PRENTICE HALL

VMware® VI and vSphere SDK

Managing the VMware Infrastructure and vSphere

STEVE JIN

The Library of Congress Cataloging-in-Publication data is on file.

ISBN-13: 978-0-137-15363-3
ISBN-10: 0-137-15363-5
Text printed in the United States on recycled paper at R.R. Donnelly in Crawfordsville, Indiana.
First printing September 2009

Editor-in-Chief
Karen Gettman
Acquisitions Editor
Jessica Goldstein
Development Editor
Sheri Cain
Managing Editor
Kristy Hart
Project Editor
Jovana San Nicolas-Shirley
Copy Editor
Karen A. Gill
Indexer
Erika Millen
Proofreader
Williams Woods Publishing Services
Publishing Coordinator
Romny French
Cover Designer
Chuti Prasertsith
Compositor
Jake McFarland

Preface

Virtualization is not a new concept, but it is changing the computing industry in a profound way. Server virtualization is now #1 on enterprises' budget lists. According to analysts, it will continue to be the highest impact trend, changing infrastructure and operations through 2012.

Virtualization became popular for two reasons. First, the hardware (especially x86–based servers) capacity has increased so much that most servers are under utilized. The market demand is strong for consolidating servers and saving operation and management cost. Virtualization has clearly provided a proven solution for this demand. Second, the social awareness of environment protection and energy saving has put green technology into the spotlight. Virtualization addresses this social requirement well by saving electricity consumption.

Today, more than 4 million virtual machines are installed. With the acceleration of hypervisor toward commodity because of increased competition, even more virtual machines will be running along the way.

In this virtualization game, VMware is by far the market leader, with 100 percent coverage of Fortune 100 enterprises. In 2007, the company achieved $1.3 billion revenue and has been growing steadily about 80 percent since its inception in 1998. There were 100,000 customers and 10,000 partners in 2008. The VMware annual conference, VMworld, attracted more than 14,000 attendees in 2008.

With the proliferation of the VMware platform, the demand for management of the virtualization environment is clear. In general, for every dollar spent on the initial infrastructure investment, $6 or more is needed for management and operation. VMware has done a great job of providing best-of-the-breed management products, including VirtualCenter server and other value-added solutions, such as Site Recovery Manager.

Clearly, VMware cannot do everything. As a platform company, VMware needs to enable partners and customers to come up with solutions to the integration, customization, and automation of VMware platforms. To achieve these, you need a solid understanding and hands-on expertise of VI SDK. This is where this book fits into the big picture of whole virtualization technology and industry.

This book helps you with the basic concepts of virtualization, the VMware VI SDK, and how to effectively use the SDK in your projects.

Who Should Read This Book

This book is for anyone interested in virtual system management of VMware platforms. It specifically targets the following audiences:

- System administrators who want to automate, customize, and optimize the VMware virtualization platforms. This book uncovers many interfaces under the hook of VI Client, RCLI, VI PowerShell, and so on.
- Software architects, engineers, and solution developers who want to design and develop applications with VI SDK. Possible vendors include the following:
    - Hardware vendors who use the VI SDK for developing management software to manage their specific system or device
    - Independent software vendors who develop applications or components using the SDK
    - System integrators who develop solutions targeting specific industry sectors
    - VMware competitors who use VI SDK as a reference for their own designs
- Technical managers, including program managers, who oversee virtualization projects for a good sense of the SDK.
- Researchers and students interested in VMware virtualization technology and might use it in their projects.
- Anyone interested in system management of virtualization platforms.

Prerequisites

To read this book, you need to have the following skills and knowledge:

- **Basic Java programming skills**—Most of our code samples are written in Java. It's not because VI SDK favors Java over others, but because Java is a popular programming language that runs on all major platforms and has the widest audience today.

  If you use other programming languages, the samples are still helpful even though they cannot be used as they are. The methods and data objects don't change much across the languages. Reading the Java sample for these can help you develop in other languages as well.

- **OS virtualization basics, especially the VMware virtualization platforms**—They are the management target of the SDK; therefore, a solid understanding of how things work in the virtual world offers an advantage toward understanding the model behind the APIs.
- **Web Services**—It's optional, and only needed to understand the VI SDK Web Services interfaces. This book covers a little background when it gets there.

This book gives you in-depth knowledge, the best practices of VMware VI SDK development, and hands-on experience with many useful samples that can be easily modified for your own automation or application development.

Structure of This Book

This book takes a pragmatic approach to show you how to program or script VI SDK for your work. As a long-time software professional, I know how software engineers think and work. We don't start a new technology by reading hundreds of pages of documents. Instead, we start with samples and read some documents when we have doubts with some concepts and API usages. With this in mind, this book provides you with many useful samples[1] that you can use as is or adapt to your projects.[2]

Although samples are important, so are the basic concepts and best practices. I show you both the big pictures and details that can be easily ignored or missed. While helping VMware strategic partners with their development projects and

[1] For easy reading, I highlighted all Java keywords in bold in the samples.

[2] These samples are intended to illustrate the usage and best practices of the APIs. They are not necessarily comprehensive or production ready.

community members with their questions, I have seen how the SDK can be misused. I explain some of these pitfalls and how to effectively avoid them. I also introduce how to best use these APIs. This book is not just another reference book.

Better than a typical SDK book, this book introduces the VI Java API I have created. With the API, you can build much shorter, faster, and more importantly much more readable and maintainable code. Most of the samples used in this book are written using this higher-level API.

Most chapters are organized around various management tasks, which are supported by VI SDK managed objects. When applicable, the related managed objects involved are listed in the following overview as well as the beginning of each chapter so that you can easily locate a chapter given a managed object.

This book contains 18 chapters and 6 appendixes:

- **Chapter 1, "VMware Infrastructure Overview"**—This chapter introduces the virtualization basics and VMware products, especially VMware Infrastructure. It explains how VI SDK fits in the big picture.
- **Chapter 2, "VI SDK Basics"**—Here, you examine the VI SDK from the bottom up. This chapter covers the Web Services API, the object model, and various tools that help to familiarize you with the SDK.
- **Chapter 3, "Hello VI"**—In this chapter, you learn how to set up the development environment and run your first "Hello World" sample code. Debugging techniques are also included here.
- **Chapter 4, "Using `PropertyCollector` and `SearchIndex`"**—Exclusive attention is devoted in this chapter to how to retrieve properties and search managed entities using `PropertyCollector` and `SearchIndex`. `PropertyCollector` is one of the most often-used services; it's also regarded as one of the most difficult ones in the SDK.
- **Chapter 5, "Introducing the VI Java API"**—This chapter examines the open source Java API, which is built on top of the Web Services API. Using this API instead of Web Services, you can have much shorter, faster, and more readable code.
- **Chapter 6, "Managing Inventory"**—The structure of inventory and how to manage it are covered here. Also covered are the View family of managed objects, which can be used in GUI applications.

