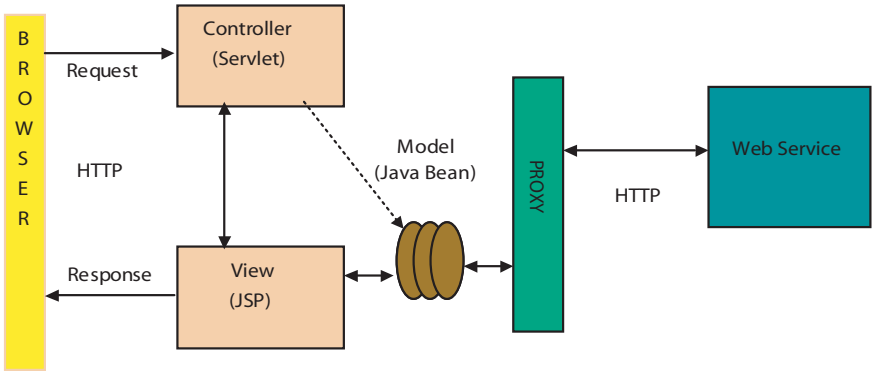
```
// Client has identified object OBJ as an object on which to perform putUpdates.
Lock (proxy object table);
OBJ.editing = true; // Mark OBJ as currently being edited by client.
Unlock (proxy object table);

Create Change Req list for putUpdates call.
PutUpdates(); // non versioned put updates
GetUpdates(NON BLOCKING on OBJ only);

Lock (proxy object table);
ApplyUpdates on OBJ
OBJ.editing = false;
Unlock (proxy object table);
```

# Web-based Monitoring and Management Application

This section describes the architecture of the monitoring and management application, followed by a brief description of the user interface.

## Web Application Architecture

This web-based monitoring and management application uses the proxy layer architecture (Proxy Layer Abstraction on page 243) that abstracts the Web service layer then presents it to the client through JavaBeans. In this section, we describe the proxy layer and its use by the Web application, which employs the standard Model-View-Control (MVC) design.




The architecture used in the Web application is an open framework that combines the use of JavaServer Pages (JSP), servlets, JavaBeans, and the Web service proxy technologies. It takes advantage of the predominant strengths of these technologies, by using JSP to generate the presentation layer, servlets to perform process-intensive tasks, and JavaBeans to define logic.

The servlet acts as the controller and is in charge of the request processing and the creation of any beans or objects that are used by JSP. Depending on the user's actions, the servlet also decides which JSP page is forwarded the request.

There is no processing logic within the JSP page itself. The JSP page is simply responsible for retrieving any objects or beans that may have been previously created by the servlet, and for extracting the dynamic content from that servlet for insertion within static templates.

The proxy layer abstracts the Web service layer. It is responsible for interfacing with the Web service and obtaining information from the Web service. The proxy layer takes this information and

presents it to client applications through JavaBeans. The Web service is transparent to the client applications, which sees only the proxy layer.

This architectural approach typically results in the cleanest separation of the presentation layer from content, leading to a clear delineation of the roles and responsibilities of the developers and page designers on your programming team.

### Controller

In the Web reference application, all URL requests go through the servlet, which is known as the Controller.

Following the Model View Control (MVC) methodology, this servlet does the following:

1. The Controller accepts HTTP requests for JSP pages.
2. The Controller next decides the eventHandler that should be called for each request,
3. The Controller then hands the request over to the eventHandler for processing,
4. If no error has occurred, the Controller lets the eventHandler forward the request to the corresponding JSP page.

The logic behind a Controller servlet is a need for basic control over the application flow, while sparing the JSP pages from containing the following:

- Information about which JSP pages require the submission of forms
- Java code that creates the objects needed for rendering the JSP page

Upon initialization (when the first request goes through this servlet), the Controller servlet reads a properties file that contains a map from event names to eventHandler class definitions. The Controller then uses reflection to create the eventHandler's objects, and stores these in a private hashMap.

Going forward, the Controller looks into every request that goes through it, trying to find the event parameter and attempting to retrieve the corresponding eventHandler from the hashMap. The Controller then calls the eventHandler's process method, and gives it the request.

EventHandlers typically create the needed objects for the JSP target pages, and notifies the Controller if anything goes wrong, through exceptions.

If the eventHandler's processing throws an exception, then the Controller switches to the "error" eventHandler. Once the processing has finished, the Controller calls upon the eventHandler's forward method, letting the eventHandler's logic decide which JSP page to target.

**Note:** The classes mapped to event names must be made available in `/webapps/[app name]/WEB-INF/classes/...classpath` in order for the corresponding eventsHandler objects to be created.

**Note:** The Web container is instructed to route requests through the Controller servlet. You can find these instructions in the Web container's standard `WEB-INF` directory and in the standard `web.xml` file.

**Additional Resources**

Refer to the following Web sites for additional information about JavaBeans, JavaServer Pages, and Java servlets.

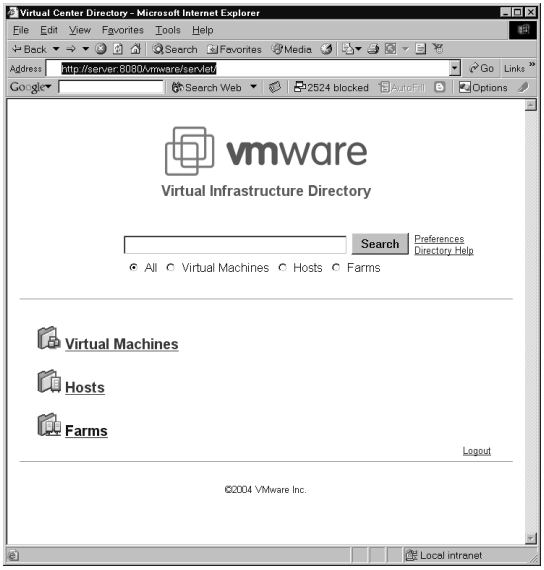
- JavaBeans — *java.sun.com/products/javabeans/*
- JSP — *java.sun.com/products/jsp*
- Servlet — *java.sun.com/products/servlet/index.jsp*

## Using the Web-based Monitoring and Management Application

This Web-based monitoring and management application is a Java application that allows you to create, perform power operations, edit, and clone a virtual machine through a browser interface. Open the browser and log into the application by entering your user name, password, and Web service URL.

In this guide, we provide a brief description of the application. Read `SDK/SDK-README.html` for complete information on installing and deploying this application.

When shutting down this application, be sure to shut down the Tomcat server as well in order to stop this sample application completely.



