Please note that you can always find the most up-to-date technical documentation on our Web site at http://www.vmware.com/support/. The VMware Web site also provides the latest product updates.
# Table of Contents

**Introduction** ___________________________________________________7  
Introducing the VMware Scripting APIs _______________________________8  
   Supported Products ___________________________________________________________________8  
   Intended Audience ____________________________________________________________________9  
   Getting Support from VMware ________________________________________________________9  
Using the VMware Scripting APIs _____________________________________________10  
Installing the VMware Scripting API ___________________________________________11  
   GSX Server ________________________________________________________________________11  
   ESX Server ________________________________________________________________________11  
Installing the VMware Scripting API on a Windows Machine _______________________11  
Installing VmPerl Scripting API on a Linux Machine ________________________________12  

**Using VmCOM** _________________________________________________________15  
   VmCOM Objects ____________________________________________________________________16  
   VmConnectParams __________________________________________________________________17  
   VmServerCtl ______________________________________________________________________18  
      Property ______________________________________________________________________18  
      Methods ______________________________________________________________________18  
   VmCollection _____________________________________________________________________20  
   VmCtl __________________________________________________________________________21  
      Properties ____________________________________________________________________21  
      Methods _____________________________________________________________________23  
   VmQuestion ______________________________________________________________________28  
Symbolic Constant Enumerations _____________________________________________29  
   VmExecutionState __________________________________________________________________29  
   VmPowerOpMode ____________________________________________________________________29  
   VmProdInfoType ____________________________________________________________________30  
   VmProduct _________________________________________________________________________31  
   VmPlatform ______________________________________________________________________31  
Using VmCOM to Pass User-Defined Information Between a Running Guest  
Operating System and a Script _____________________________________________32  
   GuestInfo Variables __________________________________________________________________32  
   Sending Information Set in a VmCOM Script to the Guest Operating System __________33
Using Sample VmCOM Programs

Copyright Information
MiniMUI Visual Basic Sample Program
JScript and VBScript Sample Programs
  JScript Sample Program 1
  VBScript Sample Program 2
  VBScript Sample Program 3

Using VmPerl

VMware::VmPerl::ConnectParams
VMware::VmPerl::Server
VMware::VmPerl::VM
  Additional Information on get_tools_last_active
VMware::VmPerl::Question
Symbolic Constants
  VM_EXECUTION_STATE_<XXX> Values
  VM_POWEROP_MODE_<XXX> Values
  Infotype Values
  VM_PRODINFO_PRODUCT_<XXX> Values
  VM_PRODINFO_PLATFORM_<XXX> Values

Using VmPerl to Pass User-Defined Information Between a Running Guest Operating System and a Script

GuestInfo Variables
  Sending Information Set in a VmPerl Script to the Guest Operating System
  Sending Information Set in the Guest Operating System to a VmPerl Script
Reusing a Single Server Object  129

Index  131
This release of VMware™ scripting APIs version 2.1.5 comprises two components: VmCOM and VmPerl.

VmCOM is a Component Object Model (COM) interface for languages such as Microsoft® Visual Basic®, Microsoft® Visual Basic® Scripting Edition (also known as VBScript), Microsoft® Visual C++® and JScript®. You may install the VmCOM Scripting API on machines with the Microsoft® Windows® operating system.

VmPerl is an application programming interface (API) that utilizes the Perl scripting language. You may install the VmPerl Scripting API on machines with the Microsoft Windows or Linux operating system.
Introducing the VMware Scripting APIs

You may install the Scripting APIs on the GSX Server host and on remote workstations, connecting to GSX Server or ESX Server.

Although the interfaces for VmCOM and VmPerl are different, both components are functionally equivalent. Depending on your operating system, you can either use VmCOM or VmPerl to accomplish the same tasks.

VMware has designed VmCOM and VmPerl to provide task automation and simple, single-purpose user interfaces. The Scripting APIs are not intended for building advanced interactive user interfaces.

For example, you can use the VMware Scripting APIs to perform power operations (start, stop, suspend or reset) on VMware servers and virtual machines, locally and remotely across servers. You can also use the API to register and unregister virtual machines and gather information about them, including sending and receiving configuration to a virtual machine. You can also send properties you define, from the host or a script, into a virtual machine’s guest operating system and vice versa.

We provide example scripts and applications demonstrating possible uses for the Scripting APIs. The directory in which you installed VmCOM contains two subdirectories; MiniMUI, that contains a sample Visual Basic project that uses VmCOM, and SampleScripts, that contains sample VmCOM scripts. Similarly, the directory in which you installed VmPerl contains a subdirectory, SampleScripts, that contains sample VmPerl scripts.

Supported Products

We support the installation of the VmCOM and VmPerl Scripting APIs on VMware™ GSX Server™ 2.x, GSX Server 3.x and VMware ESX Server™ 2.x. For more information see:
Intended Audience
This manual is written for programmers that are familiar with either the Perl language or the Component Object Model (COM) interface for programming languages. Readers of this manual should be comfortable with developing system administration and system monitoring programs and general debugging techniques. In addition, developers who use this manual should be familiar with the operation and management of VMware GSX Server, the host operating system used for this application, and VMware ESX Server.

Getting Support from VMware
See www.vmware.com/support/developer or full details on the VMware Scripting APIs support policy.
Using the VMware Scripting APIs

By using the VMware Scripting APIs, you can access and administer virtual machines without using a local or remote console. The virtual machines — or the server for that matter — do not have to be running in order to use the VMware Scripting APIs.

Note: We support a maximum of two scripting connections to a virtual machine at a time. Each VmCOM object and Perl module is described in the following chapters and includes the methods, the properties, and their usage. In addition, sample scripts and lists of error codes are provided. For VmCOM sample scripts, see Using Sample VmCOM Programs on page 35 and for VmPerl scripts, see Using Sample VmPerl Scripts on page 69. For the list of error codes, see Error Codes on page 100.

Note: For more information about VMware API development, see www.vmware.com/support/developer.
Installing the VMware Scripting API

GSX Server
You have the option of installing the VMware Scripting API on your GSX Server when you installed the software.

However, if you want to run VMware Scripting APIs on a remote workstation, you need to install VmCOM or VmPerl on that machine. Your administrator will provide you with the appropriate script or executable file, or ask you to download it from the VMware Management Interface (requires customization).

ESX Server
To download the VMware Scripting API packages, go to the Download page, www.vmware.com/download of the VMware Web site. Click Download next to the appropriate ESX Server version. Enter your email address and password, then accept the end user license agreement, to get to the packages. Click on the appropriate API package link.

The links are:
- COM API EXE — COM Scripting API for Windows operating systems. Use this package on a Windows client machine.
- Perl API EXE— Perl Scripting API for Windows operating systems. Use this package on a Windows client machine.
- Perl API Compressed Tar Archive — Perl Scripting API for Linux operating systems. Use this package on a Linux client machine.

Installing the VMware Scripting API on a Windows Machine
You have a choice of installing either the VmCOM or the VmPerl Scripting API.

1. Choose Start > Run and browse to the directory where you saved the downloaded installer file (the name is similar to VMware-VmPERLAPI-<xxxx>.exe or VMware-VmCOMAPI-<xxxx>.exe, where <xxxx> is a series of numbers representing the version and build numbers).
2. The installer starts. Click Next.
3. Acknowledge the end user license agreement (EULA). Select Yes, I accept the terms in the license agreement, then click Next.
4. Choose the directory in which to install the Scripting API. To install it in a directory other than the default, click Change and browse to your directory of choice. If the directory does not exist, the installer creates it for you. Click Next.
Note: Windows and the Microsoft Installer limit the path length to 255 characters for a path to a folder on a local drive and 240 characters for a path to a folder on a mapped or shared drive. If the path to the Scripting API program folder exceeds this limit, an error message appears. You must select or enter a shorter path.

5. Click **Install**. The installer begins copying files to your machine.
6. Click **Finish**. The VMware Scripting API is installed.

If you install VmCOM, two folders named MiniMUI and SampleScripts are created in the same directory as the VmCOM Scripting API. The MiniMUI folder contains a sample Microsoft Visual Basic 6 project that uses VmCOM. The SampleScripts folder contains VBScript and JScript samples using the VmCOM Scripting API. See Using Sample VmCOM Programs on page 35 for additional information.

If you install VmPerl, a SampleScripts (Samples) folder is created in the same directory as the VmPerl Scripting API. The SampleScripts folder contains sample scripts using the VmPerl Scripting API. See Using Sample VmPerl Scripts on page 69 for additional information on the sample scripts.

At any time, you can decide to remove this software from your system by running the installer and selecting the Remove option. Alternately, use Add/Remove Programs in the Control Panel to remove the Scripting API.

**Installing VmPerl Scripting API on a Linux Machine**

You can install only the VmPerl Scripting API on a Linux machine. VmCOM is not supported.

1. Copy the VmPerl package to the machine on which you want to run the VMware Scripting API.
2. In a terminal window, become root so you can carry out the installation.
3. Untar the package
   
   ```
   tar xzf VMware-VmPERLAPI-v.v.v-###.tar.gz
   ```
   where v.v.v is the specific version number and ### is the build number.
4. Change to the directory where you expanded the package.
   
   ```
   cd vmware-api-distrib
   ```
5. Run the install script.
   
   ```
   ./vmware-install.pl
   ```
6. Press Enter to read the end user license agreement (EULA). You may page through it by pressing the space bar. If the Do you accept? prompt doesn’t appear, press Q to get to the next prompt.
7. Choose the directory to install the VmPerl executable files or accept the default location.
CHAPTER 1 Introduction

This directory includes the uninstall script for the VmPerl API.

8. Choose the directory to install the VmPerl library files or accept the default location.

This directory includes the sample scripts for the VmPerl API. The SampleScripts directory contains example scripts that demonstrate use of the VmPerl API. You may customize these scripts for your particular organization. See Using Sample VmPerl Scripts on page 69 for more information on the sample scripts.

This completes the VmPerl API installation.

At any time, you can decide to remove this software from your system by invoking the following command as root:

<executable_files_directory>/vmware-uninstall-api.pl
Using VmCOM

The VmCOM component exposes VmServerCtl and VmCtl as the primary objects for communicating with VMware components. VmConnectParams, VmCollection and VmQuestion are support objects used as inputs or outputs to the methods and properties of the primary objects.

A VmServerCtl object represents a server and exports server-level services, such as virtual machine enumeration and registration. A VmCtl object represents a virtual machine on a particular server and provides virtual machine specific methods such as power operations. You must first activate the VmServerCtl or VmCtl object by calling its Connect() method before accessing any other method.

The Connect() method requires a VmConnectParams input parameter containing the host identifier and user credentials supplied for authentication. If the host identifier is not supplied or is undefined, the authentication is performed on the local system. If the user name and password are also not supplied, the current user is authenticated on the local machine. Otherwise, you may supply the user name and password for authentication as that user.

Unlike the VmServerCtl object, VmCtl.Connect() also takes a string specifying the configuration file name of the virtual machine that will be connected.
Once a `VmServerCtl` object is connected, you can enumerate the virtual machines on the server, and register or unregister the virtual machines. You can obtain a list of virtual machines on a particular server from the `VmServerCtl.RegisteredVmNames` property. This property returns a collection object named `VmCollection`. The collection’s elements comprise virtual machine configuration file names and you can enumerate these elements using, for example, the `for each` syntax in Visual Basic. If you know the configuration file name of a specific virtual machine, you can connect the `VmCtl` object directly without using a `VmServerCtl` object.

You can use languages such as Visual Basic or Visual C++ to access VmCOM components. For example, to use VmCOM from Visual Basic, choose `Project > References`, and enable the check box for `VMware VmCOM <version> Type Library`. If this entry is not present, verify that the VMware product is installed correctly.

To use VmCOM from another language, refer to the documentation for that language. Look for the section in the documentation that describes ActiveX® components or the COM interface for that language.

**VmCOM Objects**
The VmCOM component provides the following objects:

- `VmConnectParams`
- `VmServerCtl`
- `VmCollection`
- `VmCtl`
- `VmQuestion`
VmConnectParams

This object supplies connection information and user credentials to `VmServerCtl.Connect()` or `VmCtl.Connect()` and exposes the properties listed in the following table. All VmConnectParams properties allow you to retrieve (GET) and modify (PUT) these properties.

The security for your connection depends upon the security configuration of your VMware server. If you’re connecting to a VMware server or a virtual machine on a server, then the connections is encrypted as long as the VMware server is configured to encrypt connections.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hostname</td>
<td>string</td>
<td>GET/PUT</td>
<td>Retrieves and sets the name of a server, where Hostname is the server’s hostname or IP address. If Hostname is not given or undefined, the authentication is performed on the local system. The C library connects to the local host and uses current user information when it connects. However, this user information is not passed back to VmConnectParams. Otherwise, you may supply the user name and password for authentication as that user.</td>
</tr>
<tr>
<td>Port</td>
<td>integer</td>
<td>GET/PUT</td>
<td>Retrieves and sets the TCP port to use when connecting to the server. Its default value is 0 (zero), indicating the default port number (902) should be used. Otherwise, enter the correct port number. A port number set to a negative value is treated as an incorrect value and the default port number is used instead.</td>
</tr>
<tr>
<td>Username</td>
<td>string</td>
<td>GET/PUT</td>
<td>Retrieves and sets the name of a user on the server.</td>
</tr>
<tr>
<td>Password</td>
<td>string</td>
<td>GET/PUT</td>
<td>Retrieves and sets the user’s password on the server.</td>
</tr>
</tbody>
</table>
VmServerCtl

The VmServerCtl object represents a VMware server running on a particular machine.

Property

The VmServerCtl object includes the properties listed in the following table. All of the properties can be retrieved (GET); some of the properties can also be modified (PUT).

Note: The Resource property applies only to ESX Server.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegisteredVmNames</td>
<td>string</td>
<td>GET</td>
<td>Returns a VmCollection of strings specifying the configuration file names of the virtual machines currently registered on the server. The server must be connected using Connect(), or this property throws an error.</td>
</tr>
<tr>
<td>Resource(&lt;variable_name&gt;)</td>
<td>string</td>
<td>GET/PUT</td>
<td>Accesses the value of a ESX Server system resource variable identified by the string &lt;variable_name&gt;. The property throws an error if it accesses an undefined system variable. See VMware ESX Server System Resource Variables on page 120 for a list of server system variables.</td>
</tr>
</tbody>
</table>

Methods

The VmServerCtl object also exposes the methods listed in the following table. Except where noted otherwise, these methods are synchronous; the method does not return until it finishes its operation, fails, or times out. Most operations time out after 2 minutes.

Note: ESX Server 2.x supports a maximum of 200 registered virtual machines per server.
## CHAPTER 2 Using VmCOM

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object.Connect(&lt;params&gt;)</td>
<td>The <code>Connect()</code> method connects the object to a VMware GSX Server or VMware ESX Server where <code>params</code> is a <code>VmConnectParams</code> object that specifies the system and user information. There is no method to disconnect from a server. To reconnect to a different server, destroy the <code>VmServerCtl</code> object, create a new one, then call its <code>Connect()</code> method. The total number of connected <code>VmCtl</code> and <code>VmServerCtl</code> objects cannot exceed 62. The <code>Connect()</code> method fails with error code <code>vmErr_INSUFFICIENT_RESOURCES</code> if this limit is reached. In order to connect new objects, destroy one or more connected <code>VmCtl</code> or <code>VmServerCtl</code> objects. For example, you can do this by setting an object to <code>Nothing</code> in Visual Basic.</td>
</tr>
<tr>
<td>object.RegisterVm(&lt;vmName&gt;)</td>
<td>The <code>RegisterVm</code> method registers a virtual machine on a server where <code>vmName</code> is a string specifying the virtual machine's configuration file name.</td>
</tr>
<tr>
<td>object.UnregisterVm(&lt;vmName&gt;)</td>
<td>The <code>UnRegisterVm</code> method unregisters a virtual machine from a server where <code>vmName</code> is a string specifying the virtual machine's configuration file name.</td>
</tr>
</tbody>
</table>
VmCollection

The VmCollection object is a collection of variants that are typically strings. You can enumerate its elements by using the for each syntax in Visual Basic. You can individually access each element by passing its index to the Item property, or by using the VmCollection(index_as_integer) array syntax in Visual Basic. The first element’s index is always the integer 1 (one).

Both VmServerCtl.RegisteredVmNames and VmQuestion.Choices return a VmCollection of strings.

The VmCollection object includes the read-only (GET) properties listed in the following table:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>integer</td>
<td>GET</td>
<td>Gets the number of elements in the collection.</td>
</tr>
<tr>
<td>Item(index_as_integer)</td>
<td>string</td>
<td>GET</td>
<td>Gets the element at the specified index.</td>
</tr>
</tbody>
</table>
**VmCtl**

The VmCtl object represents a virtual machine running on a particular server machine and exposes symbolic constant enumerations, properties and methods.

**Properties**

The VmCtl object includes the properties listed in the following table. All of the properties can be retrieved (GET); some of these properties can also be modified (PUT).

**Note:** The last four properties that are listed in the following table apply only to ESX Server.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecutionState</td>
<td>VmExecutionState</td>
<td>GET</td>
<td>Current state of the virtual machine; powered on, powered off, suspended, or stuck. For more information on VmExecutionState, see VmExecutionState on page 29.</td>
</tr>
<tr>
<td>PendingQuestion</td>
<td>VmQuestion</td>
<td>GET</td>
<td>Returns a VmQuestion object if the virtual machine is currently in the vmExecutionState_Stuck state. Otherwise, an error is thrown.</td>
</tr>
<tr>
<td>GuestInfo(keyName)</td>
<td>string</td>
<td>GET/PUT</td>
<td>Accesses a shared variable identified by the string keyName. For additional information, see Using VmCOM to Pass User-Defined Information Between a Running Guest Operating System and a Script on page 32.</td>
</tr>
<tr>
<td>Config(keyName)</td>
<td>string</td>
<td>GET/PUT</td>
<td>Accesses the value of a configuration variable identified by the string keyName. When a virtual machine process is spawned on the server, the process reads configuration variables from the virtual machine's configuration file into memory. If you write a configuration variable by using the Config() property, the new value is written into memory and is discarded when the virtual machine process terminates. You cannot change the value of a configuration variable in a virtual machine's configuration file. The property throws an error if it accesses an undefined configuration variable. Do not change the memory size while a virtual machine is suspended. First power off the virtual machine, then change its memory size.</td>
</tr>
</tbody>
</table>

---

CHAPTER 2 Using VmCOM

21
### Property Name | Property Type | Access Type | Description
--- | --- | --- | ---
ConfigFileName | string | GET | Returns the configuration file name for the virtual machine. This method fails if the VmCtl object is not connected.
Heartbeat | integer | GET | Returns the current heartbeat count generated by the VMware Tools service running in the guest operating system. The count is initialized to zero when the virtual machine is powered on. The heartbeat count is typically incremented at least once per second when the VMware Tools service is running under light load conditions. The count stays constant if this service is not running.
ToolsLastActive | integer | GET | Returns an integer indicating how much time has passed, in seconds, since the last heartbeat was detected from the VMware Tools service. This value is initialized to zero when the virtual machine powers on. It stays at zero until the first heartbeat is detected, after which the value is always greater than zero until the virtual machine is power-cycled again. For additional information, see the next section.
DeviceIsConnected(devName) | Boolean | GET | Returns True if the specified device is connected. Otherwise, False is returned.
ProductInfo(infoType) | integer, VmProduct or VmPlatform | GET | Returns an integer representing the value of the product information field specified by infoType, which is of type VmProdInfoType. See VmProdInfoType on page 30 for a list of valid types and return values.
Uptime | integer | GET | Accesses the uptime of the guest operating system on the virtual machine.
Pid | integer | GET | Returns the process ID of a running virtual machine.
Resource(<variable_name>) | string | GET/PUT | Accesses the value of a virtual machine resource variable identified by the string <variable_name>. The property throws an error if it accesses an undefined variable. See Virtual Machine Resource Variables for ESX Server on page 124 for a list of virtual machine variables.
Id | string | GET | Returns the unique (world) ID for a running virtual machine.

**Note:** This property applies only to ESX Server.
Additional Information on ToolsLastActive

If the guest operating system is heavily loaded, this value may occasionally reach several seconds. If the service stops running, either because the guest operating system has experienced a failure or is shutting down, the `ToolsLastActive` value keeps increasing.

You can use a script with the `ToolsLastActive` property to monitor the start of the VMware Tools service, and once started, the health of the guest operating system. If the guest operating system has failed, the `ToolsLastActive` property indicates how long the guest has been down. The following table summarizes how you may interpret the `ToolsLastActive` property values.

<table>
<thead>
<tr>
<th>ToolsLastActive Property Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The VMware Tools service has not started since the power-on of the virtual machine.</td>
</tr>
<tr>
<td>1</td>
<td>The VMware Tools service is running and is healthy.</td>
</tr>
<tr>
<td>2, 3, 4, or 5</td>
<td>The VMware Tools service could be running, but the guest operating system may be heavily loaded or is experiencing temporary problems.</td>
</tr>
<tr>
<td>Greater than 5</td>
<td>The VMware Tools service stopped running, possibly because the guest operating system experienced a fatal failure, is restarting, or is shutting down.</td>
</tr>
</tbody>
</table>

Methods

The `VmCtl` object includes the methods listed in the following table.

You can connect to a virtual machine, start, stop, suspend and resume virtual machines, query and modify the configuration file settings, and connect and disconnect devices.

Except where noted otherwise, these methods are synchronous; the method does not return until it finishes its operation, fails or times out. Most operations time out after 2 minutes, except for power operations, which time out after 4 minutes.
Note: Two methods, `object.AddRedo` and `object.Commit`, that are listed in the following table, apply only to ESX Server. Similarly, the `object.SetRunAsUser` and `object.RunAsUser` methods apply only to GSX Server.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object.Connect(&lt;params&gt;, &lt;vmName&gt;)</code></td>
<td>The <code>Connect()</code> method establishes a connection with a virtual machine where <code>params</code> is a <code>VmConnectParams</code> object that specifies the system and user information and <code>vmName</code> is a string specifying the virtual machine's configuration file name. You should use this as the first method invoked on a <code>VmCtl</code> object. You must first activate the <code>VmCtl</code> object by calling its <code>Connect()</code> method before accessing any other method or property. There is no method to disconnect from a virtual machine. To reconnect to a different virtual machine, destroy the <code>VmCtl</code> object, create a new one, then call its <code>Connect()</code> method. The total number of connected <code>VmCtl</code> and <code>VmServerCtl</code> objects cannot exceed 62. The <code>Connect()</code> method fails with error code <code>vmErr_INSUFFICIENT_RESOURCES</code> if this limit is reached. In order to connect new objects, destroy one or more connected <code>VmCtl</code> or <code>VmServerCtl</code> objects. For example, you can do this by setting an object to <code>Nothing</code> in Visual Basic.</td>
</tr>
<tr>
<td><code>object.Start(&lt;mode&gt;)</code></td>
<td>The <code>Start()</code> method powers on a previously powered-off virtual machine or resumes a suspended virtual machine, where <code>mode</code> is a <code>VmPowerOpMode</code> object that specifies the Start operation's behavior. For more information, see <code>VmPowerOpMode</code> on page 29. If the virtual machine is powered off, then it is powered on. If it is suspended, this method resumes the virtual machine. If the virtual machine is in any other state, the <code>Start()</code> method fails and throws an error.</td>
</tr>
<tr>
<td><code>object.Stop(&lt;mode&gt;)</code></td>
<td>The <code>Stop()</code> method shuts down and powers off a virtual machine where <code>mode</code> is a <code>VmPowerOpMode</code> object that specifies the Stop operation's behavior. For more information, see <code>VmPowerOpMode</code> on page 29. This method always fails if the virtual machine is not in the <code>vmExecutionState_On</code> state.</td>
</tr>
<tr>
<td><code>object.Reset(&lt;mode&gt;)</code></td>
<td>The <code>Reset()</code> method shuts down, then reboots a virtual machine where <code>mode</code> is a <code>VmPowerOpMode</code> object that specifies the operation's behavior. For more information, see <code>VmPowerOpMode</code> on page 29. This method always fails if the virtual machine is not in the <code>vmExecutionState_On</code> state.</td>
</tr>
</tbody>
</table>
object.Suspend(<mode>)

The `Suspend()` method suspends a virtual machine where `mode` is a `VmPowerOpMode` object that specifies the Suspend operation's behavior. It saves the current state of the virtual machine to a suspend file. For more information, see `VmPowerOpMode` on page 29.