  The managed objects covered include `ManagedEntity`, `Folder`, `Datacenter`, `CustomFieldsManager`, `View`, `ManagedObjectView`, `ViewManager`, `ContainerView`, `InventoryView`, and `ListView`.

- **Chapter 7, "Managing Host Systems"**—In this chapter, focus is on the hypervisor on which virtual machines are running. It covers all the aspects except networking and storage, which are covered in detail in Chapters 10 and 11, respectively.

  The managed objects discussed are `HostSystem`, `HostDateTimeSystem`, `HostBootDeviceSystem`, `HostDiagnosticSystem`, `HostCpuSchedulerSystem`, `HostFirmwareSystem`, `HostHealthStatusSystem`, `HostAutoStartManager`, `HostMemorySystem`, and `HostPatchManager`.

- **Chapter 8, "Managing Virtual Machines, Snapshots, and VMotion"**—A virtual machine is the equivalent of a physical machine in the virtual world. This chapter shows how to manage its life cycle, change its configuration, find out more about the guest OS running on it, and migrate it and its storage live. Also covered are the virtual machine snapshots, including how they are structured and how to manage them.

  This chapter focuses on three managed objects: `VirtualMachine`, `CustomizationSpecManager`, and `VirtualMachineSnapshot`.

- **Chapter 9, "Managing Clusters and Resource Pools"—**Clustering is an advanced feature in which multiple hosts are grouped for high availability, load balancing, and energy saving, among other things. This chapter introduces how to manage VMware HA and DRS/DPM clusters. Resource pools and various resource allocation policies are also introduced here.

  This chapter covers three managed objects: `ComputeResource`, `ClusterComputeResource`, and `ResourcePool`.

- **Chapter 10, "Managing Networking"**—Networking is an important and sometimes confusing aspect of virtualization. This chapter guides you through the basic concepts of how to manage the virtual switches, port groups, virtual NIC, and network policies, as well as how to manage SNMP, network services, firewalls, and more.

  This chapter covers these managed objects: `HostNetworkSystem`, `Network`, `HostFirewallSystem`, `HostSnmpSystem`, `HostServiceSystem`, and `HostVMotionSystem`.

- **Chapter 11, "Managing Storage and Datastores"—**Storage is one of the most confusing parts because it involves many enterprise-level storage systems and how ESX virtualizes them. This chapter introduces basic concepts and how to perform various tasks from storage to datastore and files.

This chapter discusses the following managed objects: `HostStorageSystem`, `HostDatastoreSystem`, `Datastore`, `HostDatastoreBrowser`, and `FileManager`.

- **Chapter 12, "Events and Alarms"**—This chapter introduces what events and alarms are. It also shows how to retrieve events and how to set up alarms to monitor the virtual systems.

  The managed objects in focus are `EventManager`, `EventHistoryCollector`, `Alarm`, and `AlarmManager`.
- **Chapter 13, "Performance Monitoring"**—Performance is one of the biggest concerns people have when coming to virtualization. This chapter introduces the basic concepts of performance monitoring and how to retrieve performance statistics and monitor performance in real time.

  The sole managed object covered is `PerformanceManager`.
- **Chapter 14, "Task and `ScheduledTask`"**—This chapter introduces tasks and scheduled tasks. It shows how to monitor/cancel a task and how to set up scheduled tasks for automation.

  The managed objects covered are `Task`, `TaskManager`, `TaskHistoryCollector`; `ScheduledTask`, and `ScheduledTaskManager`.
- **Chapter 15, "User and License Administration"—**Here, you learn how user and license management work in VMware Infrastructure and how you can manage them using the SDK.

  The managed objects discussed are `AuthorizationManager`, `HostLocalAccountManager`, `UserDirectory`, `SessionManager`, and `LicenseManager`.
- **Chapter 16, "Extending the VI Client"**—The VI SDK provides an extension API. This chapter introduces how it works and what it takes to plug into the VI Client.

  The managed object involved is `ExtensionManager`.
- **Chapter 17, "Scripting the VI SDK with Jython, Perl, and PowerShell"**—This chapter introduces development of scripts with three major scripting languages that are commonly used in system administration: VI Perl, PowerShell, and Jython (Python).
- **Chapter 18, "Advanced Topics"**—This chapter covers topics that are important but do not fit in previous chapters (for example, multithreading, versioning, best practices for performance and scalability, and I18N).

The managed objects involved are `OptionManager`, `DiagnosticManager`, and `HostSystem`.

- **Appendix A, "The Managed Object Types"**— This appendix lists all the managed object types in VI SDK 2.5 and vSphere 4.
- **Appendix B, "The Performance Counters"**— This appendix lists all the performance counters you might need to retrieve performance statistics.
- **Appendix C, "Cmdlets in the VI Toolkit (for Windows)"**— This appendix includes the cmdlets in the toolkit 1.0.
- **Appendix D, "Unified Modeling Language"**—This appendix provides you with the basic knowledge to understand this book's UML diagrams.
- **Appendix E, "VI SDK Web Services"**— This appendix examines the Web Services in greater detail than Chapter 2.
- **Appendix F, "What Is New in vSphere 4 SDK?"**—This appendix summarizes the changes in the newly released vSphere 4 SDK, which is the next version after VI SDK 2.5.

How to Read This Book

This book is organized in an order best for most readers. It starts with an overview of virtualization technology and VMware VI products, trying to give you the big picture of virtualization and the importance of system management in a virtualized environment. Chapter 1 is recommended, but it's not required.

This book then covers the VI SDK basics (Chapter 2), how to set up the development environment (Chapter 3), and the basic managed objects `PropertyCollector` and `SearchIndex` (Chapter 4). Chapter 3 is a must-read if you want to get hands on with the samples. Chapter 4 is optional, but crucial to understand the implementation of VI Java API discussed in the chapter after.

Chapter 5 is a must-read for Java developers, because after this chapter, the Java API is used primarily. You don't necessarily read how the API is designed, but definitely how it should be used.

The chapters following Chapter 5 cover different parts/aspects of the VI SDK, from inventory management, host and virtual machine management, and storage to networking, performance statistics, and event management. You can randomly read these chapters, which are in braces ({}) in the following list, depending on your interest and preference.

Not all readers are application developers. More often than not, system administrators write scripts to automate daily tasks. If you're only looking for scripting, you can jump directly to Chapter 17, where three basic scripting languages are covered.

The following summarizes the critical reading paths based on your interests:

- **General knowledge**—Chapters 1 and 2
- **Java development**—Chapters 3, 5, {2, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18}
- **Jython development**—Chapters 2, 5, 17, {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18}
- **VI PowerShell development**—Chapter 17, Appendix C, {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18}
- **VI Perl development**—Chapters 2, 17, {6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18}

Chapter 18

Advanced Topics

This chapter discusses several topics that do not quite fit into the previous chapters. However, you may find them useful in your application development:

- Managing global settings with `OptionManager`
- Collecting logs using `DiagnosticManager`
- Sharing sessions with multiple clients
- Using single sign-on from the VI SDK to CIM
- Downloading and uploading files using HTTP access
- Versioning VI SDKs
- Multithreading with the VI SDK
- Following best practices for performance and scalability
- Considering internationalization

Managed Objects: `OptionManager`, `DiagnosticManager`, `HostSystem`

Managing Global Settings with `OptionManager`

`OptionManager` manages key/value pairs as a mechanism for flexible settings. It can be reached from both `ServiceInstance` and `HostSystem`.