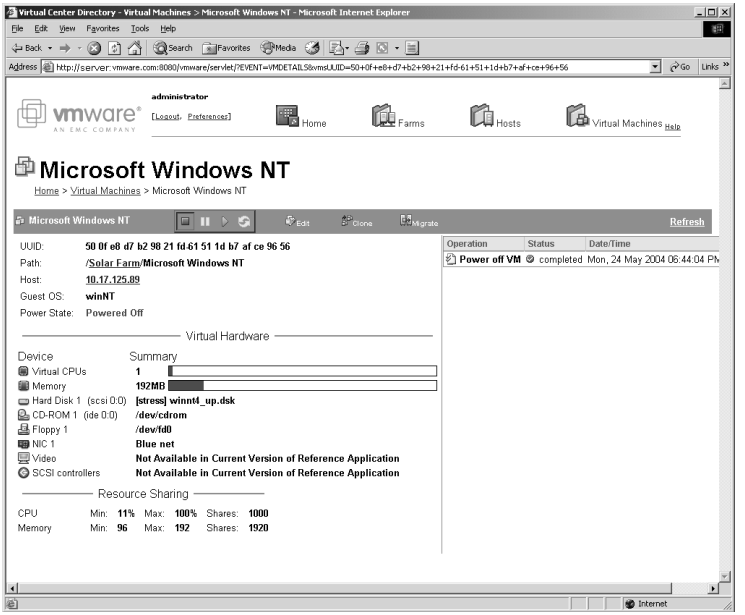
The Overview page appears with the following links.

- Virtual Machines — Clicking on this link displays a page listing all the virtual machines in VirtualCenter.
- Hosts — Clicking on this link displays a page listing all the hosts in VirtualCenter.
- Farms — Clicking on this link displays a page listing all the Farms in VirtualCenter.

In addition to providing an inventory listing, the Web-based monitoring and management application has the following features:

- Search for virtual machines, hosts, or Farms.
- Provides details about each virtual machine and host.
- Perform power operations on virtual machines, and edit the CPU, networking, memory, and so on.
- Create, clone, or migrate a virtual machine.

The following figure shows the details page for a virtual machine.



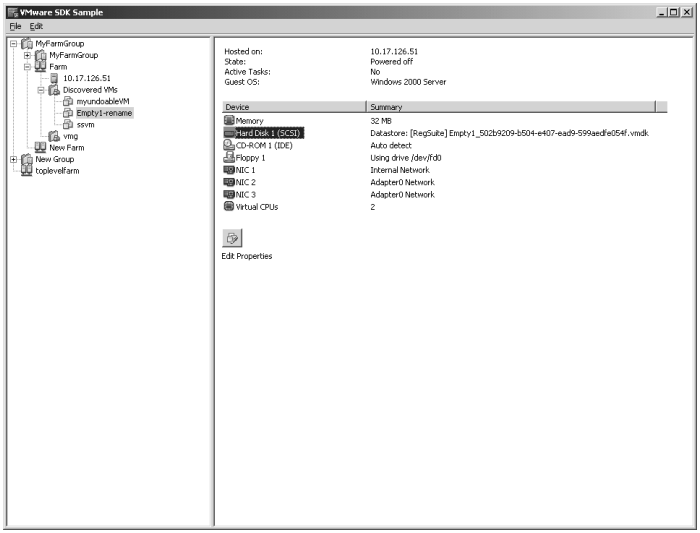
# Inventory and Virtual Machine Provisioning Application

An inventory and virtual machine provisioning GUI (VMware SDK Sample) application, built using the Standard Widget Toolkit (SWT) classes from Eclipse, is included in the samples directory. This Java application is also built on top of the proxy and kit layers (Proxy Layer Abstraction on page 243) that are provided with the reference Web application (Web-based Monitoring and Management Application on page 246).

This application includes the following features:

- Presents the `/vcenter` directory as a tree diagram.
- Provides basic information about each object.
- Includes a virtual machine creation wizard.
- Allows users to provision and customize a virtual machine by changing some of its properties.

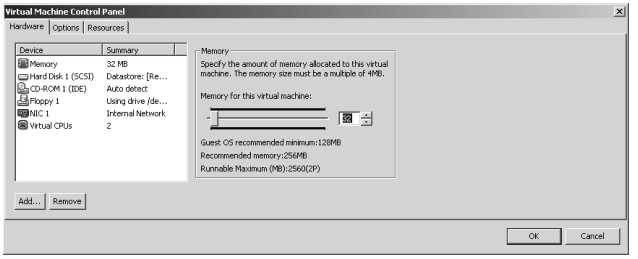
The VMware SDK Sample application opens in a two-pane window. The left pane presents a hierarchical diagram of the `/vcenter` directory. The right pane (object information screen) displays the details about the object selected in the left pane.



Users can create a new virtual machine by choosing **File > Create a new VM** or pressing **Ctrl-C** as a shortcut. Users specify the destination location, the guest operating system, the virtual machine name, the networking, and the virtual disk size.

After obtaining this information, the virtual machine creation wizard invokes the "create virtual machine" operation on the kit layer. The kit layer creates the VirtualMachineSpec using the information gathered by the virtual machine creation wizard, then invokes the Create operation. Once the virtual machine has been created, a virtual disk is added by using the settings specified by the user in the virtual machine creation wizard.

Clients can configure virtual machines by using the **Edit Properties** button on the right pane (virtual machine information screen). The Virtual Machine Control Panel window appears that lists the various devices currently configured for the virtual machine.




Users can modify the virtual machine's memory, networking, and virtual disk information through the **Resources** tab. (The kit layer modifies this information by using the PutUpdates operation.) Users can also add new devices or remove existing devices in the **Hardware** tab. Similarly, users can rename the virtual machine through the **Options** tab.

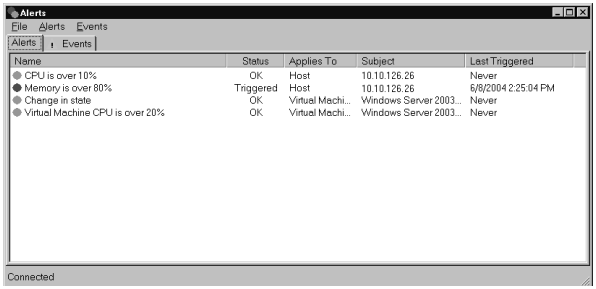
# Alerts Application

The alerts application is a C# application developed with Microsoft Visual Studio .NET. It demonstrates how the client can listen asynchronously for notifications from the Web service server. It allows the user to define simple alerts, that produces an event whenever the condition of the alert is satisfied.

For example, the alert condition can be "CPU resource of host XXX is more than NN percent for longer than MM minutes". Another example is "Virtual machine XXX is powered off" or "Something in the state of virtual machine XXX changed".