This method always fails if the virtual machine is not in the `vmExecutionState_On` state.

**Method**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object.Suspend(&lt;mode&gt;)</td>
<td>The <code>Suspend()</code> method suspends a virtual machine where <code>mode</code> is a <code>VmPowerOpMode</code> object that specifies the Suspend operation's behavior. It saves the current state of the virtual machine to a suspend file. For more information, see <code>VmPowerOpMode</code> on page 29. This method always fails if the virtual machine is not in the <code>vmExecutionState_On</code> state.</td>
</tr>
</tbody>
</table>

**Stack:** This method applies only to ESX Server.

**Note:** This method applies only to ESX Server.

object.AddRedo(<diskDevName>)

This method adds a redo log to a running virtual SCSI disk specified by `<diskDevName>`, that is associated with the virtual machine specified by the `VmCtl` object. Changes made to the virtual disk accumulate in the new redo log. This disk must be an ESX Server virtual disk stored on a VMFS volume.

The virtual disk can be in persistent, undoable or append mode. The redo log for a virtual disk in persistent mode uses the file name of the virtual disk with `.REDO` appended to it (for example, if the disk is called `vm.dsk`, the redo log is called `vm.dsk.REDO`). A virtual disk in undoable or append mode already has a redo log associated with it, so the new redo log you create is called `vm.dsk.REDO.REDO`. The parent is the existing redo log, `vm.dsk.REDO`, whose parent is the existing redo log, `vm.dsk.REDO`.

This method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk already has two redo logs associated with it.

If you add a redo log using the `AddRedo()` method, but do not commit your changes with the `Commit()` method, then the redo is automatically committed when the virtual machine is powered off.

**Method**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object.AddRedo(&lt;diskDevName&gt;)</td>
<td>This method adds a redo log to a running virtual SCSI disk specified by <code>&lt;diskDevName&gt;</code>, that is associated with the virtual machine specified by the <code>VmCtl</code> object. Changes made to the virtual disk accumulate in the new redo log. This disk must be an ESX Server virtual disk stored on a VMFS volume. The virtual disk can be in persistent, undoable or append mode. The redo log for a virtual disk in persistent mode uses the file name of the virtual disk with <code>.REDO</code> appended to it (for example, if the disk is called <code>vm.dsk</code>, the redo log is called <code>vm.dsk.REDO</code>). A virtual disk in undoable or append mode already has a redo log associated with it, so the new redo log you create is called <code>vm.dsk.REDO.REDO</code>. The parent is the existing redo log, <code>vm.dsk.REDO</code>, whose parent is the existing redo log, <code>vm.dsk.REDO</code>. This method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk already has two redo logs associated with it. If you add a redo log using the <code>AddRedo()</code> method, but do not commit your changes with the <code>Commit()</code> method, then the redo is automatically committed when the virtual machine is powered off.</td>
</tr>
</tbody>
</table>
This method commits the changes in a redo log to a running virtual SCSI disk specified by `<diskDevName>` that is associated with the virtual machine specified by `$vm`.

- `<level>` can be 0 or 1. When `<level>` is 0, there can be one or two redo logs associated with the disk. If `<level>` is 0, then the top-most redo log (the redo log being modified) is committed to its parent. For example, if there is currently only the disk `vm.dsk` with a single redo log `vm.dsk.REDO`, then the changes in `vm.dsk.REDO` are committed to `vm.dsk`. If a second REDO log `vm.dsk.REDO.REDO` has been added, then the changes in `vm.dsk.REDO.REDO` are committed to `vm.dsk.REDO`.

- `<level>` can be 1 only when there are two redo logs associated with the disk, `vm.dsk.REDO` and `vm.dsk.REDO.REDO`. When `<level>` is 1, the changes in the next-to-top REDO log, `vm.dsk.REDO`, are committed to `vm.dsk`. In this case, the virtual machine is not frozen while the redo log is being committed. Also, when the log is committed, `vm.dsk.REDO.REDO` is renamed to `vm.dsk.REDO`.

- `<freeze>` can be 0 or 1. If `<freeze>` is 0, then the virtual machine is not frozen when changes are committed, though it runs more slowly. If `<freeze>` is 1, then the virtual machine is frozen until the commit operation finishes. If `<level>` is 0, then the virtual machine must be frozen when changes are committed and `<freeze>` is ignored.

- `<wait>` can be 0 or 1. If `<wait>` is 0, then the method returns as soon as the commit begins. If `<wait>` is 1, then the method does not return until the commit completes.

The method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk currently has no redo logs.
The `AnswerQuestion()` method replies to a question where `question` is a `VmQuestion` object that represents the question that requires an answer and `choice` represents the index of the selected answer to the question. The index is an integer and the first choice’s index is always 1 (one). The second choice’s index is 2, and so on.

When a virtual machine is in the `vmExecutionState_Stuck` state and requires user input to continue, use this method to answer the current question or dismiss the current error message.

First, get a `VmQuestion` object from `VmCtl.PendingQuestion`. You can retrieve the possible choices and their respective indices from the `VmQuestion.Choices` property. Then, use the `AnswerQuestion` method to answer the question.

The `ConnectDevice()` method sets a virtual device to the connected state where `devName` is a string that identifies the virtual device you want to connect. The virtual machine must be powered on for this method to succeed, otherwise a `vmErr_BADSTATE` error is returned.

Use the `Config()` property to set configuration parameters relevant to the virtual device before calling the `ConnectDevice()` method. The following code example illustrates connecting a virtual drive to a CD image file:

```plaintext
vm.Config("ide1:0.devicetype") = "cdrom-image"
vm.Config("ide1:0.filename") = "/iso/foo.iso"
vm.ConnectDevice("ide1:0")
```

The `DisconnectDevice()` method sets a virtual device to the disconnected state where `devName` is a string that identifies the virtual device you want to disconnect. The virtual machine must be powered on for this method to succeed, otherwise a `vmErr_BADSTATE` error is returned.

The `SetRunAsUser(<myuseraccount>, <mypassword>)` method runs the virtual machine as the user specified by the `<myuseraccount>` and `<mypassword>`. Note: This method applies only to GSX Server 3.1.

The `RunAsUser` method returns the name of the user running the virtual machine. Note: This method applies only to GSX Server 3.1.
VmQuestion

The VmQuestion object is created and returned by VmCtl.PendingQuestion(). It describes a question or error condition requiring user input. Once the script selects one of the possible answers, it passes the object and the selected answer as inputs to VmCtl.AnswerQuestion().

The VmQuestion object includes the read-only (GET) properties listed in the following table:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>string</td>
<td>GET</td>
<td>Gets the question text.</td>
</tr>
<tr>
<td>Choices</td>
<td>string</td>
<td>GET</td>
<td>Gets a VmCollection of strings representing a list of possible answers to the question.</td>
</tr>
<tr>
<td>Id</td>
<td>integer</td>
<td>GET</td>
<td>Gets an integer used internally by the VmCOM component to identify the question.</td>
</tr>
</tbody>
</table>
Symbolic Constant Enumerations

The `VmCtl` object exposes the following symbolic constant enumerations, where each element of an enumeration is a symbolic constant:

- `VmExecutionState`
- `VmPowerOpMode`
- `VmProdInfoType`
- `VmProduct`
- `VmPlatform`

**VmExecutionState**

The `VmExecutionState` symbolic constant enumeration specifies the state (or condition) of a virtual machine. The possible values are listed in the following table:

<table>
<thead>
<tr>
<th><code>VmExecutionState</code> Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vmExecutionState_On</code></td>
<td>The virtual machine is powered on.</td>
</tr>
<tr>
<td><code>vmExecutionState_Off</code></td>
<td>The virtual machine is powered off.</td>
</tr>
<tr>
<td><code>vmExecutionState_Suspended</code></td>
<td>The virtual machine is suspended.</td>
</tr>
<tr>
<td><code>vmExecutionState_Stuck</code></td>
<td>The virtual machine requires user input. The user must answer a question or dismiss an error.</td>
</tr>
<tr>
<td><code>vmExecutionState_Unknown</code></td>
<td>The virtual machine is in an unknown state.</td>
</tr>
</tbody>
</table>

**VmPowerOpMode**

The `VmPowerOpMode` symbolic constant enumeration specifies the behavior of a power transition (start, stop, reset, or suspend) method.

During a soft power transition, the VMware Tools service runs a script inside the guest operating system. For example, the default scripts that run during suspend and resume operations, respectively release and renew DHCP leases, for graceful integration into most corporate LANs. You may also customize these scripts. For more information on using these scripts, see your VMware product documentation.

The following table includes the possible values for a `VmPowerOpMode` symbolic constant enumeration.
VmPowerOpMode Values

<table>
<thead>
<tr>
<th>Description</th>
<th>vmPowerOpMode_Soft</th>
</tr>
</thead>
<tbody>
<tr>
<td>To succeed, soft power transitions require the current version of the VMware Tools service to be installed and running in the guest operating system.</td>
<td></td>
</tr>
<tr>
<td>Start when a virtual machine is suspended — After resuming the virtual machine, it attempts to run a script in the guest operating system to restore network connections by renewing the DHCP lease. The Start() operation always succeeds. However, if the VMware Tools service is not present or is malfunctioning, the running of the script may fail.</td>
<td></td>
</tr>
<tr>
<td>Start when virtual machine is powered off — After powering on the virtual machine, the operation attempts to run a script in the guest operating system when the VMware Tools service becomes active. This default script does nothing during this operation as there is no DHCP lease to renew. The Start() operation always succeeds. However, if the VMware Tools service is not present or is malfunctioning, the running of the script may fail.</td>
<td></td>
</tr>
<tr>
<td>Stop — Attempts to shut down the guest operating system and then powers off the virtual machine.</td>
<td></td>
</tr>
<tr>
<td>Reset — Attempts to shut down the guest operating system, then reboots the virtual machine.</td>
<td></td>
</tr>
<tr>
<td>Suspend — Attempts to run a script in the guest operating system that safely disables network connections (such as releasing a DHCP lease) before suspending the virtual machine.</td>
<td></td>
</tr>
</tbody>
</table>

| vmPowerOpMode_Hard |
| Start — Starts or resumes a virtual machine without running any scripts, a standard power on or resume. |
| Stop, reset or suspend — Immediately and unconditionally powers off, resets, or suspends the virtual machine. |

| vmPowerOpMode_TrySoft |
| First attempts to perform the power transition operation with vmPowerOpMode_Soft. If this fails, the same operation is performed with vmPowerOpMode_Hard. |

VmProdInfoType Values

<table>
<thead>
<tr>
<th>Description</th>
<th>vmProdInfo_Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>The VMware product type is returned as VmProduct. For more information on VmProduct, see the following section.</td>
<td></td>
</tr>
</tbody>
</table>

| vmProdInfo_Platform |
| The host platform type is returned as VmPlatform. For more information on VmPlatform, see VmPlatform on page 31. |

| vmProdInfo_Build |
| The product’s build number. |

| vmProdInfo_Version_Major |
| The product’s major version number. |

VmProdInfoType symbolic constant enumeration specifies the type of product information when reading the ProductInfo property.
VmProduct

The VmProduct symbolic constant enumeration denotes a VMware product type. The ProductInfo property returns this information when the requested product information type is vmProdInfo_Product.

<table>
<thead>
<tr>
<th>VmProduct Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmProduct_WS</td>
<td>The product is VMware Workstation</td>
</tr>
<tr>
<td>vmProduct_GSX</td>
<td>The product is VMware GSX Server</td>
</tr>
<tr>
<td>vmProduct_ESX</td>
<td>The product is VMware ESX Server</td>
</tr>
<tr>
<td>vmProduct_UNKNOWN</td>
<td>The product type is unknown</td>
</tr>
</tbody>
</table>

VmPlatform

The VmPlatform symbolic constant enumeration denotes a VMware machine’s platform type. The ProductInfo property returns this information when the requested product information type is vmProdInfo_Platform.

<table>
<thead>
<tr>
<th>VmPlatform Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmPlatform_WINDOWS</td>
<td>The host platform is a Microsoft Windows operating system</td>
</tr>
<tr>
<td>vmPlatform_LINUX</td>
<td>The host platform is a Linux operating system</td>
</tr>
<tr>
<td>vmPlatform_VMNX</td>
<td>The host platform is the ESX Server console operating system</td>
</tr>
<tr>
<td>vmPlatform_UNKNOWN</td>
<td>The host platform is unknown</td>
</tr>
</tbody>
</table>
Using VmCOM to Pass User-Defined Information Between a Running Guest Operating System and a Script

When the guest operating system is running inside a virtual machine, you can pass information from a script (running in another machine) to the guest operating system, and from the guest operating system back to the script, through the VMware Tools service. You do this by using a class of shared variables, commonly referred to as GuestInfo. VMware Tools must be installed and running in the guest operating system before a GuestInfo variable can be read or written inside the guest operating system.

For example, create and connect a VmCtl object, assuming the virtual machine is powered off. Next, set the GuestInfo variable with the VmCOM API. Then, power on the virtual machine and use the VMware Tools service to retrieve the variable. See Sending Information Set in a VmCOM Script to the Guest Operating System on page 33 for an example of this procedure.

See your server documentation for more information about VMware Tools.

GuestInfo Variables

You pass to the virtual machine variables you define yourself. What you pass is up to you, but you might find it useful to pass items like the virtual machine’s IP address, Windows system ID (SID, for Windows guest operating systems) or machine name.

This is useful in situations where you want to deploy virtual machines on a network using a common configuration file, while providing each machine with its own unique identity. By providing each virtual machine with a unique identifying string, you can use the same configuration file to launch the same nonpersistent virtual disk multiple times in a training or testing environment, where each virtual machine would be unique on the network. Note that in the case of persistent or undoable disks, each virtual disk file must be copied into its own directory if it shares its file name with another virtual disk file.

When a virtual machine process is created on the server, all GuestInfo variables are initially undefined. A GuestInfo variable is created the first time it is written.

You identify a GuestInfo variable with a key name. You can define and create any number of GuestInfo variable key names. The information you pass is temporary, lasting until the virtual machine is powered off and all consoles connected to the virtual machine are closed.
Sending Information Set in a VmCOM Script to the Guest Operating System

To send information from a VmCOM script to a running guest operating system, you use the GuestInfo() property. You need to specify the string value of the configuration variable identified by keyName.

For example, you might want to deploy virtual machines for a training class. When a virtual machine starts, you want to display a banner welcoming the student to the class. You can pass their name from a VmCOM script to the guest operating system on a student’s virtual machine.

If you have not already done so, connect a VmCtl object and set the student’s name for this virtual machine to “Susan Williams”:

```python
vm.GuestInfo("name") = "Susan Williams"
```

This statement passes a string “name” to the guest operating system. A script in the guest operating system reads the string, then calls a command (specific to the guest operating system) to set the student’s name in the banner. (This operation is explained in the following section).

This setting lasts until you power off the virtual machine and close all connected consoles.

Retrieving the Information in the Guest Operating System

In the running guest operating system, you use the VMware Tools service to retrieve variables set for the virtual machine. You can then use this passed “name” string inside a guest operating system startup sequence. Use the following to read the GuestInfo variable keyName.

In a Windows guest operating system:

```bash
VMwareService.exe --cmd "info-get guestinfo.<keyName>"
```

In a Linux guest operating system:

```bash
/etc/vmware-tools/vmware-guestd --cmd 'info-get guestinfo.<keyName>'
```

For example, to get the current value for the “name” variable, you can type the following in a Linux guest operating system:

```bash
/etc/vmware-tools/vmware-guestd --cmd 'info-get guestinfo.name'
```

Sending Information Set in the Guest Operating System to a VmCOM Script

Similarly, in a virtual machine’s guest operating system, you can use the VMware Tools service to set GuestInfo variables for the virtual machine. Use the following to write the GuestInfo variable keyName.

In a Windows guest operating system:

```bash
VMwareService.exe --cmd "info-set guestinfo.<keyName> <value>"
```
In a Linux guest operating system:

```
/etc/vmware-tools/vmware-guestd --cmd 'info-set guestinfo.<keyName> <value>'
```

Continuing with the previous example, Susan Williams prefers "Sue". To set the value of "Sue Williams" for the "name" variable, type the following in a Linux guest operating system:

```
/etc/vmware-tools/vmware-guestd --cmd 'info-set guestinfo.name Sue Williams'
```

**Retrieving Information in a VmCOM Script**

With the VmCOM API, you use the `GuestInfo(keyName)` property to retrieve information set in the guest operating system, into a VmCOM script running on any machine, including GSX Server or any remote workstation that can connect to the virtual machine.

For example, to retrieve Sue's name set by the VMware Tools service, query the guest operating system by using the VmCOM API:

```python
str = vm.GuestInfo("name")
```
This section contains sample VmCOM programs written by VMware to demonstrate example uses of the VmCOM API. You can modify them to suit the needs of your organization.

These sample programs are installed with the VmCOM component. During installation, two folders named MiniMUI and SampleScripts are created in the same directory as the Scripting API. The MiniMUI folder contains a sample VmCOM project that you may open with Microsoft Visual Basic 6. The SampleScripts folder contains VBScript and JScript samples using the VmCOM Scripting API.

**Note:** You may also obtain these sample scripts from the VMware Web site. The scripts on the Web site are saved with a .TXT extension for online viewing. Remove the .TXT extension before using these scripts.
Copyright Information

Each sample script and sample program included with the VmCOM Scripting API includes a copyright. However, for brevity, we do not include this copyright in its entirety with each sample script and sample program in this manual. Instead, we include the first line of the copyright followed by ellipses, to indicate its placement. The complete copyright is as follows:

Copyright (c) 1998-2004 VMware, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the software in this file (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The names "VMware" and "VMware, Inc." must not be used to endorse or promote products derived from the Software without the prior written permission of VMware, Inc.

Products derived from the Software may not be called "VMware", nor may "VMware" appear in their name, without the prior written permission of VMware, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL VMware, Inc. BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
MiniMUI Visual Basic Sample Program

The MiniMUI sample program illustrates the use of VmCOM interfaces from a Visual Basic application. It is a control panel application allowing users to get status information and to perform power operations on virtual machines registered on a particular server.

The source code demonstrates how to:

- initialize a server object
- enumerate virtual machines on a server
- perform power operations on a virtual machine
- handle errors and get status information
- answer a question for a stuck virtual machine

To run the program, open the project file in Visual Basic. The source for the MiniMUI application is in the MiniMUI folder in the VmCOM Scripting API directory. The following image shows the application’s main window.
JScript and VBScript Sample Programs

The sample scripts included in the SampleScripts folder are designed to run under the Windows Script Host environment, which is included with all Microsoft Windows 2000 and subsequent compatible operating systems. To run a script under a different environment, such as an ASP or HTML page, refer to that environment's documentation.

Each sample program comprises two files: a script, with a .js (JScript) or .vbs (VBScript) extension, and the accompanying Windows Script File with the same name and the .wsf extension. For example, the first sample program consists of the files sample1.js and sample1.wsf. Both the script and the associated .wsf file must be in the same directory when you execute the sample program.

To execute a sample program, type the following in a command line window:

```
cscript //nologo sample<n>.wsf
```

where <n> is the sample program number.

**Note:** The cscript command loads the Windows Script Host environment and is included with the supported operating system. The .js or .vbs script contains the program's actual logic. The associated .wsf file defines and initializes an execution environment for the script. In this example, the .wsf file loads VmCOM's type library to allow the script to use VmCOM's symbolic constants. For more information on symbolic constants, see Properties on page 21.

JScript Sample Program 1

This JScript program connects to the local server and lists all registered virtual machines. If a virtual machine is in the stuck state, the pending question is also displayed.

The source for the sample program 1 script is in the SampleScripts folder in the VmCOM Scripting API directory.

You can also find it on the VMware Web site, saved with a .TXT extension for online viewing, at www.vmware.com/support/developer/scripting-API/doc/sample1.js.txt.

```
//
//  VmCOM JScript Sample Script (sample1)
//  Copyright (c) 1998-2004 VMware, Inc.
//  .
//  .
//  This program is for educational purposes only.
//  It is not to be used in production environments.
//  .
//  Description:
//  .
```
CHAPTER 3 Using Sample VmCOM Programs

// This script displays the virtual machines on the local server.
// It prints the configuration file path and current execution
// state of each VM. If a VM is in the stuck state, the current
// question and its choices are also printed.

// Instructions for Windows 2000 and later operating systems:
// - save the contents of this file to a file named 'sample1.js'
//   unless it's already named that way
// - there should be an accompanying file named 'sample1.wsf'
//   It is placed in the same directory as this file during
//   product installation. This file is responsible for setting
//   up the Windows Script Host environment and loading the
//   VmCOM type library, thereby enabling this script to
//   reference symbolic constants such as vmExecutionState_On
// - in a command line window, type:
//   cscript //nologo sample1.wsf

cp = WScript.CreateObject("VmCOM.VmConnectParams");
server = WScript.CreateObject("VmCOM.VmServerCtl");
server.Connect(cp);
vmCollection = server.RegisteredVmNames
for (j = 1; j <= vmCollection.count; j++) {

  vmName = vmCollection(j);
  vm = WScript.CreateObject("VmCOM.VmCtl");
  vm.Connect(cp, vmName);

  str = "config path=\" + vmName + "\" OS=\" + vm.Config("guestOS") + " state=\";
  execStateString = State2Str(vm);
  str += execStateString;
  if (execStateString == "STUCK") {
    question = vm.PendingQuestion;
    str += " pending question=\" + question.text + "\" choices=\";

    choices = question.choices
    for (i = 1; i <= choices.count; i++) {
      str += "[\" + choices(i) + "] \";
    }
  }
  WScript.Echo(str);
function State2Str(vm) {
    switch (vm.ExecutionState) {
        case vm.ExecutionState_On:
            return "ON";
            break;
        case vm.ExecutionState_Off:
            return "OFF";
            break;
        case vm.ExecutionState_Suspended:
            return "SUSPENDED";
            break;
        case vm.ExecutionState_Stuck:
            return "STUCK";
            break;
        default:
            return "UNKNOWN";
            break;
    }
}

The source for the sample program 1 accompanying Windows Script File is in the SampleScripts folder in the VmCOM Scripting API directory.

You can also find it on the VMware Web site, saved with a .TXT extension for online viewing, at www.vmware.com/support/developer/scripting-API/doc/sample1.wsf.txt.

Note: If you are using Microsoft® Internet Explorer as your browser, select View > Source to view the file. Alternately, right-click this link and download this file.

<job id="Sample1">
    <reference object="VmCOM.VmCtl" />
    <script language="JScript" src="sample1.js" />
</job>

**VBScript Sample Program 2**

This VBScript sample program 2 provides similar functionality to sample program 1. It also connects to the local server and lists all registered virtual machines. If a virtual machine is in the stuck state, the pending question is displayed.

In addition, sample program 2 also illustrates how to handle a virtual machine that is waiting for input to a question (that is, the virtual machine is in the `vmExecutionState_Stuck` state).

For example, if a virtual machine is configured with an undoable disk and a redo log is found, this
script automatically keeps the redo log during a shutdown operation or appends the redo log during a power-on operation.

**Note:** The script’s question-answering code is highly dependent on the version of your server product and the language used in the question. This script can malfunction with a newer version of the server product or different language version of the VMware server product. This sample program is for example purposes only and is written for VMware GSX Server.