When connecting directly to an ESX server, you can get the `OptionManager` object from the `ServiceInstance`, but it doesn't have much information. The `OptionManager` object from the `HostSystem` has a real data setting that you can find and modify from the Advanced Setting dialog box of the ESX system in the VI Client.

When connecting to a VirtualCenter server, you can get an `OptionManager` object from the `ServiceInstance` with VirtualCenter server settings. You can find and change these settings in the VirtualCenter Management Server Configuration dialog box in the VI Client. The `OptionManager` object from a `HostSystem` is the same that you would get from the `HostSystem` on ESX.

The `OptionManager` type has two properties defined:

- **`supportedOption`**—List of `OptionDef` objects. The `OptionDef` data object, whose UML diagram is shown in Figure 18-1, is extended from `ElementDescription`. The `ElementDescription` has a string property key, which normally uses a dot-separated notation to indicate its position in the whole hierarchy, such as `mail.smtp.port` for the SMTP port option under the Mail section.

  `OptionDef` includes `OptionType`, which has one property indicating whether the option is read-only. `OptionType` has six subtypes, each of which represents a data type of the option. Most of them include `defaultValue` for the option. `FloatOption`, `IntOption`, and `LongOption` have the same structure but different data types for the properties.
- **`Setting`**—List of `OptionValue` data objects to hold real key/value pairs. The key is one of the keys in the `OptionDef` objects in `supportedOption`. The value, as you expect, is defined as `xsd:anyType` because it can be any value as defined in the six types in `OptionDef`.

The `optionManager` defines two methods:

- `queryOption()` retrieves a specific node or nodes in the option hierarchy. It takes in a string parameter name as the key to the `OptionDef`. You can provide a full key, or the starting part of the key ending with dot, such as `snmp`,

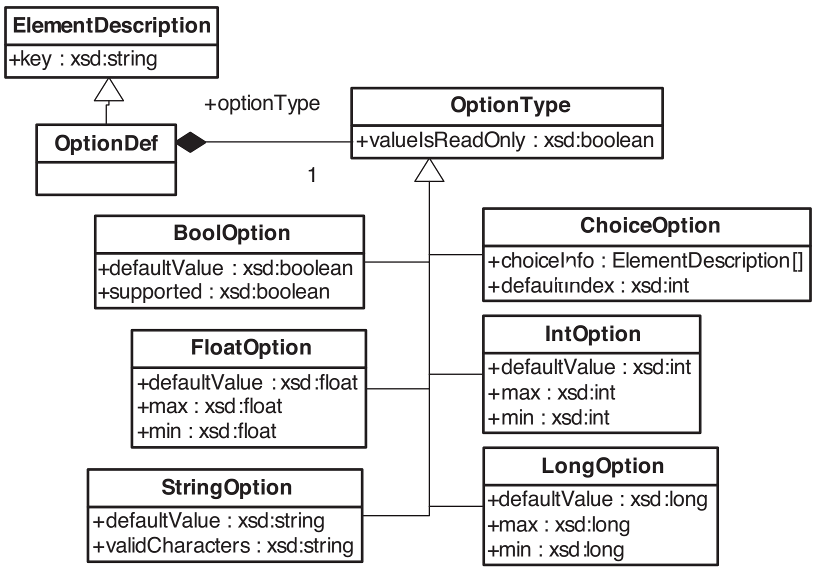


Figure 18-1 The `OptionDef` and its subtypes

which returns all the `OptionValue` objects under the `snmp` node. You must include the dot for the parameter to be valid.

- `updateOptions()` updates one or more options specified by the `OptionValue[]` parameter. The changes are done atomically; they are all changed, or none are changed.

For both methods, the option key must be valid, or the `InvalidName` fault is thrown. For `updateOptions()`, you must provide a valid value for the option as required in the `OptionDef`; otherwise, the `InvalidArgument` fault is thrown.

These two methods do not change the available `OptionDef` objects. In fact, you cannot add a new `OptionDef` or remove an existing `OptionDef`. You can add a new `OptionValue` without matching `OptionDef`, but that is not a typical use case.

In the VirtualCenter Management Server Configuration dialog box of the VI Client, you can add more key/value pairs when you select Advanced Settings. Whatever key you enter in is prefixed with `config` in the `OptionManager` object. Just use the Managed Object Browser (MOB) to check it out. Awkwardly, the next time you bring back the same dialog box, the newly added key/value pair doesn't display.

Collecting Logs Using `DiagnosticManager`

The `DiagnosticManager` provides services to access low-level debugging logs or generate diagnostic bundles for either the VirtualCenter server or ESX server.

The managed object defines no property but three methods:

- **`queryDescriptions()`** provides a list of diagnostic files for a given system. It takes in an optional parameter `host` for specifying the `HostSystem` to extract information from. When you connect to the ESX server directly, the parameter isn't needed. When you connect to the VirtualCenter server and the parameter isn't specified, the method assumes you're looking for VirtualCenter logs. The return of this method is an array of `DiagnosticManagerLogDescriptor` data objects. The data object includes six properties, as shown in Figure 18-2.

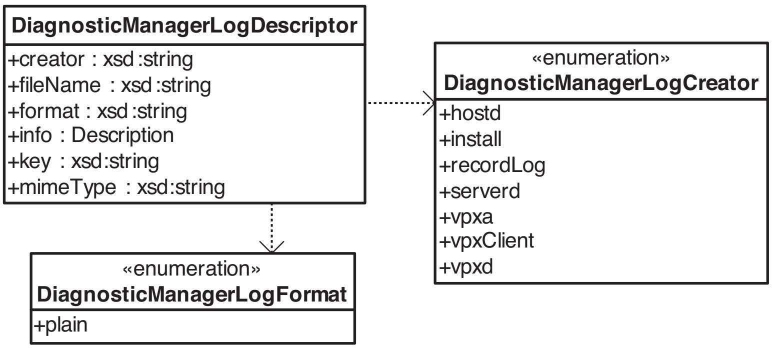


Figure 18-2 The `DiagnosticManagerLogDescriptor` data object

The `creator` property represents the component that creates the log; the value has to be one of the string values defined in the `DiagnosticManagerLogCreator` enumeration type. The filename is the full path to the log file on the ESX server, such as `/var/log/vmware/hostd.log`, or a simple filename on the VirtualCenter server, such as `vpxd-0.log`. The format has only one choice—`plain`—as defined in the `DiagnosticManagerLogFormat` enumeration type. The property `key` is used by other methods for browsing or downloading; it can have values like `vpxd:vpxd-0.log` on VirtualCenter, or `hostd`, `messages`, `vmkernel`, `vmksummary`, `vmkwarning`, or `vpxa` on the ESX server.

