VMWARE® VI AND VSPHERE SDK
The author works for VMware as a senior member of the technical staff. The opinions expressed here are the author's personal opinions. Content published here was not read or approved in advance by VMware and does not necessarily reflect the views and opinions of VMware. This is the author's book—not a VMware book.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact

International Sales
international@pearson.com

Visit us on the Web: informit.com/ph

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

Copyright © 2010 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax (617) 671-3447

The VMware VI SDK is Copyright © 2008 VMware, Inc. All Rights Reserved. The VI Java APIs are open-source software and are governed by the BSD license.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL VMWARE, INC. OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Text printed in the United States on recycled paper at R.R. Donnelly in Crawfordsville, Indiana.
First printing September 2009
To Maggie and Melody
This page intentionally left blank
# Table of Contents

## Preface

### CHAPTER 1: VMware Infrastructure Overview

Introducing OS Virtualization
  - What Virtualization Is
  - History of Virtualization
  - Benefits
  - Solutions

VMware Products
  - VMware Infrastructure
  - ESX Server
  - VirtualCenter Server
  - VI Client

VMware Management APIs
  - VI SDK
  - CIM APIs
  - VIX API
  - GuestSDK
  - VMware VMCI
  - Legacy APIs
  - Other APIs

## CHAPTER 2: VI SDK Basics

Overview of the VI SDK
  - What Is Included in VI SDK 2.5?
  - Object Model
  - Unified Interface with Different Support

Managed Objects
  - Inheritance Hierarchy
  - Inventory Hierarchy
  - Properties
  - Methods
  - Synchronous versus Asynchronous Methods

ManagedObjectReference

Data Objects
  - Property Path Notation
  - Enumeration Types
  - Fault Types

Using Tools to Become Familiar with Object Model
  - VI Client
  - Web Access
  - Managed Object Browser (MOB)
# Table of Contents

Web-Based Datastore Browser 45  
Using the API Reference 46  
VMware Online Communities and Conferences 47

**CHAPTER 3: HELLO VI 49**  
Setting Up the Environment 49  
Setting Up the VMware Infrastructure 49  
Installing a Java JDK 50  
Downloading the VI SDK, AXIS, and the VI Java API 50  
Setting Up the Eclipse IDE 51  
Creating Your First Application 52  
Debugging VI SDK Applications 58  
Using Logs 58  
Monitoring SOAP Messages with HTTP 62  
Common Bugs 62

**CHAPTER 4: USING PROPERTYCOLLECTOR AND SEARCHINDEX 66**  
PropertyCollector 67  
Getting Properties from a Single Managed Object 67  
Getting Properties of Multiple Managed Objects 73  
Monitoring Changes Using PropertyFilter 78  
SearchIndex 86  
When to Use Which 91

**CHAPTER 5: INTRODUCING VI JAVA API 93**  
Challenges with the VI SDK 94  
Lack of Managed Object Types 94  
Confusing Method Signatures 95  
Extra Long Code for Its Purpose 95  
Design Objectives 95  
Architecture Overview 97  
Layered Structure 97  
Object Hierarchy 98  
Yet Another Hello VM 99  
How to Use the VI Java API 101  
Typical Application Flow 101  
Switching Between the API and Web Services 103  
Getting Help 104  
Key Managed Object Types 105  
ManagedObject 106  
ServerConnection 122  
ServiceInstance 125  
ExtensibleManagedObject 136  
ManagedEntity 137  
Utility Classes 141  
MorUtil 141  
InventoryNavigator 144  
Other Classes 149  
High-Performance Web Services Engine 149  
Motivation 150  
Architecture 150

**CHAPTER 6: MANAGING INVENTORY 153**  
Inventory Hierarchical Structure 153  
Managed Entities 156  
ManagedEntity 156  
Folder 160  
Datacenter 161  
Basic Inventory Operations 162  
Adding a Folder or a Datacenter 163
# Table of Contents

Deleting an Existing Entity 164  
Renaming an Existing Entity 165  
Moving Existing Entities 167  
Managing Custom Fields 168  
Using Views 170  
ViewManager 170  
View and ManagedObjectView 171  
ContainerView 171  
InventoryView 172  
ListView 172  
Sample Code 173

## Chapter 7: Managing Host Systems 179

HostSystem Managed Object 179  
Retrieving Information 182  
Host Capabilities 182  
Runtime Information 183  
Configuration 183  
Hardware Information 184  
Quick Summary 186  
Power Management 186  
Querying the Memory Requirement for Virtual Machines 189  
Managing Connections 189  
Configuring the Host 190  
Configuring Auto Start/Stop of Virtual Machines 191  
Managing the Time of the Host 192  
Configuring the Booting Device 196  
Configure the Diagnostic Partition 196  
Configuring the CPU Scheduler 198  
Managing Firmware 199  
Managing the Health Status 200  
Managing the Host Memory 200  
Patch Management 202

## Chapter 8: Managing Virtual Machines, Snapshots, and VMotion 204

Virtual Machine 204  
Files of a Virtual Machine 205  
VirtualMachine Managed Object 206  
Retrieving Information 209  
Virtual Machine Capabilities 209  
Virtual Machine Configuration 209  
Runtime Information 210  
Guest Operating System Information 212  
Quick Summary 213  
Virtual Machine File Layout 214  
Resource Allocation 215  
Reconfiguring Virtual Machine 215  
Power Operations 222  
Managing the Virtual Machine Life Cycle 226  
Creating a New Virtual Machine 226  
Registering a Virtual Machine 231  
Cloning a Virtual Machine 232  
Destroying a Virtual Machine 234  
Converting To/From Templates 235  
Upgrading a Virtual Machine 235  
Unregistering a Virtual Machine 236  
Managing the Guest Operating System 236  
Upgrading VMware Tools 237  
Customizing the Guest OS 239  
Managing the Customization Spec 240
Table of Contents

Managing Virtual Machine
  Snapshots 243
  Snapshot Hierarchy 243
  Creating, Restoring, and Removing Snapshots 244
  VirtualMachineSnapshot Managed Object 245
  Migrating Live Virtual Machines (VMotion) 251
  Storage VMotion 254

CHAPTER 9: MANAGING CLUSTERS AND RESOURCE POOLS 257
  Managing Resources with ResourcePool 257
  Virtual Resource Management Basics 258
  ResourcePool Managed Object 259
  DRS and HA Clustering 262
    High Availability 262
    Distributed Resource Scheduler 263
  ComputeResource Managed Object 264
  ClusterComputeResource Managed Object 268
  Managing Clusters 270
    Creating a New Cluster 270
    Adding Hosts to a Cluster 270
    Reconfiguring a Cluster 271
    Removing a Host from a Cluster 272
    Removing a Cluster 272
  Interacting with the DRS Cluster 272
  Querying the Environment for Virtual Machines 278

CHAPTER 10: MANAGING NETWORKING 283
  Key Concepts 283
    Virtual Switch 284
    Port Group 284
    Virtual NIC 284
  HostNetworkSystem Managed Object 285
    Properties 285
    Overview of Methods 287
  Managing Virtual Network Components 289
    Adding a Virtual Switch 290
    Adding a New Port Group to a Virtual Switch 290
    Adding a Virtual NIC (VNIC) 290
  Managing Service Console Networking 292
  Configuring the Host Networking 293
    Changing the IP Routing Configuration 293
    Changing the DNS Configuration 293
    Changing the Network Configuration 293
    Defining the Host Network Policy 294
  Network Managed Object 295
  Managing SNMP with HostSnmpSystem 296
  Managing the Firewall with HostFirewallSystem 299
  Managing Network Services with HostServiceSystem 304
  Configuring VMotion Network Using HostVMotionSystem 308
# Table of Contents