The source for the sample program 2 script is in the SampleScripts folder in the VmCOM Scripting API directory.

You can also find it on the VMware Web site, saved with a .TXT extension for online viewing, at www.vmware.com/support/developer/scripting-API/doc/sample2.vbs.txt.

```vbs
' VmCOM VBScript Sample Script (sample2)
' Copyright (c) 1998-2004 VMware, Inc.
' .
' .
' This program is for educational purposes only.
' It is not to be used in production environments.
' .
' Description:
' .
' This script displays the virtual machines on the local server.
' It prints the configuration file path and current execution state of each VM. If a VM is in the stuck state, the current question and its choices are also printed.
' Additionally, if a VM is stuck on an undoable disk related question, the script automatically answers 'Keep' on a power-off and 'Append' on a power-on.
' .
' NOTE: the question-answering logic used is language and product dependent, and is only provided for illustration purposes only!
' .
' Instructions for Windows 2000 and later operating systems:
' .
' - save the contents of this file to a file named 'sample2.vbs'
' unless it's already named that way
' .
' - there should be an accompanying file named 'sample2.wsf'
' It is placed in the same directory as this file during product installation. This file is responsible for setting up the Windows Script Host environment and loading the VmCOM type library, thereby enabling this script to reference symbolic constants such as vmExecutionState_On
```
Set cp = CreateObject("VmCOM.VmConnectParams")
Set server = CreateObject("VmCOM.VmServerCtl")

server.Connect cp
Set vmCollection = server.RegisteredVmNames

for each vmName in vmCollection
  Set vm = CreateObject("VmCOM.VmCtl")
  vm.Connect cp,vmName
  s = "path=" & vmName & " state=" & State2Str(vm) & " os=" & vm.Config("guestos")

  if vm.ExecutionState = vmExecutionState_Stuck then
    Set q = vm.PendingQuestion
    Set choices = q.choices
    s = s & " question= '" & q.text & "' choices="
    for each choice in choices
      s = s & "+[" & choice & "] "
    next

    ' If this looks like an undoable disk save question,
    ' automatically answer 'Append' or 'Keep'
    '
    ' NOTE: this code makes a lot of assumptions about the product
    ' and the language used, and may break under some environments.
    ' It is shown for illustration purposes only!

    Set r = new RegExp
    r.pattern = "undoable disk"
    r.ignorecase = True
    Set matches = r.Execute(q.text)

    if matches.count > 0 then
      for i = 1 to choices.count
        if choices(i) = "Append" or choices(i) = "Keep" then
          WScript.Echo(s)
          s = " --> Automatically selecting '" & q.choices(i) & '" as answer"
          vm.AnswerQuestion q,i
          exit for
        end if
      next
    end if

next
end if
    WScript.Echo(s)
next

function State2Str(vm)
    select case vm.ExecutionState
        case vmExecutionState_On
            State2Str = "ON"
        case vmExecutionState_Off
            State2Str = "OFF"
        case vmExecutionState_Suspended
            State2Str = "SUSPENDED"
        case vmExecutionState_Stuck
            State2Str = "STUCK"
        case else
            State2Str = "UNKNOWN"
    end select
end function

The source for the sample program 2 accompanying Windows Script File is in the SampleScripts folder in the VmCOM Scripting API directory.
You can also find it on the VMware Web site, saved with a .TXT extension for online viewing, at www.vmware.com/support/developer/scripting-API/doc/sample2.wsf.txt.

**Note:** If you are using Microsoft Internet Explorer as your browser, select View > Source to view the file. Alternately, right-click this link and download this file.

    <job id="Sample2">
        <reference object="VmCOM.VmCtl" />
        <script language="VBScript" src="sample2.vbs" />
    </job>

**VBScript Sample Program 3**

This VBScript sample program lists, then starts locally registered virtual machines that are not already running on a server. This script powers on powered-off virtual machines and resumes suspended virtual machines that have the line "autostart=true" in their configuration files.
This script includes a slight delay after starting each virtual machine. This delay balances the load on the server. Do not start many virtual machines in rapid succession without this delay.
You can use a script like the following to start selected virtual machines automatically after a server boots. However, this script must be configured as a service for it to run without requiring a login from a user.
Tools exist that allow any application, including a script, to run as a service. One example is the *instsrv* and *srvany* programs from the Microsoft Windows 2000 Resource Kit. If you use *srvany* to implement the service, then configure your service to launch the *cscript* program. Set the program's argument to the path of the script's *.wsf* file. Refer to the Microsoft Windows 2000 Resource Kit documentation for more details. If you choose to use a different tool, then refer to your specific tool's documentation to configure the script to run as a service.

The source for the sample program 3 script is in the SampleScripts folder in the VmCOM Scripting API directory.

You can also find it on the VMware Web site, saved with a .TXT extension for online viewing, at www.vmware.com/support/developer/scripting-API/doc/sample3.vbs.txt.

```
VmCOM VBScript Sample Program 3
Copyright (c) 1998-2004 VMware, Inc.

This program is for educational purposes only.
It is not to be used in production environments.

Description:
This script gets a list of virtual machines registered on
the local server. It attempts to power-on each VM that
is not already running and has a line in the config file:
autostart=true

Instructions for Windows 2000 and Windows XP host:
- save the contents of this file to a file named 'sample3.vbs'
- there should be an accompanying file named 'sample3.wsf'
  It is placed in the same directory as this file during
  product installation. This file is responsible for setting
  up the Windows Script Host environment and loading the
  VmCOM type library, thereby enabling this script to
  reference symbolic constants such as vmExecutionState_On
- in a command line window, type:
cscript //nologo sample3.wsf
```
Set connect_params = CreateObject("VmCOM.VmConnectParams")

' By default, connects to the local server.
' To connect to a remote server, uncomment these lines and set
' the values appropriately.
' connect_params.hostname = "<host>"
' connect_params.username = "<user>"
' connect_params.password = "<password>"
' And use this if your port number is different
' connect_params.port = 902

Set vm_server = CreateObject("VmCOM.VmServerCtl")

' Handle errors non-fatally from here on
On Error Resume Next

' Try connecting to server a few times. It's possible the VMware services
' are still in the process of starting up. We'll wait a maximum of
' 12 * 10 = 120 seconds = 2 minutes
' connected = false
for tries = 1 to 12
    vm_server.Connect connect_params
    if Err.number = 0 then
        connected = true
        exit for
    end if
    WScript.Echo "Could not connect to server: " & Err.Description
    WScript.Echo "Retrying in 10 seconds ..."
    WScript.Sleep 10000
    Err.clear
next

if not connected then
    WScript.Echo "Failed to connect to server. Giving up."
    WScript.Quit
end if

' Get a list of all VMs from the server.
Set vmlist = vm_server.RegisteredVmNames

for each config in vmlist
    ' Connect to the VM
    Set vm = CreateObject("VmCOM.VmCtl")

vm.Connect connect_params, config

if Err.Number <> 0 then
    WScript.Echo "Could not connect to VM " & config & ": " & Err.Description
    Err.Clear
else
    ' Check that the VM should be started automatically
    auto_start = vm.Config("autostart")
    if Err.Number <> 0 then
        if Err.Number <> vmErr_NOPROPERTY then
            WScript.Echo "Could not read autostart variable: " & Err.Number & ": " & Err.Description
        else
            WScript.Echo "This VM is not configured for autostart: " & config
        end if
        Err.Clear
    else
        if auto_start = "true" or auto_start = "TRUE" then
            ' Check that the VM is powered off
            power_state = vm.ExecutionState
            if Err.Number <> 0 then
                WScript.Echo "Error getting execution state: " & Err.Number & ": " & Err.Description
                Err.Clear
            else
                if power_state = vmExecutionState_Off or power_state = vmExecutionState_Suspended then
                    WScript.Echo "Powering on " & config
                    vm.Start(vmPowerOpMode_Soft)
                    if Err.Number <> 0 then
                        WScript.Echo "Error powering on " & config & ": " & Err.Description
                        Err.Clear
                    else
                        ' Wait between starting up VMs to smooth out the load on the server
                        WScript.Sleep 5000
                    end if
                end if
            end if
        end if
    end if
next
The source for the sample program 3 accompanying Windows Script File is in the SampleScripts folder in the VmCOM Scripting API directory.
You can also find it on the VMware Web site, saved with a .TXT extension for online viewing, at www.vmware.com/support/developer/scripting-API/doc/sample3.wsf.txt.

**Note:** If you are using Microsoft Internet Explorer as your browser, select **View > Source** to view the file. Alternately, right-click this link and download this file.

```xml
<job id="sample3">
  <reference object="VmCOM.VmCtl" />
  <script language="VBScript" src="sample3.vbs" />
</job>
```
Using VmPerl

The VmPerl interface provides controlled access to VMware servers and virtual machines. You can incorporate VmPerl function calls in a Perl script you write to automate the day-to-day functioning of your server and virtual machines.

The VmPerl API consists of four modules or packages:

- **VMware::VmPerl::ConnectParams** — that provides connection information and authentication (user credentials) when connecting to a server.
- **VMware::VmPerl::Server** — that controls interaction with a GSX Server or ESX Server machine.
- **VMware::VmPerl::VM** — that controls interaction with a particular virtual machine on a GSX Server or ESX Server.
- **VMware::VmPerl::Question** — that provides for user interaction when there is a question or error condition requiring a response.

VMware::VmPerl::Server and VMware::VmPerl::VM are the primary modules for communicating with VMware components. VMware::VmPerl::ConnectParams and VMware::VmPerl::Question are support modules used as inputs or outputs to the methods and properties of the primary modules.

A VMware::VmPerl::Server object represents a server and exports server-level services, such as virtual machine enumeration and registration. A VMware::VmPerl::VM object represents a virtual
machine on a particular server and provides virtual machine specific methods including power
operations. You activate the VMware::VmPerl::Server or VMware::VmPerl::VM object by calling its
`connect()` method before accessing any other method.

The `connect()` method requires a `$connectparams` input parameter containing the host
identifier and user credentials supplied for authentication. If the host identifier is not supplied or is
undefined, the authentication is performed on the local system. If the user name and password are
also not supplied, the current user is authenticated on the local machine. Otherwise, you may
supply the user name and password for authentication as that user.

Unlike a VMware::VmPerl::Server object, `$vm->connect()` also takes the string `$vm_name`
specifying the configuration file name of the virtual machine that will be connected.

Once a VMware::VmPerl::Server object is connected, you can enumerate the virtual machines
on the server, and register or unregister the virtual machines. You can obtain a list of virtual machines
on a particular server by using the `$server->registered_vm_names()` method. This
method returns an array of strings specifying the configuration file names of the virtual machines
currently registered on the server. If you know the configuration file name of a specific virtual
machine, you can connect the VMware::VmPerl::VM object directly without using a
VMware::VmPerl::Server object.
VMware::VmPerl::ConnectParams

VMware::VmPerl::ConnectParams::new($hostname, $port, $username, $password) connects to the given hostname and network port and authenticates the connection with the supplied user name and password.

The VMware::VmPerl::ConnectParams module supplies connection information and user credentials to the $server->connect() or $vm->connect() methods and exposes the methods listed in the following table. All VMware::VmPerl::ConnectParams methods have both read and write permissions, allowing you to retrieve (GET) and set (PUT) the values.

The security for your connection depends upon the security configuration of your VMware server. If you’re connecting to a VMware server or a virtual machine on a VMware server, then the connections are encrypted as long as the VMware server is configured to encrypt connections.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$connectparams-&gt;get_hostname()</td>
<td>Returns the defined value on success or undef (undefined value) on failure or if the value is not set. Set the value and retry the API call. $connectparams-&gt;set_hostname($hostname) Gets or sets the name of a server, where $hostname is the server’s hostname or IP address. If $hostname is not given or undefined, the authentication is performed on the local system. The C library connects to the local host and uses current user information when it connects. However, this user information is not passed back to $connectparams. Otherwise, you may supply the user name and password for authentication as that user.</td>
</tr>
<tr>
<td>$connectparams-&gt;get_port()</td>
<td>Returns the defined value on success or undef (undefined value) on failure or if the value is not set. Set the value and retry the API call. $connectparams-&gt;set_port($port) Gets or set the TCP port to use when connecting to the server. Its default value is 0 (zero), indicating the default port number (902) should be used. Otherwise, enter the correct port number. A port number set to a negative value is treated as an incorrect value and the default port number is used instead.</td>
</tr>
<tr>
<td>$connectparams-&gt;get_username()</td>
<td>Returns the defined value on success or undef (undefined value) on failure or if the value is not set. Set the value and retry the API call. $connectparams-&gt;set_username($username) Gets or set the name of a user on the server.</td>
</tr>
<tr>
<td>$connectparams-&gt;get_password()</td>
<td>Returns the defined value on success or undef (undefined value) on failure or if the value is not set. Set the value and retry the API call. $connectparams-&gt;set_password($password) Gets or set the user’s password on the server.</td>
</tr>
</tbody>
</table>
VMware::VmPerl::Server

The VMware::VmPerl::Server module represents a VMware server running on a particular machine.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$server-&gt;connect($connectparams)</td>
<td>Connects the object to a VMware GSX Server or a VMware ESX Server where $connectparams specifies the system and user information. The total number of connected VMware::VmPerl::VM and VMware::VmPerl::Server objects cannot exceed 62. The connect() method fails with error code VM_E_INSUFFICIENT_RESOURCES if this limit is reached. In order to connect new objects, destroy one or more connected VMware::VmPerl::VM or VMware::VmPerl::Server objects.</td>
</tr>
<tr>
<td>$server-&gt;get_last_error()</td>
<td>Gets details about the last error that occurred in an array of form [$error_num, $error_string].</td>
</tr>
<tr>
<td>$server-&gt;is_connected()</td>
<td>Use this method to determine whether or not a connection exists to the server specified by $server.</td>
</tr>
<tr>
<td>$server-&gt;registered_vm_names()</td>
<td>Gets an array of strings specifying the configuration file names of the virtual machines currently registered on the server. The array is indexed beginning at 0 (zero). The server must be connected using the connect() method, or this method throws an error.</td>
</tr>
</tbody>
</table>

The remaining methods only work after you connect to the server with $server->connect().

Note: Two methods, $server->get_resource and $server->set_resource, that are listed in the following table, apply only to ESX Server.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$server-&gt;registered_vm_names()</td>
<td>Returns a list of virtual machine configuration file names, an empty list if no virtual machines are registered or if there is a failure. You can use $vm-&gt;get_last_error to determine if an error occurred or there are no registered virtual machines.</td>
</tr>
<tr>
<td>$server-&gt;register_vm($vm_name)</td>
<td>Registers a virtual machine on a server where $vm_name is a string specifying the virtual machine's configuration file name.</td>
</tr>
</tbody>
</table>
### Method

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$server-&gt;unregister_vm($vm_name)</code></td>
<td>Unregisters a virtual machine from a server where <code>$vm_name</code> is a string specifying the virtual machine's configuration file name.</td>
</tr>
<tr>
<td><code>$server-&gt;get_resource(&quot;system.&lt;variable_name&gt;&quot;)</code></td>
<td>Returns the defined value on success or <code>undef</code> (undefined value) on failure.</td>
</tr>
<tr>
<td><code>$server-&gt;set_resource(&quot;system.&lt;variable_name&gt;, &lt;value&gt;&quot;)</code></td>
<td>Gets or sets the value of the ESX Server system resource variable specified by <code>system.&lt;variable_name&gt;</code>. For a list of ESX Server system variables, see VMware ESX Server System Resource Variables on page 120.</td>
</tr>
</tbody>
</table>

**Note:** These methods apply only to ESX Server.
VMware::VmPerl::VM

The VMware::VmPerl::VM object represents a virtual machine running on a particular server.
You can connect to a virtual machine, start, stop, suspend and resume virtual machines, query and modify the configuration file settings, and connect and disconnect devices.
Except where noted otherwise, these methods are synchronous; the method does not return until it finishes its operation, fails or times out. Most operations time out after 2 minutes, except for power operations, which time out after 4 minutes.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vm-&gt;connect($connectparams, $vm_name)</td>
<td>Establishes a connection with a virtual machine using the specified parameters where $connectparams specifies the system and user information and $vm_name is a string specifying the virtual machine’s configuration file name. The total number of connected VMware::VmPerl::VM and VMware::VmPerl::Server objects cannot exceed 62. The connect() method fails with error code VM_E_INSUFFICIENT_RESOURCES if this limit is reached. In order to connect new objects, destroy one or more connected VMware::VmPerl::VM or VMware::VmPerl::Server objects.</td>
</tr>
<tr>
<td>$vm-&gt;get_last_error()</td>
<td>Gets details about the last error that occurred in an array of form [$error_num, $error_string].</td>
</tr>
<tr>
<td>$vm-&gt;is_connected()</td>
<td>Use this method to determine whether or not a connection exists to the virtual machine specified by $vm.</td>
</tr>
</tbody>
</table>

The remaining methods only work after you connect to the virtual machine with $vm->connect().

Note: The following table includes some ESX Server-specific methods, and are specifically noted.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vm-&gt;start($mode)</td>
<td>Returns the defined value on success or <strong>undef</strong> (undefined value) on failure. Powers on a previously powered-off virtual machine or resumes a suspended virtual machine where <strong>$mode</strong> specifies the operation's behavior based on the value of the VMware::VmPerl::VM_POWEROP_MODE_&lt;XXX&gt; where &lt;XXX&gt; is HARD, SOFT, or TRYSOFT. If <strong>$mode</strong> is not specified, the default mode is VM_POWEROP_MODE_SOFT. For more information, see VM_POWEROP_MODE_&lt;XXX&gt; Values on page 63. <strong>Note:</strong> If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM_POWEROP_MODE_HARD as the mode or the operation will fail. If the virtual machine is powered off, then it is powered on. If it is suspended, this method resumes the virtual machine. If the virtual machine is in any other state, the <code>start()</code> method fails and throws an error.</td>
</tr>
<tr>
<td>$vm-&gt;stop($mode)</td>
<td>Returns the defined value on success or <strong>undef</strong> (undefined value) on failure. Shuts down and powers off a virtual machine where <strong>$mode</strong> specifies the operation's behavior based on the value of the VMware::VmPerl::VM_POWEROP_MODE_&lt;XXX&gt; where &lt;XXX&gt; is HARD, SOFT, or TRYSOFT. If <strong>$mode</strong> is not specified, the default mode is VM_POWEROP_MODE_SOFT. For more information, see VM_POWEROP_MODE_&lt;XXX&gt; Values on page 63. <strong>Note:</strong> If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM_POWEROP_MODE_HARD as the mode or the operation will fail. This method always fails if the virtual machine is not in the VM_EXECUTION_STATE_ON state.</td>
</tr>
<tr>
<td>$vm-&gt;reset($mode)</td>
<td>Returns the defined value on success or <strong>undef</strong> (undefined value) on failure. Shuts down, then reboots a virtual machine where <strong>$mode</strong> specifies the operation's behavior based on the value of the VMware::VmPerl::VM_POWEROP_MODE_&lt;XXX&gt; where &lt;XXX&gt; is HARD, SOFT, or TRYSOFT. If <strong>$mode</strong> is not specified, the default mode is VM_POWEROP_MODE_SOFT. See VM_POWEROP_MODE_&lt;XXX&gt; Values on page 63. <strong>Note:</strong> If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM_POWEROP_MODE_HARD as the mode or the operation will fail. This method always fails if the virtual machine is not in the VM_EXECUTION_STATE_ON state.</td>
</tr>
</tbody>
</table>
$vm->suspend($mode)

Returns the defined value on success or undef (undefined value) on failure.

Suspends a virtual machine where $mode specifies the operation’s behavior based on the value of the VMware::VmPerl::VM_POWEROP_MODE_<XXX> where <XXX> is HARD, SOFT, or TRYSOFT. It saves the current state of the virtual machine to a suspend file. If $mode is not specified, the default mode is VM::POWEROP_MODE_SOFT. For more information, see VM::POWEROP_MODE_<XXX> Values on page 63.

Note: If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM_POWEROP_MODE_HARD as the mode or the operation will fail.

This method always fails if the virtual machine is not in the VMware::VmPerl::VM_EXECUTION_STATE_ON state.

$vm->add_redo($disk)

Returns the defined value on success or undef (undefined value) on failure.

Note: This method applies only to ESX Server.

This method adds a redo log to a running virtual SCSI disk specified by $disk, that is associated with the virtual machine specified by $vm. Changes made to the virtual disk accumulate in the new redo log. This disk must be a ESX Server virtual disk stored on a VMFS volume.

The virtual disk can be in persistent, undoable or append mode. The redo log for a virtual disk in persistent mode uses the file name of the virtual disk with .REDO appended to it (for example, if the disk is called vm.dsk, the redo log is called vm.dsk.REDO). A virtual disk in undoable or append mode already has a redo log associated with it, so the new redo log you create is called vm.dsk.REDO.REDO, whose parent is the existing redo log, vm.dsk.REDO.

This method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk already has two redo logs associated with it.