The `mimeType` is Multipurpose Internet Mail Extensions (MIME). With the format as `plain`, the `mimeType` is limited to `text/plain`.

- **`browseDiagnosticLog()`** allows you to "browse" a log file that is identified by the key as returned from the `queryDescriptions()` method. In addition to the key parameter, you can optionally specify two integers as the starting line and the number of lines in the log file you want to browse. If not specified, the starting line defaults to the top, and the number of lines defaults to the total from the starting to the end of the log. To browse the entire log, just leave the two optional parameters empty.

  The return from the method is a `DiagnosticManagerLogHeader` data object, which has properties such as `lineStart` for the starting line and `lineEnd` for the ending line, as well as an array of strings as log entries in the log file.

- **`generateLogBundles_Task()`** creates diagnostic bundles on the server side to be downloaded. A diagnostic bundle includes log files and other configurations, such as a virtual machine configuration that can be used to investigate potential server issues. The method has a boolean parameter, `includeDefault`, that specifies whether to include the default server, and optionally an array of `HostSystems` while connecting to a VirtualCenter server.

  The return of the method, as its name suggests, is a `Task`. When it's done successfully, the `info.result` property contains a list of `DiagnosticManagerBundleInfo` data object. The data object includes two properties: a system pointing to the `HostSystem` from which the diagnostic bundle is generated; and `url`, which represents the location where you can download the bundle. When you connect to the ESX server directly, `url` might have `*` as a placeholder for the real host name. Just replace it with the real host name or IP address before downloading it.

```
https://*:443/downloads/esxsupport-5224f0a4-becc-cf30-0d30-1bc28a26f7ce.tgz
```

As the file extension suggests, the bundle is a compressed archive file. Use tools like 7-Zip to extract the files inside. Depending on your environment, the bundle could have thousands of files for you to do a thorough analysis of the system.

Sharing Sessions Among Different Applications

When a client first logs into VC or ESX server, a username and password are required to authenticate the user and grant access privileges. In the successive interactions with VC server, no username and password are needed. The HTTP session ID is instead used to track the user.

This HTTP session is different from the `UserSession` discussed earlier. Their session IDs are in the same format, but they have different values. You can find one sample in the following code.

The HTTP session ID is essentially an HTTP cookie that is pushed from the server when the connection with the server is established. A `UserSession` is created after the login succeeds. There is mapping from the HTTP session to the user session on the server side so that the consecutive SOAP requests carrying an HTTP session ID automatically assume the privileges of a user. In other words, if your request has the same HTTP session ID as a current user, the system takes you as the user. There won't be a new `UserSession` for the new client.

In some of the cases, different clients need to share a user session. For example, in a VI Client plug-in, a Web application needs to use the session ID passed in from the VI Client in URL so that it can interact with the VC Server as if from the login user in the VI Client.

Now let's look at how to get the session ID and how to use it from another client.

Getting the Session ID

The code to get the session ID is as follows:

```
...
VimPortType vimService = null;
ManagedObjectReference mor =  null;
ServiceContent serviceContent = null;

VimServiceLocator serviceLocator = new VimServiceLocator();
serviceLocator.setMaintainSession(true);
```

```
try
{
      vimService = serviceLocator.getVimPort(new URL(urlStr));
      ManagedObjectReference serviceRef = new ManagedObjectReference();
      serviceRef.setType("ServiceInstance");
      serviceRef.set_value("ServiceInstance");
      serviceContent = imService.retrieveServiceContent(serviceRef);

      if(serviceContent.getSessionManager()!=null)
      {
            vimService.login(serviceContent.getSessionManager(), username,
password, null);
      }
}
catch (Exception e)
{
      System.err.println("Exception: " + e.getMessage() );
}

VimBindingStub vimStub = (VimBindingStub) vimService;
org.apache.axis.client.Call call = vimStub._getCall();
org.apache.axis.MessageContext msgContext = call.getMessageContext();
String sessionString = (String) msgContext.getProperty(
org.apache.axis.transport.http.HTTPConstants.HEADER_COOKIE);

System.out.println(sessionString);
...
```

Notice that after logging in, the code converts the `VimPortType` to `VimBindingStub` and gets the `Call`, `MessageContext` objects. From the `MessageContext`, the session ID is extracted and printed.

In most cases, you only use the interface `com.vmware.vim.VimPortType` for operations in the VI SDK. This is, however, not enough to access the session information. The `VimPortType` interface provides a login service, but it does *not* expose session information.

To share a session, just grab the real implementation class, which is `com.vmware.vim.VimBindingStub`. With this implementation class, you can get and set session information, as shown in the previous samples.

```
public class com.vmware.vim.VimBindingStub extends org.apache.axis.client.Stub
implements com.vmware.vim.VimPortType ()
```

The content of `sessionString` looks like this:

```
vmware_soap_session="B3240D15-34DF-4BB9-B902-A844FDF42E85"
```

This sample code is not always needed. In the case of the VI client plug-in, the VI client extracts the session ID and passes it in the URL to the Web application, which can use it to interact with the VC Server.

Using Session ID

Now, let's see how another client can use the session ID:

```
VimPortType vimService = null;
ManagedObjectReference mor = null;
ServiceContent serviceContent = null;

VimServiceLocator serviceLocator = new VimServiceLocator();
serviceLocator.setMaintainSession(true);

try
{
      vimService = serviceLocator.getVimPort(
                new URL("https://localhost/sdk"));
}
catch (Exception e)
{
      System.err.println("Exception: " + e.getMessage() );
}

VimBindingStub vimStub = (VimBindingStub) vimService;
vimStub._setProperty(
org.apache.axis.transport.http.HTTPConstants.HEADER_COOKIE,
"vmware_soap_session=\"B3240D15-34DF-4BB9-B902-A844FDF42E85\"");
...
```

This code's flow differs from the last code's flow because it doesn't require you to log in. The `VimPortType` is casted to `VimBindingStub`, and the session ID is set as the HTTP cookie.

Essentially, the session ID is a cookie string. It can be sent from one client to others in string format, in a local file, a URL, or messaging through the network. When setting the session, include "`vmware_soap_session`."

From the last line on, the consecutive code can send SOAP requests as if they were from the previous client, which prints the session ID.

While sharing the session, it's critical for the user of the session *not* to log out of the session. As a rule of thumb, whoever logs in first should close the session—nobody else. It's up to the designer of the client applications to track the numbers of clients who are sharing the same session.

Further Discussion

This introduction covers how to share the sessions in the code samples using Java. The previous samples actually assume using AXIS as the underlying Web Services engine. If you are using a different Web Services engine, find out how to retrieve the HTTP session and change the code accordingly.