## CHAPTER 11: MANAGING STORAGE AND DATASTORES 310
- Key Concepts 311
  - SCSI 311
  - SAN 312
  - iSCSI 312
  - HBA 312
  - VMFS 313
  - NAS 313
- HostStorageSystem Managed Object 313
- HostStorageDeviceInfo 314
- HostFileSystemVolumeInfo 315
- Discovering HBAs 321
- Managing iSCSI 322
  - Manipulating Discovery List 322
  - Configuring iSCSI 323
- Managing Disk Partitioning 324
- Managing the VMFS File System 326
- Managing Storage Multipath 327
- Managing Datastores Using HostDatastoreSystem 328
  - Creating a New Datastore 328
  - Extending a VMFS Datastore 331
  - Removing an Existing Datastore 332
  - Managing the Swap Datastore 332
  - Configuring the Datastore Principal 332
- Datastore Managed Object 333
- Search Datastore with HostDatastoreBrowser 335
- Managing Datastores with FileManager 341
- Managing Virtual Disks with VirtualDiskManager 344

## CHAPTER 12: EVENTS AND ALARMS 349
- Event Data Object 349
- EventManager Managed Object 351
- Retrieving Historical Events with EventHistoryCollector 356
- HistoryCollector Managed Object 357
- EventHistoryCollector Managed Object 358
- Alarm Managed Object 360
  - Triggering Conditions 362
  - Follow-Up Actions 362
  - Reconfiguring an Alarm 365
  - Removing an Alarm 366
- AlarmManager Managed Object 366
  - Creating a New Alarm 374
  - Finding Existing Alarms 377
  - Getting Alarm State 378

## CHAPTER 13: PERFORMANCE MONITORING 379
- Basic Terminologies 379
- Performance Counter 380
- Performance Metric 382
- Intervals 382
- Real-Time versus Historical Performance Statistics 384
- PerformanceManager Managed Object 385
- Querying Performance Metadata 390
- Querying Performance Statistics 391
- Using the queryPerf Method 391
Using the queryPerfComposite Method 394
Monitoring Performance in Real Time 399
Configuring Historical Intervals 405

CHAPTER 14: TASK AND SCHEDULED TASK 407
Task Managed Object 407
  Monitoring a Task 409
  Canceling a Task 410
  Fixing Task Timeout 410
TaskManager Managed Object 411
Retrieving Historical Tasks with TaskHistoryCollector 416
ScheduledTask Managed Object 422
ScheduledTaskManager Managed Object 424
  Specifying Task Schedule 425
  Specifying an Action 425

CHAPTER 15: USER AND LICENSE ADMINISTRATION 432
Security Model 432
  Privileges 433
  Roles 433
  Permissions 434
AuthorizationManager Managed Object 435
  Managing Roles 438
  Managing Permissions 439
  Querying Permissions 440
Looking Up Users with UserDirectory 443
Managing ESX Users with HostLocalAccountManager 445
Managing Sessions Using SessionManager 449
License Management 454

CHAPTER 16: EXTENDING VI CLIENT 465
VMware Infrastructure Extensibility Story 465
ExtensionManager Managed Object 466
How a VI Client Plug-In Works 468
Understanding Extension Configuration 469
Registering and Managing Your Plug-In 471
Developing Your Backend Web Application 479
  Parsing Information from the VI Client 479
  Connecting Back to the VirtualCenter 480
Security Discussion 483

CHAPTER 17: SCRIPTING THE VI SDK WITH JYTHON, PERL, AND POWERSHELL 484
Scripting with Jython 484
  What Is Needed? 485
  HelloVM.py 485
  Further Discussion 488
VI Perl Toolkit 488
Installing VI Perl Toolkit 489
  HelloVM.pl 490
Application Flow Using VI Perl 491
<table>
<thead>
<tr>
<th>Perl View Objects for Managed Entities</th>
<th>492</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using Data Objects</td>
<td>493</td>
</tr>
<tr>
<td>Working with Non-Entity View Objects</td>
<td>494</td>
</tr>
<tr>
<td>Further Reading</td>
<td>495</td>
</tr>
<tr>
<td>VI Toolkit (for Windows)</td>
<td>495</td>
</tr>
<tr>
<td>Installing VI Toolkit (for Windows)</td>
<td>496</td>
</tr>
<tr>
<td>Common Cmdlets</td>
<td>496</td>
</tr>
<tr>
<td>Cmdlet Pipeline</td>
<td>499</td>
</tr>
<tr>
<td>Running the Cmdlets in a Script File</td>
<td>499</td>
</tr>
<tr>
<td>Web Service Access Cmdlets</td>
<td>500</td>
</tr>
<tr>
<td>Further Reading</td>
<td>501</td>
</tr>
</tbody>
</table>

**CHAPTER 18: ADVANCED TOPICS**  502

- Managing Global Settings with OptionManager  503
- Collecting Logs Using DiagnosticManager  505
- Sharing Sessions Among Different Applications  507
- Getting the Session ID  507
- Using Session ID  509
- Further Discussion  510
- Using Single Sign-On from the VI SDK to CIM  511
- Downloading and Uploading Files Using HTTP Access  515
- Multithreading with the VI SDK  518
- Versioning  519
- Namespace  519
- Compatibility  523
- API Deprecation  524
- Following Best Practices for Performance and Scalability  524
- Considering Internationalization  527

**APPENDIX A: THE MANAGED OBJECT TYPES**  530

**APPENDIX B: THE PERFORMANCE COUNTERS**  538

**APPENDIX C: CMDLETS IN THE VI TOOLKIT (FOR WINDOWS)**  554

**APPENDIX D: UNIFIED MODELING LANGUAGE**  580
- Class Diagram  580
- Visibility  581
- Inheritance/Generalization  581
- Association  581
- Object Diagram  582
- Modeling XML  583

**APPENDIX E: VI SDK WEB SERVICES**  585
- SOAP  585
- Two WSDL Files  587
- Stub Code Generation  590
- From Managed Object to WSDL to Generated Stub  592

**APPENDIX F: WHAT IS NEW IN VSphere 4 SDK?**  594
- Changes of Existing APIs  595
- Profile Management  595
- Distribute Virtual Switch  596
- Host Management  597
- OVF Support  597
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>VirtualApp Support</td>
<td>598</td>
</tr>
<tr>
<td>New Compatibility Checkers</td>
<td>599</td>
</tr>
<tr>
<td>Licensing Change</td>
<td>600</td>
</tr>
<tr>
<td>Localization Support</td>
<td>600</td>
</tr>
<tr>
<td>Resource Planning</td>
<td>601</td>
</tr>
<tr>
<td>Migrating Your Existing Applications</td>
<td>602</td>
</tr>
</tbody>
</table>
Acknowledgments

I thank VMware founders Dr. Mendel Roseblum, Diane Greene, Scott Devine, Dr. Edward Wang, and Edouard Bugnion for creating the great technology that I can write about today. I also thank the many engineers who have followed their footsteps to push the virtualization technology to state-of-the-art, and the many other colleagues who brought the company to its dominant position in the marketplace.

I’d like to thank Carter Shanklin, Mark Menkhus, Robert D. Petruska, and Cody Bunch for reading early drafts of this manuscript and working hard to improve its clarity and accuracy. I could not have written this book without their help, and any mistakes that remain are my own.