If you add a redo log using the $vm->add_redo () method, but do not commit your changes with the $vm->commit () method, then the redo is automatically committed when the virtual machine is powered off.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vm-&gt;suspend($mode)</td>
<td>Suspends a virtual machine where $mode specifies the operation’s behavior based on the value of the VMware::VmPerl::VM_POWEROP_MODE_&lt;XXX&gt; where &lt;XXX&gt; is HARD, SOFT, or TRYSOFT. It saves the current state of the virtual machine to a suspend file. If $mode is not specified, the default mode is VM::POWEROP_MODE_SOFT. For more information, see VM::POWEROP_MODE_&lt;XXX&gt; Values on page 63. Note: If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM_POWEROP_MODE_HARD as the mode or the operation will fail. This method always fails if the virtual machine is not in the VMware::VmPerl::VM_EXECUTION_STATE_ON state.</td>
</tr>
<tr>
<td>$vm-&gt;add_redo($disk)</td>
<td>This method adds a redo log to a running virtual SCSI disk specified by $disk, that is associated with the virtual machine specified by $vm. Changes made to the virtual disk accumulate in the new redo log. This disk must be a ESX Server virtual disk stored on a VMFS volume. The virtual disk can be in persistent, undoable or append mode. The redo log for a virtual disk in persistent mode uses the file name of the virtual disk with .REDO appended to it (for example, if the disk is called vm.dsk, the redo log is called vm.dsk.REDO). A virtual disk in undoable or append mode already has a redo log associated with it, so the new redo log you create is called vm.dsk.REDO.REDO, whose parent is the existing redo log, vm.dsk.REDO. This method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk already has two redo logs associated with it. If you add a redo log using the $vm-&gt;add_redo () method, but do not commit your changes with the $vm-&gt;commit () method, then the redo is automatically committed when the virtual machine is powered off.</td>
</tr>
</tbody>
</table>
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| $vm->commit($disk, $level, $freeze, $wait) | Returns the defined value on success or *undef* (undefined value) on failure.  
This method commits the changes in a redo log to a running virtual SCSI disk specified by `$disk` that is associated with the virtual machine specified by `$vm`.  
$level can be 0 or 1. When $level is 0, there can be one or two redo logs associated with the disk. If $level is 0, then the top-most redo log (the redo log being modified) is committed to its parent. For example, if there is currently only the disk `vm.dsk` with a single redo log `vm.dsk.REDO`, then the changes in `vm.dsk.REDO` are committed to `vm.dsk`. If a second REDO log `vm.dsk.REDO.REDO` has been added, then the changes in `vm.dsk.REDO.REDO` are committed to `vm.dsk.REDO`.  
$level can be 1 only when there are two redo logs associated with the disk, `vm.dsk.REDO` and `vm.dsk.REDO.REDO`. When $level is 1, the changes in the next-to-top REDO log `vm.dsk.REDO.REDO` are committed to `vm.dsk`. In this case, the virtual machine is not frozen while the redo log is being committed. Also, when the log is committed, `vm.dsk.REDO.REDO` is renamed to `vm.dsk.REDO`.  
$freeze can be 0 or 1. If $freeze is 0, then the virtual machine is not frozen when changes are committed, though it runs more slowly. If $freeze is 1, then the virtual machine is frozen until the commit operation finishes. If $level is 0, then the virtual machine must be frozen when changes are committed and $freeze is ignored.  
$wait can be 0 or 1. If $wait is 0, then the method returns as soon as the commit begins. If $wait is 1, then the method does not return until the commit completes.  
The method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk currently has no redo logs. |
| $vm->get_connected_users() | Returns a list of local and remote connected users, and their IP addresses. This list includes remote console connections, API connections, and Web-based management interface connections to the specified virtual machine. |
| $vm->get_execution_state() | Returns the virtual machine's current state: powered on, powered off, suspended, or stuck. For a list of the execution states, see [VM_EXECUTION_STATE_<XXX> Values on page 63]. |
| $vm->get_guest_info($key_name) | It accesses a shared variable identified by the string $key_name.  
If you write a GuestInfo variable by using the `set_guest_info()` method, the new value is written into memory and is discarded when the virtual machine process terminates.  
For additional information, see [Using VmPerl to Pass User-Defined Information Between a Running Guest Operating System and a Script on page 66]. |
<p>| $vm-&gt;set_guest_info($key_name, $value) | Returns the defined value on success or <em>undef</em> (undefined value) on failure. |</p>
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$vm-&gt;get_config_file_name()</td>
<td>Returns a string containing the configuration file name for the virtual machine. This method fails if the VMware::VmPerl::VM object is not connected.</td>
</tr>
<tr>
<td>$vm-&gt;get_config($key_name)</td>
<td>Accesses the value of a configuration variable identified by the string $key_name. When a virtual machine process is spawned on the server, the process reads configuration variables from the virtual machine's configuration file into memory. If you write a configuration variable by using the $set_config() method, the new value is written into memory and is discarded when the virtual machine process terminates. You cannot change the value of a configuration variable in a virtual machine's configuration file. The method throws an error if it accesses an undefined configuration variable. Do not change the memory size while a virtual machine is suspended. First power off the virtual machine, then change its memory size.</td>
</tr>
<tr>
<td>$vm-&gt;set_config($key_name, $value)</td>
<td>Returns the defined value on success or undefined (undefined value) on failure.</td>
</tr>
<tr>
<td>$vm-&gt;get_product_info($infotype)</td>
<td>Gets information about the product. For additional information, see Infotype Values on page 64.</td>
</tr>
<tr>
<td>$vm-&gt;get_heartbeat()</td>
<td>Returns the current heartbeat count generated by the VMware Tools service running in the guest operating system. The count is initialized to zero when the virtual machine is powered on. The heartbeat count is typically incremented at least once per second when the VMware Tools service is running under light load conditions. The count stays constant if the service is not running.</td>
</tr>
<tr>
<td>$vm-&gt;get_tools_last_active()</td>
<td>Returns an integer indicating how much time has passed, in seconds, since the last heartbeat was detected from the VMware Tools service. This value is initialized to zero when the virtual machine powers on. It stays at zero until the first heartbeat is detected, after which the value is always greater than zero until the virtual machine is power-cycled again. For additional information, see Additional Information on get_tools_last_active on page 60.</td>
</tr>
<tr>
<td>$vm-&gt;get_pending_question()</td>
<td>Returns a Vmware::VmPerl::Question object if the virtual machine is currently in the VM_EXECUTION_STATE_STUCK state. Use $question-&gt;get_text() to retrieve the actual question text. For additional information, see VMware::VmPerl::Question on page 62.</td>
</tr>
</tbody>
</table>
### Method Description

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$vm-&gt;answer_question($question, $choice)</code></td>
<td>Replies to a question where $question represents the question and $choice represents the index of the selected answer to the question. The index is a number associated with an answer. The first choice's index is always 0. The second choice's index is 1, and so on. Use this method to answer the current question or dismiss the current error message when a virtual machine is in the VM_EXECUTION_STATE_STUCK state and requires user input to continue. First, get a VMware::VmPerl::Question object from the VMware::VmPerl::VM object's <code>get_pending_question()</code> method. You can retrieve the possible choices and their respective indices from the VMware::VmPerl::Question object's <code>get_choices()</code> method. Then, use the <code>answer_question()</code> method to answer the question.</td>
</tr>
<tr>
<td><code>$vm-&gt;device_is_connected($dev_name)</code></td>
<td>Determines the connection state where $dev_name identifies the virtual device.</td>
</tr>
<tr>
<td><code>$vm-&gt;connect_device($dev_name)</code></td>
<td>Sets a virtual device to the connected state where $dev_name identifies the virtual device you want to connect. The virtual machine must be powered on for this method to succeed, otherwise a VM_E_BADSTATE error is returned. Use the <code>set_config()</code> method to set configuration parameters relevant to the virtual device before calling the <code>connect_device()</code> method. The following code example illustrates connecting a virtual drive to a CD image file: <code>$vm-&gt;set_config(&quot;ide1:0.devicetype&quot;) = &quot;cdrom-image&quot;</code> <code>$vm-&gt;set_config(&quot;ide1:0.filename&quot;) = &quot;/iso/foo.iso&quot;</code> <code>$vm-&gt;connect_device(&quot;ide1:0&quot;)</code></td>
</tr>
<tr>
<td><code>$vm-&gt;disconnect_device($dev_name)</code></td>
<td>Sets a virtual device to the disconnected state where $dev_name is a string identifying the virtual device you want to disconnect. The virtual machine must be powered on for this method to succeed, otherwise a VM_E_BADSTATE error is returned.</td>
</tr>
</tbody>
</table>

```perl
$vm->answer_question($question, $choice)

Returns the defined value on success or undef (undefined value) on failure.

$vm->device_is_connected($dev_name)

Returns the defined value on success or false on failure (if the device is not connected or if there is a failure). You can use $vm->get_last_error to determine if an error occurred or if the device is not connected.

$vm->connect_device($dev_name)

Returns the defined value on success or undef (undefined value) on failure. Sets a virtual device to the connected state where $dev_name identifies the virtual device you want to connect. The virtual machine must be powered on for this method to succeed, otherwise a VM_E_BADSTATE error is returned. Use the `set_config()` method to set configuration parameters relevant to the virtual device before calling the `connect_device()` method. The following code example illustrates connecting a virtual drive to a CD image file: $vm->set_config("ide1:0.devicetype") = "cdrom-image" $vm->set_config("ide1:0.filename") = "/iso/foo.iso" $vm->connect_device("ide1:0")

$vm->disconnect_device($dev_name)

Returns the defined value on success or undef (undefined value) on failure. Sets a virtual device to the disconnected state where $dev_name is a string identifying the virtual device you want to disconnect. The virtual machine must be powered on for this method to succeed, otherwise a VM_E_BADSTATE error is returned.
```
### Additional Information on `get_tools_last_active`

If the guest operating system is heavily loaded, this value may occasionally reach several seconds. If the service stops running, either because the guest operating system has experienced a failure or is shutting down, the value keeps increasing.

You can use a script with the `get_tools_last_active()` method to monitor the start of the VMware Tools service, and once started, the health of the guest operating system. If the guest operating system has failed, the `get_tools_last_active()` method indicates how long the guest has been down. The following table summarizes how you may interpret the `get_tools_last_active()` method values:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$vm-&gt;get_resource (&lt;variable_name&gt;)</code></td>
<td>Gets or sets the value of the virtual machine resource variable specified by <code>&lt;variable_name&gt;</code>. For a list of virtual machine resource variables, see Virtual Machine Resource Variables for ESX Server on page 124.</td>
</tr>
<tr>
<td><code>$vm-&gt;set_resource (&lt;variable_name&gt;, &lt;value&gt;)</code></td>
<td>Returns 1 on success or <code>undef</code> (undefined value) on failure.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_uptime()</code></td>
<td>Accesses the uptime of the guest operating system on the virtual machine.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_id()</code></td>
<td>Returns the unique (world) ID for a running virtual machine.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_pid()</code></td>
<td>Returns the process ID of a running virtual machine.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_capabilities()</code></td>
<td>Returns the access permissions for the current user. This number is a bit vector, where 4=read, 2=write, and 1=execute. For a user with all three permissions, a value of 7 is returned when this property is used in a script.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_remote_connections()</code></td>
<td>Returns the number of remotely connected users. This value includes the number of remote consoles, Scripting APIs, and Web-based management interface connections to the virtual machine.</td>
</tr>
<tr>
<td><code>$vm-&gt;set_runas_user($user, $password)</code></td>
<td>Runs the virtual machine as the user specified by the $user and $password.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_runas_user()</code></td>
<td>Returns the name of the user running the virtual machine.</td>
</tr>
<tr>
<td>get_tools_last_active Method Value</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0</td>
<td>The VMware Tools service has not started since the power-on of the virtual machine.</td>
</tr>
<tr>
<td>1</td>
<td>The VMware Tools service is running and is healthy.</td>
</tr>
<tr>
<td>2, 3, 4, or 5</td>
<td>The VMware Tools service could be running, but the guest operating system may be heavily loaded or is experiencing temporary problems.</td>
</tr>
<tr>
<td>Greater than 5</td>
<td>The VMware Tools service stopped running, possibly because the guest operating system experienced a fatal failure, is restarting, or is shutting down.</td>
</tr>
</tbody>
</table>
VMware::VmPerl::Question

The VMware::VmPerl::Question method describes a question or error condition requiring input. The script selects one from the list of possible answers.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$question-&gt;get_text()</td>
<td>Returns the defined value on success or undef (undefined value) on failure.</td>
</tr>
<tr>
<td>$question-&gt;get_choices()</td>
<td>Returns the defined value on success or undef (undefined value) on failure.</td>
</tr>
<tr>
<td>$question-&gt;get_id()</td>
<td>Gets an integer used internally by VmPerl to identify the question.</td>
</tr>
</tbody>
</table>
Symbolic Constants

The VMware::VmPerl::VM object exposes the following symbolic constants:

- **VM_EXECUTION_STATE_<XXX> Values**
- **VM_POWEROP_MODE_<XXX> Values**
- Infotype Values
- **VM_PRODINFO_PRODUCT_<XXX> Values**
- **VM_PRODINFO_PLATFORM_<XXX> Values**

### VM_EXECUTION_STATE_<XXX> Values

VM_EXECUTION_STATE_<XXX> values specify the state (or condition) of a virtual machine. The possible values are listed in the following table:

<table>
<thead>
<tr>
<th>Execution state Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM_EXECUTION_STATE_ON</td>
<td>The virtual machine is powered on.</td>
</tr>
<tr>
<td>VM_EXECUTION_STATE_OFF</td>
<td>The virtual machine is powered off.</td>
</tr>
<tr>
<td>VM_EXECUTION_STATE_SUSPENDED</td>
<td>The virtual machine is suspended.</td>
</tr>
<tr>
<td>VM_EXECUTION_STATE_STUCK</td>
<td>The virtual machine requires user input. The user must answer a question or dismiss an error.</td>
</tr>
<tr>
<td>VM_EXECUTION_STATE_UNKNOWN</td>
<td>The virtual machine is in an unknown state.</td>
</tr>
</tbody>
</table>

### VM_POWEROP_MODE_<XXX> Values

VMware::VmPerl::VM::POWEROP_MODE_<XXX> specifies the behavior of a power transition (start, stop, reset, or suspend) method. If $mode is not specified, the default mode is VM::POWEROP_MODE_SOFT. However, if you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM::POWEROP_MODE_HARD as the mode or the operation will fail.

During a soft power transition, the VMware Tools service runs a script inside the guest operating system. For example, the default scripts that run during suspend and resume operations, respectively release and renew DHCP leases, for graceful integration into most corporate LANs. You may also customize these scripts. For more information on these scripts, see your VMware product documentation. Refer to the section on executing scripts.

The possible values are listed in the following table:
### Powerop_mode Values

<table>
<thead>
<tr>
<th>Powerop_mode Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM_POWEROP_MODE_SOFT</td>
<td>Start when a virtual machine is suspended — After resuming the virtual machine, the operation attempts to run a script in the guest operating system to restore network connections by renewing the DHCP lease. The Start0 operation always succeeds. However, if the VMware Tools service is not present or is malfunctioning, the running of the script may fail. Start when virtual machine is powered off — After powering on the virtual machine, it attempts to run a script in the guest operating system when the VMware Tools service becomes active. This default script does nothing during this operation as there is no DHCP lease to renew. The Start0 operation always succeeds. However, if the VMware Tools service is not present or is malfunctioning, the running of the script may fail. Stop — Attempts to shut down the guest operating system and then powers off the virtual machine. Reset — Attempts to shut down the guest operating system, then reboots the virtual machine. Suspend — Attempts to run a script in the guest operating system that safely disables network connections (such as releasing a DHCP lease) before suspending the virtual machine.</td>
</tr>
<tr>
<td>VM_POWEROP_MODE_HARD</td>
<td>Start — Starts or resumes a virtual machine without running any scripts, a standard power on or resume. Stop, reset or suspend — Immediately and unconditionally powers off, resets, or suspends the virtual machine.</td>
</tr>
<tr>
<td>VM_POWEROP_MODE_TRYSOFT</td>
<td>First attempts to perform the power transition operation with VM_POWEROP_MODE_SOFT. If this fails, the same operation is performed with VM_POWEROP_MODE_HARD.</td>
</tr>
</tbody>
</table>

### Infotype Values

$\text{infotype}$ specifies the product information for the get_product_info() method.

<table>
<thead>
<tr>
<th>Infotype Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM_PRODINFO_PRODUCT</td>
<td>The VMware product is returned as VmProduct. For more information on VmProduct, see the following section.</td>
</tr>
<tr>
<td>VM_PRODINFO_PLATFORM</td>
<td>The host’s operating system is returned as VmPlatform. For more information on VmPlatform, see VM_PRODINFO_PLATFORM_&lt;XXX&gt; Values on page 65.</td>
</tr>
<tr>
<td>VM_PRODINFO_BUILD</td>
<td>The product’s build number.</td>
</tr>
<tr>
<td>VM_PRODINFO_VERSION_MAJOR</td>
<td>The product’s major version number.</td>
</tr>
</tbody>
</table>
CHAPTER 4 Using VmPerl

VM_PRODINFO_PRODUCT_<XXX> Values
The get_product_info method returns the VMware product when the requested $infotype is VM_PRODINFO_PRODUCT_<XXX>.

<table>
<thead>
<tr>
<th>VM_PRODINFO_PRODUCT Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM_PRODUCT_WS</td>
<td>The product is VMware Workstation.</td>
</tr>
<tr>
<td>VM_PRODUCT_GSX</td>
<td>The product is VMware GSX Server.</td>
</tr>
<tr>
<td>VM_PRODUCT_ESX</td>
<td>The product is VMware ESX Server.</td>
</tr>
<tr>
<td>VM_PRODUCT_UNKNOWN</td>
<td>The product is unknown.</td>
</tr>
</tbody>
</table>

VM_PRODINFO_PLATFORM_<XXX> Values
The get_product_info method returns the host's platform when the requested $infotype is VM_PRODINFO_PLATFORM_<XXX>.

<table>
<thead>
<tr>
<th>VM_PRODINFO_PLATFORM Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM_PLATFORM_WINDOWS</td>
<td>The platform is a Microsoft Windows operating system.</td>
</tr>
<tr>
<td>VM_PLATFORM_LINUX</td>
<td>The platform is a Linux operating system.</td>
</tr>
<tr>
<td>VM_PLATFORM_VMNX</td>
<td>The platform is the VMware Service Console.</td>
</tr>
<tr>
<td>VM_PLATFORMUNKNOWN</td>
<td>The platform is unknown.</td>
</tr>
</tbody>
</table>
Using VmPerl to Pass User-Defined Information Between a Running Guest Operating System and a Script

When the guest operating system is running inside a virtual machine, you can pass information from a script (running in another machine) to the guest operating system, and from the guest operating system back to the script, through the VMware Tools service. You do this by using a class of shared variables, commonly referred to as GuestInfo. VMware Tools must be installed and running in the guest operating system before a GuestInfo variable can be read or written inside the guest operating system.

For example, create and connect a VMware::VmPerl::VM object, assuming the virtual machine is powered off. Next, set the GuestInfo variable with the VmPerl API. Then, power on the virtual machine and use the VMware Tools service to retrieve the variable. See Sending Information Set in a VmPerl Script to the Guest Operating System on page 67 for an example of this procedure.

See your VMware product documentation for more information about VMware Tools.

GuestInfo Variables

You pass to the virtual machine variables you define yourself. What you pass is up to you, but you might find it useful to pass items like the virtual machine’s IP address, Windows system ID (SID, for Windows guest operating systems) or machine name.

This is useful in situations where you want to deploy virtual machines on a network using a common configuration file, while providing each machine with its own unique identity. By providing each virtual machine with a unique identifying string, you can use the same configuration file to launch the same nonpersistent virtual disk multiple times in a training or testing environment, where each virtual machine would be unique on the network. Note that in the case of persistent or undoable disks, each virtual disk file must be copied into its own directory if it shares its file name with another virtual disk file.

When a virtual machine process is created on the server, all GuestInfo variables are initially undefined. A GuestInfo variable is created the first time it is written.

You identify a GuestInfo variable with a key name. You can define and create any number of GuestInfo variable key names. The information you pass is temporary, lasting until the virtual machine is powered off and all consoles connected to the virtual machine are closed.

For an example showing how the VMware guest service can be invoked in a Perl script, see the sample Perl script to get the IP address of a guest operating system on Setting a Virtual Machine’s IP Address Configuration Variable on page 88.
Sending Information Set in a VmPerl Script to the Guest Operating System

To send information from a VmPerl script to a running guest operating system, you use VmPerl API's $vm->set_guest_info() method. You need to specify a variable name ($key_name) and its value ($value).

For example, you might want to deploy virtual machines for a training class. When a virtual machine starts, you want to display a banner welcoming the student to the class. You can pass their name from a VmPerl script to the guest operating system on a student's virtual machine.

If you have not already done so, connect a VMware::VmPerl::VM object and set the student's name for this virtual machine to "Susan Williams":

    $vm->set_guest_info("name", "Susan Williams");

This statement passes a string "name" to the guest operating system. You can write a script that reads the string, then calls a command (specific to the guest operating system) to set the student’s name in the banner. This operation is explained in the following section.

This setting lasts until you power off the virtual machine and close all connected consoles.

Retrieving the Information in the Guest Operating System

In the running guest operating system, you use the VMware Tools service to retrieve variables set for the virtual machine. You can then use this passed "name" string inside a guest operating system startup sequence. Use the following to read the GuestInfo variable key_name.

In a Windows guest operating system:

    VMwareService.exe --cmd "info-get guestinfo.<key_name>"

In a Linux guest operating system:

    /etc/vmware-tools/vmware-guestd --cmd 'info-get guestinfo.<key_name>'

For example, to get the current value for the "name" variable, you can type the following in a Linux guest operating system:

    /etc/vmware-tools/vmware-guestd --cmd 'info-get guestinfo.name'

Sending Information Set in the Guest Operating System to a VmPerl Script

Similarly, in a virtual machine's guest operating system, you can use the VMware Tools service to set GuestInfo variables for the virtual machine. Use the following to write the GuestInfo variable key_name.

In a Windows guest operating system:

    VMwareService.exe --cmd "info-set guestinfo.<key_name> <value>"
In a Linux guest operating system:

```
/etc/vmware-tools/vmware-guestd --cmd 'info-set guestinfo.<key_name> <value>'
```

Continuing with the previous example, Susan Williams prefers “Sue”. To set the value of “Sue Williams” for the “name” variable, type the following in a Linux guest operating system:

```
/etc/vmware-tools/vmware-guestd --cmd 'info-set guestinfo.name Sue Williams'
```

### Retrieving Information in a VmPerl Script

With the VmPerl API, you use the `$vm->get_guest_info()` method to retrieve information set in the guest operating system, into a VmPerl script running on any machine, including GSX Server or any remote workstation that can connect to the virtual machine.

For example, to retrieve Sue’s name set by the VMware Tools service, query the guest operating system by using the VmPerl API:

```
$vm->get_guest_info('name')
```
Using Sample VmPerl Scripts

This section contains sample Perl scripts written by VMware to demonstrate example uses of the VmPerl API. You can modify these scripts to suit the needs of your organization. These scripts are located in the SampleScripts subdirectory in the VmPerl directory or on the VMware Web site.

**Note:** The scripts on the Web site are saved with a .TXT extension for online viewing. Remove the .TXT extension before using these scripts.

The sample scripts illustrate:

- Listing the Virtual Machines on the Server
- Starting All Virtual Machines on a Server
- Checking a Virtual Machine’s Power Status
- Monitoring a Virtual Machine’s Heartbeat
- Answering Questions Posed by a Virtual Machine
- Suspending a Virtual Machine
- Setting a Virtual Machine’s IP Address Configuration Variable
- Getting a Virtual Machine’s IP Address
• Adding a Redo Log to a Virtual Disk (ESX Server only)
• Committing a Redo Log to a Virtual Disk without Freezing the Virtual Machine (ESX Server only)

**Note:** If you plan on using the VMware Perl API remotely on a Windows machine, you must copy your scripts into the same directory in which you installed the VMware Perl API.
Copyright Information

Each sample script and sample program included with the VmPerl Scripting API includes a copyright. However, for brevity, we do not include this copyright in its entirety with each sample script and sample program in this manual. Instead, we include the first line of the copyright followed by ellipses, to indicate its placement. The complete copyright is as follows:

Copyright (c) 1998-2004 VMware, Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of the software in this file (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The names "VMware" and "VMware, Inc." must not be used to endorse or promote products derived from the Software without the prior written permission of VMware, Inc.

Products derived from the Software may not be called "VMware", nor may "VMware" appear in their name, without the prior written permission of VMware, Inc.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL VMware, Inc. BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
Listing the Virtual Machines on the Server

You can use a script like the following to generate a list of all the registered virtual machines on a server. You need to know the name of the machine and you must provide a valid user name and password to connect to the server.

This script (enumerate.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/enumerate.pl.txt.

#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
#
#
# enumerate.pl
#
# This script lists all of the registered virtual machines
# on the server specified by hostname.
#
# usage:
#   enumerate.pl <hostname> <user> <password>
#
BEGIN {
  if ($^O eq "MSWin32") {
    @INC = {
      # Set the path to your VmPerl Scripting directory if different
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005',
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005\MSWin32-x86');
    }
  }

use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;
use strict;

my ($server_name, $user, $passwd) = @ARGV;

# Use the default port of 902. Change this if your port is different.
my $port = 902;

# Create a new VMware::VmPerl::Server to connect to the server
# To connect to the remote server, use the following line:
my $connect_params =
   VMware::VmPerl::ConnectParams::new($server_name, $port, $user, $passwd);

# To connect to a local server, you would use the following line:
# my $connect_params =
#   VMware::VmPerl::ConnectParams::new(undef, $port, $user, $passwd);

# To connect to a local server as the current user, you would use the
# following line:
# my $connect_params = VMware::VmPerl::ConnectParams::new();

# Establish a persistent connection with server
my $server = VMware::VmPerl::Server::new();
if (!$server->connect($connect_params)) {
    my ($error_number, $error_string) = $server->get_last_error();
    die "Could not connect to server: Error $error_number: $error_string\n";
}

print \n"The following virtual machines are registered:\n";

# Obtain a list containing every config file path registered with the server.
my @list = $server->registered_vm_names();
if (!defined($list[0])) {
    my ($error_number, $error_string) = $server->get_last_error();
    die "Could not get list of VMs from server: Error $error_number: "$error_string\n";
}

print "$\n" foreach (@list);

# Destroys the server object, thus disconnecting from the server.
undef $server;
Starting All Virtual Machines on a Server

You can use a script like the following to start all virtual machines that are not already running on a server. This script powers on powered-off virtual machines and resumes suspended virtual machines that have the line "autostart=true" in their configuration files.