In this sample application, the generated alert event is simply recorded in a list of events. You can imagine that a more sophisticated version of this application would allow the user to configure an action associated with each alert event, including sending an email or paging someone, running a script, and so on

**Note:** The alert events described here are not related to the event objects in the VMware SDK event data model. The alert events, that are part of this alerts application, only exist as part of this application.




The alerts application window comprises two tabs: an Alerts tab and an Events tab. The Alerts tab lists the alerts that the user has defined. The Events tab displays the events that have been generated.

If an alert is triggered, it generates an event when it is triggered. The icon of the alert changes from green to red and the icon remains red as long as the alert condition alert is satisfied.

## Building and Running the Alerts Application

The easiest way to build and run the alerts application is to open the `alerts.sln` file in Microsoft Visual Studio .NET and choose **Debug > Start** (or press **F5**).

## Alerts Application Source Files

Here is a brief description of some of the more interesting source files:

- `vmaService_proxy.cs` — This file takes function calls and packages them into SOAP messages that are sent out over the network in HTTP format. It is automatically generated by running the `wsdlProxyGen.exe` tool and using the `vma.wsdl` file as input. See VMware SDK Applications Developed with .NET on page 271.

  This tool automatically chooses `vma` as the namespace for the classes that it generates. For the sake of clarity, we have manually enclosed the `vma` namespace inside the `VMware` namespace, by adding `namespace VMware { ... }` around everything. This is an optional step and it is not necessary for the code (generated by the `wsdlProxyGen.exe` tool) to work.

- `VmaClient.cs` -— This file is a very simple wrapper for `vmaService_proxy.cs`. It provides the Connect and Disconnect methods. The `VmaClient.cs` file is useful because other objects can have a reference to this file instead of a direct reference to the `vmaService` object, in case it is necessary to destroy the `vmaService` object.

- `CertPolicy.cs` — Implements an instance of ICertificatePolicy, which gives us control of how we want to handle bad server-side certificates.

- `ChangeListener.cs` — A simple way to listen and respond to notifications from the Web service server. The ChangeListener class maintains a hash table of handle-->object entries. It queues a request for GetUpdates for all handles in the hash table. Whenever the request completes, the ChangeListener code uses the handle to refetch the object for which a change occurred. A more sophisticated implementation could use the description of the change and apply it to the in-memory object, rather than refetching the whole object.

  The ChangeListener uses the fact that the proxy class contains an asynchronous version for each method. The GetUpdates method can be invoked asynchronously by using the BeginGetUpdates/EndGetUpdates pairs of methods.

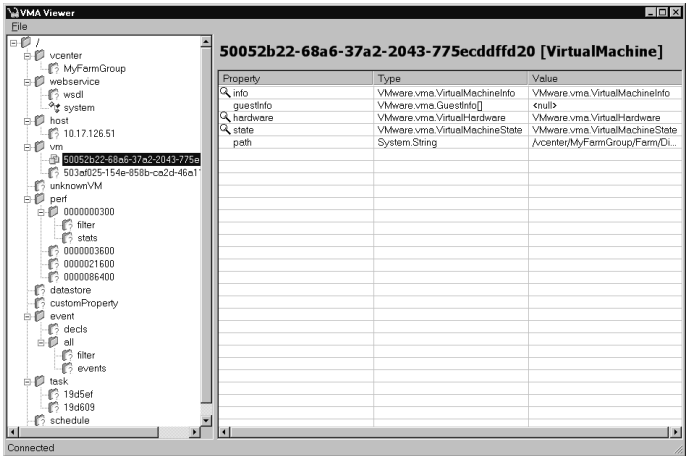
  The BeginGetUpdates method is used to queue the request. The client code provides a completion callback function (the .NET term is "delegate") that is invoked when a response has been received, or in this case, GetUpdatesDone. The completion callback then calls EndGetUpdates to finish processing the response and then immediately queues a new request by calling BeginGetUpdates again.

  By passing `true` to the second argument of BeginGetUpdates, we make the GetUpdates operation "wait" until there are changes. Effectively, this means that GetUpdatesDone is not called until there are updates to be reported. By using the approach of asynchronous method invocation together with making the GetUpdates method "wait", we have effectively made the Web service server push data to the client, instead of having the client poll for the data.

# VMA Viewer Application

The VMA Viewer is a C# application developed with Microsoft Visual Studio .NET. It allows the user to browse the namespace hierarchy exposed by the VMware VirtualCenter Web Service. It also allows the user to view the data structures at each node in the hierarchy as well as invoke some of the operations of the VMware SDK.

Although VMA Viewer is provided as a sample application, it is also useful as a way to familiarize yourself with the SDK.




The user interface window comprises a two-pane window; a namespace tree (left pane) and a view (right pane). Each time the user selects a node in the namespace tree, the contents of that node are refetched and displayed in the view pane on the right. Data structures are visualized in a generic way as a table of fields and values. Non-scalar fields are shown with a magnifying glass icon, which indicates that users can drill down by clicking on them. Grayed-out fields indicate that the data for these fields are not supplied.

Currently, the data is not automatically refreshed from the Web service server. The user must click or otherwise select a node in the namespace tree to refresh the contents of that node.

## Building the VMA Viewer Application

The easiest way to build and run the VMA Viewer application is to open the `vmaviewer.sln` file in Microsoft Visual Studio .NET and choose **Debug > Start** (or press **F5**).

## VMA Viewer Application Source Files

Here is a brief description of some of the more interesting source files:

- `vmaService_proxy.cs`, `VmaClient.cs`, and `CertPolicy.cs` — These are the same as the similarly named files of the Alerts application. See Alerts Application Source Files on page 253.
- `View*.cs` — This set of files implements the various types of right view panes.
- `PropCascade.cs` and `PropGrid.cs` - These files are used to implement the generic data structure viewer with drill-down ability.

# PerfMon Application (C#)

The PerfMon application comprises four sample applications, each demonstrating an implementation of a different aspect of monitoring performance data. These four samples are wrapped by the command-line application PerfMon. For more information on performance monitoring, see Collecting Performance Data on page 119 (Java samples) and Collecting Performance Data on page 224 (Perl samples).

The four sample applications, and some of the more interesting source file, are described below:

- `vmaService_proxy.cs` — This is the same as the similarly named file of the Alerts application. See Alerts Application Source Files on page 253.
- `VmaClient.cs` — This file is a wrapper console application for the performance monitoring samples.
- `PerfMonitor.cs` — This file demonstrates using an existing VirtualCenter perf collector with the GetContents and GetUpdates operations, to monitor performance statistics. Because the VirtualCenter perf collector statistics are disabled by default, you must set the `periodicPerfRefreshEnable` variable in the `vmaConfig.xml` file to TRUE for this sample to work properly.