Although the sample is only in Java, clients that share a session can write in any language that Web Services supports. For example, the session ID can be extracted by a Java client and consumed by a Perl client. If Perl is used for a re-use session, refer to the *VI Perl Toolkit Programming Guide*.[1]

Note that the session file format used in the VI Perl Toolkit is not simply the session ID string, but something more. The format of the session file is as follows:

```
#LWP-Cookies-1.0
Set-Cookie3: vmware_soap_session="\"52dc490b-a6e7-0e65-65a7-a926b924e72c\"";
path="/"; domain=192.20.143.205; path_spec; discard; version=0
```

I leave it to you to figure out how to construct such a session file with a known session ID and how to use a session file saved in Perl.[2] You might have similar exercises with other language bindings.

[1] www.vmware.com/support/developer/viperltoolkit/doc/perl_toolkit_guide_idx.html
[2] Hint: Consider the load_session() method. Check out the VI Perl programming guide, pages 35–36, for details (www.vmware.com/support/developer/viperltoolkit/viperl16/doc/viperl_proggd.pdf).

As mentioned, passing the session ID poses a security risk especially over the wire. The session ID is enough for a client to carry over all the privileges of the original user—whatever operation the user can do, the consumer client is able to do the same. This session cookie can also be used to access files over HTTP. Even worse, the session ID does not expire, so it allows enough time to prepare an attack. To avoid security issues, carefully consider whom to share a session with and how to send the session ID.

Using Single Sign-On from the VI SDK to CIM

When a client connects to the ESX server via the VI SDK, it has to log in. If it wants to further access the Common Information Model (CIM), it has to log in again using the same username and password. So the client has to either get input from users again or save the username/password somewhere. Neither of these two approaches is good for system security.

In some cases, like VI Client plug-in development, the Web application gets only the session string. It's impossible to get the password, which presumably can also be used for CIM access. So it is necessary for the VI SDK to provide a mechanism to issue tokens that can be used for CIM service login.

Starting with VI 3.5, VMware provides a new API to make possible single sign-on from the VI SDK to CIM. The following discusses how to get it done with sample Java code.

VI SDK 2.5 provides `acquireCimServicesTicket()` on a `HostSystem` managed object to get a `HostServiceTicket` object. The definition of the `HostServiceTicket` data object and related classes are shown in Figure 18-3. Some of its properties, such as `host`, `port`, and `sslThumpprint`, could be `null` because the API reference indicates "need not to be set."

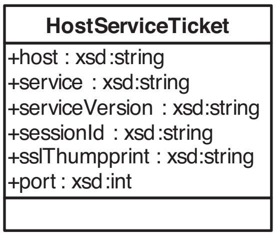


Figure 18-3 The `HostServiceTicket` data object class and related classes

The `sessionId` is a string with a format as follows. It can be used as both username and password for CIM access login.

```
5259c389-9891-c650-b108-e10a0ff5c781
```

Now let's look at a Java program that shows how it can be done (see Listing 18-1).

Listing 18-1

`CimTicket.java`

```
package vim25.samples.mo.cim;
import java.net.URL;
import java.util.Enumeration;

import org.sblim.wbem.cim.CIMNameSpace;
import org.sblim.wbem.cim.CIMObject;
import org.sblim.wbem.cim.CIMObjectPath;
import org.sblim.wbem.client.CIMClient;
import org.sblim.wbem.client.PasswordCredential;
import org.sblim.wbem.client.UserPrincipal;

import com.vmware.vim25.HostServiceTicket;
import com.vmware.vim25.mo.Folder;
import com.vmware.vim25.mo.HostSystem;
import com.vmware.vim25.mo.InventoryNavigator;
import com.vmware.vim25.mo.ServiceInstance;

public class CimTicket
{
  public static void main(String[] args) throws Exception
  {
    if(args.length!=3)
    {
      System.out.println("Usage: java CimTicket <url> " +
          "<username> <password>");
      return;
    }
    String urlStr = args[0];
    String username = args[1];
    String password = args[2];

    ServiceInstance si = new ServiceInstance(new URL(urlStr),
        username, password, true);
    Folder rootFolder = si.getRootFolder();

    HostSystem host = (HostSystem) new InventoryNavigator(
        rootFolder).searchManagedEntities("HostSystem")[0];

    System.out.println(host.getName());
    HostServiceTicket ticket = host.acquireCimServicesTicket();
    System.out.println("\nHost Name:" + ticket.getHost());
    System.out.println("sessionId=" + ticket.getSessionId());
    System.out.println("sslThumpprint="
        + ticket.getSslThumbprint());
    System.out.println("serviceVersion="
        + ticket.getServiceVersion());
    System.out.println("service=" + ticket.getService());
    System.out.println("port=" + ticket.getPort());
    retrieveCimInfo(urlStr, ticket.getSessionId());
    si.getServerConnection().logout();
  }

  private static void retrieveCimInfo(
      String urlStr, String sessionId)
  {
    String serverUrl = urlStr.substring(0,
        urlStr.lastIndexOf("/sdk"));
    String cimAgentAddress = serverUrl + ":5989";
    String namespace = "root/cimv2";
    UserPrincipal userPr = new UserPrincipal(sessionId);
    PasswordCredential pwCred = new PasswordCredential(
        sessionId.toCharArray());

    CIMNameSpace ns = new CIMNameSpace(
        cimAgentAddress, namespace);
    CIMClient cimClient = new CIMClient(ns, userPr, pwCred);
    CIMObjectPath rpCOP = new CIMObjectPath(
        "CIM_RegisteredProfile");

    System.out.println("Looking for children of " +
        "CIM_RegisteredProfile");

    long enumerationStart = System.currentTimeMillis();
    Enumeration rpEnm = cimClient.enumerateInstances(rpCOP);
    long enumerationStop = System.currentTimeMillis();
    System.out.println("Enumeration completed in: " +
      (enumerationStop - enumerationStart) / 1000 + " sec.\n");

    while (rpEnm.hasMoreElements())
    {
      CIMObject rp = (CIMObject) rpEnm.nextElement();
      System.out.println(" Found: " + rp);
    }
  }
}
```

The console printout is shown here. Given the size limit, only the first several lines are listed:

```
test.acme.com
Host Name:test.acme.com
sessionId=5259c389-9891-c650-b108-e10a0ff5c781
sslThumpprint=null
serviceVersion=1.0
service=CimInterfaces
port=null
Looking for children of CIM_RegisteredProfile
Enumeration completed in: 0 sec.

 Found: instance of OMC_RegisteredSensorProfile {
      string AdvertiseTypeDescriptions[];

      uint16 AdvertiseTypes[] = {3};

      string Caption;

      string Description;
```

This API is supported by default in ESXi 3.5, but not in classic ESX. Users can manually enable it by tweaking a configuration file in classic ESX. In classic 3.5 U2, it's enabled as default. You can also use this API with VC server as long as the `HostSystem` it manages supports the feature.