This script includes a slight delay after starting each virtual machine. This delay balances the load on the server. Do not start many virtual machines in rapid succession without this delay.

This script (startallvms.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/startallvms.pl.txt.

```perl
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
#
# startallvms.pl
#
# This script powers on all VMs on the system that are not
# already running.
#
# usage:
#   startallvms.pl <hostname> <user> <password>
#
BEGIN {
  if ($^O eq "MSWin32") {
    @INC = (
      # Set the path to your VmPerl Scripting directory if different
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl15\site_perl\5.005',
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl15\site_perl\5.005\MSWin32-x86');
    }
  }

use VMware::VmPerl;
use VMware::VmPerl::VM;
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;
use strict;

my ($server_name, $user, $passwd) = @ARGV;

# Change this to your port if it is different.
my $port = 902;
```
# Create a ConnectParams object
my $connect_params = VMware::VmPerl::ConnectParams::new($server_name,$port,$user,$passwd);

# Create a Server object
my $server = VMware::VmPerl::Server::new();

# Establish a persistent connection with server
if (!$server->connect($connect_params)) {
    my ($error_number, $error_string) = $server->get_last_error();
    die "Could not connect to server: Error $error_number: $error_string\n";
}

# Get a list of all virtual machine configuration files registered
# with the server.
my @list = $server->registered_vm_names();

if(!defined($list[0])) {
    my ($error_number, $error_string) = $server->get_last_error();
    die "Could not get list of VMs: Error $error_number: $error_string\n";
}

my $config;
foreach $config (@list) {
    my $vm = VMware::VmPerl::VM::new();
    # Connect to the VM, using the same ConnectParams object.
    if ($vm->connect($connect_params, $config)) {
        my ($error_number, $error_string) = $server->get_last_error();
        print STDERR "Could not connect to VM $config: Error $error_number: "$error_string\n";
    } else {
        # Only power on VMs with the config setting autostart = "true"
        my $autostart = $vm->get_config("autostart");
        if($autostart && $autostart =~ /true/i) {
            # Only try this for VMs that are powered off or suspended.
            my $power_state = $vm->get_execution_state();
            if (!defined($power_state)) {
                my ($error_number, $error_string) = $server->get_last_error();
                print STDERR "Could not get execution state of VM $config: Error ".
            } else {

        }
    }
}
"$error_number: $error_string\n";
} elsif ($power_state == VM_EXECUTION_STATE_OFF ||
    $power_state == VM_EXECUTION_STATE_SUSPENDED) {

    print "Powering on $config...\n";
    if (!$vm->start()) {
        # If an error occurs, report it and continue
        my ($error_number, $error_string) = $server->get_last_error();
        print STDERR "Could not power on VM $config: Error ".
        "$error_number: $error_string\n";
    } else {

        # Delay slightly between starting each VM.
        # This prevents too much initial load on the server.
        # Warning: starting many VMs in rapid succession
        # is not recommended.

        sleep 5;
    }

    # Destroys the virtual machine object, thus disconnecting from the virtual machine.
    undef $vm;
}

# Destroys the server object, thus disconnecting from the server.
undef $server;
Checking a Virtual Machine’s Power Status

You can use a script like the following to determine whether a virtual machine is running, suspended or powered off. Once you know its power status, you can use this information in conjunction with other scripts to start, stop or suspend a virtual machine.

This script (status.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/status.pl.txt.

```perl
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
#
# status.pl
#
# This script returns the current power status (on, off, suspended) of the
# virtual machine specified by config on the server defined by hostname.
#
# usage:
# status.pl <path_to_config_file> [server user password]
#
# If server, user and password are not given, connect to the local server
# as the current user.
#
BEGIN { 
  if ($^O eq "MSWin32") { 
    @INC = ( 
      # Set the path to your VmPerl Scripting directory if different
      'C:\Program Files\VMware\VmPerl Scripting API\perl5\site_perl\5.005',
      'C:\Program Files\VMware\VmPerl Scripting API\perl5\site_perl\5.005\MSWin32-x86');
  }
}

use VMware::VmPerl;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;
use strict;

# Retrieves a pre-defined constant value.
sub vm_constant { 
  my $constant_str = shift;
  return VMware::VmPerl::constant($constant_str, 0);
} 
```
if ($@ARGV < 1) {
    print "Usage $0: <path_to_config_file> [<server> <user> <password>]\n";
    exit(1);
}

my $state_string_map = {};
my @state_strings = ("VM_EXECUTION_STATE_ON", "VM_EXECUTION_STATE_OFF", "VM_EXECUTION_STATE_SUSPENDED", "VM_EXECUTION_STATE_STUCK", "VM_EXECUTION_STATE_UNKNOWN" );

foreach my $state_string (@state_strings) {
    $state_string_map->{vm_constant($state_string)} = $state_string;
}

# Read in parameters.
my ($cfg_path, $server_name, $user, $passwd) = @ARGV;

# Use the default port of 902. Change this if your port is different.
my $port = 902;

my $connect_params = VMware::VmPerl::ConnectParams::new($server_name, $port, $user, $passwd);

my $vm = VMware::VmPerl::VM::new();
if (!$vm->connect($connect_params, $cfg_path)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not connect to vm: Error $error_number: $error_string\n";
}

# Get the power status of the virtual machine.
my $cur_state = $vm->get_execution_state();
if (!defined($cur_state)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not get execution state: Error $error_number: $error_string\n";
}
print "The execution state of $cfg_path is: $state_string_map->{$cur_state}\n";

# Destroys the virtual machine object, thus disconnecting from the virtual machine.
undef $vm;
CHAPTER 5  Using Sample VmPerl Scripts

Monitoring a Virtual Machine’s Heartbeat

The following sample Perl script provides one method to monitor a virtual machine’s heartbeat. If the heartbeat is lost or is not detected, the script powers on a second instance of the virtual machine.

This script (hb_check.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/hbcheck.pl.txt.

#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
# .
# .
# .
#
# hbcheck.pl
#
# You can use this script to check the virtual machine specified by
# ConfigToCheck for a heartbeat within a certain interval in seconds.
# If no heartbeat is received within the specified Interval, then this
# script will forcefully shutdown ConfigToCheck, and start ConfigToStart.
# 
# usage:
#  hbcheck.pl <ConfigToCheck> <ConfigToStart> [Interval]
#
BEGIN {
  if ($^O eq "MSWin32") {
    @INC = ( 
      # Set the path to your VmPerl Scripting directory if different
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005',
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005\MSWin32-x86');
  }
}

# Import required VMware Perl modules and version.
use VMware::VmPerl;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;
use strict;

# Display the script usage.
sub usage() {
  print STDERR "Usage: hbcheck.pl <config_to_check> <config_to_start> [interval_in_secs]\n";
  exit(1);
}

# Retrieves a pre-defined constant value.
sub vm_constant {
    my $constant_str = shift;
    return VMware::VmPerl::constant($constant_str, 0);
}

# Read in command line options.
usage() unless (scalar(@ARGV) == 3 || scalar(@ARGV) == 2);
my $cfg_to_check = shift;
my $cfg_to_start = shift;
my $interval = shift;

# Set the interval to 30 seconds if it is not specified.
$interval ||= 30;

# Connect to the local host on the default port as the current user.
# Change the port number if it is different.
my $connect_params = VMware::VmPerl::ConnectParams::new(undef, 902, undef, undef);

# Initialize the object for the virtual machine we want to check.
my $vm = VMware::VmPerl::VM::new();
if (!$vm->connect($connect_params, $cfg_to_check)) {
    my ($error_number, $error_string) = $vm->get_last_error();
    die "Could not connect to virtual machine at $cfg_to_check:
    Error $error_number: $error_string"
}

# Check to see if the virtual machine is powered on; if not, end.
my $vm_state = $vm->get_execution_state();
if (!($vm_state eq vm_constant("VM_EXECUTION_STATE_ON"))) {
    # Destroys the virtual machine object, thus disconnecting from the virtual machine
    undef $vm;
    die "The virtual machine $cfg_to_check is not powered on. Exiting."
}

# Maintain the last read heartbeat value for comparison.
# The heartbeat count begins at zero, so a value of -1 ensures
# at least one comparison.
my $last_hb = -1;

while ($vm->is_connected()) {

    # Get the current heartbeat count.  This should steadily increase
    # as long as VMware tools is running inside the virtual machine.
    my $hb = $vm->get_heartbeat();
    unless (defined $hb) {

my ($error_number, $error_string) = $vm->get_last_error();
die "Could not get virtual machine heartbeat:
".
"Error $error_number: $error_string
";
}

if ($hb == $last_hb) {
    # Since we don't have a heartbeat, we need to do something
    # about it. Let's shut this virtual machine down, and then start
    # the backup virtual machine (specified by $vm_to_start).
    # Use the "TRYSOFT" mode to shutdown gracefully if possible.
    $vm->stop(vm_constant("VM_POWEROP_MODE_TRYSOFT"));
    undef $vm;

    # Initialize the new virtual machine object.
    my $vm_to_start = VMware::VmPerl::VM::new();
    if (!$vm_to_start->connect($connect_params, $cfg_to_start)) {
        my ($error_number, $error_string) = $vm_to_start->get_last_error();
        die "Could not connect to virtual machine at $cfg_to_start:
" .
        "Error $error_number: $error_string\n"
    }

    # Start the new virtual machine and clean up.
    my $start_ok = $vm_to_start->start();
    unless ($start_ok) {
        my ($error_number, $error_string) = $vm_to_start->get_last_error();
        undef $vm_to_start;
        die "Could not start virtual machine $cfg_to_start:
" .
        "Error $error_number: $error_string\n"
    }
    undef $vm_to_start;
    die "Lost heartbeat of $cfg_to_check, powered on $cfg_to_start.\n";
} else {
    # Wait $interval seconds before checking for the virtual machine's heartbeat.
    print "Got heartbeat count $hb\n";
    sleep ($interval);
}
$last_hb = $hb;
Answering Questions Posed by a Virtual Machine

You can use a script like the following to answer a question posed by a virtual machine in a stuck state; that is, one that is waiting for user acknowledgment before it can complete an operation such as suspending or resuming the virtual machine. The script allows the question to be answered at the command line, saving you the effort of connecting to the virtual machine from a console or the VMware Management Interface in order to answer the question.

This script (answer_question.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/answerquestion.pl.txt.

#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
# answerquestion.pl
#
# You can use this script to check if the virtual machine specified by
# config is stuck. If it's stuck, you can answer any question posed by this
# virtual machine to allow it to continue.
#
# usage:
#   answerquestion.pl <config-file>

BEGIN {
    if ($^O eq "MSWin32") {
        @INC = (
            # Set the path to your VmPerl Scripting directory if different
            "C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005",
            "C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005\MSWin32-x86"),
        )
    }

    # Import the required VMware Perl modules and version.
    use VMware::VmPerl;
    use VMware::VmPerl::VM;
    use VMware::VmPerl::ConnectParams;
    use VMware::VmPerl::Question;
    use strict;
# Read in command line options.
my $cfg = shift or die "Usage: $0 <config-file>\n";

# Connect to the local host on the default port as yourself.
my $connect_params = VMware::VmPerl::ConnectParams::new();

# Initialize the object for the virtual machine we want to check.
my $vm = VMware::VmPerl::VM::new();
my $vm_ok = $vm->connect($connect_params, $cfg);
unless ($vm_ok) {
    my ($err, $errstr) = $vm->get_last_error();
    undef $vm;
    die "Could not connect to vm; error $err: $errstr\n";
}

# Check the power state of the virtual machine. If it's stuck, get the
# question and list the possible responses.
my $state = $vm->get_execution_state();
if (!defined($state)) {
    my ($err, $errstr) = $vm->get_last_error();
    undef $vm;
    die "Could not get execution state of vm; error $err: $errstr\n";
}

if ($state ne VM_EXECUTION_STATE_STUCK) {
    print "There is no question to answer.\n";
} else {
    my $q = $vm->get_pending_question();
    unless (defined($q)) {
        undef $vm;
        die "Could not get the pending question.\n";
    }
    my $text = $q->get_text();
    unless (defined($text)) {
        undef $vm;
        die "Could not get the text of the pending question.\n";
    }
    my @choices = $q->get_choices();
    unless (defined($choices[0])) {
        undef $vm;
        die "Could not get the choices to answer the pending question.\n";
    }
    # Print question and choices for user:
    print "\n" . $q->get_text() . "\n";
my $answer;
do {
    prompt(@choices);
    $answer = get_answer();
} until (valid_answer($answer, @choices));

my $op_ok;
$op_ok = $vm->answer_question($q, $answer-1);
unless ($op_ok) {
    my ($err, $errstr) = $vm->get_last_error();
    undef $vm;
    die "Could not answer pending question; error $err: $errstr\n";
}

# Destroys the virtual machine object, thus disconnecting from the virtual machine.
undef $vm;

#------------------------------------------------
# Prints answer choices, prompts user for an answer number.
sub prompt {
    my @choices = shift;
    print "To answer the question, type the number that corresponds to one of the answers below:\n";
    for (my $i = 0; $i <= $#choices; $i++) {
        print " \t" . ($i + 1) . " . $choices[$i]\n";
    }
    print "Final answer? \n";
}

# Reads user's answer number.
sub get_answer {
    my $answer;
    chop($answer = <STDIN>);
    print "\n";

    # Remove unintentional whitespace.
    $answer =~ s/\(\s*\)\(.*?\)\(\s*\)\$/2/;
    return $answer;
}

# Checks if an answer number is within the valid range of choices.
sub valid_answer {
    my $answer = shift;
    my @choices = shift;

$answer--; # convert to 0-based indexing.
if ($answer < 0 || $answer > $#choices) {
  my $num = scalar(@choices);
  print "Valid answer numbers are from 1 to $num; please try again.\n";
  return 0;
} else {
  return 1;
}
Suspending a Virtual Machine

A script like the following allows you to suspend a virtual machine remotely without connecting to it through a remote console or the VMware Management Interface.

This script (suspend.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/suspend.pl.txt.

```
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
# suspend.pl
#
# This script suspends to disk the virtual machine specified by config on
# the server defined by hostname.
#
# usage:
#   suspend.pl hostname user password config
BEGIN {
  if ($^O eq "MSWin32") {
    @INC = (
      # Set the path to your VmPerl Scripting directory if different
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005',
      'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005\MSWin32-x86');
  }
}
use VMware::VmPerl;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;
use strict;
if (@ARGV < 1) {
  print "Usage $0: <path_to_config_file> [<server> [<user> <password>]]\n";
  exit(1);
}
my ($cfg_path, $server_name, $user, $passwd) = @ARGV;
# Use the default port of 902. Change this if your port is different.
my $port = 902;
# Connect to the local host on the default port as yourself.
```
my $connect_params = VMware::VmPerl::ConnectParams::new($server_name,$port,$user,$passwd);

# Create a new VMware::VmPerl::VM object to interact with a virtual machine.
my $vm = VMware::VmPerl::VM::new();

# Establish a persistent connection with virtual machine.
if (!$vm->connect($connect_params, $cfg_path)) {
    my ($errorNumber, $errorString) = $vm->get_last_error();
    # Destroys the virtual machine object, thus disconnecting from the virtual machine.
    undef $vm;
    die "Cannot connect to vm: Error $errorNumber: $errorString\n";
}

# Gets the Power status of the virtual machine to determine if it is running.
my $curState = $vm->get_execution_state();
if ($curState != VM_EXECUTION_STATE_ON) {
    print "Can only suspend a powered on Virtual Machine.\n";
} else {
    # Suspends the running vm.
    if (!$vm->suspend()) {
        my ($errorNumber, $errorString) = $vm->get_last_error();
        print "Couldn't suspend: Error $errorNumber: $errorString\n";
    }
}

# Destroys the virtual machine object, thus disconnecting from the virtual machine.
undef $vm;
Setting a Virtual Machine’s IP Address Configuration Variable

This Perl script invokes the VMware guest operating system service to set a virtual machine’s IP address “ip” configuration variable. This sample script complements the following sample script that retrieves a virtual machine’s IP address “ip” configuration variable. The saveguestip.pl script runs inside a virtual machine, while the getguestip.pl sample script runs in the host operating system or another machine. See Getting a Virtual Machine’s IP Address on page 91.

For more information on passing information between a script and a guest operating system, see Using VmPerl to Pass User-Defined Information Between a Running Guest Operating System and a Script on page 66.

This script (saveguestip.pl, formerly known as configsetip.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/saveguestip.pl.txt.

#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
#
# saveguestip.pl
#
# This script demonstrates the use of the VMware guest service to set
# a configuration variable from within a running virtual machine’s guest
# operating system. It stores the guest operating system’s IP address.
# The host can retrieve the IP address with a corresponding script.
#
# usage:
#   saveguestip.pl
#
# NOTE:
# This script should be run from within a running virtual machine’s guest
# operating system. The corresponding script getguestip.pl can be run
# from the host operating system.

if (@ARGV != 0) {
    print "Usage: $0\n";
    exit(1);
}
my($err);

# Get the IP for the Guest
my($ip) = (undef);
$ip = &get_ip();

if(!defined($ip)) {
    die "$0: Could not get guest ip\n";
} else {
    print "$0: guest ip is $ip\n";
}

# Sets the ip address configuration variable.
$err = &set_ip_variable();
if($err != 0) {
    die "$0: Could not set guest ip\n";
}

# Captures IP address from the OS.
sub get_ip {
    my ($myip, @iparr) = (undef, []);

    # For Windows Guest OS.
    if ($^O eq "MSWin32") {
        $$_ = `ipconfig`;
        @iparr = /IP Address.*?\(\d+\.\d+\.\d+\.\d+)/ig;
        $myip = $iparr[0];
    }

    # For Linux Guest OS.
    # Please ensure that ifconfig is in your path. The root user has it by default.
    else {
        $$_ = `ifconfig`;
        @iparr = /inet addr:.*\(\d+\.\d+\.\d+\.\d+)/ig;
        $myip = $iparr[0];
    }

    return $myip;
}

# Stores the IP address in the guestinfo name space.
sub set_ip_variable {
    if ($^O eq "MSWin32") {
        # Please ensure that VMwareService is in your path.
        # VMwareService needs double quotes around the command.
my $cmd = "VMwareService -cmd " . "" . "info-set guestinfo.ip $ip" . ""; 
 system($cmd); 
} 
else { 
    # Please ensure that vmware-guestd is found in the path used below 
    system("/etc/vmware/vmware-guestd --cmd 'info-set guestinfo.ip $ip'")); 
} 
return $?; 
}
Getting a Virtual Machine’s IP Address

This script runs in the host operating system (or another machine) and invokes the VMware Perl API to retrieve the value of the "ip" variable (a virtual machine's IP address). This sample script complements the preceding sample script (Setting a Virtual Machine’s IP Address Configuration Variable on page 88), that sets a virtual machine's IP address configuration variable in the guest operating system.

For more information on passing information between a script and a guest operating system, see Using VmPerl to Pass User-Defined Information Between a Running Guest Operating System and a Script on page 66.

This script (getguestip.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/getguestip.pl.txt.

#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
# getguestip.pl
#
# This script returns the value of the guest_info variable 'ip' set by
# the guest OS in a virtual machine on a given server.
#
# usage:
# getguestip.pl <path_to_config_file> [<server> <user> <password>]
#
BEGIN {
    if ($^O eq "MSWin32") {
        @INC = (
            # Set the path to your VmPerl Scripting directory if different
            'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005',
            'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005\MSWin32-x86');
        } else {
            @INC = (
                # Set the path to your VmPerl Scripting directory if different
                'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005',
                'C:\Program Files\VMware\VMware VmPerl Scripting API\perl5\site_perl\5.005\MSWin32-x86');
        }
    }
}

use VMware::VmPerl;
use VMware::VmPerl::VM;
use VMware::VmPerl::ConnectParams;
use strict;

if (@ARGV ne 1 && @ARGV ne 4) {

```perl
print "Usage $0: <path_to_config_file> [server] [user] [password]\n";
exit(1);
}

# Read in parameters.
my ($cfg_path, $server_name, $user, $passwd) = @ARGV;

# Use the default port of 902. Change this if your port is different.
my $port = 902;

# If $server_name, $user, and $passwd are missing, connect to localhost as current user.
my $connect_params = VMware::VmPerl::ConnectParams::new($server_name,$port,$user,$passwd);

my $vm = VMware::VmPerl::VM::new();
if (!$vm->connect($connect_params, $cfg_path)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  undef $vm;
  die "Could not connect to vm: Error $error_number: $error_string\n";
}

# Get the IP address of the virtual machine.
my $ip = $vm->get_guest_info('ip');
if (!defined($ip)) {
  my ($error_number, $error_string) = $vm->get_last_error();
  undef $vm;
  die "Could not get IP address: Error $error_number: $error_string\n";
}
if (!$ip) {
  undef $vm;
  die "The guest OS did not set the variable 'ip'.\n";
}
print "The IP address of $cfg_path is: $ip\n";

# Destroys the virtual machine object, thus disconnecting from the virtual machine.
undef $vm;
```
Adding a Redo Log to a Virtual Disk (ESX Server only)

You can add a redo log to a virtual SCSI disk in a running virtual machine on ESX Server. You can specify the disk on the command line or let the script give you a choice of disks to select that are associated with the virtual machine.

For example, you can write a script to back up a virtual disk. Add a new redo log, then back up the virtual disk. The virtual disk is no longer changing as all data is now written to the redo log.

For more information about the `$vm->add_redo()` method, please see VMware::VmPerl::VM on page 54.

This script (`addredo.pl`), saved with a .TXT extension for online viewing, can be found on the VMware Web site at [www.vmware.com/support/developer/scripting-API/doc/addredo.pl.txt](http://www.vmware.com/support/developer/scripting-API/doc/addredo.pl.txt).

Note: This script applies only to ESX Server.

```perl
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
# addredo.pl
#
# This script takes a specification of a server name, user, password, and the path for the config file of a virtual machine on that server. It then displays the disks in the virtual machine configuration, allows the user to choose a disk and then adds a redo log to that disk. The user can also specify the disk directly as the fifth argument on the command line.
#
# usage:
# addredo.pl server user password config [virtual disk]
#
BEGIN {
  if ($^O eq "MSWin32") {
    @INC = ("./5.00503/lib",
      "/5.00503/lib/MSWin32-x86/auto",
      "/5.00503/lib/MSWin32-x86",
      "/site/5.00503/lib",
      "/site/5.00503/lib/MSWin32-x86/auto",
```
use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;
use VMware::VmPerl::VM;
use strict;

if (@ARGV != 4 && @ARGV != 5) {
    print "Usage $0: server user password path_to_config_file [virtual disk]\n";
    exit(1);
}

my ($serverName, $user, $passwd, $cfg, $disk) = @ARGV;
my $port = 902;

# Open up a VM object for the virtual machine associated with the
# specified config file.
my $params = VMware::VmPerl::ConnectParams::new($serverName, $port, $user, $passwd);
my $vm = VMware::VmPerl::VM::new();
my $err = $vm->connect($params, $cfg);
if (!defined($err)) {
    my ($errorNumber, $errorString) = $vm->get_last_error();
    die "Cannot connect to vm: Error $errorNumber: $errorString\n";
}

# Add a REDO log to the specified disk
$err = $vm->add_redo($disk);
if (!defined($err)) {
    my ($errorNumber, $errorString) = $vm->get_last_error();
    die "Cannot add redo log: Error $errorNumber: $errorString\n";
}

$vm->disconnect();
Committing a Redo Log to a Virtual Disk without Freezing the Virtual Machine (ESX Server only)

To use this script, you must have a virtual disk with two redo logs (<disk>.REDO and <disk>.REDO.REDO). You can use this script to commit a virtual disk’s redo log (<disk>.REDO) to its virtual disk.