  Run this sample by typing:

  ```
  PerfMon Basic <URL> <username> <password>
  ```
- `FilteredPerfMonitor.cs` — This file demonstrates creating a filtered perf collector from its parent 5-minute VirtualCenter perf collector, then using the GetContents and GetUpdates operations to monitor performance statistics. This sample also demonstrates how to use the QueryPerfData2 operation to get historical performance data, following a successful GetUpdates operation. This sample demonstrates obtaining CPU statistics for the two sources specified in the argument list (such as two virtual machines or two hosts).

  Run this sample by typing:

  ```
  PerfMon Filtered <URL> <username> <password> <src1path> <src2path>
  ```
- `QueryPerfCollector.cs` — This file demonstrates how to retrieve historical statistics after creating a new VirtualCenter perf collector.

  Run this sample by typing:

  ```
  PerfMon QueryPerfData <URL> <username> <password> <name for
  new collector> <hostPath> <startDate for sample collection>
  ```
- `QueryPerfData2Sample.cs` — This file demonstrates how to retrieve historical statistics after creating a new VirtualCenter perf collector with extended statistics for CPU, memory, and network usage, by using the QueryPerfData2 operation.

Run this sample by typing:

```
PerfMon QueryPerfData2 <URL> <username> <password> <name for new
collector> <hostPath> <startDate for sample collection>
```

- `CertPolicy.cs` — Implements an instance of ICertificatePolicy, which gives us control of how we want to handle bad server-side certificates.

# WS-I Test Application

The WS-I test application is a command-line application that demonstrates the use of several SDK methods such as Create, Delete, Rename, GetContents, GetInfo, and so on.

## WS-I Test Application Source Files

The TestOps application's source files are located in the `WebService\samples\vb\TestOps` directory. Here is a brief description of the files:

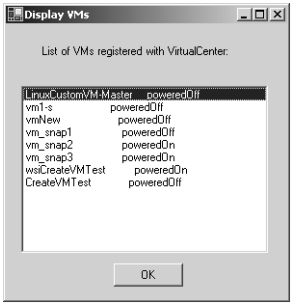
- `vmaService_proxy.cs` and `CertPolicy.cs` — These are the same as the similarly named files of the Alerts application. See Alerts Application Source Files on page 253.
- `VmaClient.cs` -— This file is a very simple test that invokes various operations such as Login, Logout, GetContents, GetInfo, Create a virtual machine, Clone a virtual machine, Rename a virtual machine, virtual machine power operations, and so on. By testing all these operations, this file is a fairly comprehensive test of the VMware VirtualCenter Web Service.

## Running the WS-I Test Application

You can compile this test from within Visual Studio. It needs three input parameters: the URL of the Web service, the user name, and the password. This test assumes that there is one host connected in VirtualCenter.

# SimpleListing Application

SimpleListing is a Visual Basic application developed with Microsoft Visual Studio .NET. It displays a listing of all the virtual machines connected to VirtualCenter, along with their current states.




## Building and Running the SimpleListing Application

The easiest way to build and run the SimpleListing application is to open the `SimpleListing.sln` file in Microsoft Visual Studio .NET 2003 and choose **Debug** > **Start** from the main menu (or press **F5**).

## SimpleListing Application Source Files

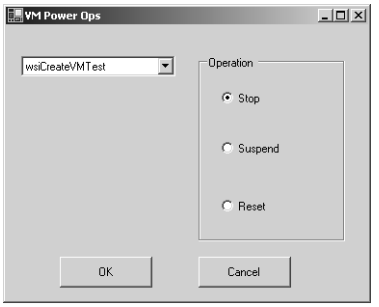
The SimpleListing application's source files are located in the `WebService\samples\vb\SimpleListing` directory. Here is a brief description of some of these files:

- `Reference.vb` — This file contains the VB stubs generated from `vma.wsdl`. It takes function calls and packages them into SOAP messages that are sent out over the network in HTTP format. It is automatically generated by running the `wsdlProxyGen.exe` tool (included in the VMware SDK package) and using the `vma.wsdl` file as input.
- `VmaClient.vb` — This file is a very simple wrapper for the `Reference.vb` file. It provides the Connect and Disconnect methods. The `VmaClient.vb` file is very useful because other objects can have a reference to this file, instead of a direct reference to the `Reference.vb` file.
- `CertPolicy.vb` — This file implements an instance of ICertificatePolicy, which gives us control of how we want to handle bad server-side certificates.
- `Main.vb` — This file comprises the Main class that displays the forms and makes API calls to retrieve information about connected virtual machines.
- `Start.vb` — This file represents the starting point for the SimpleListing application.

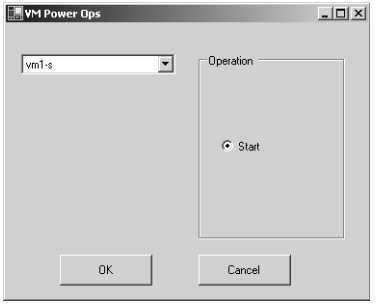
- `frmLogin.vb` — This file comprises a form that displays a login dialog box to log onto the Web service.
- `frmDisplay.vb` — This file comprises a form that displays the listing of virtual machines and their current states.

# VMPowerOps Application

The VMPowerOps application enables the user to start, stop, suspend, or reset any virtual machine connected to VirtualCenter. The user's choices depends on the current state of the virtual machine. Users can stop, suspend, or reset a powered-on virtual machine.




However, users can only power on a powered-off virtual machine, or resume a suspended virtual machine.




## Building and Running the VMPowerOps Application

The easiest way to build and run the VMPowerOps application is to open the `VMPowerOps.sln` file in Microsoft Visual Studio .NET 2003 and choose **Debug > Start** from the main menu (or press **F5**).

## VMPowerOps Application Source Files

The SimpleListing application's source files are located in the `WebService\samples\vb\VMPowerOps` directory. Here is a brief description of some of these files:

- `Reference.vb`, `VmaClient.vb`, `CertPolicy.vb`, `Main.vb`, `Start.vb`, `frmLogin.vb` — These are the same as the similarly named files of the SimpleListing application. See SimpleListing Application Source Files on page 259.
- `frmVMPowerOps` — This file comprises a form that displays a choice of power operations for the selected `virtual machine`. It makes a call to the respective SDK method, depending on the chosen power operation. `frmVMPowerOps` also includes a subroutine that monitors the task returned by the method, for either a successful completion or a failure.