I’d also like to thank those teachers, colleagues, and managers who have done so much for me: Cheng Wu, Jun Li, Liman Deng, Jincai Xu, Wenjian Chen, Jeane Chen, Shauchi Ong, George Hung, Steve Nameroff, Frank Yang, Charles Yan, Bingxue Xu, Yingqiu Sui, Sujit Panikatt, Lawrence Jacobs, Marianna Tessel, Carter Shanklin, Alan Tan, Luke Dion, Jeff Hu, Beng-Hong Lim, Colleen Lam, Pablo Roesch, Gilbert Lau, Melissa Ercoli Cotton, Budianto Bong, Tobin Edwards, Lucas Nguyen, Kimberly Wang, Prasad Pimplaskar, Giampiero Caprino, Andrew Zhu, Frank Li, Mike Chen. Without their support, I would not be where I am in my career today and would never have been in a position to write this book. For their teaching, mentoring, inspiration, and support, I am truly grateful.

I thank Jessica Goldstein, Karen Gettman, Romny French, Sheri Cain, Jovana San Nicolas-Shirley, Karen Gill, Kristy Hart, Doug Ingersoll, Chuti Prasertsith, Jake McFarland, and the entire team at Prentice Hall for their great support and professionalism. I feel so blessed to have worked with the best publishing team.
This page intentionally left blank
Steve Jin is a senior member of technical staff at VMware, where he provides guidance to strategic partners, such as IBM, HP, Dell, NetApp, and BEA, who build applications using VI (vSphere) SDK. In his spare time, he created VI (vSphere) Java API opensource project (http://vijava.sf.net), which is widely used by various commercial companies and developers. Jin received his BA, MS, and Ph.D. degrees in control theory (EE) from prestigious Tsinghua University in Beijing. Prior to his current job, Jin worked at IBM Research, Rational Software, and ASDC in various engineering and management roles.

Jin is the author of two software engineering books published for the Springer Tsinghua Press and the China Electronics Industry Press.
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.
Preface

- **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.


- **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.


- **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 `smtp`,...
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](image)

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.

- **`generateLog Bundles_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:

```java
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();
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:

```plaintext
#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
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 `sslThumbprint`, could be `null` because the API reference indicates “need not to be set.”

![HostServiceTicket](image)

**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`

```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("sslThumbprint="
    + 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
sslThumbprint=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 is used instead of the one 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:

\[
\text{http(s):/}/<\text{hostname}>/\text{folder}[/<\text{path}>]?\text{dcPath=<datacenter\_path>}[&\text{dsName=<datastore\_name>}]\]

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\%(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\%(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.

```java
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.

```
<table>
<thead>
<tr>
<th>Stack Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays.copyOf(byte[], int) line: 2786</td>
</tr>
<tr>
<td>PosterOutputStream(ByteArrayOutputStream).write(byte[], int, int) line: 94</td>
</tr>
<tr>
<td>PosterOutputStream.write(byte[], int, int) line: 61</td>
</tr>
</tbody>
</table>
```

**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.

```java
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:

```java
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.

```java
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;
    }
```
Chapter 18—Advanced Topics

... 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 HttpURLConnection. 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.

```java
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`.

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

Listing 18-2

```java
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.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";
            HttpsURLConnection conn = (HttpsURLConnection) 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[] get AcceptedIssuers()
    {
        return null;
    }
    public void checkServerTrusted(X509Certificate[] certs,
Versioning

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 `configurationEx` 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. 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/](http://www.w3.org/TR/ws-i18n/)

■ **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
them when localized. The only exception is when you want to localize your application to a locale that the VirtualCenter Server does not support.

From the SessionManager, you can find out the server default locale and a list of locales for which the server has localized the messages. You can tell easily whether the server supports your locale.

Beyond these messages, the error message can be localized. The LocalizedMethodFault is a wrapper around MethodFault that provides a localized message for the wrapped fault. As already emphasized, LocalizedMethodFault is not a fault type, but a data object type.

So far, the discussion has mainly focused on the localized information from the server side. You can also send localized text from the client to the server side. For example, you can change the virtual machine name through the VI SDK. Even if you are working with an English-only version of VirtualCenter/ESX server, you can set a virtual machine’s name as a double-byte character string. On the English-only VI Client, you might see just rectangles because they don’t render well. If you open MOB to the virtual machine page, you might see the name show up correctly. That means the double-byte name has been saved correctly and your application can retrieve it using the VI SDK.

In general, you should be a little conservative with sending non-English text to the server from the API. Unless necessary, try to restrict to ASCII characters only. Some of the text should never be localized. For example, the keys in the system that identify certain resources should never be localized. Most of the time, they are not visible anyway.

The VI SDK is only part of your I18N story. To have a truly localized VI SDK application, make sure your client-side language, Web Services stub, and application are ready to handle the double-byte character. If you are using a language such as Java, it’s less of a problem because it’s built from the ground up to handle a double-byte string.

**Summary**

In this chapter we discussed several topics that do not fit into previous chapters but are important for a successful VI SDK application. Among them are how to manage global settings; how to collect logs; how to share sessions; how to download and upload files to datastores; how to sign on from the VI SDK to CIM without a password; how to handle different VI SDK versions; how to use multithreads with the VI SDK; how to achieve performance and scalability; and how to localize your VI SDK applications.
Symbols

# (protected visibility), 581
+ (protected visibility), 581
- (private visibility), 581

A

acquireCimServicesTicket(), 181
acquireLocalTicket(), SessionManager, 452
actions
  Alarm, 362
  specifying with ScheduledTaskManager, 425-426
addAuthorizationRole() method, 438
addCustomFieldDef(), 169
AddDatastore.java, 330-331
addHost_Task(), 270
adding
  Datacenter, inventory operations, 163-164
  Folder, inventory operations, 163-164
  hosts to clusters, 270-271
  virtual NIC, 290-292
  virtual switches, 290
addInternetScsiSendTargets(), 323
AddNIC.java, 290-292
addVirtualNic() method, 290-292
addVirtualSwitch() method, 290
Alarm, 360-361
  creating tasks, 364
  follow-up actions, 362
  invoking methods, 365
reconfiguring alarms, 365-366
removing alarms, 366
running scripts, 363
sending email, 362-363
sending SNMP trap, 364-365
triggering conditions, 362
AlarmInfo, 361
AlarmManager, 366
  alarms, creating, 374
  CreateVmAlarm.java, 374-377
  getAlarmState(), 378
  PrintAlarmManager.java, 367-373
  properties, 366
alarms
  creating, 374
    CreateVmAlarm.java, 374-377
  finding existing, 377
AlarmSpec, 361
AlarmState data object, 378
AlarmTriggeringAction, 362
AndAlarmExpression, 362
answerVM() method, 223
API, 93
API deprecation, compatibility, 524
API reference, 46-47
APIs, 12, 15
  CMI APIs, 14
  GuestSDK, 14
  new features, 595
  switching between Web Services and (VI Java API), 103-104
VI SDK. See VI SDK
VIX API, 14
VMCI, 15
application flow
VI Java API, 101-103
VI Perl Toolkit, 491
applications
creating, 52-55, 58
debugging, 58
common bugs, 62-65
logs, 58-62
monitoring SOAP messages with HTTP, 62
existing migrating applications, 602
architecture
high-performance Web Services engine (VI Java API), 150-151
VI Java API, 97
HelloVM.java, 99-101
layered structure, 97
object hierarchy, 98-99
assignUserToGroup(), 446
association, class diagrams (UML), 581
asynchronous methods versus synchronous methods, 26-27
attachVmfsExtent(), 327
AuthorizationDescription, 435
AuthorizationManager, 435
ListAuthorization.java, 436-438
managing permissions, 439-440
querying permissions, 440-442
managing roles, 438
AuthorizationPrivilege, 435
AuthorizationRole, 435
auto start, configuring for virtual machines, 191-192
auto stop, configuring for virtual machines, 191-192
AXIS, 591
downloading, 50-51
backupFirmwareConfiguration(), 199
batch processing methods, 526
best practices for performance and scalability, 524-526
booting devices, configuring, 196
browseDiagnosticLog(), DiagnosticManager, 506
bugs, VI SDK applications, 62
connection timeout, 63
invalid arguments, 64
invalid state, 65
no permission, 63
null pointer, 62
canceling tasks, 410
cancelTask(), 410
cancelWaitForUpdates(), 78
capabilities, virtual machines, 209
Catalog, 600
catalogName, 600
catalogUri, 601
challenges with VI SDK, 94
extra long code, 95
lack of managed object types, 94
method signatures, 95
changes, monitoring with PropertyFilter, 78-85
check*_Task(), 600
checkCompliance_Task(), 596
checkCustomizationResources(), 241
checkLicenseFeature(), 455
checkMigrate_Task(), 600
checkRelocate_Task(), 600
child entities, 434
childEntity, Folder, 160
childType, Folder, 160
CIM (Common Information Model), 14
single sign-on from VI SDK, 511-515
CimTicket.java, 512-515
class diagrams, UML, 580
association, 581
inheritance, 581
visibility, 581
clearComplianceStatus(), 596
cloneVApp_Task(), 599
CloneVM.java, 233-234
cloneVM_Task(), 232
cloning virtual machines, 232-233
CloneVM.java, 233-234
closeInventoryViewFolder(), 172
INDEX