You specify the disk on the command line or let the script give you a choice to select of disks that are associated with the virtual machine. The virtual machine is not frozen when the redo log(<disk>.REDO) is committed to the virtual disk, but the script waits until the commit finishes.

For example, you keep the virtual disk of a virtual machine in undoable mode. At some point, you may want to commit your changes and back up the entire contents of the virtual disk. You can add a (second) new redo log using the $vm->add_redo() method. The original redo log is no longer changing; all data is now written to the new redo log.

You can then commit the original redo log to the base virtual disk by using the $vm->commit() method with $level with a value of 1. The presence of the second redo log allows you to commit changes from the original redo log to the virtual disk without freezing the virtual disk.

Once the commit is done, you can back up the virtual disk.

For more information about the $vm->commit() method, please see VMware::VmPerl::VM on page 54.

This script (commitnext.pl), saved with a .TXT extension for online viewing, can be found on the VMware Web site at www.vmware.com/support/developer/scripting-API/doc/commitnext.pl.txt.

Note: This script applies only to ESX Server.

#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
#
# commitnext.pl
#
# This script takes a specification of a server name, user, password, and the path for the config file of a virtual machine on that server. It then displays the disks in the virtual machine
# configuration and allows the user to choose a disk. It then commits
# the next-to-top redo log (there must be at least two redo logs) of
# that disk to its. The virtual machine is not frozen during the committing
# process, but the script waits until the commit finishes. The user can
# also specify the disk directly as the fifth argument on the command line.
# usage:
# commitnext.pl server user password config [virtual disk]

BEGIN {
    if ($^O eq "MSWin32") {
        @INC = ("./5.00503/lib",
                "./5.00503/lib/MSWin32-x86/auto",
                "./5.00503/lib/MSWin32-x86",
                "/site/5.00503/lib",
                "/site/5.00503/lib/MSWin32-x86/auto",
                "/site/5.00503/lib/MSWin32-x86");
    } else {
        push(@INC,
               "/usr/lib/perl5/site_perl/5.005/i386-linux",
               "/usr/lib/perl5/5.00503",
               ".");
    }
}

use VMware::VmPerl;
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;
use VMware::VmPerl::VM;
use strict;

if (@ARGV != 4 && @ARGV != 5) {
    print "Usage $0: server user password path_to_config_file [virtual disk]\n";
    exit(1);
}

my ($serverName, $user, $passwd, $cfg, $disk) = @ARGV;
my $port = 902;

# Open up a VM object for the virtual machine associated with the
# specified config file.
my $params = VMware::VmPerl::ConnectParams::new($serverName, $port, $user, $passwd);
my $vm = VMware::VmPerl::VM::new();
my $err = $vm->connect($params, $cfg);
if (!defined($err)) {
    my ($errorNumber, $errorString) = $vm->get_last_error();
die "Cannot connect to vm: Error $errorNumber: $errorString\n";

# Make the API timeout much larger (this is 8 minutes in milliseconds),
# since the commit may take a while.
$vm->set_timeout(480000);

# Commit the next-to-top REDO log
if (!$vm->commit($disk, 1, 0, 1)) {
    my ($errorNumber, $errorString) = $vm->get_last_error();
    die "Cannot commit redo log: Error $errorNumber: $errorString\n";
}

$vm->disconnect();
This chapter includes information to help you use the VMware Scripting APIs. In particular, we describe VMware Scripting API errors. We also describe how you can use Event Viewer to view and manage event logs for virtual machines on a Windows machine.
Error Codes

The following sections describe error handling in the VMware Scripting APIs.

Error Handling for the VmCOM Library

VmCOM methods and properties throw error exceptions when they fail. VmCOM supports the ISupportErrorInfo interface for detailed error reporting.

For example, in Visual Basic, use standard error trapping and examine the err object to retrieve detailed error information. The object's Description field contains a string describing the failure. The Number field contains a VmCOM error code. For more information on VmCOM error codes, see Common VmCOM and VmPerl Errors on page 101.

If a remote virtual machine or server unexpectedly disconnects, most operations fail, giving you either the vmErr_NOTCONNECTED or vmErr_DISCONNECT error code. You cannot reconnect to an existing VmCtl or VmServerCtl object. Instead, destroy the object (for example, Set obj = Nothing in Visual Basic), then create a new object and call Connect() on it.

If a virtual machine operation fails with error code vmErr_NEEDINPUT, obtain a VmQuestion object from VmCtl.PendingQuestion property and examine the question or error description. Then call AnswerQuestion() to answer the question or dismiss the error.

Error Handling for the VmPerl Library

The error codes listed in the following section apply to, and can be returned by, all of the VmPerl modules.

When a $server method returns an error, use $server->get_last_error() in a script to retrieve the error code and, optionally, its description. For example, to return an error code and a description of the error in your scripts, use:

```perl
my ($ret, $string) = $server->get_last_error();
```

Alternately, to return only the error code in your scripts, use:

```perl
my $ret = $server->get_last_error();
```

When a $vm method returns undef, use $vm->get_last_error() in a script to retrieve the error code and, optionally, its description.

For example, to return an error code and a description of the error in your scripts, use:

```perl
my ($ret, $string) = $vm->get_last_error();
```

Alternately, to return only the error code, in your scripts, use:

```perl
my $ret = $vm->get_last_error();
```
Common VmCOM and VmPerl Errors

The following table is a partial list of common VmCOM and VmPerl errors. Any error code not listed in this table indicates an internal failure in VmCOM, VmPerl or another VMware component.

<table>
<thead>
<tr>
<th>VmCOM Error Code</th>
<th>VmPerl Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmErr_BADSTATE</td>
<td>VM_E_BADSTATE</td>
<td>You attempted to move a virtual machine from a valid state to an invalid one. For example, you tried to restore a non-suspended virtual machine or power on an already powered-on virtual machine. Either change the virtual machine's state (for example, from powered on to suspended) or attempt a different operation.</td>
</tr>
<tr>
<td>vmErr_BADVERSION</td>
<td>VM_E_BADVERSION</td>
<td>The version of the VmCOM component/VmPerl module and the VMware server product are incompatible.</td>
</tr>
<tr>
<td>vmErr_DISCONNECT</td>
<td>VM_E_DISCONNECT</td>
<td>The network connection to the virtual machine was lost.</td>
</tr>
<tr>
<td>vmErr_INSUFFICIENT_RESOURCES</td>
<td>VM_E_INSUFFICIENT_RESOURCES</td>
<td>The operation failed because an internal or system limit was exceeded. For example, the <code>Connect()</code> method may return this error if the maximum number of connected objects has been reached.</td>
</tr>
<tr>
<td>vmErr_INVALIDARGS</td>
<td>VM_E_INVALIDARGS</td>
<td>The specified arguments are not valid for this operation.</td>
</tr>
<tr>
<td>vmErr_INVALIDVM</td>
<td>VM_E_INVALIDVM</td>
<td>The specified virtual machine configuration file does not exist. The path to the configuration file may have been entered incorrectly or the virtual machine is not registered.</td>
</tr>
<tr>
<td>vmErr_NEEDINPUT</td>
<td>VM_E_NEEDINPUT</td>
<td>The operation did not complete because the virtual machine is stuck and waiting for user input; that is, the user must answer a question or acknowledge an error before the virtual machine can continue its operation.</td>
</tr>
<tr>
<td>vmErr_NETFAIL</td>
<td>VM_E_NETFAIL</td>
<td>A network failure or misconfiguration prevented the operation from completing.</td>
</tr>
<tr>
<td>vmErr_NOACCESS</td>
<td>VM_E_NOACCESS</td>
<td>The operation could not be completed because of an access violation (a permissions problem).</td>
</tr>
<tr>
<td>vmErr_NOMEM</td>
<td>VM_E_NOMEM</td>
<td>Your system has run out of memory. Shut down some processes to free up memory.</td>
</tr>
<tr>
<td>vmErr_NOPROPERTY</td>
<td>VM_E_NOPROPERTY</td>
<td>The requested variable or property name does not exist.</td>
</tr>
<tr>
<td>VmCOM Error Code</td>
<td>VmPerl Error Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>vmErr_NOTCONNECTED</td>
<td>VM_E_NOTCONNECTED</td>
<td>An operation was attempted on a disconnected virtual machine. Connect the virtual machine before performing this operation.</td>
</tr>
<tr>
<td>vmErr_NOTSUPPORTED</td>
<td>VM_E_NOTSUPPORTED</td>
<td>The attempted operation is not supported by your version of VMware server.</td>
</tr>
<tr>
<td>vmErr_PROXYFAIL</td>
<td>VM_E_PROXYFAIL</td>
<td>The Scripting API could not connect to the server because of a proxy failure. You see this error only if you have configured your remote workstation to use a Web proxy. For more information on using a Web proxy, see your VMware product documentation.</td>
</tr>
<tr>
<td>vmErr_TIMEOUT</td>
<td>VM_E_TIMEOUT</td>
<td>There is no response to the request (the operation timed out).</td>
</tr>
<tr>
<td>vmErr_UNSPECIFIED</td>
<td>VM_E_UNSPECIFIED</td>
<td>An unspecified error has occurred.</td>
</tr>
<tr>
<td>vmErr_VMBUSY</td>
<td>VM_E_VMBUSY</td>
<td>You attempted to connect to a virtual machine that is under the control of a local console running on the server.</td>
</tr>
<tr>
<td>vmErr_VMEXISTS</td>
<td>VM_E_VMEXISTS</td>
<td>You attempted to register a virtual machine that is already registered.</td>
</tr>
<tr>
<td>vmErr_VMINITFAILED</td>
<td>VM_E_VMINITFAILED</td>
<td>The virtual machine process could not be started on the server.</td>
</tr>
</tbody>
</table>
Event Logging

If you are running GSX Server on a Windows machine, you can use Event Viewer to view the following types of events for virtual machines:

- **Power transitions**
  By default, Event Viewer logs an event whenever the virtual machine changes power state (on, off, or suspended).

- **Messages**
  Messages occur whenever an error condition exists in a virtual machine. The Event Viewer logs a message with its type (hint, warning, error, or question), the text of the message, and the choices to acknowledge a message.

- **Message answers**
  When a message is acknowledged, the answer is logged with the message that is answered and the choice that was selected as the answer for that message.

By default, the Event Viewer logs all three types of events. However, you may turn off logging for one or more of these event types by editing the `config.ini` file.

1. Change directories to the VMware GSX Server program directory. The default location is `C:\Program Files\VMware\VMware GSX Server`.
2. Edit the `config.ini` file with a text editor of your choice. Add one or more of the following configuration variables. Each configuration variable turns off event logging for that event type.

   ```
   eventlog.win.power = "FALSE"
   eventlog.win.message = "FALSE"
   eventlog.win.answer = "FALSE"
   ```

**Using the Event Viewer**

1. Open the Event Viewer application. This application is typically in the Administrative Tools folder. Refer to your operating system's documentation for additional information on this application.
2. Open the Application Log file.
   The Event Viewer is displayed as shown in the following image.
You can use the filtering feature in Event Viewer to see selected events on a virtual machine. All virtual machine events are stored in the “Virtual Machines” category. By contrast, all `serverd` and `authd` events are stored in the default “None” category.

Each event type has an event ID. For example, all virtual machine power transition events share the event ID 1100. You may use this event ID to filter virtual machine events. The event IDs for virtual machines are listed in the following table.

<table>
<thead>
<tr>
<th>Event ID</th>
<th>Event Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1100</td>
<td>Power transition events</td>
</tr>
<tr>
<td>1101</td>
<td>Message events</td>
</tr>
<tr>
<td>1102</td>
<td>Message answer events</td>
</tr>
</tbody>
</table>

Right-click on a single event log and select Properties. The Event Properties window is displayed with additional details about the event as shown in the following image.
CHAPTER 6 Error Codes and Event Logging

Reading the Event Log
Each event always begins with a string that describes what happened to the virtual machine.

Power Transitions
The Event Viewer logs virtual machine power transitions as Windows information type events (EVENTLOG_INFORMATION_TYPE). Each power transition event log begins with a simple string indicating the new power state of the virtual machine. Power transition event log strings follow. In these examples, D:\foo.vmx is the path to the configuration file for the virtual machine.

- Virtual machine powered on (was powered off): D:\foo.vmx.
- Virtual machine powered off (was powered on): D:\foo.vmx.
- Virtual machine suspended (was powered on): D:\foo.vmx.

Messages
The Event Viewer logs messages with a severity appropriate for the message:

- VMware hints have an “info” type and are logged as a Windows information type event (EVENTLOG_INFORMATION_TYPE).
- VMware warnings have a “warning” type and are logged as a Windows warning type event (EVENTLOG_WARNING_TYPE).
- VMware errors have a “error” type and are logged as a Windows error type event (EVENTLOG_ERROR_TYPE).
- VMware questions have a “question” type and are logged as a Windows information type event (EVENTLOG_INFORMATION_TYPE).

Each message event log begins with a simple string indicating that a message was received. The message event log includes the type of message and the message text. Example message event log strings follow.

This first example is for a message hint.

- Virtual machine received hint: D:\foo.vmx.
  Don’t forget to install VMware Tools inside this virtual machine. Wait until your guest operating system finishes booting, then choose 'VMware Tools Install...' from the Settings menu in VMware GSX Server. Then follow the instructions that are provided.
  [Ok]

This second example is for an error message.

- Virtual machine received error: D:\foo.vmx
  Failed to resume disk ide0:0. The disk was modified since the virtual machine was suspended.
Error encountered while trying to restore ide0:0 state from file \foo.vmss.
[OK]

This third example is for a question.

Virtual machine received question: D:\foo.vmdk.
Select an action for the redo log of undoable disk D:\foo.vmdk.
[Commit, Discard, Keep]

Message Answers
The Event Viewer logs message answers as Windows information type events (EVENTLOG_INFORMATION_TYPE). Each message answer event log begins with a simple string indicating that an answer to a message was received. The message answer event log includes the type of message, the message text, and the answer.
An example message answer event log string follows.

Virtual machine received answer "Discard": D:\foo.vmdk.
Select an action for the redo log of undoable disk D:\foo.vmdk.
You can use the `vmware-cmd` utility to perform various operations on a virtual machine, including registering a virtual machine (on the local server), getting the power state of a virtual machine, setting configuration variables, and so on.

**Note:** The previous `vmware-control` utility is deprecated. If you are using scripts with the `vmware-control` utility, update your scripts with the new `vmware-cmd` utility or they will not work with VMware GSX Server 2.x, GSX Server 3 or ESX Server 2.x.

By default, the `vmware-cmd` utility is installed in the `/usr/bin` directory (Linux operating system) or in C:\Program Files\VMware\VMware VmPerl Scripting API (Windows operating system).
vmware-cmd Utility Options

The *vmware-cmd* utility takes the following options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-H</td>
<td>Specifies an alternate host other than the local host. If the -H option is used, then the -U and -P options must also be specified.</td>
</tr>
<tr>
<td>-O</td>
<td>Specifies an alternative port. The default port number is 902.</td>
</tr>
<tr>
<td>-U</td>
<td>Specifies the username.</td>
</tr>
<tr>
<td>-P</td>
<td>Specifies the user's password.</td>
</tr>
<tr>
<td>-h</td>
<td>Prints a help message, listing the options for this utility.</td>
</tr>
<tr>
<td>-q</td>
<td>Turns on the quiet option with minimal output. The specified operation and arguments are not specified in the output.</td>
</tr>
<tr>
<td>-v</td>
<td>Turns on the verbose option.</td>
</tr>
</tbody>
</table>
vmware-cmd Operations on a Server

The syntax for this utility on a server is:

    vmware-cmd -s <options> <server-operation> <arguments>

The vmware-cmd utility performs the following operations on a VMware server.

<table>
<thead>
<tr>
<th>Server Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmware-cmd -l</td>
<td>Lists the virtual machines on the local server. Unlike the other server operations, this option does not require the -s option.</td>
</tr>
<tr>
<td>vmware-cmd -s register &lt;vm-cfg-path&gt;</td>
<td>Registers a virtual machine specified by &lt;vm-cfg-path&gt; on the server.</td>
</tr>
<tr>
<td>vmware-cmd -s unregister &lt;vm-cfg-path&gt;</td>
<td>Unregisters a virtual machine specified by &lt;vm-cfg-path&gt; on the server.</td>
</tr>
<tr>
<td>vmware-cmd -s getresource &lt;vm-cfg-path&gt; &lt;variable_name&gt;</td>
<td>Gets the value of the ESX Server system resource variable specified by system.&lt;variable_name&gt;. For a list of ESX Server system variables, see VMware ESX Server System Resource Variables on page 120.</td>
</tr>
<tr>
<td>Note: These methods apply only to ESX Server.</td>
<td></td>
</tr>
<tr>
<td>vmware-cmd -s setresource &lt;variable_name&gt; &lt;value&gt;</td>
<td>Sets the value of the ESX Server system resource variable specified by system.&lt;variable_name&gt;. For a list of ESX Server system variables, see VMware ESX Server System Resource Variables on page 120.</td>
</tr>
<tr>
<td>Note: These methods apply only to ESX Server.</td>
<td></td>
</tr>
</tbody>
</table>
vmware-cmd Operations on a Virtual Machine

The syntax for this utility on a virtual machine is:

```
vmware-cmd <options> <vm-cfg-path> <vm-operation> <arguments>
```

The `vmware-cmd` utility performs the following operations on a virtual machine, where `vm-cfg-path` represents the complete path to the virtual machine's configuration file.

**Note:** The following table includes some ESX Server and GSX Server-specific methods, and are specifically noted.

<table>
<thead>
<tr>
<th>Virtual Machine Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; getstate</code></td>
<td>Retrieves the execution state of a virtual machine: on, off, suspended, stuck (requires user input) or unknown.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; start &lt;powerop_mode&gt;</code></td>
<td>Powers on a previously powered-off virtual machine or resumes a suspended virtual machine. Hard, soft or trysoft specifies the behavior of the power operation <code>&lt;powerop_mode&gt;</code>. If <code>&lt;powerop_mode&gt;</code> is not specified, the default behavior is soft. For more information, see <code>&lt;powerop_mode&gt; Values on page 115</code>.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; stop &lt;powerop_mode&gt;</code></td>
<td>Shuts down and powers off a virtual machine. Hard, soft or trysoft specifies the behavior of the power operation <code>&lt;powerop_mode&gt;</code>. If <code>&lt;powerop_mode&gt;</code> is not specified, the default behavior is soft. For more information, see <code>&lt;powerop_mode&gt; Values on page 115</code>.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; reset &lt;powerop_mode&gt;</code></td>
<td>Shuts down, then reboots a virtual machine. Hard, soft or trysoft specifies the behavior of the power operation <code>&lt;powerop_mode&gt;</code>. If <code>&lt;powerop_mode&gt;</code> is not specified, the default behavior is soft. For more information, see <code>&lt;powerop_mode&gt; Values on page 115</code>.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; suspend &lt;powerop_mode&gt;</code></td>
<td>Suspends a virtual machine. Hard, soft or trysoft specifies the behavior of the power operation <code>&lt;powerop_mode&gt;</code>. If <code>&lt;powerop_mode&gt;</code> is not specified, the default behavior is soft. For more information, see <code>&lt;powerop_mode&gt; Values on page 115</code>.</td>
</tr>
</tbody>
</table>
### CHAPTER 7 vmware-cmd Utility

vmware-cmd <vm-cfg-path> addredo <disk_device_name>

**Note:** This operation applies only to ESX Server.

This operation adds a redo log to a running virtual SCSI disk specified by `<disk_device_name>`, that is associated with the virtual machine specified by `<vm-cfg-path>`. Changes made to the virtual disk accumulate in the new redo log. This disk must be a ESX Server virtual disk stored on a VMFS volume.

The virtual disk can be in persistent, undoable or append mode. The redo log for a virtual disk in persistent mode uses the file name of the virtual disk with .REDO appended to it (for example, if the disk is called `vm.dsk`, the redo log is called `vm.dsk.REDO`). A virtual disk in undoable or append mode already has a redo log associated with it, so the new redo log you create is called `vm.dsk.REDO.REDO`, whose parent is the existing redo log, `vm.dsk.REDO`.

This operation fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk already has two redo logs associated with it.

If you add a redo log using the `vmware-cmd addredo` command, but do not commit your changes with the `vmware-cmd commit` command, then the redo is automatically committed when the virtual machine is powered off.

<table>
<thead>
<tr>
<th>Virtual Machine Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmware-cmd &lt;vm-cfg-path&gt; addredo &lt;disk_device_name&gt;</td>
<td>This operation adds a redo log to a running virtual SCSI disk specified by <code>&lt;disk_device_name&gt;</code>, that is associated with the virtual machine specified by <code>&lt;vm-cfg-path&gt;</code>. Changes made to the virtual disk accumulate in the new redo log. This disk must be a ESX Server virtual disk stored on a VMFS volume. The virtual disk can be in persistent, undoable or append mode. The redo log for a virtual disk in persistent mode uses the file name of the virtual disk with .REDO appended to it (for example, if the disk is called <code>vm.dsk</code>, the redo log is called <code>vm.dsk.REDO</code>). A virtual disk in undoable or append mode already has a redo log associated with it, so the new redo log you create is called <code>vm.dsk.REDO.REDO</code>, whose parent is the existing redo log, <code>vm.dsk.REDO</code>. This operation fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk already has two redo logs associated with it. If you add a redo log using the <code>vmware-cmd addredo</code> command, but do not commit your changes with the <code>vmware-cmd commit</code> command, then the redo is automatically committed when the virtual machine is powered off.</td>
</tr>
</tbody>
</table>
This method commits the changes in a redo log to a running virtual SCSI disk specified by `<disk_device_name>` that is associated with the virtual machine specified by `<vm-cfg-path>`.

`<level>` can be 0 or 1. When `<level>` is 0, there can be one or two redo logs associated with the disk. If `<level>` is 0, then the top-most redo log (the redo log being modified) is committed to its parent. For example, if there is currently only the disk `vm.dsk` with a single redo log `vm.dsk.REDO`, then the changes in `vm.dsk.REDO` are committed to `vm.dsk`. If a second REDO log `vm.dsk.REDO.REDO` has been added, then the changes in `vm.dsk.REDO.REDO` are committed to `vm.dsk.REDO`. `<level>` can be 1 only when there are two redo logs associated with the disk, `vm.dsk.REDO` and `vm.dsk.REDO.REDO`. When `<level>` is 1, the changes in the next-to-top REDO log, `vm.dsk.REDO`, are committed to `vm.dsk`. In this case, the virtual machine is not frozen while the redo log is being committed. Also, when the log is committed, `vm.dsk.REDO.REDO` is renamed to `vm.dsk.REDO`.

`<freeze>` can be 0 or 1. If `<freeze>` is 0, then the virtual machine is not frozen when changes are committed, though it runs more slowly. If `<freeze>` is 1, then the virtual machine must be frozen when changes are committed and `<freeze>` is ignored.

`<wait>` can be 0 or 1. If `<wait>` is 0, then the method returns as soon as the commit begins. If `<wait>` is 1, then the method does not return until the commit completes.

The method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk currently has no redo logs.

<table>
<thead>
<tr>
<th>Virtual Machine Operation</th>
<th>Description</th>
</tr>
</thead>
</table>
| `vmware-cmd <vm-cfg-path> commit <disk_device_name> <level> <freeze> <wait>` | This method commits the changes in a redo log to a running virtual SCSI disk specified by `<disk_device_name>` that is associated with the virtual machine specified by `<vm-cfg-path>`.