# PerfMon Application (Visual Basic)

The PerfMon application comprises four sample applications, each demonstrating an implementation of a different aspect of monitoring performance data. These four samples are wrapped by the command-line application PerfMon. For more information on performance monitoring, see Collecting Performance Data on page 119 (Java samples) and Collecting Performance Data on page 224 (Perl samples).

The four sample applications, and some of the more interesting source file, are described below:

- `Reference.vb`, `CertPolicy.vb` — These are the same as the similarly named files of the SimpleListing application. See SimpleListing Application Source Files on page 259.
- `VmaClient.vb` — This file is a wrapper console application for the performance monitoring samples.
- `PerfMonitor.vb` — This file demonstrates using an existing VirtualCenter perf collector with the GetContents and GetUpdates operations, to monitor performance statistics. Because the VirtualCenter perf collector statistics are disabled by default, you must set the `periodicPerfRefreshEnable` variable in the `vmaConfig.xml` file to TRUE for this sample to work properly.

  Run this sample by typing:

  ```
  PerfMon Basic <URL> <username> <password>
  ```
- `FilteredPerfMonitor.vb` — This file demonstrates creating a filtered perf collector from its parent 5-minute VirtualCenter perf collector, then using the GetContents and GetUpdates operations to monitor performance statistics. This sample also demonstrates how to use the QueryPerfData2 operation to get historical performance data, following a successful GetUpdates operation. This sample demonstrates obtaining CPU statistics for the two sources specified in the argument list (such as two virtual machines or two hosts).

  Run this sample by typing:

  ```
  PerfMon Filtered <URL> <username> <password> <src1path> <src2path>
  ```
- `QueryPerfCollector.vb` — This file demonstrates how to retrieve historical statistics after creating a new VirtualCenter perf collector.

  Run this sample by typing:

  ```
  PerfMon QueryPerfData <URL> <username> <password> <name for
  new collector> <hostPath> <startDate for sample collection>
  ```
- `QueryPerfData2Sample.vb` — This file demonstrates how to retrieve historical statistics after creating a new VirtualCenter perf collector with extended statistics for CPU, memory, and network usage, by using the QueryPerfData2 operation.

Run this sample by typing:

```
PerfMon QueryPerfData2 <URL> <username> <password> <name for new
collector> <hostPath> <startDate for sample collection>
```

# TestOps Application

The TestOps application is a command-line application that demonstrates the use of several SDK methods such as Create, Delete, Rename, GetContents, GetInfo, and so on.

## TestOps Application Source Files

The TestOps application's source files are located in the `WebService\samples\vb\TestOps` directory. Here is a brief description of the files:

- `Reference.vb` and `CertPolicy.vb` — These files are the same as the similarly named files in the SimpleListing application. See SimpleListing Application Source Files on page 259.
- `VmaClient.vb` — This file is a very simple test that invokes various operations such as Login, Logout, GetContents, GetInfo, Create a virtual machine, Clone a virtual machine, Rename a virtual machine, virtual machine power operations, and so on. By testing all these operations, this file is a fairly comprehensive test of the VMware VirtualCenter Web Service.

## Running the TestOps Application

You can compile this test from within Visual Studio. It needs three input parameters: the URL of the Web service, the user name, and the password. This test assumes that there is one host connected in VirtualCenter.

CHAPTER 10

# Client Development Environments

This chapter describes the different development environments that you can use to develop your client application.

# Selecting a Development Environment

A Web service development environment facilitates the building of Web service servers and clients. Two popular examples of such environments for Java are Axis from Apache and the Websphere Software Developer Kit (WSDK) for Web services from IBM.

The Web service development environments are also integrated into the more popular integrated development environments (IDEs); for example, Websphere WSDK is integrated into Eclipse. Similarly, Microsoft's Visual Studio .NET supports Web service development for its constituent languages. The VMware SDK has been tested with all the platforms previously listed.

The Web service development environments support a variety of capabilities such as generating server code, generating WSDL from other language APIs, generating client code, and so on. While server side development is the typical emphasis of Web service development environments, we require the environment simply to generate client code from the WSDL file provided in the SDK. We provide the Web service server.

We use the development environment as follows:

1. Collect all the library files (`.dll` or `.jar`) that comprise the client-side common infrastructure. The generated stub files and client code make calls to these libraries in order to communicate with the server.
2. Generate stub files (for example, in Java) corresponding to the WSDL shipped with the SDK. Compile these stubs. The interfaces of these stubs serve as the APIs you use when building your client code.
3. Write your client code using the APIs of the generated stubs as well as the APIs of the library files.
4. Compile your client code.

# IBM Websphere Software Developer Kit

WebSphere is IBM's e-business software that enables companies to develop, deploy, and integrate e-business applications.

## Installing the IBM Websphere Software Developer Kit

We use the IBM WSDK Version 5.1 for the examples in this guide. You can download the WSDK from *www-106.ibm.com/developerworks/webservices/wsdk/*. This download includes IBM's Java SDK (version 1.3) and a test version of the Websphere application server. We recommend that you use the Java SDK that comes with this download.

1. Install IBM WSDK on the machine you plan to use to develop client applications. Refer to the WSDK Installation Guide at *www-106.ibm.com/developerworks/webservices/wsdk/install_guide.html*.
2. After completing the installation, include the top level `bin` directory and the `bin` directory of the Java SDK to your path.
3. To verify correct installation, type `WSDL2Client`.
4. Type `java -version` to make sure the Java SDK works and prints the IBM version number.

## Sample Application Developed by Using the IBM Websphere Software Developer Kit

We developed the simple client application described in Creating a Simple Client on page 52 with the IBM Websphere SDK. This section includes the sample code, as well as explaining the logic used in creating this application.

## Running the MoveVM Java Sample with the IBM Websphere Software Developer Kit

There is an issue with running the MoveVM Java sample with stubs generated by IBM WSDK. The stub generator generates the name of the VirtualDiskDestination parameter with the name "VirtualDiskDestination" instead of "disk". This error causes the conversion of the parameters to fail, because the Web service expects "disk" and the parameter's name is wrong. (Unbounded elements in complexTypes that are Method parameter elements are not handled correctly.)

To fix this problem, you need to edit the `VmaBindingStub.java` class file.

**Note:** If you are using the pre-built Java stubs contained in the VMware SDK package, then we have already edited the `VmaBindingStub.java` class file with this fix, and no change is required. However, if you are regenerating these stubs yourself, then complete the following steps.

1. Generate the stub files. See Generating the Stub Files on page 50.
2. Search for the term `_moveVMOperation` in `VmaBindingStub.java` in the `<client starting directory>\com\vmware\vma` directory.