The SBLIM Java client[3] is used instead of the one[4] used with the VI SDK CIM sample. The latter has a bug whereby it truncates the password after 16 characters, so it fails the CIM login.

Downloading and Uploading Files Using HTTP Access

In VI 3.5, VMware introduced a new feature to enable applications to download a file from and upload a file to the datastores of ESX servers.

The syntax of the URLs is as follows:

```
http(s)://<hostname>/folder[/<path>]?dcPath=<datacenter_path>[&dsName=<datastore
_name>]
```

Following are some sample URLs:

```
https://18.17.218.228/folder?dcPath=Datacenter
```

which lists all the datastores in the datacenter whose path is `Datacenter`.

```
https://18.17.218.228/folder?dcPath=Datacenter&dsName=storage1%20(1)
```

which lists all the folders in the datastore named `storage1 (1)` whose path is `Datacenter`.

```
https://18.17.218.228/folder/SuSe_server10/SuSe_server10-
flat.vmdk?dcPath=Datacenter&dsName=storage1%20(1)
```

which points to the `vmdk` file called `SuSe_server10-flat.vmdk`.

[3] http://sblim.wiki.sourceforge.net/CimClient
[4] http://sourceforge.net/projects/wbemservices

VMware has shipped a sample code with VI SDK 2.5 showing how to upload a virtual machine to an ESX server (`com.vmware.samples.httpfileaccess.ColdMigration`). The sample code works by uploading files whose sizes are 40MB or smaller. In most of the cases, virtual machine disk files are much bigger than 40MB; therefore, an `OutOfMemoryError` is almost always thrown for such an execution.

Given the average virtual disk size, the `ColdMigration` sample is almost useless if it cannot work with files bigger than 40MB.

An easy solution seems to be increasing the Java heap size using a parameter to the Java virtual machine. But that is not a good solution given the size of the virtual machines, sometimes 100GB or bigger. It would be hard to find such a large memory and assign it to a Java virtual machine. So let's pin down the root cause of the problem and find an alternative solution.

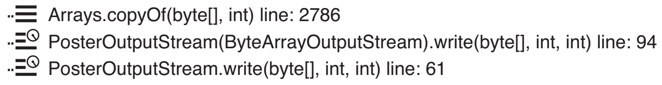
After debugging the sample, the following section of code is found to throw exceptions.

```
OutputStream out = conn.getOutputStream();
FileInputStream in = new FileInputStream(
   new File(localFilePath));

byte[] buf = new byte[1024];
int len = 0;
while ((len = in.read(buf)) > 0) {
   out.write(buf, 0, len);
}
conn.getResponseMessage();
conn.disconnect();
out.close();
```

Simply reading the code, it seems okay. To figure out in which iteration the problem happens, a conditional breakpoint is set to catch `OutOfMemoryError`.

The next run reveals the stack shown in Figure 18-4 when the exception happened.



```
Arrays.copyOf(byte[], int) line: 2786
PosterOutputStream(ByteArrayOutputStream).write(byte[], int, int) line: 94
PosterOutputStream.write(byte[], int, int) line: 61
```

Figure 18-4 Partial calling stack when `OutOfMemoryError` happens

The error was thrown at the following highlighted line. The argument `newLength` is 67,108,864. No wonder we have a problem with this huge array.

```
public static byte[] copyOf(byte[] original, int newLength)
{
  byte[] copy = new byte[newLength];
  System.arraycopy(original, 0, copy, 0,
       Math.min(original.length, newLength));
  return copy;
}
```

Further reading discloses that `PostOutputStream(ByteArrayOutputStream).write()` tries to buffer all the data to be sent. It's not going to work with big size uploading.

The question becomes whether it can use a different output class. The `PosterOutputStream` is returned from the `getOutputStream()` method:

```
public synchronized OutputStream getOutputStream()
    throws IOException
{
  return delegate.getOutputStream();
}
```

Because the source code of Sun's `HttpURLConnection` class is not available in the `src.zip`, a search on the Web finds code samples like the following. It shows that the `getOutputStream()` method can actually return subtypes of `OutputStream` other than `PosterOutputStream` depending on the variable `fixedContentLength` and `chunkLength`.

```
public synchronized OutputStream getOutputStream()
{
...
  if(streaming())
  {
   if(fixedContentLength != -1)
     strOutputStream = new StreamingOutputStream(
       ps, fixedContentLength);
   else if(chunkLength != -1)
     strOutputStream = new StreamingOutputStream(
       new ChunkedOutputStream(ps, chunkLength), -1);
   return strOutputStream;
   }
...
   if(poster == null)
        poster = new PosterOutputStream();
   return poster;
 }
```

By exploring the HttpURLConnection class, you can find the methods to set the two variables: setFixedLengthStreamingMode(int) and setChunkedStreamingMode(int), respectively.

Because the file size is known beforehand, just use the first method and get StreamingOutputStream. It works fine by inserting the following line before conn.getOutputStream():

```
conn.setFixedLengthStreamingMode(fileSize);
```

The issue with the sample code is not a bug of VMware Infrastructure per se. The root cause of the issue is the misuse of Java APIs.

You could argue that PosterOutputStream should *not* be the default OutputStream. In practice, uploading a huge file is not a typical use case of the HttpConnection. Most API users use it to download content of any size and upload a relatively small file. After all, the HttpURLConnection class has provided an alternative to solve the problem.

Note that the argument to setFixedLengthStreamingMode is an integer, meaning it can only hold up to about 2.14GB, which is not enough for a disk file. Uploading bigger files requires using setChunkedStreamingMode(int). As of SDK 2.5, the chunked streaming is not supported on the ESX server side.

Multithreading with the VI SDK

Multithreading is the norm in application development, especially GUI-related applications and server applications. In these applications, you might need several threads, each of which is assigned a specific task. When you develop your application that uses VI SDK, you need to consider using multithreading.

Overall, the AXIS generated VI SDK stubs are thread-safe, meaning you can safely issue multiple calls in different threads. For each call, a new Call object is created and not shared with other method invocations.

While using multiple threads, be aware (and careful) of the following:

- There normally shouldn't be more than one operation currently running with one entity. Most of the invocations are fast, so you don't notice there is such a limitation. But some operations are slow. When a second call comes in before the first finishes, a `TaskInProgress` fault might be thrown. In that fault type, you get the first running task's `MOR`.

  To avoid this, you can put a `synchronized` keyword on every operation in your Java code. This prevents the same application from issuing the second call on the same entity before the first finishes. The `synchronized` keyword cannot prevent other applications, either running on the same machine or not, from issuing a second call to the same managed entity on the server side. When that happens, you can do nothing about it but catch the fault and try again. The lock is essentially on the server side.
- Pay extra attention to the `PropertyCollector`. The `waitForUpdate()` defined there is a synchronized method that reports the result as specified by the `PropertyFilter` objects from version to version. You get all the results for all the `PropertyFilter` objects. If you have multiple threads calling the `waitForUpdate()`, you get multiple duplicated updates. It then makes sense to have one backend thread to call the `waitForUpdate()`.