B
backend Web applications
developing, 479
connecting back to VirtualCenter, 480-482
parsing information from the VI Client, 479-480
security, 483
ClusterComputeResource, 268-270
properties, 268
ClusterProfile, 595
clusters
adding hosts to, 270-271
creating new, 270
DRS, 272-275
DrsApp.java, 275-277
EmailMessenger.java, 277-278
HA (High Availability), 262-263
reconfiguring, 271
removing, 272
removing hosts from, 272
cmdlet pipeline, VI Toolkit (for Windows), 499
cmdlets, VI Toolkit (for Windows), 496-498, 554
running in a script file, 499-500
Web Service Access cmdlets, 500-501
CMI APIs, 14
collecting logs with DiagnosticManager, 505-506
command, 469
Common Information Model. See CIM
compatibility, virtualization, 5
compatibility checkers, 599-600
components of virtual networks, 289
port groups, adding to virtual switches, 290
virtual NIC, adding, 290-292
virtual switches, adding, 290
computeDiskPartitionInfo(), 325
ComputeResource, 155, 264, 266-267
properties of, 264
CloneVM_Task() method, 433
conferences on VMware, 47-48
configurations
host systems, 183-184
virtual machines, 209
configureDatastorePrincipal(), 332
configureLicenseSource(), 456
ConfigureSnmpSystem.java, 298-299
configuring
booting devices, 196
CPUscheduler, 198-199
diagnostic partitions, 196-198
historical intervals, 405-406
host networking, 293
DNS configuration, changing, 293
host network policy, defining, 294-295
IP routing configuration, changing, 293
network configuration, changing, 293
hosts, 190-191
iSCSI, 323-324
VMotion with VMotionSystem, 308
Connect-VIServer (VMware), 498
connecting back to VirtualCenter, 480-482
connection timeout, 63
connections, 189-190
ContainerView, 171
CopyDatastoreFile_Task(), 342
CopyFile.java, 343-344
COS, 292
cost savings, virtualization, 6
counterId, 382
CPU scheduler, configuring, 198-199
createAlarm(), 374
createClusterEx(), 524
createCollectorForEvent(), 352
createCollectorForTasks(), 412
createCustomizationSpec(), 241
createDiagnosticPartition(), 197
CreateFolderDatacenter.java, 163-164
createGroup(), 446
createIpPool(), 599
createLocalDatastore(), 328
createNasDatastore(), 329
createPerfQuerySpec(), 405
createResourcePool(), 260
CreateRole.java, 441-442
createScheduledTask(), 422-424
CreateScheduledTasks.java, 426-430
createSnapshot_Task(), 244
createTask(), 412
CreateTaskAction, 364
CreateUser(), 446
CreateVM.java, 227-231
CreateVmAlarm.java, 374-377
createVmfsDatastore(), 329
createVM_Task(), 227
custom fields, managing, 168-169
CustomFieldsManager, 168-169
customization specifications, guest OS, 240-243
CustomizationSpec, 239
CustomizationSpec data object, 238
CustomizationSpecManager, 240
customizeVM_Task(), 237
customizing guest OS, 239-240
customization specification, 240-243
INDEX

data object, 293
data object type, 27-30
data objects, 30-32
  AlarmInfo, 361
  AlarmSpec, 361
  AlarmState, 378
  CustomizationSpec, 238
  enumeration types, 34-35
  Event, 349-351
  EventFilterSpec, 353
  GuestInfo, 212
  HostDiskPartitionInfo, 325
  HostDnsConfig, 293
  HostFileSystemVolumeInfo, 315-316
  HostFirewallInfo, 302
  HostNicTeamingPolicy, 295
  HostScsiDiskPartition, 327
  HostSnmpDestination, 298
  HostStorageDeviceInfo, 314-315
  PerfCompositeMetric, 394
  PerfCounterInfo, 380
  PerfEntityMetricBase, 393
  PerfInterval, 383
  PerfProviderSummary, 390
  PerfQuerySpec, 392
  property path notation, 33-34
  ResourceConfigSpec, 215
  VI Perl Toolkit, 493-494
  VirtualMachineConfigSpec, 215-216
  VirtualMachineFileLayout, 214
  VirtualMachineSnapshotInfo, 244
  virtualMachineSummary, 214
Data Transfer Object, 30
data types, 591
Datacenter, 161-162
  adding, 163-164
  properties, 161
Datacenter managed object, 155
Datastore, 328, 333-335
  configureDatastorePrincipal(), 332
  FileManager, 341-342
  CopyFile.java, 343-344
  properties, 334
  datastore
    creating new, 328-330
    removing, 332
    searching with HostDatastoreBrowser, 335-340
    swapping, 332
    VMFS datastore, 331-332
  datastoreBrowser, 279
datastoreBrowser, 279
datastoreBrowserSearchResult, 337
  HostDatastoreBrowserSearchSpec, 336
  HostDiskPartitionInfo, 325
  HostDnsConfig, 293
  HostFileSystemVolumeInfo, 315-316
  HostFirewallInfo, 302
  HostNicTeamingPolicy, 295
  HostScsiDiskPartition, 327
  HostSnmpDestination, 298
  HostStorageDeviceInfo, 314-315
  PerfCompositeMetric, 394
  PerfCounterInfo, 380
  PerfEntityMetricBase, 393
  PerfInterval, 383
  PerfProviderSummary, 390
  PerfQuerySpec, 392
  property path notation, 33-34
  ResourceConfigSpec, 215
  VI Perl Toolkit, 493-494
  VirtualMachineConfigSpec, 215-216
  VirtualMachineFileLayout, 214
  VirtualMachineSnapshotInfo, 244
  virtualMachineSummary, 214
disableFeature(), LicenseManager, 457
disableMultipathPath(), 327
disableRuleset(), 302
disconnectHost_Task(), 181
discovery lists, manipulating (iSCSI), 322
disk partitioning, 324-326
distributed virtual switch (DVS), 596-597
distributing virtual switch (DVS), 349
dNS configuration, changing, 293
doesCustomizationSpecExist(), 241
DHCP, COS, 292
diagnostic partitions, configuring, 196-198
DiagnosticManager, collecting logs, 505
  browseDiagnosticLog(), 506
  generateLogBundles_Task(), 506
  queryDescriptions(), 505
Distributed Management Task Force (DTMF), 597
Distributed Resource Scheduler (DRS), 349
doesCustomizationSpecExist(), 241
doesCustomizationSpecExist(), 241
doesCustomizationSpecExist(), 241

downloading
AXIS, 50-51
files, using HTTP Access, 515-518
VI Java API, 50-51
VI SDK, 50-51
DRS (Distributed Resource Scheduler), 188,
257, 263, 349
DRS cluster, 272-275
DrsApp.java, 275-277
EmailMessenger.java, 277-278
DrsApp.java, 275-277
DMTF (Distributed Management Task Force), 597
duplicatecustomizationSpec(), 242
DVS (distributed virtual switch), 596-597