`<level>` can be 0 or 1. When `<level>` is 0, there can be one or two redo logs associated with the disk. If `<level>` is 0, then the top-most redo log (the redo log being modified) is committed to its parent. For example, if there is currently only the disk `vm.dsk` with a single redo log `vm.dsk.REDO`, then the changes in `vm.dsk.REDO` are committed to `vm.dsk`. If a second REDO log `vm.dsk.REDO.REDO` has been added, then the changes in `vm.dsk.REDO.REDO` are committed to `vm.dsk.REDO`. `<level>` can be 1 only when there are two redo logs associated with the disk, `vm.dsk.REDO` and `vm.dsk.REDO.REDO`. When `<level>` is 1, the changes in the next-to-top REDO log, `vm.dsk.REDO`, are committed to `vm.dsk`. In this case, the virtual machine is not frozen while the redo log is being committed. Also, when the log is committed, `vm.dsk.REDO.REDO` is renamed to `vm.dsk.REDO`.

`<freeze>` can be 0 or 1. If `<freeze>` is 0, then the virtual machine is not frozen when changes are committed, though it runs more slowly. If `<freeze>` is 1, then the virtual machine must be frozen when changes are committed and `<freeze>` is ignored.

`<wait>` can be 0 or 1. If `<wait>` is 0, then the method returns as soon as the commit begins. If `<wait>` is 1, then the method does not return until the commit completes.

The method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk currently has no redo logs. |

<p>| <code>vmware-cmd &lt;vm-cfg-path&gt; setconfig &lt;variable&gt; &lt;value&gt;</code> | Sets a configuration variable for the virtual machine connected to the remote console. |
| <code>vmware-cmd &lt;vm-cfg-path&gt; getconfig &lt;variable&gt;</code> | Retrieves the value for a configuration variable for the virtual machine connected to the remote console. |
| <code>vmware-cmd &lt;vm-cfg-path&gt; setguestinfo &lt;variable&gt; &lt;value&gt;</code> | Writes a GuestInfo variable into memory. The variable is discarded when the virtual machine process terminates. |
| <code>vmware-cmd &lt;vm-cfg-path&gt; getguestinfo &lt;variable&gt;</code> | Retrieves the value for a GuestInfo variable. |</p>
<table>
<thead>
<tr>
<th>Virtual Machine Operation</th>
<th>Description</th>
</tr>
</thead>
</table>
| `vmware-cmd <vm-cfg-path> getproductinfo <prodinfo>`          | Returns information about the product, where `<prodinfo>` is product, platform, build, majorversion (product’s major version number), minorversion (product’s minor version number) or revision.  
If product is specified, the return value is one of the following: ws (VMware Workstation), gsx (VMware GSX Server) esx (VMware ESX Server) or unknown (unknown product type).  
If platform is specified, the return value is one of the following: windows (Microsoft Windows), linux (Linux operating system) or unknown (unknown platform type). |
| `vmware-cmd <vm-cfg-path> connectdevice <device_name>`       | Connects the specified virtual device to a virtual machine.                                                                                                                                                                                                                                                                                  |
| `vmware-cmd <vm-cfg-path> disconnectdevice <device_name>`    | Disconnects the specified virtual device from the virtual machine.                                                                                                                                                                                                                                                                            |
| `vmware-cmd <vm-cfg-path> getconfigfile`                     | Returns a string containing the configuration file name for the virtual machine. This method fails if the virtual machine is not connected.                                                                                                                                                                                                  |
| `vmware-cmd <vm-cfg-path> getheartbeat`                       | Returns the current heartbeat count generated by the VMware Tools service running in the guest operating system. The count is initialized to zero when the virtual machine is powered on.  
The heartbeat count is typically incremented at least once per second when the VMware Tools service is running under light load conditions. The count stays constant if this service is not running. |
| `vmware-cmd <vm-cfg-path> gettoolslastactive`                 | Returns an integer indicating how much time has passed, in seconds, since the last heartbeat was detected from the VMware Tools service.  
This value is initialized to zero when the virtual machine powers on. It stays at zero until the first heartbeat is detected, after which the value is always greater than zero until the virtual machine is power-cycled again. |
<p>| <code>vmware-cmd &lt;vm-cfg-path&gt; answer</code>                            | Prompts the user to answer a question for a virtual machine waiting for user input.                                                                                                                                                                                                                                                         |
| <code>vmware-cmd getresource &lt;vm-cfg-path&gt; &lt;variable_name&gt;</code>       | Gets the value of the virtual machine resource variable specified by <code>&lt;variable_name&gt;</code>. For a list of ESX Server virtual machine resource variables, see Virtual Machine Resource Variables for ESX Server on page 124.                                                                                     |
| <strong>Note:</strong> This method applies only to ESX Server.          |                                                                                                                                                                                                                                                                                                                                             |
| <code>vmware-cmd setresource &lt;vm-cfg-path&gt; &lt;variable_name&gt; &lt;value&gt;</code> | Sets the value of the virtual machine resource variable specified by <code>&lt;variable_name&gt;</code>. For a list of ESX Server virtual machine resource variables, see Virtual Machine Resource Variables for ESX Server on page 124.                                                                                     |
| <strong>Note:</strong> This method applies only to ESX Server.          |                                                                                                                                                                                                                                                                                                                                             |
| <code>vmware-cmd &lt;vm-cfg-path&gt; getuptime</code>                         | Accesses the uptime of the guest operating system on the virtual machine.                                                                                                                                                                                                           |
| <strong>Note:</strong> This method applies only to ESX Server.          |                                                                                                                                                                                                                                                                                                                                             |</p>
<table>
<thead>
<tr>
<th>Virtual Machine Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmware-cmd &lt;vm-cfg-path&gt; getid</td>
<td>Returns the unique (world) ID for a running virtual machine.</td>
</tr>
<tr>
<td><strong>Note:</strong> This method applies only to ESX Server.</td>
<td></td>
</tr>
<tr>
<td>vmware-cmd &lt;vm-cfg-path&gt; getpid</td>
<td>Returns the process ID of a running virtual machine.</td>
</tr>
<tr>
<td><strong>Note:</strong> This method applies only to ESX Server.</td>
<td></td>
</tr>
<tr>
<td>vmware-cmd &lt;vm-cfg-path&gt; getcapabilities</td>
<td>Returns the access permissions for the current user. This number is a bit vector, where 4=read, 2=write, and 1=execute. For a user with all three permissions, a value of 7 is returned when this property is used in a script.</td>
</tr>
<tr>
<td><strong>Note:</strong> This method applies only to ESX Server.</td>
<td></td>
</tr>
<tr>
<td>vmware-cmd &lt;vm-cfg-path&gt; getremoteconnections</td>
<td>Returns the number of remotely connected users. This value includes the number of remote consoles, Scripting APIs, and Web-based management interface connections to the virtual machine.</td>
</tr>
<tr>
<td><strong>Note:</strong> This method applies only to ESX Server.</td>
<td></td>
</tr>
<tr>
<td>vmware-cmd &lt;vm-cfg-path&gt; setrunasuser &lt;username&gt; &lt;password&gt;</td>
<td>Runs the virtual machine as the user specified by the &lt;username&gt; and &lt;password&gt;.</td>
</tr>
<tr>
<td><strong>Note:</strong> This method applies only to GSX Server 3.1.</td>
<td></td>
</tr>
<tr>
<td>vmware-cmd &lt;vm-cfg-path&gt; getrunasuser</td>
<td>Returns the name of the user running the virtual machine.</td>
</tr>
<tr>
<td><strong>Note:</strong> This method applies only to GSX Server 3.1.</td>
<td></td>
</tr>
</tbody>
</table>
### `<powerop_mode>` Values

The following table describes hard, soft and trysoft power operations.

<table>
<thead>
<tr>
<th>Powerop_mode Values</th>
<th>Description</th>
</tr>
</thead>
</table>
| soft                | Start when a virtual machine is suspended — After resuming the virtual machine, the operation attempts to run a script in the guest operating system. The Start operation always succeeds. However, if VMware Tools is not present or is malfunctioning, the running of the script may fail.  
Start when virtual machine is powered off — After powering on the virtual machine, it attempts to run a script in the guest operating system when the VMware Tools service becomes active. The default script does nothing during this operation as there is no DHCP lease to renew. The Start operation always succeeds. However, if VMware Tools is not present or is malfunctioning, the running of the script may fail.  
Stop — Attempts to shut down the guest operating system and then powers off the virtual machine.  
Reset — Attempts to shut down the guest operating system, then reboots the virtual machine.  
Suspend — Attempts to run a script in the guest operating system before suspending the virtual machine. |  
| hard                | Start — Starts or resumes a virtual machine without running any scripts; a standard power on or resume.  
Stop, reset or suspend — Immediately and unconditionally powers off, resets, or suspends the virtual machine. |  
| trysoft             | First attempts to perform the soft power transition operation. If this fails, the hard power operation is performed. |
vmware-cmd Utility Examples

This section includes examples of using the vmware-cmd utility on a virtual machine.

Retrieving the State of a Virtual Machine

The following examples illustrate retrieving the execution state of a virtual machine.

Change directories to the directory (folder) containing the vmware-cmd utility or include the full path to the utility when typing the following on a command line. Note that you must use double quotes when specifying a path with spaces; for example, "C:\Program Files\VMware\VMware VmPerl Scripting API\vmware-cmd".

In a Linux guest operating system:

    vmware-cmd /home/vmware/win2000.vmx getstate

where /home/vmware/win2000.vmx is the path to the virtual machine's configuration file.

In a Windows guest operating system:

    vmware-cmd C:\home\vmware\win2000.vmx getstate

where C:\home\vmware\win2000.vmx is the path to the virtual machine's configuration file.

Performing a Power Operation

The following examples illustrate performing a power operation. The first example illustrates powering on a virtual machine and the second example illustrates performing a hard reset.

Change directories to the directory (folder) containing the vmware-cmd utility or include the full path to the utility when typing the following on a command line. Note that you must use double quotes when specifying a path with spaces; for example, "C:\Program Files\VMware\VMware VmPerl Scripting API\vmware-cmd".

In a Linux guest operating system:

    vmware-cmd -v /home/vmware/win2000.vmx start

where -v indicates the verbose option, /home/vmware/win2000.vmx is the path to the virtual machine's configuration file and start is the power operation. Since a <powerop_mode> is not specified, the default soft behavior is performed.

Similarly, in a Windows guest operating system:

    vmware-cmd -q C:\home\vmware\win2000.vmx reset hard

where -q indicates the quiet option (only the results of the operation are printed), C:\home\vmware\win2000.vmx is the path to the virtual machine's configuration file and reset is the power operation. This example specifies a hard reset so the virtual machine is immediately and unconditionally reset.
Setting a Configuration Variable
The following example illustrates setting a configuration variable in a Linux guest operating system.

Change directories to the directory (folder) containing the `vmware-cmd` utility or include the full path to the utility when typing the following on a command line.

```
vmware-cmd foo.vmx setconfig ide1:0.file /tmp/cdimages/foo.iso
```

where `foo.vmx` is the virtual machine's configuration file, `ide1:0.file` is the variable and its value is `/tmp/cdimages/foo.iso`.

Connecting a Device
The following example illustrates connecting a virtual IDE device in a Windows guest operating system.

Change directories to the directory (folder) containing the `vmware-cmd` utility or include the full path to the utility when typing the following on a command line. Note that you must use double quotes when specifying a path with spaces; for example, "C:\Program Files\VMware\VMware VmPerl Scripting API\vmware-cmd".

```
vmware-cmd D:\foo.vmx connectdevice ide1:0
```

where `D:\foo.vmx` is the virtual machine's configuration file and `ide1:0` is the device name.
This chapter includes ESX Server resource variables that you can set on ESX Server and on virtual machines running on ESX Server. You can get and set these resource variables through the VmCOM resource property and the VmPerl `get_resource` and `set_resource` methods.
VMware ESX Server System Resource Variables

Use these variables to return or set statistics on ESX Server. For more information on the VmCOM resource property, see VmServerCtl on page 18. For more information on the VmPerl get_resource and set_resource methods, see VMware::VmPerl::Server on page 52.

In the following table, # represents the number for a particular CPU. For example, if an ESX Server has four physical CPUs, then # is 0, 1, 2, or 3.

<HTL> represents the host target LUN for a SCSI device. For example, for vmhba1:2:0, 1 represents the host adapter, 2 represents the target on the adapter, and 0 specifies the LUN.

<vmnic> represents the physical network interface card, for example, vmnic0.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Statistics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system.sys.cosUptime</td>
<td>INT</td>
<td>VMware Service Console uptime, in seconds.</td>
</tr>
<tr>
<td>system.sys.vmkUptime</td>
<td>INT</td>
<td>VMkernel uptime, in seconds.</td>
</tr>
<tr>
<td><strong>CPU Statistics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>system.cpu.number</td>
<td>INT</td>
<td>Number of CPUs on the ESX Server system. On hyperthreading-enabled systems,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>this number of CPUs reflects the number of logical CPUs (double the number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>of physical CPUs).</td>
</tr>
<tr>
<td>system.cpu.#</td>
<td>INT</td>
<td>Information about the CPU (specified by #).</td>
</tr>
<tr>
<td>system.cpu.#.idlesec</td>
<td>FLOAT</td>
<td>Idle time, in seconds, of the physical or logical CPU (specified by #).</td>
</tr>
<tr>
<td>system.cpu.#.usedsec</td>
<td>FLOAT</td>
<td>Time, in seconds, the physical or logical CPU (specified by #) is in use.</td>
</tr>
</tbody>
</table>

<p>| <strong>Memory Statistics</strong> |               |                                                                             |
| system.mem.avail     | INT           | Amount of memory, in KB, available for all virtual machines.               |
| system.mem.COS       | INT           | Amount of memory, in KB, allocated to the service console.                 |
| system.mem.COSavail  | INT           | Amount of free memory, in KB, available to the service console.            |
| system.mem.active     | INT           | Amount of memory (subset of system.mem.size) that has been recently used. |
| system.mem.cosUsage   | INT           | Amount of memory, in KB, currently used by the service console. The sum of |
|                       |               | system.mem.COSavail and system.mem.cosUsage should equal the value for sys |
|                       |               | tem.mem.COS.                                                               |
| system.mem.cpt-tgt    | INT           | Total amount of data, in KB, read from suspend files for all virtual      |
|                       |               | machines.                                                                 |
| system.mem.cptread    | INT           | Sum of cptread (KB read from suspend files) for all running virtual       |
|                       |               | machines.                                                                 |</p>
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.mem.mctltgt</td>
<td>INT</td>
<td>Total amount of memory, in KB, that the vmmemctl 1 balloon drivers should reclaim from all guest operating systems in running virtual machines. The value of system.mem.mctltgt should always approach this number.</td>
</tr>
<tr>
<td>system.mem.memctl</td>
<td>INT</td>
<td>Total amount of memory, in KB, that the vmmemctl 1 balloon drivers have reclaimed from all guest operating systems in running virtual machines.</td>
</tr>
<tr>
<td>system.mem.overhd</td>
<td>INT</td>
<td>Sum of the current overhead memory, in KB, for all running virtual machines.</td>
</tr>
<tr>
<td>system.mem.ovhdmax</td>
<td>INT</td>
<td>Sum of the maximum overhead memory, in KB, for all running virtual machines.</td>
</tr>
<tr>
<td>system.mem.reservedMem</td>
<td>INT</td>
<td>Reserved memory, in KB, across all running virtual machines. This number is the total of user-set minimum memory sizes and the default minimum memory sizes (50% of the specified maximum) on all virtual machines.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>This applies only to ESX Server 2.1.1 and higher.</td>
</tr>
<tr>
<td>system.mem.reservedSwap</td>
<td>INT</td>
<td>Reserved swap space, in KB, across all virtual machines. This value is equal to the difference between the max (maximum) memory size and min (minimum) memory size, for all virtual machines.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>This applies only to ESX Server 2.1.1 and higher.</td>
</tr>
<tr>
<td>system.mem.shared</td>
<td>INT</td>
<td>Memory allocated to running virtual machines that is securely shared with other virtual machines.</td>
</tr>
<tr>
<td>system.mem.sharedCommon</td>
<td>INT</td>
<td>Total amount of memory, in MB, that’s required for a single copy of shared pages in running virtual machines.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>This applies only to ESX Server 2.1.1 and higher.</td>
</tr>
<tr>
<td>system.mem.sharedVM</td>
<td>INT</td>
<td>Total amount of shared memory, in MB, for running virtual machines. This is the same value as the sum of system.mem.shared and system.mem.sharedCommon.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>This applies only to ESX Server 2.1.1 and higher.</td>
</tr>
<tr>
<td>system.mem.size</td>
<td>INT</td>
<td>Sum of the current memory size, in KB, for all running virtual machines.</td>
</tr>
<tr>
<td>system.mem.sizetgt</td>
<td>INT</td>
<td>Sum of the target memory size, in KB, for all running virtual machines.</td>
</tr>
<tr>
<td>system.mem.swapin</td>
<td>INT</td>
<td>Cumulative memory, in KB, swapped into memory since the last time ESX Server was booted.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>This applies only to ESX Server 2.1.1 and higher.</td>
</tr>
<tr>
<td>system.mem.swapout</td>
<td>INT</td>
<td>Cumulative memory, in KB, swapped out to disk since the last time ESX Server was booted.</td>
</tr>
<tr>
<td>Note:</td>
<td></td>
<td>This applies only to ESX Server 2.1.1 and higher.</td>
</tr>
<tr>
<td>system.mem.swapped</td>
<td>INT</td>
<td>Sum of currently swapped memory, in KB, for all running virtual machines. It is the same value as when system.mem.swapin is subtracted from system.mem.swapout.</td>
</tr>
</tbody>
</table>
### System Memory Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.mem.swaptgt</td>
<td>INT</td>
<td>For all running virtual machines, the target memory size, in KB, used for swapping to the VMFS swap files.</td>
</tr>
<tr>
<td>system.sysCodeSize</td>
<td>INT</td>
<td>VMkernel code size (memory used to store the executable instructions of the VMkernel).</td>
</tr>
<tr>
<td>system.sysHeapSize</td>
<td>INT</td>
<td>Amount of memory allocated, in KB, to the VMkernel heap file.</td>
</tr>
<tr>
<td>system.sysUsage</td>
<td>INT</td>
<td>Amount of overhead memory, in KB, currently used by the VMkernel.</td>
</tr>
<tr>
<td>system.totalMem</td>
<td>INT</td>
<td>Total amount of memory, in KB, on ESX Server.</td>
</tr>
<tr>
<td>system.totalSwap</td>
<td>INT</td>
<td>Total amount of swap space, in MB, on ESX Server.</td>
</tr>
<tr>
<td>system.vmiMaxSize</td>
<td>INT</td>
<td>Memory size of the largest virtual machine that can be powered on, if the minimum (min) size is unspecified.</td>
</tr>
<tr>
<td>system.vmkUsage</td>
<td>INT</td>
<td>Amount of memory, in KB, currently used by the VMkernel.</td>
</tr>
<tr>
<td>system.vmkKernel</td>
<td>INT</td>
<td>Amount of memory, in MB, currently allocated to the VMkernel.</td>
</tr>
</tbody>
</table>

### Disk Statistics

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.disk.&lt;HTL&gt;.KBread</td>
<td>INT</td>
<td>Total data read (in KB) for the target SCSI device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>system.disk.&lt;HTL&gt;.KBwritten</td>
<td>INT</td>
<td>Total data written (in KB) for the target SCSI device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>system.disk.&lt;HTL&gt;.busResets</td>
<td>INT</td>
<td>Total number of bus resets for the target SCSI device specified by &lt;HTL&gt;. We convert bus and device resets to virtual disk files (excluding raw devices) into aborts, so if you compare counters from the virtual machine with this information, then they do not always match.</td>
</tr>
<tr>
<td>system.disk.&lt;HTL&gt;.cmds</td>
<td>INT</td>
<td>Total number of commands for the target SCSI device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>system.disk.&lt;HTL&gt;.cmdsAborted</td>
<td>INT</td>
<td>Total number of commands aborted for the target SCSI device specified by &lt;HTL&gt;. We convert bus and device resets to virtual disk files (excluding raw devices) into aborts, so if you compare counters from the virtual machine with this information, then they do not always match.</td>
</tr>
<tr>
<td>system.disk.&lt;HTL&gt;.reads</td>
<td>INT</td>
<td>Total number of reads for the target SCSI device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>system.disk.&lt;HTL&gt;.writes</td>
<td>INT</td>
<td>Total number of writes for the target SCSI device specified by &lt;HTL&gt;.</td>
</tr>
</tbody>
</table>
### Network Statistics

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.net.&lt;device&gt;.totKBRx</td>
<td>INT</td>
<td>Total amount of data received (in KB) on the virtual switch attached to <code>&lt;device&gt;</code>; This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic.</td>
</tr>
<tr>
<td>system.net.&lt;device&gt;.totKBTx</td>
<td>INT</td>
<td>Total amount of data transmitted (in KB) on the virtual switch attached to <code>&lt;device&gt;</code>; This data includes both local (virtual machine to virtual machine) and remote (virtual machine to physical network) traffic.</td>
</tr>
<tr>
<td>system.net.&lt;device&gt;.totPktsRx</td>
<td>INT</td>
<td>Total number of packets received on the virtual switch attached to <code>&lt;device&gt;</code>; This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic. <strong>Note:</strong> In releases prior to ESX Server 2.1.1, there may be inconsistencies when non-unicast packets (broadcasts, multicasts, and unicasts to virtual NICs in promiscuous mode) are received.</td>
</tr>
<tr>
<td>system.net.&lt;device&gt;.totPktsTx</td>
<td>INT</td>
<td>Total number of packets transmitted on the virtual switch attached to <code>&lt;device&gt;</code>; This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic. <strong>Note:</strong> In releases prior to ESX Server 2.1.1, there may be inconsistencies when non-unicast packets are transmitted.</td>
</tr>
</tbody>
</table>

### Miscellaneous Statistics

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>system.worlds</td>
<td>string</td>
<td>Space-delimited set of IDs listing running virtual machines.</td>
</tr>
<tr>
<td>system.net.adapters</td>
<td>string</td>
<td>Space-delimited set of bonds and physical network adapters, for example, bond0 or vmnet_0.</td>
</tr>
<tr>
<td>system.disk.HTL</td>
<td>string</td>
<td>Space-delimited set of disks specified as a set of host target LUN (HTL), for example, vmhba0:0:0 or vmhba1:0:1.</td>
</tr>
</tbody>
</table>
Virtual Machine Resource Variables for ESX Server

Use these variables to return or set resource statistics on virtual machines running on ESX Server. For more information on the VmCOM resource property, see VmCtl on page 21. For more information on the VmPerl get_resource and set_resource methods, see VMware::VmPerl::VM on page 54.

In the following table, <HTL> represents the host target LUN for a SCSI device that is assigned to a particular virtual machine. For example, for vmhba1:2:0, 1 represents the host adapter, 2 represents the target on the adapter, and 0 specifies the LUN.

Similarly, <MAC> represents the media access control (MAC) address for virtual network adapters.

The virtual machine session lasts until all remote connections are closed, including remote consoles and remote control connections through the API.

**Note:** You can set a configuration variable as many times as you want through the API (per virtual machine session), but only the latest change is kept in the virtual machine session.

All of the variables in the following table can be queried by using the $vm->get_resource() method. Some of these variables can also be modified with the $vm->set_resource() method. These variables that can be both queried and modified are indicated, where appropriate, in the table.

Following the table, we include some tips on using these resource variables efficiently. See Using ESX Server Virtual Machine Resource Variables Efficiently on page 129.

**Note:** The following table includes a brief description for each of the resource variables. For additional information, refer to the man pages for cpu(8), hyperthreading(8), mem(8), and diskbw(8).