   There should be a `ParameterDesc`:

   ```
   new
   com.ibm.ws.webservices.engine.description.ParameterDesc(com.ibm.ws.
   webservices.engine.utils.QNameTable.createQName("urn:vma1",
   "VirtualDiskDestination"),
   com.ibm.ws.webservices.engine.description.ParameterDesc.IN,
   com.ibm.ws.webservices.engine.utils.QNameTable.createQName("urn:vma
   1", "VirtualDiskDestination"),
   com.vmware.vma.VirtualDiskDestination[].class, false, false),
   ```

3. Change the mention of `VirtualDiskDestination` in the First `createQName` call in this line to `disk`. Do not change any other occurrences of `VirtualDiskDestination` in this file. This line should now resemble the following:

   ```
   new
   com.ibm.ws.webservices.engine.description.ParameterDesc(com.ibm.ws.
   webservices.engine.utils.QNameTable.createQName("urn:vma1",
   "disk"),
   com.ibm.ws.webservices.engine.description.ParameterDesc.IN,
   com.ibm.ws.webservices.engine.utils.QNameTable.createQName("urn:vma
   1", "VirtualDiskDestination"),
   com.vmware.vma.VirtualDiskDestination[].class, false, false),
   ```

4. Recompile the files without regenerating the stubs.

# Microsoft Visual Studio .NET and .NET Framework

The .NET framework is a runtime environment for running .NET applications. In this respect, it is similar to the C-runtime library, although it has many more components than a single DLL. In order to run a .NET application, you must first install the appropriate version of the .NET framework (for VMware SDK this is version 1.1). Unlike a typical run-time library, the .NET framework is not distributed together with the applications that use it. Instead, users must install it. The .NET framework is available at *msdn.microsoft.com/downloads/*.

Microsoft Visual Studio .NET is a development environment for building a variety of Windows applications, including .NET applications. It also includes extensive support for building .NET-based Web Services applications, such as Web service server and Web service client applications. While Microsoft Visual Studio .NET is necessary to develop and debug a .NET application, it is not required to run a released version of such an application; only the .NET framework is needed. To obtain a copy of Microsoft Visual Studio .NET, you can either purchase it directly from Microsoft or receive it as part of the MSDN subscription.

You can obtain more information about Visual Studio and the .NET Framework by clicking on their links in the Developer Centers section of *msdn.microsoft.com*.

## VMware SDK Applications Developed with .NET

The C# and Visual Basic sample applications contained in the VMware SDK package are developed using Microsoft Visual Studio .NET 2003 (Version 7.1) and version 1.1 of the .NET framework.

**Note:** Using earlier versions of the Microsoft Visual Studio .NET will not work for these samples.

There are four C# applications:

- Alerts — Alerts Application on page 252
- VMA Viewer — VMA Viewer Application on page 254
- Perfmon — PerfMon Application (C#) on page 256
- WS-ITest — WS-I Test Application on page 258

Similarly, there are four Visual Basic applications:

- SimpleListing — SimpleListing Application on page 259)
- VMPowerOps — VMPowerOps Application on page 261
- PerfMon — PerfMon Application (Visual Basic) on page 263
- TestOps — TestOps Application on page 265

To develop a Web service client application using Microsoft Visual Studio.NET, you must first use a WSDL proxy generator tool to generate proxy code based on the WSDL file. Essentially, the proxy code exposes all Web service operations as methods of a C# or Visual Basic class. The file that contains the proxy code is then added to the Microsoft Visual Studio.NET project, used to develop the client application. This enables the client application code to invoke Web service operations programatically by calling the methods of the proxy class.

Microsoft Visual Studio .NET includes a tool for generating proxy code from a WSDL file, called `wsdl.exe`. This tool does not correctly generate code with some WSDL files, including our `vma.wsdl`. (For more information, refer to *support.microsoft.com/default.aspx?scid=kb;en-us;326790*.)

Therefore, we've included our own proxy code generator, `wsdlProxyGen.exe` in the VMware SDK package. This tool has a simple GUI interface and clients can use `wsdlProxyGen.exe` to generate stubs from the WSDL file. However, we do not guarantee that this tool works for WSDL files other than `vma.wsdl`.

Read `SDK/SDK-README.html` in the VMware SDK package for additional information about `wsdlProxyGen.exe`.

# Apache Axis

Apache Axis is a modular, flexible, and high performing SOAP implementation designed around a streaming model. You can obtain Apache Axis at *ws.apache.org/axis*. This download package contains the Web services tool kit.

1. Install Apache Axis Web services tool kit.
2. Include the following `.jar` files to your classpath: `axis.jar`, `commons-logging.jar`, `commons-discovery.jar`, `jaxrpc.jar`, `wsdl4j.jar` and `saaj.jar`.
3. Set the JAVAHOME variable to a J2SE 1.4 Java installation.
4. Generate stubs by using the Axis WSDK. For example:

```
java  org.apache.axis.wsdl.WSDL2Java -Nurn:vma1=com.vmware.vma vma.wsdl
```

   This command generates the stubs under `com/vmware/vma`.

5. Compile these stubs into `vma.jar`.
6. Add `vma.jar` to your classpath.
7. Run the sample code with the `vma.jar` file built against the Axis WSDK. You do not need to make any changes to the sample code to run against Axis WSDK. The provided samples will run, as is, against the new `vma.jar` and `axis.jar` files.
8. See Creating a Simple Client on page 52 for information on how to write a simple client application using the VMware SDK.

# SOAP::LITE for Perl

SOAP::Lite is an open source collection of Perl modules that provides a simple and lightweight interface to the Simple Object Access Protocol (SOAP). It is currently the standard for designing Perl-based Web service applications.

However, SOAP::Lite is limited in its support for WSDL schemas. By default, it supports the RPC/Literal schema whereas our `vma.wsdl` uses the Document/Literal schema. To get around this issue, all SOAP messages are slightly modified to adhere to the Doc/Literal schema before transmission. With this modification, SOAP::Lite works with our Web service and can support the full complement of SDK operations.

To run the Perl samples, you must have installed the following:

- Perl interpreter — Enables you to run your Perl program, available at *www.perl.com*.
- SOAP::Lite — Enables SOAP interaction, available at *soaplite.com/download*.
- Tie::IxHash — Enables ordering of hash elements, available at *search.cpan.org*.
- HTTP::Cookies — Enables session maintenance, available at *search.cpan.org*.
- Bundle::LWP — Enables you to define all prerequisite modules for libwww-perl, available at *search.cpan.org*.
- Crypt::SSLeay — Provides SSL (`https`) support for Perl-based Web service clients, available at *search.cpan.org*. This module is required if you want the Perl samples to work with an SSL connection. In addition, you must have installed OpenSSL on your system (*www.openssl.org*).