Eclipse IDEs, setting up, 51-52
EmailMessenger.java, 277-278
enableFeature(), LicenseManager, 457
enableMultipathPath(), 327
enableRuleset(), 302
encapsulation, virtualization, 5
encryptionKey, CustomizationSpecManager, 241
energy, reducing consumption through
virtualization, 6
enterMaintenanceMode_Task(), 181
entities, inventory operations
deleting, 164-165
moving, 167-168
renaming, 165-166
enumeration types, data objects, 34-35
EnvironmentBrowser, 278-281
environments
Java JDK, installing, 50
querying for virtual machines, 278-281
VMware infrastructure, setting up, 49-50
estimateDatabaseSize(), 601
ESX
COS, 292
firewall rules, 300-301
fixing task timeouts, 410
PastDay interval, 384
physical NIC, 285
predefined roles, 435
virtual machines, files, 205-206
ESX servers, 10, 18
logs 60-61
ESX users, managing with
HostLocalAccountManager, 445-449
EsxAccountManager.java, 447
ESXi, Firmware, 199-200
esxtop, 379
Event data object, 349-351
EventAlarmExpression, 362
EventArgument, 351
EventFilterSpec, 353
EventHistoryCollector, 358-359
QueryHistoricalEvents.java, 359-360
retrieving historical events with, 356
EventManager, 351-353
createCollectorForEvent(), 352
logUserEvent(), 352
postEvent(), 352
properties, 351
queryEvents(), 352
QueryEvents.java, 354-358
events, historical events (retrieving with
EventHistoryCollector), 356
exitMaintenanceMode_Task(), 181
exportVApp(), 599
extendVmfsDataStore(), 332
extensibility, VMware infrastructure, 465-466
ExtensibleManagedObject, VI Java API, 136-137
ExtensibleManagedObject.java, 136-137
ExtensibleManagedObject, 168
extension configuration, 469-470
extension.properties, 477
ExtensionManager, 466
findExtension(), 466
getPublicKey(), 466
registerExtension, 467
setPublicKey(), 467
unregisterExtension(), 467
updateExtension(), 467
ExtMgrUtil.java, 471-477

fault types, 35-37
in VI SDK 2.5, 36-37
faults, 35-36
file layout, virtual machines, 214, 218
FileManager, 341-342
CopyFile.java, 343-344
files
downloading with HTTP Access, 515-518
uploading with HTTP Access, 515-518
of virtual machines, 205-206
findByDatastorePath(), SearchIndex, 89
findByDnsName(), SearchIndex, 89
findByInventoryPath(), SearchIndex, 90
findByIp(), SearchIndex, 91
findByUuid(), SearchIndex, 91
findChild(), SearchIndex, 89
findExtension(), ExtensionManager, 466
finding alarms, 377
firewall rules, ESX, 300-301
firewalls, HostFirewallSystem, 301-302
   TurnOnFirewallPolicy.java, 303-305
Firmware, 199-200
fixing task timeouts, 410-411
FlipNetworkService.java, 306-308
Folder, 160-161
    adding, 163-164
Folder object, 154
ForEach-Object, 497
formatVmfs(), 326

G
G11N, 527
generalization, class diagrams (UML), 581
generated stubs, 592-593
generateLogBundles_Task(),
   DiagnosticManager, 506
Get-Member, 497
Get-VM (VMware), 498
Get-VMHost (VMware), 498
getAlarmState(), 378
getCustomizationSpec(), 243
GetMultiPerf.java, 395-399
getOutputStream() method, 517
getPublicKey(), ExtensionManager, 466
global settings, managing with OptionManager,
   503-504
       queryOption(), 503
       setting, 503
       supportedOption, 503
       updateOptions(), 504
GroupAlarmAction, 362
guest operating system information, virtual
   machines, 212-213
guest operating systems, 236-237
guest OS, customizing, 239-240
       customization specifications, 240-243
guestHeartbeatStatus property, 212
GuestInfo data object, 212

H
HA (High Availability), 262-263
hardware information, host systems, 184
HBAs, 312, 321-322
health status, 200
HelloVI.java, 55
HelloVM.java, 99-101
HelloVM.pl, VI Perl Toolkit, 490
HelloVM.py, 485-487
help, VI Java API, 104-105
historical structure of inventory, 153-155
historical events, retrieving with
   EventHistoryCollector, 356
historical intervals, configuring, 405-406
historical performance statistics versus real time,
   384-385
HistoryCollector, 357-358
host capabilities, 182
host management, 597
host memory, 200-202
host networking
   configuring, 293
      DNS configuration, changing, 293
      IP routing configuration, changing, 293
      network configuration, changing, 293
   policies, defining, 294-295
host systems
   booting devices, configuring, 196
   retrieving information, 182
      configuration, 183-184
      hardware information, 184
      host capabilities, 182
      runtime information, 183
      summary, 186
   time managing, 192-195
HostAutoStartManager, 191-192
HostConfigInfo, 183-184
HostCpuSchedulerSystem, 198
HostDatastoreBrowser, 335-337
   SearchDatastore.java, 337-340
HostDatastoreBrowserSearchResult data object, 337
HostDatastoreBrowserSearchSpec data object, 336
HostDatastoreSystem, 328
   AddDatastore.java, 330-331
   configureDatastorePrincipal(), 332
datastore
   creating new, 328-330
   removing, 332
   swapping, 332
   VMFS datastore, 331
inventory operations, 162
  adding folders or datacenter, 163-164
deleting existing entities, 164-165
  adding folders or datacenter, 163-164
  moving existing entities, 167-168
  renaming existing entities, 165-166
InventoryNavigator, VI Java API, 144-149
InventoryNavigator.java, 144-149
InventoryView, 172
invoking methods, Alarm, 365
IP routing configuration, changing, 293
IpPoolManager, 599
iSCSI, 322
  configuring, 323-324
  discovery lists, manipulating, 322
iSCSI (Internet SCSI), 312
isolation, virtualization, 5
Java, data types, 591
Java JDK, installing, 50
JEWSE (Just Enough Web Service Engine), 150-151
Jython, scripting, 484-485, 488
  HelloVM.py, 485-487
Key, 469
  key strings, plug-ins, 478
lastModified, 601
levels, performance counters, 381
LicenseAssignmentManager, 600
LicenseManager, 454, 457-458, 600
  PrintLicense.java, 458-462
  properties of, 454
  SetLicenseSource.java, 462-463
licensing, changes to, 600
ListAllUsers.java, 443-444
ListAuthorization.java, 436-438
ListView, 172
locale, 480, 601
localization, new features, 600-601
LocalizationManager, 600
LocalizedMethodFault, 37
login(), SessionManager, 451
loginBySSPI(), SessionManager, 453
logout(), SessionManager, 451
logs
  collecting with DiagnosticManager, 505
  browseDiagnosticLog(), 506
  generateLogBundles_Task(), 506
  queryDescriptions(), 505
debugging VI SDK applications, 58
  ESX server logs, 60-61
  VirtualCenter logs, 59-60
  VM logs, 61-62
  ESX server, 60-61
  VirtualCenter, 59-60
  VM, 61-62
logUserEvent(), 352
looking up users with UserDirectory, 443-445
Maintenance, 187
makeDirectory(), 341
managed entities, 434
  inventory management, 156
  Datacenter, 161-162
    Folder, 160-161
    ManagedEntity, 156, 159-160
  Privileges, 160
Managed Object Browser. See MOB
managed object types
  VI Java API, 105-106
    ExtensibleManagedObject, 136-137
    ManagedEntity, 137-141
    ManagedObject, 106-122
    ServerConnection, 122-125
    ServiceInstance, 125-136
  VI SDK 2.5, 530-535
    vSphere SDK 4, 535-537
managed objects, 19
Alarm. See Alarm
  AlarmManager. See AlarmManager
AuthorizationManager. See AuthorizationManager
ClusterComputeResource, 268-270
ComputeResource, 264-267
ContainerView, 171
CustomizationSpecManager. See
  CustomizationSpecManager
Datastore. See Datastore
  EventHistoryCollector. See EventHistoryCollector
  EventManager. See EventManager
ExtensionManager. See ExtensionManager
HistoryCollector. See HistoryCollector
HostDatastoreSystem. See HostDatastoreSystem
INDEX