  You can implement the Publisher-Subscriber design pattern, in which the backend thread is the publisher and any other objects/threads are subscribers who can receive notifications when interested updates come. This implementation requires more effort, but it is worthwhile in big-server applications.

Versioning

The VI SDK exposes the features of the VMware Infrastructure. With the rapid changes of the VI, the VI SDK has evolved accordingly. As it stands today, there are three major versions of the VI SDK: 2.0, 2.5, and 4.0 (a.k.a vSphere SDK). The version 2.0 is not compatible with the other two.

Namespace

As discussed earlier, the most important component in the SDK is the WSDL file from which the client stubs can be generated. The two versions' WSDLs use two different namespaces: VI SDK 2.0 uses `urn:vim`, and VI SDK 2.5 uses `urn:vim25`. Interestingly, the VI SDK 2.5 package also includes a previous version of WSDL. Therefore, you can use the VI SDK 2.5 package to develop applications with 2.0 interfaces.

When client-side stubs are generated, the package names are up to the developers. The pregenerated stubs use `com.vmware.vim` in SDK 2.0 and `com.vmware.vim25` in SDK 2.5. If your application needs to work with two versions of platforms, you have two package names even though the class definitions are similar, if not the same.

Tying the version with a namespace complicates the application development. If you have to work with two versions of VI in your application, you must include two JARs. These JARs actually include many duplicated classes that are essentially the same, even though they end up in two package names. For example, there are two `ManagedObjectReference` types. When you enter the type name, the compiler cannot easily decide which one to use. To avoid the confusion, just include the full package name in your code.

The following code detects the version of the target. The basic idea is to get the `vimService.wsdl` file, as shown in Listing 18-2, with the following URL, and then parse the XML file for the version number.

```
https://<server-name>/sdk/vimService?wsdl
```

When you want to discover the namespace of a target server, simply call the utility like the following. If the target supports SDK 2.5, the version is `urn:vim25Service`.

```
String version = VerUtil.getTargetNameSpace("192.168.143.209");
```

Listing 18-2
`VerUtil.java`

```
package vim25.samples.mo.util;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.security.KeyManagementException;
import java.security.NoSuchAlgorithmException;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLSession;
import javax.net.ssl.TrustManager;
import javax.net.ssl.X509TrustManager;

public class VerUtil
{
  /**
   * Retrieve the target server's namespace
   * @param target, either IP or host name
   * @return the namespace, e.g. urn:vim25Service
   */
  public static String getTargetNameSpace(String target)
  {
    String version = "";
    try
    {
      trustAllHttpsCertificates();
      HttpsURLConnection.setDefaultHostnameVerifier(
          new HostnameVerifier()
          {
            public boolean verify(String urlHostName,
                SSLSession session)
            {
              return true;
            }
          });

      String urlStr = "https://"+ target
          + "/sdk/vimService?wsdl";
      HttpURLConnection conn = (HttpURLConnection) new URL(
          urlStr).openConnection();
      conn.connect();
      BufferedReader in = new BufferedReader(
          new InputStreamReader(conn.getInputStream()));
      StringBuffer xmlWSDL = new StringBuffer();
      String line;
      while ((line=in.readLine())!= null)
      {
        xmlWSDL.append(line);
      }

      int start = xmlWSDL.indexOf("targetNamespace")
                     + "targetNamespace".length();
      start = xmlWSDL.indexOf("\"", start);
      int end = xmlWSDL.indexOf("\"", start+1);
      version = xmlWSDL.substring(start+1, end);
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    return version;
  }

  private static void trustAllHttpsCertificates()
    throws NoSuchAlgorithmException, KeyManagementException
  {
    TrustManager[] trustAllCerts = new TrustManager[1];
    trustAllCerts[0] = new TrustAllManager();
    SSLContext sc = SSLContext.getInstance("SSL");
    sc.init(null, trustAllCerts, null);
    HttpsURLConnection.setDefaultSSLSocketFactory(
        sc.getSocketFactory());
  }

  private static class TrustAllManager
    implements X509TrustManager
  {
    public X509Certificate[] getAcceptedIssuers()
    {
      return null;
    }
    public void checkServerTrusted(X509Certificate[] certs,
        String authType)
      throws CertificateException
    {
    }
    public void checkClientTrusted(X509Certificate[] certs,
        String authType)
    throws CertificateException
    {
    }
  }

  public static void main(String[] args)
  {
    String ver = getTargetNameSpace("10.20.143.205");
    System.out.println("ver:" + ver);
  }
}
```

Compatibility

Can the applications developed with 2.0 still work with the newer VI product? Yes. The newer version of VI supports both the current and older versions of WSDL generated stubs.

Because the application still uses the old interfaces, they cannot, however, retrieve the new properties and call the newly added methods. So how can you access new properties and methods?

There could be two different solutions. First, rewrite the application to the newer version of SDK. Then the application no longer works with lower versions of VI platforms. For example, the application built on top of SDK 2.5 does not work with ESX 3.0 or VirtualCenter 2.0.

The second solution is to keep the old code as it is and choose to access new properties and new methods after detecting that the target is newer. Of course, the logic could be more complicated and the code could be less straightforward. But the gain would be application compatibility, which allows it to work with both older and newer versions of VI.

In the second solution, to access the newly defined properties or methods, you must convert the old `ManagedObjectReference` to the newer `ManagedObjectReference`. Because the definitions are the same except for the package name and name space, the conversion is straightforward. With the new `MOR` object, you can retrieve new properties and invoke new methods.

There is an easier way: Put all the ESX, old or new versions, under the newer VC2.5 and then connect to it. VC 2.5 handles the versioning for you, and you don't need to worry about it. Only VC needs an upgrade. The newer version of VC can manage the old version of ESXes.

API Deprecation

With the evolution of the VI SDK, some of the interfaces are deprecated in favor of new ones. As a simple naming convention, the new interfaces normally have an `Ex` suffix in their names. For example, the old interface to create a cluster was `createCluster()`, and the new interface is `createClusterEx()`.

These new interfaces normally come with new data objects with similar `Ex` suffixes. The `createClusterEx()` method, for example, has a new parameter type `ClusterConfigSpecEx` instead of `ClusterConfigSpec` for the old method.

Because of interface changes, some of the managed objects might have new properties of new types. For instance, the `ComputeResource` has a `configrationEx` property of `ClusterConfigInfoEx`.

Although for compatibility reasons the new interfaces are still supported, you should use the new interfaces especially for new development for better compatibility and more features.

Note that not all the deprecated methods have replacements. The `destroyNetwork()` method is such a case. When the network is no longer in use, the system removes it automatically like the garbage collection in Java, thereby making it unnecessary.