**Note:** These Perl modules must be properly installed in order to run the Perl samples. Refer to the documentation at these Web sites for instructions on installing these modules.

## Testing a SOAP::Lite Installation

To verify that your installation of SOAP::Lite is correct, run the example script `dynamic4.pl` available under the `SOAP-Lite-0.55/examples/WSDL` directory.

```
perl  SOAP-Lite-0.55/examples/WSDL/dynamic4.pl
```

This script uses the publicly available StockQuote wsdl and displays a numeric value on success.

**Note:** If you're working behind a proxy/firewall to reach the Web service, then include the following line of code at the beginning of the script:

```
$ENV{HTTP_proxy} = "http://<proxy_server:port_number>";
```

CHAPTER 11

# Troubleshooting

This chapter describes some troubleshooting tips if you have difficulties with the VMware VirtualCenter Web Service. We also describe how you can customize the location and to some extent, the format of your dump file.

# Troubleshooting the VMware SDK

1. Reset the VMware VirtualCenter and the VMware VirtualCenter Web Service (they should both be on the same Windows machine) by completing the following. Refer to the section on "Finishing VirtualCenter Web Service Installation" in the *VMware VirtualCenter Users' Manual* for detailed instructions.
   a. Stop the Web service.
   b. Stop and restart VirtualCenter.
   c. Restart the Web service.
2. Refer to your VirtualCenter documentation and verify that the Web service is running. Perform the Web service post-installation verification steps, including connecting to the Web service through a browser.

   If you are unable to connect to the Web service, first double-check your username and password and retry the request. If you see an XML document similar to the following, then check the Web service log file, as described in Web Service Can't Connect to VirtualCenter on page 279.

```
- <env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <env:Body>
- <env:Fault>
  <env:faultcode xmlns:ns="urn:vma1">ns:FaultInfo</env:faultcode>
  <env:faultstring>Service Unavailable</env:faultstring>
  <env:faultactor>vma</env:faultactor>
- <env:detail>
- <ns:FaultInfo xmlns:ns="urn:vma1">
  <kind>BadRequest</kind>
  <code>503</code>
  <info>Service Unavailable</info>
  </ns:FaultInfo>
  </env:detail>
  </env:Fault>
  </env:Body>
  </env:Envelope>
```

3. If the Web service is running, but you are unable to connect to it and see a ConnectionException, then the Web service may be busy servicing other requests. The client should wait for a few seconds and then retry the request. Refer to the sample code below or at `/SDK/WebService/samples/java/sampleapp/src/com/vmware/sample/ExceptionHandling` (ConnectRetry) for an example of how to do this.

```
boolean success = false;
int numRetries = 0;
int maxRetries = 3;

while (!success && numRetries < maxRetries) {
    try {
        // connect to VMA and invoke operations
        VmaService vmaservice = new VmaServiceLocator();
        VmaPortType serviceConnection =
            new VmaBindingStub(new URL(args[0]), vmaservice);
        ((Stub) serviceConnection)._setProperty(
            Stub.SESSION_MAINTAIN_PROPERTY,
            Boolean.TRUE);

        System.out.println("Connection succeeded");
        // Logs in to the server.
        serviceConnection.login(args[1], args[2]);
        //.. some other operations here
        serviceConnection.logout();
        success = true;

    } catch (Exception ex) {
        if (ex.getMessage().startsWith("java.net.ConnectException")) {
            System.out.println("Connection to VMA failed. Retry after 3 secs");
            synchronized (this) {
                wait(3 * 1000);
            }
            numRetries++;
            if (numRetries >= maxRetries) {
                System.out.println("Unable to connect to WebService after 3 retries. Aborting.");
            }
        }
    }
}
```

4. Start one of the SDK sample applications.

   a. Go to the directory where the VMware SDK package contents were unzipped.

   b. Run the following command. If it is different, then substitute the port number you chose during the Web service installation.

   ```
   ./runsample.bat com.vmware.sample.Simple.SimpleClient http://
   <host_name>:8080 <username> <password>
   ```

   This command runs the simple client application that lists the UUIDs of all the virtual machines in VirtualCenter.

5. Attempt to run the other small sample applications.
   - Run the GetUpdates sample, the PutUpdates sample, and so on, contained in the SDK package. Read `SDK/SDK-README.html` for more information on how to run the various sample applications.
   - Run the Web-based monitoring and management application (Web-based Monitoring and Management Application on page 246).

# Problems Connecting to VMware VirtualCenter

If a client program has issues connecting to VirtualCenter, then you may not notice this problem until the client attempts to log into the Web service and it returns a fault. When this occurs, there are two typical causes: either the client can't connect to the Web service, or the Web service can't connect to VirtualCenter.

## Client Can't Connect to the Web Service

If this is the problem, then the Web service returns a PermissionDenied fault for the Login operation. The client is using an invalid user name or password in the Login request. Choose a valid user name and password, and retry the Login request.

In this case, nothing is logged in the Web service log file (by default, `vma.txt`), unless the `/vcenter` subject log level has been changed from its default value of "info" to "verbose". (See vmaConfig.xml File on page 16.) If the log level is "verbose", then the following error message is logged:

```
Authentication of user <username> with VirtualCenter service on
host <hostname>, port <port_number> failed: Bad username/password
```

## Web Service Can't Connect to VirtualCenter

If this is the problem, then the Web service returns a BadRequest fault with error code 503 (ServiceUnavailable) for the Login operation. The Web service is in a bad or disconnected state. There are three possible causes: the Web service is unable to log into the VirtualCenter service, the VirtualCenter service is not running, or the VirtualCenter port number is incorrect. You must examine the Web service log file (by default, `vma.txt`), to determine the source of the problem.

- If the VirtualCenter service is running, but the Web service is unable to log in (as a user with administrative privileges), then the following error message is logged.

  ```
  VirtualCenter web service failed to connect to VirtualCenter
  service on host <hostname>, port <port_number> as <administrator>:
  Bad username/password
  ```

  Configure the Web service with a valid user name and password (with administrative privileges) for the Web service to connect to VirtualCenter. Use the `vma` command-line program, as described in the section on "Changing Web Service Options After Installation" in the *VMware VirtualCenter User's Manual*.

- If the VirtualCenter service is not running, then the following error message is logged.

```
VirtualCenter web service failed to connect to VirtualCenter
service on host <hostname>, port <port_number>: Connection refused
```

1. Check the VirtualCenter service, and restart it.
2. Stop, then restart the Web service. Refer to section titled "Changing VMware Web Service Options After Installation" in the *VMware VirtualCenter User's Manual* for detailed instructions.