HostFirewallSystem, 301-302
   TurnOnFirewallPolicy.java, 303-305
HostLocalAccountManager. See HostLocalAccountManager
HostNetworkSystem. See HostNetworkSystem
HostStorageSystem. See HostStorageSystem
HostSystem. See HostSystem
inheritance hierarchy, 19-21
inventory hierarchy, 21
InventoryView, 172
LicenseManager. See LicenseManager
ListView, 172
ManagedObjectView, 171
methods, 24-26
   synchronous versus asynchronous, 26-27
Network. See Network
PerformanceManager. See PerformanceManager
Properties, 21, 23-24
PropertyCollector. See PropertyCollector
ResourcePool. See ResourcePool
ScheduledTask. See ScheduledTask
ScheduledTaskManager. See ScheduledTaskManager
SessionManager. See SessionManager
Task. See Task
TaskManager. See TaskManager
UserDirectory. See UserDirectory
View, 171
ViewManager. See ViewManager
VirtualDiskManager. See VirtualDiskManager
VirtualMachine, 206-208
VirtualMachineSnapshot, 245-246
VMSnapshot.java, 246-250
ManagedObject, 105
 VI Java API, 106-122
ManagedObject, java, 109-122
ManagedObjectReference, 22, 27-30
ManagedObjectView, 171
managing
custom fields, 168-169
ESX users with HostLocalAccountManager, 445-449
global settings with OptionManager, 503-504
permissions, AuthorizationManager, 439-442
plug-ins, 471, 478-479
   extension.properties, 477
   ExtMgrUtil.java, 471-477
   roles, AuthorizationManager, 438
   sessions with SessionManager, 449-453
mapping, from WSDL to .NET data types, 584
md5sum, 601
Measure_Object, 497
memory requirements, querying for virtual machines, 189
mergePermissions(), 440
merging permissions, 440
messages, SOAP, 585-587
method signatures, challenges with VI SDK, 95
MethodAction, 365, 426
MethodFault, 36
   methods
   HostNetworkSystem, 287-289
   invoking (Alarm), 365
   managed objects, 24-26
   synchronous versus asynchronous, 26-27
MetricAlarmExpression, 362
Microsoft PowerShell, 495-496
MigrateVM.java, 253-254
migrating, existing applications, 602
migration, DRS cluster, 273
missingProperty, 71
MOB (Managed Object Browser), 42-44
   registering VI client plug-ins, 478
   resource URLs, 44
modeling XML, 583
modifyListView(), 173
moduleName, 601
monitoring
   changes with propertyFilter, 78-85
   performance in real time, 399-404
   SOAP messages with HTTP, 62
tasks, 409
MonitorVM.java, 80-85
MOR objects, 28-30
morof, 479
MorUtil, 141
 VI Java API, 141-144
INDEX

MorUtil.java, 141-143
motivation, high-performance Web Services
   engine (VI Java API), 150
mountToolInstaller(), 239
MoveDatacenter.java, 167-168
moveDatastoreFile_Task(), 342
moveHostInto_Task(), 271
moveIntoResourcePool(), 261
moveInto_Task(), 271
moving existing entities, inventory operations,
   167-168
multipathing, 327
multithreading, with VI SDK, 518-519

N
namespaces
   compatibility, 523-524
   versioning, 519-523
naming conventions, 23
NAS (network attached storage), 310, 313
NAS-backed datastore, 328
.NET data types, mapping from WSDL, 584
.NET objects, Web Service Access cmdlets, 500
Network, 295-296
   Properties, 295-296
network configuration, changing, 293
network interface card (NIC), 283-284
network services, HostServiceSystem 305-306
   FlipNetworkService.java, 306-308
networking
   host networking. See host networking
   HostNetworkSystem. See
   HostNetworkSystem
   port groups, 284
   service console networking, 292-293
   virtual network components. See virtual
      network components
virtual networking, 283
virtual NIC, 284
virtual switch (vSwitch), 284
VMotion, HostVMotionSystem, 308
new features
   APIs, changes of existing APIs, 595
   compatibility checkers, 599-600
   distributed virtual switch (DVS), 596-597
   host management, 597
   licensing, 600
   localization support, 600-601
   migrating existing applications, 602
   OVF support, 597-598
   profile management, 595-596
   resource planning, 601
   VirtualApp support, 598-599
NIC (network interface cards), virtual NIC, 284
NIC teaming, 294
no permission, 63
nonentity view objects, VI Perl Toolkit, 494
null pointer, 62

O
obj, 71
object diagrams, UML, 582
object hierarchy, VI Java API, 98-99
Object Management Group (OMG), 580
object model, VI SDK, 17-18
objects, managed objects. See managed objects
offloading setting, host network policy, 295
OMG (Object Management Group), 580
online communities, 47-48
Open Virtualization Format (OVF), 597-598
openInventoryViewFolder(), 172
OptionDef, 503-504
OptionManager, managing global settings, 503-504
   queryOption(), 503
   setting, 503
   supportedOption, 503
   updateOptions(), 504
OrAlarmExpression, 362
OS
   guest operating systems, 236-237
   virtual machines, 212
   overwriteCustomizationSpec(), 243
OVF (Open Virtualization Format), 597-598
OvfManager, 598