Following Best Practices for Performance and Scalability

This section is not intended to be a general guide for improving performance and scalability; instead, it focuses on how to get better performance and scalability from the VI SDK.

The general principles for performance and scalability still hold; for example, don't optimize your code unless you have to.

When designing a VI SDK application, consider the following:

- Use VirtualCenter instead of individual hypervisors as a target server for the VI SDK. VirtualCenter has more functionalities than ESX. From a scalability point of view, your application can scale with VirtualCenter.

  If your application tries to manage 100 hosts, for instance, you must have 100 connections if you're talking to individual hosts. When a virtual machine is moved from one host to the other, you have to track it down from host to host. If you use a VirtualCenter managing these ESXes, you can shift the burden to the VirtualCenter. One connection to VirtualCenter saves almost all the tedious work for you.

  There is a limitation with VirtualCenter server in terms of the number of hypervisors and virtual machines it can manage.[5] Your application can always connect to multiple VirtualCenter servers to scale beyond one VirtualCenter coverage.

- Use as few sessions as possible. The sessions take system resources and use locks on the server side. The slowdown ultimately affects all the VI SDK clients in that the calls to the server are slower to return. This is, of course, out of the control of a single client. Even if your client behaves perfectly well, it might still be affected by others.

  If your application is deployed with many concurrent clients, the one-client, one-session approach doesn't scale, especially when your target is VirtualCenter. Instead, consider having your own backend server that connects to a server with a single session.

- Avoid a big dataset in a single call. Most of the time, you should be fine. It can be a problem when it comes to retrieving performance data, which could be several megabytes of data returned. This puts the pressure on both the server and the client, where the data has to be marshaled to and unmarshaled from SOAP XML. If the client side uses the DOM parser, it also uses a lot of memory.

  In general, specify as much criteria as possible to restrict your dataset as small as possible. In the performance statistics case, you should use the CSV format over the array for better performance.

[5] Normally 200 ESX hosts and 2,000 virtual machines in VI3. When the clustering feature is on, numbers will be fewer.

- Use batch processing methods when possible. Some operations can have batch processing in which multiple entities can be manipulated. For example, the `ClusterComputeResource` has two methods: `moveHostInto_Task()` and `moveInto_Task()`. The former moves one host into a cluster at a time, and the latter moves multiple hosts at a time. The latter works faster, in that it causes fewer rounds of communication.

  There is a problem with batch processing methods in the VI SDK: they are not atomic. If something goes wrong, the VI SDK just stops there and returns. Whatever is or is not yet processed stays as it is. The VI SDK doesn't roll back the already processed one. So when faults happen, you need to take care of the half-baked cake by yourself. Or, just go with the one-call, one-entity scheme. This is a trade-off between performance and atomicity.

You can always pass in one item in a method that expects an array of items; just avoid the complexity to handle the atomicity. For example, you want performance data for a list of managed entities, but you don't know which of them might no longer be valid. Instead of retrieving them all, you can simply retrieve one at a time. Just remember to catch the exception so that it doesn't exit the loop.

- Design and implement your local cache. It's not worthwhile if it is a simple utility application. But for a big application that requires extensive interaction with VirtualCenter or ESX, it makes a lot of sense.

  First, your application can have instant access of cached information. Second, it saves the number of calls to the server as well as the workload on the server. The same server can work faster and serve more clients.

  Whenever you have a cache and care about freshness, you must consider how to sync it with the server. The VI SDK has a `waitForUpdate()` method that blocks the current thread and returns when updates come up on the server. Clearly, you shouldn't have multiple threads waiting for update, but one to keep the cache synchronized with the server. Be careful with the synchronization of multiple threads on the client application.

  VI Java API 2.0 includes a caching framework that handles most of the burden for you. All you need to do is specify what properties to cache and monitor on what managed objects, and then retrieve the properties in the same way as from a hash table. It's multithread safe and can be used in large applications.

The caching framework is designed for caching, mainly speeding up the second time retrieval. A big company reported three times performance gain by retrieving the properties using the caching framework even the first time. Nice surprise.

Considering Internationalization

With today's global market, a software vendor has to consider the internationalization (I18N) issue to better serve users in different areas and maximize the return on the product investment.

There are two basic meanings. First, you have to design your software so that it is localizable. In other words, you have to use the right APIs that can handle double byte characters. Sometimes people call this globalization (G11N).

Second, you should provide localized versions of your software so that users can read and use their native languages. Sometimes people call this localization.

In most cases, you externalize all the text strings that are visible to end users from the code to the resource files and translate them into different languages. Then localizing the software is as easy as combining the code and localized resource files. This is the way VirtualCenter server is localized. Depending on the programming language and platform, the resource files can be organized differently and might have another format. For example, Java uses properties files, yet C++ on Windows uses resource dlls.

That said, I18N is a broad topic that does much more than what is briefly covered here. Further discussion is beyond the scope of this book, but you can find more detailed information online.

As discussed, the VI SDK is essentially a set of Web Services interfaces. The WS-I18N[6] summarizes four internationalization patterns[7] that can be applied with Web Services when deployed.

[6] www.w3.org/TR/ws-i18n/

[7] Copyright © 2008 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231

- **Locale neutral**—Most aspects of most services are not particularly locale affected. For example, a service that adds two integers is locale neutral.
- **Data driven**—Aspects of the data determine how it is processed, rather than the configuration of either the requester or the provider.
- **Service determined**—The service has a particular setting built into it. For example, this service always runs in the French for France locale. Or, commonly, the service will run in the host's default locale. It may even be a deployment decision that controls which locale or preferences are applied to the service's operation.
- **Client influenced**—The service's operation can use a locale preference provided by the end user to affect its processing. This is called "influenced" because not every request may be honored by the service. (The service may only implement behavior for certain locales or international preference combinations.)

If the VI SDK has to be put into a category, it's client influenced because the service provider tracks the client locale and responds accordingly. The VI SDK is indeed complicated, and most of the services and properties of managed objects are locale neutral.

When you first log into the system using the `SessionManager`, you can provide your locale so that the server knows your locale and responds in successive requests/responses. Your locale is held in the `currentSession.locale` of `SessionManager`.

The locale mainly affects the properties in some managed objects, such as the description properties of `TaskManager`, `AlarmManager`, `AuthorizationManager`, `EventManager`, `PerformanceManager`, and `ScheduledTaskManager`. Although they are all named `description`, they are different data object types inherited from the `Description` data object that includes two string properties: `label` and `summary`. These two properties can be displayed as they are at the client side; therefore, they should be localized.

Besides the subtypes, the `Description` data object can be included in other data objects. For example, the `DiagnosticManagerLogDescriptor` data object, which is a return type of the `queryDescriptions()` method defined in `DiagnosticManager`, includes `Description` as a localized description.

If your applications need to display any description about the task, alarm, and so on, you should always get it from the description properties. This not only saves you the time to write the descriptions, but it saves you the time to translate

Index