- If the Web service is using the incorrect port number to connect to VirtualCenter, then the following error message is logged.

  ```
  VirtualCenter web service failed to connect to VirtualCenter
  service on host <hostname>, port <port_number> : No connection could
  be made because the target machine actively refused it.
  ```

  1. Stop the Web service. Refer to the section titled "Changing VMware Web Service Options After Installation" in the *VMware VirtualCenter User's Manual* for detailed instructions.
  2. Edit the `vmaConfig.xml` file with the correct VirtualCenter port number. See vmaConfig.xml File on page 16.
  3. Restart the Web service.

# Viewing the Dump File

If the VMware VirtualCenter Web Service crashes, a crash dump file is created. This section describes how to find this dump file, and how to turn on detailed error logging.

When reporting a problem with the Web service, a dump file and detailed logs that you provide may be very useful in helping VMware quickly debug the problem.

Over time, these dump files (`.dmp` files) may accumulate. You may to remove them to reclaim storage space.

## Customizing the Dump File

Dump files are created when an unhandled exception occurs in the Web service. By default, the dump file is summarized as a minidump file. You have the option of customizing the minidump file by determining its location and to some extent, by configuring its format and changing it to a full crash dump file or by changing the logging to a verbose format.

If you want to customize the minidump files, then make your changes to the Web service configuration file, `vmaConfig.xml`, located in `C:\Documents And Settings\All Users\Application Data\VMware\VMware VirtualCenter\VMA`.

By default, the name for the minidump file is `vma-<pid>.dmp` and the format of the minidump file is Normal+DataSegs+HandleData, which is enough for callstack lookup.

1. Open `vmaConfig.xml` and look for the `<service>` element.
2. (Optional) Change the default directory location for the minidump files. For example, typing the following:

   ```
   <coreDumpDir>c:\MyTemp</coreDumpDir>
   ```

   changes the location of the minidump files to the `C:\MyTemp` directory. By default, the minidump files are located in `C:\Documents And Settings\All Users\Application Data\VMware\VMware VirtualCenter\VMA`.
3. (Optional) Change the format of the minidump file to a full dump file.

   ```
   <coreDumpFullMemory>true</coreDumpFullMemory>
   ```

   The default minidump format has less detail than a full dump, but is sufficient for debugging.
4. (Optional) Enable more verbose logging from the Web service by changing the `<eventlog level>` element from "`info`" to "`verbose`".

   The resulting detailed log is saved in `vma.txt`, located in `C:\Documents And Settings\All Users\Application Data\VMware\VMware VirtualCenter\VMA`.

CHAPTER 12

# Glossary

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

***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.

***Container —*** A container describes an interior object in a hierarchy, such as a file system directory or a Server Farm in VMware VirtualCenter.

***Event declaration —*** Type of event (alert, error, warning, info, or user) along with its name, arguments, and its message format.

***Farm —*** A Farm is a set of host machines and VirtualMachineGroups that collectively act as a single host. This concept is similar to a "Server Farm" in VMware VirtualCenter.

***File —*** A file contains raw data, such as an image, or other auxiliary information.

***FQDN —*** Fully Qualified Domain Name of a host includes both the host name and the domain name. For example, a host named `esx1` in the domain `vmware.com` is represented as `esx1.vmware.com`.

***Group —*** A group is a set of users and groups.

***Host Machine —*** A host machine is a system capable of managing and executing virtual machines.

***IBM WebSphere —*** WebSphere is IBM's e-business software that enables companies to develop, deploy, and integrate e-business applications.

***Item —*** An item is a <name, value> pair for an element of information about an object. An item is normally represented as an element in XML. However, in some cases, it may be necessary to represent a collection of Items in a single element for performance reasons.

***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).

***Link —*** A link is a hyperlink as in HTML; that is, a path to another object. As in the Web, links may be relative to the current object path, relative to the current server's object root, or on a specific server, as interpreted by the current client's host name resolver.

***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 powered-off virtual machine between hosts.

***Name —*** A name is either a path that refers to an object (like an URL or URI in Web terms) or the name of an item of information in the server.

***Operation —*** An operation describes the interaction between a client and the Web service.

***Provisioning (a virtual machine) —*** The process of creating a functioning virtual machine by assigning resources such as CPU, memory, and virtual hardware, and then deploying a system image.

***Remote console —*** An interface to a virtual machine that provides non-exclusive access to a virtual machine from both the server on which the virtual machine is running, and from workstations connected to that server.

***Service console —*** The VMware Service Console provides ESX Server system management functions and interfaces. It is installed as the first component, and is used to bootstrap the ESX Server installation and configuration. It also boots the system and initiates execution of the virtualization layer and resource manager. In ESX Server, the service console is implemented by using a modified Linux® distribution.

***Service Host —*** The Web service executes on the service host.

***Session token —*** Upon successful login, the Web service issues a token and as part of the HTTP header, passes it to the client in the response. Specifically the token is a cookie that should be passed back by the client with each request. The token identifies the session to the Web service and its format is opaque to the client. Session tokens can be passed across multiple connections to the server. A session token expires after a period of inactivity, and may also expire after a certain period of time, even if it is being actively used.

***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 stub is a local procedure in 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.

***Target —*** A target is the object that corresponds to a request URI.

***Uptime —*** Uptime is the total elapsed time since the host or virtual machine was last restarted. Uptime may be computed using the last bootTime or by looking up the interval from the cumulative PerfSample.

***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.

***View —*** A view is an XML document that contains information about service objects, particularly virtual machines and hosts. Use a view to access virtual machines and other top level objects through the Web service.

***View Body —*** The view body is the XML document that describes that object's current state that is obtained by using the GetContents operation.

***View Definition —*** A view definition is an XML document that specifies the elements that appear in a view. View definitions typically specify the items of interest in the view, but may also include additional elements for presentation or computation, related to the items.

***Virtual Machine —*** A virtual machine contains system and configuration state for a machine and may execute on a host machine.

***Virtual Machine Array —*** A virtual machine array is a set of virtual machines that may be operated on collectively. This concept is currently called a "VM Group" or "VM Folder" in VMware VirtualCenter.

***VMA —*** The VMA is the VMware VirtualCenter Web Service, which provides a Web services interface that client programs may use to talk, by using the SOAP protocol.

***Web Service —*** A programming interface based on SOAP and WSDL.

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

APPENDIX A

# Revision History

The following table lists the revision history for the *Virtual Infrastructure SDK Programming Guide*.

| Date | Description |
|---|---|
| May 11, 2006 | No change in content for VirtualCenter 1.4 release. |
| September 22, 2005 | Added mention of support for VMware VirtualCenter 1.3. |
| November 30, 2004 | GA version of this guide. |

# Index