P
parsing information from VI Client, 479-480
partial vim.wsdl, 588
patches, 202
patterns, Publisher-Subscriber, 519
PerfCompositeMetric, 394
PerfCounterInfo data object, 380
PerfEntityMetricBase, 393
PerfEntityMetricCSV, 393
PerfInterval data object, 383
PerfInterval parameter, 405
performance
    best practices for, 524-526
    historical statistics versus real time, 384-385
    monitoring in real time, 399-404
performance counters, 380-382
    levels, 381
    VI SDK, 538
performance metadata, querying, 390-391
performance metrics, 382
performance statistics, querying, 391
    queryPerf(), 391-394
    queryPerfComposite(), 394-399
PerformanceManager, 385, 390
    PrintPerfMgr.java, 386-389
    properties, 385
PerfProviderSummary data object, 390
PerfQuerySpec, 391-392, 405
    properties, 392
Perl view objects, VI Perl Toolkit, 492-493
permissions, 434-435
    managing with AuthorizationManager, 439-442
    merging, 440
    querying, 440-442
    removing, 439
    setting, 439
plug-ins
    key strings, 478
    registering and managing, 471, 478-479
        extension.properties, 477
        ExtMgrUtil.java, 471-477
    VI Client plug-ins, 468
policies, defining host network policies, 294-295
port groups, 284
    adding to virtual switches, 290
PosterOutputStream, 517
postEvent(), 352
power consumption, reducing through
    virtualization, 6
power management, 186-189
    DRS cluster, 273
power operations, virtual machines, 222-223
    VMPowerOps, 224-226
powerOffVM_Task(), 222
powerOnVApp_Task(), 599
powerOnVM_Task(), 222
PowerShell, cmdlets, 497
powerUpHostFromStandBy_Task(), 181
PrintAlarmManager.java, AlarmManager, 367-373
PrintLicense.java, 458-462
PrintPerfMgr.java, PerformanceManager, 386-389
PrintStorageSystem.java, 316-321
PrintTaskManager.java, 413-415
private (·), 581
privileges, 433
    managed entities, 160
    required for reconfiguring virtual machines, 217
profile management, new features, 595-596
properties
    AlarmManager, 366
    ClusterComputeResource, 268
    ComputeResource, 264
    Datacenter, 161
    DataStore, 334
    EventManager, 351
    getting from multiple managed objects,
        PropertyCollector, 73-78
    getting from single managed objects,
        PropertyCollector, 67-73
    HostNetworkSystem, 285-287
    HostStorageSystem, 313
    HostSystem, 180
    LicenseManager, 454
    managed objects, 21-24
    ManagedEntity, 157-159
    Network, 295-296
    PerformanceManager, 385
    PerfQuerySpec, 392
    resourceConfig, 215
    ResourcePool, 260
    retrieving, from a single managed object, 67
    SessionManager, 450
    TaskManager, 411-412
    VirtualMachine, 207-208
property path notation, data objects, 33-34
PropertyCollector, 67, 519
    getting properties from a single managed object, 67-73
    getting properties of multiple managed objects, 73-78
    monitoring changes using PropertyFilter, 78-85
    versus SearchIndex, 91
PropertyCollector2.java, 74-77
PropertyCollectorUtil, VI Java API, 149
PropertyFilter, monitoring changes, 78-85
propSet, 71
protected (#), 581
PSOD (Purple Screen of Death), 196
public (+), 581
Publisher-Subscriber design pattern, 519
Purple Screen of Death (PSOD), 196

queryAssignedLicenses(), 600
queryAvailableDisksForVmfs(), 331
queryAvailablePartition(), 197
queryAvailablePerfMetric(), 391
queryBootDevices(), 196
queryComplianceStatus(), 596
queryConfigOption(), 279
queryConfigOptionDescription, 281
queryConfigTarget(), 281
queryDateTime(), 193
queryDescriptions(), DiagnosticManager, 505
queryEvents(), 352-353
queryEvents.java, EventManager, 354-358
queryExpressionMetadata(), 596
queryFirmwareConfigUploadURL(), 199
queryHistoricalEvents.java, EventHistoryCollector, 359-360
queryHostConnectionInfo(), 181, 189
querying
  environments for virtual machines, 278-281
  memory requirements for virtual machines, 189
  performance metadata, 390-391
  performance statistics, 391
    queryPerf(), 391-394
    queryPerfComposite(), 394-399
  permissions, 440-442
queryIpPools(), 599
queryLicenseSourceAvailability(), LicenseManager, 457
queryLicenseUsage(), LicenseManager, 457
queryMemoryOverhead(), 181, 189
queryMemoryOverheadEx(), 181, 189
queryMotionCompatibilityEx_Task(), 600
queryOption(), OptionManager, 503
queryPartitionCreateDesc(), 197
queryPartitionCreateOptions(), 198
queryPerf(), 390-394
queryPerfComposite(), 390, 394
  GetMultiPerf.java, 395-399
queryPerfCounter(), 391
queryPerfCounterByLevel(), 391
queryPerfProviderSummary(), 390, 399-404
querySupportedFeatures(), LicenseManager, 457
QueryVirtualDisk.java, 346-348
queryVmfsDataStoreExtendOptions(), 332

raw performance data, 399
real time
  versus historical performance statistics, 384-385
  monitoring performance in, 399-404
RealtimePerfMonitor.java, 400-404
rebootGuest(), 237
rebootHost_Task(), 181
Recommendation object, 273
recommendations, DRS cluster, 273
reconfigureCluster_Task(), 267
reconfigureHostForDAS_Task(), 181, 191
reconfigureScheduledTask(), 424
reconfigureServiceConsoleReservation(), 201
reconfigureSmpAgent(), 298
reconfigureVirtualMachineReservation(), 201
reconfiguring
  alarms, 365-366
  clusters, 271
  virtual machines, 215, 219
RemoveVmDisk.java, 219-221
reconfigVM_Task(), 215
reconnectHost_Task(), 181, 190
refreshDatastore(), 335
refreshFirewall(), 302
refreshHealthStatusSystem(), 200
refreshRecommendation(), 274
refreshServices(), 305
registerExtension, ExtensionManager, 467
registering
  plug-ins, 471, 478-479
    extension.properties, 477
    ExtMgrUtil.java, 471-477
  virtual machines, 231-232
registerVM_Task(), 231
removeAllSnapshots_Task(), 245
removeAssignedLicense(), 600
removeAuthorizationRole(), 438
removeCustomFieldDef(), 169
removeDatastore(), 332
sessions
managing with SessionManager, 449-453
sharing among different applications, 507
session IDs, getting, 507-509
session IDs, using, 509-511
setCollectorPageSize(), HistoryCollector, 358
setEntityPermissions(), 439
setField(), 169
SetHostTime.java, 194-195
setLicenseEdition(), LicenseManager, 457
SetLicenseSource.java, 462-463
setLocale(), Sessionmanager, 451
setMultipathLunPolicy, 327
setPublicKey(), ExtensionManager, 467
setScreenResolution(), 237
setting, OptionManager, 503
sharing sessions among different applications, 507
session IDs, getting, 507-511
shutdownGuest(), 236
shutdownHost_Task(), 181, 189
Simple Object Access Protocol. See SOAP
single sign-on, from VI SDK to CIM, 511-515
SLAs (Service Level Agreements), 258
Small Computer Systems Interface (SCSI), 311
snapshots, virtual machine snapshots, 243-244
hierarchy, 243-245
SNMP, HostSnmpSystem, 296-298
ConfigureSnmpSystem.java, 298-299
SOAP (Simple Object Access Protocol), 16, 585-587
SOAP messages, monitoring with HTTP, 62
solutions, virtualization, 7
Standby, 188
standbyGuest(), 237
startService(), network services, 305
StateAlarmExpression, 362
stopService(), network services, 305
storage
HBA, 312
iSCSI, 312
NAS, 313
SAN, 312
SCSI, 311
VMFS, 313
storage area network (SAN), 311-312
storage virtualization, 311
storage VMotion, 254-255
stub code generation, 590-592
stubs, generated stubs, 592-593
summaries, virtual machines, 213
summary, host systems, 186
supportedOption, OptionManager, 503
Suspended, virtual machines, 222
suspendVM_Task(), 222
swapping datastore, 332
switching between API and Web Services, VI Java
API, 103-104
synchronous methods versus asynchronous
methods, 26-27

T
Task, 407-409
canceling tasks, 410
fixing task timeouts, 410-411
monitoring tasks, 409
task schedules, specifying with
ScheduledTaskManager, 425
task timeouts, fixing, 410-411
TaskHistoryCollector, 412
retrieving historical tasks, 416-422
TaskHistoryMonitor.java, 417-422
TaskInfo, 408
TaskManager, 411, 415-416
createCollectorForTasks(), 412
createTask(), 412
PrintTaskManager.java, 413-415
properties, 411-412
TaskReason, 408
tasks
canceling, 410
creating (Alarm), 364
monitoring, 409
retrieving with TaskHistoryCollector, 416-422
templates, converting to/from virtual
machines, 235
terminateSession(), 452
TestServlet.java, 481-482
_this, 592
time, managing time of host, 192-195
title, extension configuration, 470
traffic shaping, 294
triggering conditions, Alarm, 362
TurnOnFirewallPolicy.java, 303-305

U
UML (Unified Modeling Language), 580
class diagrams, 580-581
object diagrams, 582
XML, modeling, 583
UML diagrams, 105
unassignUserFromGroup(), 446
Unified Modeling Language (UML), 580
uninstallService(), 306
unmountToolsInstaller(), 239
unregisterExtension(), ExtensionManager, 467
unregistering virtual machines, 236
unregisterVApp_Task(), 599
updateAssignedLicense(), 600
updateBootDevice(), 196
updateChildResourceConfiguration(), 262
updateConfig(), 262
updateConsoleIpRouteConfig(), 293
updateDateTime(), 193
updateDefaultPolicy(), 302
updateDiskPartitions(), 325
updateDnsConfig(), 293
updateExtension(), ExtensionManager, 467
updateFlags(), 181, 191
updateInternetScsiAlias(), 323
updateInternetScsiAuthenticationProperties(), 323
updateInternetScsiDiscoveryProperties(), 324
updateInternetScsiIPProperties(), 324
updateInternetScsiName(), 324
updateIpConfig(), 309
updateIpPool(), 599
updateIpRouteConfig(), 293
updateNetworkConfig(), 293
updateOptions(), OptionManager, 504
updatePerfInterval(), 405
updateProgress() method, 26
updateServiceMessage(), SessionManager, 452
updateServicePolicy(), 306
updateSoftwareInternetScsiEnabled(), 324
updateSystemResources(), 182, 191
updateUser(), 446
updateVAppConfig(), 599
upgrading
virtual machines, 235-236
VMware Tools, 237-239
uploading files, using HTTP Access, 515-518
url, extension configuration, 470
user groups, retrieving, 443
UserDirectory, 443-445
users, looking up with UserDirectory, 443-445
virtual machines, 3, 204-205
capabilities, 209
cloning, 232-233
    CloneVM.java, 233-234
configuration, 209
configuring auto start/stop, 191-192
creating new, 226-227
    CreateVM.java, 227-231
destroying, 234
file layout, 214, 218
files of, 205-206
guest operating system information, 212-213
power operations, 222-223
    VMPowerOps, 224-226
querying environments, 278-281
querying memory requirements for, 189
reconfiguring, 215, 219
    RemoveVmDisk.java, 219-221
registering, 231-232
resource allocation, 215
runtime information, 210
summaries, 213
templates, converting to/from, 235
unregistering, 236
upgrading, 235-236
virtual network components, 289
port groups, adding to virtual switches, 290
virtual NIC, adding, 290-292
virtual switches, adding, 290
virtual networking, 283
virtual NIC (network interface card), 283-284
    adding, 290-292
virtual resource management, 258-259
virtual switch (vSwitch), 284
virtual switches, 290
VirtualApp, 598-599
VirtualCenter, 20, 525
    connecting back to, 480-482
    DRS cluster, 272
    features in, 456
    logs, 59-60
    predefined roles, 435
    VMotion, 251
VirtualDiskManager, 346
    methods defined, 344-345
    QueryVirtualDisk.java, 346-348
virtualization, 1-3
    benefits of, 5-6
    compatibility, 5
cost saving, 6
encapsulation, 5
history of, 3-5
isolation, 5
resource sharing, 6
solutions, 7
VirtualMachine, 206-207
    file layout, 214, 218
    guest operating systems, 236-237
    properties, 207-208
VirtualMachineCapability, 209
VirtualMachineCompatibilityChecker, 599
VirtualMachineConfigInfo, 209
VirtualMachineConfigSpec, 215
VirtualMachineConfigSpec data object, 215-216
VirtualMachineConfigSummary, 213
VirtualMachineFileLayout data object, 214
VirtualMachineProvisioningChecker, 599
VirtualMachineRuntimeInfo, 210
VirtualMachineSnapshot, 245-246
    VMSnapshot.java, 246-250
VirtualMachineSnapshotInfo, 243-244
VirtualMachineSnapshotTree, 243
virtualMachineSummary data object, 214
visibility, class diagrams (UML), 581
VIX API, 14
VLAN policy, 294
VM, logs, 61-62
vm.getVimService(), 104
VMCI (Virtual Machine Communication Interface), 15
VMFS (VMware File System), 310, 313
VMFS datastore, 331-332
VMFS file systems, 326-327
VMFS-backed datastore, 328
VMKernel, 284
    host memory, 200-202
VMotion, 12, 251-252
    MigrateVM.java, 253-254
    requirements, 252
    storage VMotion, 254-255
VMPowerOps, 224-226
VmRename.java, 165-166
VMSnapshot.java, 246-250
vmSuspendedEvent, 350
VMware
    APIs, 12, 15
        CMI APIs, 14
        GuestSDK, 14
        VI SDK, 13
<table>
<thead>
<tr>
<th>INDEX</th>
</tr>
</thead>
</table>

| VIX API, 14 |
| VMCI, 15 |
| DRS, 263 |
| infrastructure, extensibility, 465-466 |
| online communities and conferences, 47-48 |
| VMware ACE, 9 |
| VMware ESX, 10 |
| VMware ESXi, 10 |
| VMware File System (VMFS), 310 |
| VMware Fusion, 9 |
| VMware HA. See HA |
| VMware infrastructure (VI), 3, 8, 257 |
| setting up 49-50 |
| VMware Lab Manager, 8 |
| VMware Lifecycle Manager, 8 |
| VMware Player, 9 |
| VMware Server, 9 |
| VMware Site Recovery Manager, 8 |
| VMware Stage Manager, 8 |
| VMware Tools, upgrading 237-239 |
| VMware VI, 9 |
| VMware Virtual Desktop Infrastructure, 9 |
| VMware VirtualCenter server, 12 |
| VMware Workstation, 8 |
| VMX files, virtual machines, 206 |
| VNIC (virtual network interface card), adding, 290-292 |
| vSphere 4 SDK, new features, 594 |
| vSphere SDK 2.5, managed object types, 535-537 |
| vSwitch (virtual switch), 284 |

**W**

`waitForUpdate()` method, 27-78

Web Access, 41

Web Service Access cmdlets, VI Toolkit (for Windows), 500-501

Web Services

- SOAP, 585-587
- stub code generation, 590-592
- switching between API and (VI Java API), 103-104
- WSDL files, 587-590

Web Services Definition Language. See WSDL

Web Services engine, VI Java API, 149

- architecture, 150-151
- motivation, 150

Web-based Datastore Browser, 45

Where-Object, 497

Windows, VI Toolkit, 495-496

- cmdlet pipeline, 499
- cmdlets, 496-498
- installing, 496
- running cmdlets in a script file, 499-500
- Web Service access cmdlets, 500-501

WSDL (Web Services Definition Language), 17

- data types, 591
- mapping to .NET data types, 584
- WSDL files, 587-590

x86 virtualization, 4

XML, modeling, 583

xmlToCustomizationSpecItem(), 243