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Statistics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cpu.number</td>
<td>INT</td>
<td>Number of CPUs configured for the virtual machine.</td>
</tr>
<tr>
<td>cpu.emin</td>
<td>INT</td>
<td>Effective min (minimum) percentage of a CPU that is assigned to the virtual machine on CPU x. cpu.emin is always greater than cpu.min. This emin value can change when the virtual machine powers on or off, and when any changes are made to CPU allocations for any running virtual machines.</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Variable Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>cpu.extrasec</td>
<td>FLOAT</td>
<td>Amount of usedsec (cumulative processor time, in milliseconds, consumed by the virtual CPU) over and above the value guaranteed by emin.</td>
</tr>
<tr>
<td>cpu.syssec</td>
<td>FLOAT</td>
<td>Amount of CPU time, in milliseconds, used by the VMkernel on behalf of the virtual machine (for example, for I/O processing). This CPU time is included in usedsec, described later in this table.</td>
</tr>
<tr>
<td>cpu.uptime</td>
<td>INT</td>
<td>Amount of elapsed time, in seconds, since the virtual machine was powered on.</td>
</tr>
<tr>
<td>cpu.usedsec</td>
<td>INT</td>
<td>Number of seconds of CPU time used by a virtual machine. Note: For an SMP virtual machine, this variable reports only the amount used by virtual CPU 0. To report the total used by all virtual CPUs, calculate the sum of all cpu.&lt;n&gt;.usedsec variables, where &lt;n&gt; is the number of virtual CPUs returned by the cpu.number variable.</td>
</tr>
<tr>
<td>cpu.waitsec</td>
<td>FLOAT</td>
<td>Number of milliseconds that a virtual machine is idle or blocked on some event.</td>
</tr>
<tr>
<td>cpu.affinity</td>
<td>string (Read-write)</td>
<td>CPU affinity set as a comma-delimited set of numbers, with each number referring to a CPU number. For example, if a CPU affinity set is 0, 1, 3, then a virtual machine runs on CPU 0, 1, or 3.</td>
</tr>
</tbody>
</table>
| cpu.htSharing | string (Read-write) | Specifies how the virtual CPUs in a virtual machine are allowed to share physical packages on a hyperthreaded system. The values are:  
  • any — Default for all virtual machines on a hyperthreaded system. The virtual CPUs may freely share packages at any time, with any other virtual CPUs.  
  • none — Virtual CPUs in this virtual machine cannot share packages with each other, or with virtual CPUs from any other virtual machines.  
  • internal — Applies only to SMP virtual machines. Virtual CPUs on a virtual machine may share packages with each other, but not with virtual CPUs from other virtual machines. |
| cpu.max      | INT           | Maximum CPU percentage for a virtual machine. The valid range of values is 0 to the number representing the total physical CPU resources. The maximum may be greater than 100 for SMP virtual machines that may use more than one full physical CPU. |
| cpu.min      | INT           | Guaranteed minimum CPU percentage reserved for a virtual machine. |
### Variable Name | Variable Type | Description
--- | --- | ---
| cpu.shares | INT | Read-write | Number of CPU shares assigned to a virtual machine.

### Memory Statistics

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mem.active</td>
<td>INT</td>
<td>Amount of memory, in KB, actively used by a virtual machine.</td>
</tr>
<tr>
<td>mem.cpt-tgt</td>
<td>INT</td>
<td>Amount of memory, in KB, that the virtual machine reads into physical memory from its suspend file. This matches the memory size of the virtual machine, or is 0 (zero), if the system is not swapping from the virtual machine's suspend file.</td>
</tr>
<tr>
<td>mem.cptread</td>
<td>INT</td>
<td>Size, in KB, of the suspend file (for a virtual machine) that has been read into memory. If the system memory is low, the suspend file is used as a special swap file for the VMkernel, when the virtual machine is resumed.</td>
</tr>
<tr>
<td>mem.mctltgt</td>
<td>INT</td>
<td>The VMkernel believes this is the size, in KB, of the \texttt{vmmemctl} balloon driver in the guest operating system for a virtual machine. This value should approach mem.memctl (next row in this table).</td>
</tr>
<tr>
<td>mem.memctl</td>
<td>INT</td>
<td>Amount of reclaimed memory, in KB, after running the \texttt{vmmemctl} module for a virtual machine.</td>
</tr>
<tr>
<td>mem.overhd</td>
<td>INT</td>
<td>Overhead memory, in KB, for a virtual machine.</td>
</tr>
<tr>
<td>mem.ovhdmax</td>
<td>INT</td>
<td>Maximum overhead memory, in KB, for a virtual machine.</td>
</tr>
<tr>
<td>mem.shared</td>
<td>INT</td>
<td>Amount of memory, in KB, that is shared through transparent page sharing either within a virtual machine or with other virtual machines running on the same server.</td>
</tr>
<tr>
<td>mem.size</td>
<td>INT</td>
<td>Size of the actual memory, in KB, for a virtual machine. This value should approach mem.sizetgt (next row in this table).</td>
</tr>
<tr>
<td>mem.sizetgt</td>
<td>INT</td>
<td>Target memory size, in KB, for a virtual machine.</td>
</tr>
<tr>
<td>mem.swapin</td>
<td>INT</td>
<td>KB swapped into memory since the last time the virtual machine was booted.</td>
</tr>
<tr>
<td>mem.swapout</td>
<td>INT</td>
<td>KB swapped out to disk since the last time the virtual machine was booted.</td>
</tr>
<tr>
<td>mem.swapped</td>
<td>INT</td>
<td>Amount of memory, in KB, swapped in and out to the VMFS partition swap file for a virtual machine. This value should approach mem.swaptgt (next row in this table).</td>
</tr>
<tr>
<td>mem.swaptgt</td>
<td>INT</td>
<td>Target memory size, in KB, to swap to the VMFS swap file for a virtual machine.</td>
</tr>
<tr>
<td>mem.affinity</td>
<td>string</td>
<td>Read-write</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Variable Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>mem.max</td>
<td>INT</td>
<td>Size of the maximum memory for a virtual machine. (This is the amount of memory a virtual machine thinks it has.) In ESX Server 2.0, mem.max is in KB. In ESX Server 2.0.1 and higher, mem.max is in MB.</td>
</tr>
<tr>
<td>mem.min</td>
<td>INT</td>
<td>Size of the minimum memory for a virtual machine. In ESX Server 2.0, mem.min is in KB. In ESX Server 2.0.1 and higher, mem.min is in MB.</td>
</tr>
<tr>
<td>mem.shares</td>
<td>INT</td>
<td>Number of memory shares assigned to a virtual machine. You can use memory shares to calculate proportional server resource allocation between virtual machines.</td>
</tr>
</tbody>
</table>

### Disk Statistics

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>disk.HTL</td>
<td>string</td>
<td>Space-delimited set of disks specified as a set of host target LUN (HTL); for example, vmba0:0:0 or vmba1:0:1.</td>
</tr>
<tr>
<td>disk.&lt;HTL&gt;.KBread</td>
<td>INT</td>
<td>Total data read (in KB) for the virtual machine on the target device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>disk.&lt;HTL&gt;.KBwritten</td>
<td>INT</td>
<td>Total data written (in KB) for the virtual machine on the target device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>disk.&lt;HTL&gt;.busResets</td>
<td>INT</td>
<td>Number of bus resets that occurred on the target device specified by &lt;HTL&gt; due to a command from the virtual machine.</td>
</tr>
<tr>
<td>disk.&lt;HTL&gt;.cmds</td>
<td>INT</td>
<td>Total number of commands made by the virtual machine on the target device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>disk.&lt;HTL&gt;.cmdsAborted</td>
<td>INT</td>
<td>Total number of commands made by the virtual machine that were aborted on the target device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>disk.&lt;HTL&gt;.reads</td>
<td>INT</td>
<td>Total number of disk reads for the virtual machine on the target device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>disk.&lt;HTL&gt;.writes</td>
<td>INT</td>
<td>Total number of disk writes for the virtual machine on the target device specified by &lt;HTL&gt;.</td>
</tr>
<tr>
<td>disk.&lt;HTL&gt;.shares</td>
<td>INT</td>
<td>Number of disk shares assigned to the virtual machine on the target device specified by &lt;HTL&gt;.</td>
</tr>
</tbody>
</table>

### Network Statistics

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>net.adapters</td>
<td>string</td>
<td>Space-delimited set of MAC addresses corresponding to virtual network adapters.</td>
</tr>
<tr>
<td>net.&lt;mac&gt;.totKBRx</td>
<td>INT</td>
<td>Total amount of data received (in KB) by the virtual adapter specified by &lt;mac&gt;. This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic.</td>
</tr>
</tbody>
</table>
### Other Virtual Machine Statistics

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>worldid</td>
<td>INT</td>
<td>World ID for a virtual machine.</td>
</tr>
<tr>
<td>pid</td>
<td>INT</td>
<td>Process ID for a virtual machine.</td>
</tr>
<tr>
<td>uptime</td>
<td>INT</td>
<td>Uptime, in seconds, of a running virtual machine.</td>
</tr>
</tbody>
</table>
Using ESX Server Virtual Machine Resource Variables Efficiently

This section describes how you can query virtual machine variables more efficiently, by minimizing overhead.

**Note:** These tips apply both to both the VmCOM and VmPerl APIs, although the examples use only the VmPerl API.

**Using the Server Object**

You can query virtual machine resource values by using either the virtual machine object or the server object. For the latter, the resource value string used in the query must be qualified with the virtual machine's world ID number.

For example, if a virtual machine's world ID is 131, then the following VmPerl queries return identical values.

```perl
$server->get_resource("vm.131.cpu.usedsec");
$vm->get_resource("cpu.usedsec");
```

However, querying through the virtual machine object has slightly more overhead than querying through the server object, so we recommend that you use the server object.

**Reusing a Single Server Object**

There is some overhead associated with creating and connecting a server object or a virtual machine object. Therefore, if you expect to query system or virtual machine resource values very frequently, then we recommend you perform all queries using a single connected server object.

For example, in VmPerl:

```perl
# one-time setup of server object
my $conn_params = VMware::VmPerl::ConnectParams::new($server_name, $port, $user, $passwd);
my $server = VMware::VmPerl::Server::new();
$server->connect($conn_params);

# query system / vm resources
$server->get_resource("system.worlds");
$server->get_resource("vm.<...>");
...
...

# one-time teardown of server object
$server->disconnect();
```
Symbols
$choice 59
$connectparams 50, 52, 54
$dev_name 59
$infotype 58, 64, 65
$key_name 57, 67–68
$mode 55
$question 59
$vm_name 54
A
access permissions 23, 60, 114
add_redo() method 56
adding 25, 56, 111
adding redo log 25, 56, 111
AddRedo() method 25
answer_question() method 59
answering a question 28, 49, 58, 62, 82–85, 103, 105–106, 113
AnswerQuestion() method 27, 28, 100
API incompatible with server 101
authd 104
C
Capabilities property 23
choice 27
Choices property 20, 27, 28
collection object 16
command, cscript 38, 44
Commit() method 26
commit() method 57
committing redo log changes 26, 57, 112
concepts
VmCOM 15
VmPerl 49
Config property 21, 27
cconfig.ini file 103
ConfigFileName property 22
configuration file for virtual machine 16, 19, 21, 24, 50, 53, 54, 58, 101, 113
configuration variable 21, 58, 88–90
91–92, 112
Connect() method 15, 17, 19, 24, 100, 101
connect() method 50, 52, 54, 101
Connect_device() method 59
ConnectDevice() method 27
connected users 23, 60, 114
connecting to a device 27, 59, 113
to a server 15, 19, 24, 49, 52, 54
to a virtual machine 24, 54, 102
connection parameters 15, 17, 19, 24, 52, 54
connection security 17, 51
connections, total number of simultaneous 19, 24, 52, 54
Count property 20
CPU statistics 120, 124–126
cscript 38, 44
D
device, connecting to 27, 59, 113
device, disconnecting from 27, 59, 113
device_is_connected() method 59
DeviceIsConnected property 22
devName 27
DHCP lease 29, 63
disconnect_device() method 59
DisconnectDevice() method 27
disconnected virtual machine 102
disconnecting from a device 27, 59, 113
disk statistics 122, 127
E
error condition requiring user input 28, 49, 59, 62, 103, 105–106
error handling 100
error, VmCOM 101–102
vmErr_BADSTATE 27, 101
vmErr_BADVERSION 101
vmErr_DISCONNECT 100, 101
vmErr_INSUFFICIENT_RESOURCES
vmErr_INVALIDARGS 101
vmErr_INVALIDVM 101
vmErr_NEEDINPUT 100, 101
vmErr_NETFAIL 101
vmErr_NOACCESS 101
vmErr_NOPROPERTY 101
vmErr_NOTCONNECTED 100, 102
vmErr_NOTSUPPORTED 102
vmErr_PROXYFAIL 102
vmErr_TIMEOUT 102
vmErr_UNSPECIFIED 102
vmErr_VMBUSY 102
vmErr_VMEXISTS 102
vmErr_VMINITFAILED 102
error, VmPerl 101–102
VM_E_BADSTATE 59, 101
VM_E_BADVERSION 101
VM_E_DISCONNECT 101
VM_E_INSUFFICIENT_RESOURCES 101
VM_E_INVALIDARGS 101
VM_E_INVALIDVM 101
VM_E_NEEDINPUT 101
VM_E_NETFAIL 101
VM_E_NOACCESS 101
VM_E_NOPROPERTY 101
VM_E_NOTCONNECTED 102
VM_E_NOTSUPPORTED 102
VM_E_PROXYFAIL 102
VM_E_TIMEOUT 102
VM_E_UNSPECIFIED 102
VM_E_VMBUSY 102
VM_E_VMEXISTS 102
VM_E_VMINITFAILED 102
ESX Server 18, 53, 109
CPU statistics 120
disk statistics 122
memory statistics 120–122
network statistics 123
set resource variable 18, 53, 109
ESX Server, resource variable 120–123
event ID 104
Event Viewer 103–106
ExecutionState property 21
G
get resource variable 18, 22, 53, 60, 109, 113
capabilities() method 60
choices() method 59, 62
_config() method 58
_config_file_name() method 58
_connected_users() method 57
_execution_state() method 57
_guest_info() method 57, 68
_heartbeat() method 58
_hostname() method 51
_id() method 60, 62
_last_error() method 52, 54, 59, 100
_password() method 51
_pending_question() method 58
_pid() method 60
_port() method 51
_product_info() method 58, 64, 65
_remote_connections() method 60
_resource() method 53, 60
text() method 58, 62
tools_last_active() method 58, 60
_uptime() method 60
_username() method 51
generating resource variables for ESX Server 18, 53, 109, 120–123
generating resource variables in a virtual machine 22, 60, 113, 124–128
guest operating system 23, 37, 60, 66–68
_uptime 22, 60, 113
GuestInfo property 21
GuestInfo variable 32–34, 66–68, 112
H
hard power transition 30, 64, 115
heartbeat 22, 58, 79–81, 113
Heartbeat property 22
host platform 31, 65, 113
hostname 17, 51
I
ID for a running virtual machine 22, 60, 114
Id property 22, 28
index 59
infoType 22
input, requiring 27, 28, 49, 58, 59, 62,
82–85, 103, 105–106, 113
installation
VmCOM 11–12
VmPerl 11–12
installrv 44
insufficient memory 101
insufficient resources 19, 24, 52, 54,
101
invalid power transition 101
is_connected() method 52, 54
ISupportErrorInfo 100
Item property 20
J
JScript 38, 38–40
K
keyName 21, 33–34
L
limits 19, 24, 52, 54, 101
Linux operating system
installing VmPerl on 12
list of virtual machines 16, 38–40, 40–
43, 43–47, 50, 72–73, 74–76, 109
M
memory 101
memory size 21, 58
memory statistics 120–122, 126–127
memory, values stored in 21, 57
messages 103
method, VmCOM
AddRedo() 25
AnswerQuestion() 27, 28, 100
Commit() 26
Connect() 15, 17, 19, 24, 100, 101
ConnectDevice() 27
DisconnectDevice() 27
RegisterVm() 19
Reset() 24
Start() 24
Stop() 24
Suspend() 25
UnRegisterVm() 19
method, VmPerl
add_redo() 56
answer_question() 59
commit() 57
connect() 50, 52, 54, 101
connect_device() 59
device_is_connected() 59
disconnect_device() 59
get_capabilities() 60
get_choices() 59, 62
get_config() 58
get_config_file_name() 58
get_connected_users() 57
get_execution_state() 57
guest_info() 57, 68
guest_heartbeat() 58
guest_hostname() 51
guest_id() 60, 62
get_last_error() 52, 54, 59, 100
get_password() 51
guest_pending_question() 58
get_pid() 60
guest_port() 51
guest_product_info() 58, 64, 65
guest_remote_connections() 50
guest_resource() 53, 60
guest_text() 58, 62
guest_tools_last_active() 58, 60
guest_uptime() 60
guest_username() 51
guest_is_connected() 52, 54
guest_unregister_vm() 52
registered_vm_names() 50, 52
reset() 55
set_config() 58, 59
set_guest_info() 57, 67
set_resource() 53, 60
start() 55
stop() 55
suspend() 56
unregister_vm() 53
MiniMUI Visual Basic project 12, 37
N
network failure 101
network port 51
network statistics 123, 127–128
no response 102
not enough memory 101
P
passing information between script and guest operating system 32–34, 66–68
password 17, 51
PendingQuestion property 21, 27, 28, 100
permission 101
Pid property 22
platform 31, 65, 113
platform information 31
port 17, 51, 108
power status of a virtual machine 21, 57, 77–78, 110
power transition 29, 63, 103, 105
hard 30, 64, 115
invalid 101
soft 30, 64, 115
trysoft 30, 64, 115
powering off a virtual machine 24, 29–30, 55, 63–64, 110, 115
powering on a virtual machine 24, 29–30, 55, 63–64, 110, 115
process ID for a running virtual machine 22, 60, 114
product information 22, 30, 31, 58, 64, 65, 113
ProductInfo property 22
property
Capabilities 23
Choices 20, 27, 28
Config 21, 27
ConfigFileName 22
Count 20
DevicesConnected 22
ExecutionState 21
GuestInfo 21
Heartbeat 22
Id 22, 28
Item 20
PendingQuestion 21, 27, 28, 100
Pid 22
ProductInfo 22
RegisteredVmNames 18
RemoteConnections 23
Resource 18, 22
Text 28
ToolsLastActive 22, 23
Uptime 22
proxy 102
proxy failure 102
Q
question 49, 58, 59, 62, 82–85, 103, 105–106, 113
R
reconnect to a virtual machine 24
redo log 25, 40, 56, 93, 111
redo log, committing changes to 26, 57, 95, 112
register_vm() method 52
registered_vm_names() method 50, 52
RegisteredVmNames property 18
registering virtual machine 52, 102, 109
RegisterVm() method 19
RemoteConnections property 23
Reset() method 24
reset() method 55
resetting a virtual machine 24, 29–30, 55, 63–64, 110, 115
Resource property 18, 22
resource variable 119–128
get 18, 22, 53, 60, 109, 113
set 18, 22, 53, 60, 109, 113
resuming a suspended machine 24, 55, 110, 115
S
sample scripts, VmCOM 12, 35–47
connecting to server and listing virtual machines 38–40, 40–43
listing and starting virtual machines 43–47
sample scripts, VmPerl 69–92
answering question for stuck virtual machine 82–85
determining power status 77–78
listing and starting virtual machines 74–76
listing virtual machines 72–73
monitoring virtual machine heartbeat 79–81
retrieving a configuration variable 91–92
setting a configuration variable 88–90
suspending a virtual machine 86–87
sample scripts. VmPerl 13
server script 32–34, 66–68
security 17, 51, 101
server connecting to 15, 19, 24, 38–40, 40–43, 49, 52, 54
incompatible with API 101
security 17, 51
virtual machines on 16
VmServerCtl 15, 18–19, 100
VMware::VmPerl::Server 49, 52–53
serverd 104
set_config() method 58, 59
set_guest_info() method 57, 67
set_resource() method 53, 60
setting resource variables for ESX Server 120–123
setting resource variables in a virtual machine 22, 60, 113, 124–128
setting resource variables in ESX Server 18, 53, 109
shared variables 32–34, 66–68
simultaneous connections 19, 24, 52, 54
soft power transition 30, 64, 115
srvany 44
Start() method 24
start() method 55
starting a virtual machine 24, 29–30, 55, 63–64, 110, 115
state of virtual machine 21, 57, 110
Stop() method 24
stop() method 55
stopping a virtual machine 24, 29–30, 55, 63–64, 110, 115
string
$key_name 57
keyName 21
Suspend() method 25
suspend() method 56
suspending machine, resuming 24, 55, 110, 115
suspending a virtual machine 25, 29–30, 56, 63–64, 86–87, 110, 115
T
TCP port 17, 51
Text property 28
time out 102
ToolsLastActive property 22, 23
trysoft power transition 30, 64, 115
U
undoable disk 40
uninstalling VmPerl 13
unregister_vm() method 53
unregistering virtual machine 109
UnregisterVm() method 19
uptime 22, 60, 113
Uptime property 22
user input 27, 28, 49, 58, 59, 62, 82–85, 103, 105–106, 113
user name 17, 51, 108
user permissions 23, 60, 114
users, connected 23, 60, 114
V
variable 21, 32–34, 58, 66–68, 112
VBScript 38, 40–43, 43–47
virtual device 22, 27, 59, 113
virtual machine 22, 49, 54–61, 66, 113
configuration file 16, 19, 21, 24, 50, 53, 54, 58, 101, 113
connecting to 24, 54, 102
CPU statistics 124–126
disconnected 102
disk statistics 127
event logging 103–106
execution state 29, 63
heartbeat 22, 58, 79–81, 113
ID 22, 60, 114
list of 16, 19, 21, 24, 50, 72–73, 74–76, 109
memory size 21, 58
memory statistics 126–127
<table>
<thead>
<tr>
<th>Method</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmErr_INVALIDVM</td>
<td>101</td>
</tr>
<tr>
<td>vmErr_NEEDINPUT</td>
<td>100, 101</td>
</tr>
<tr>
<td>vmErr_NETFAIL</td>
<td>101</td>
</tr>
<tr>
<td>vmErr_NOACCESS</td>
<td>101</td>
</tr>
<tr>
<td>vmErr_NOPROPERTY</td>
<td>101</td>
</tr>
<tr>
<td>vmErr_NOTCONNECTED</td>
<td>100, 102</td>
</tr>
<tr>
<td>vmErr_NOTSUPPORTED</td>
<td>102</td>
</tr>
<tr>
<td>vmErr_PROXYFAIL</td>
<td>102</td>
</tr>
<tr>
<td>vmErr_TIMEOUT</td>
<td>102</td>
</tr>
<tr>
<td>vmErr_UNSPECIFIED</td>
<td>102</td>
</tr>
<tr>
<td>vmErr_VMBUSY</td>
<td>102</td>
</tr>
<tr>
<td>vmErr_VMINITFAILED</td>
<td>102</td>
</tr>
<tr>
<td>VmExecutionState</td>
<td>29</td>
</tr>
<tr>
<td>vmName</td>
<td>24</td>
</tr>
<tr>
<td>VmPerl</td>
<td></td>
</tr>
<tr>
<td>add_redo() method</td>
<td>56</td>
</tr>
<tr>
<td>answer_question() method</td>
<td>59</td>
</tr>
<tr>
<td>commit() method</td>
<td>57</td>
</tr>
<tr>
<td>concepts</td>
<td>49</td>
</tr>
<tr>
<td>connect() method</td>
<td>50, 52, 54, 101</td>
</tr>
<tr>
<td>connect_device() method</td>
<td>59</td>
</tr>
<tr>
<td>device_is_connected() method</td>
<td>59</td>
</tr>
<tr>
<td>disconnect_device() method</td>
<td>59</td>
</tr>
<tr>
<td>error handling</td>
<td>100</td>
</tr>
<tr>
<td>get_capabilities() method</td>
<td>60</td>
</tr>
<tr>
<td>get_choices() method</td>
<td>59, 62</td>
</tr>
<tr>
<td>get_config() method</td>
<td>58</td>
</tr>
<tr>
<td>get_config_file_name() method</td>
<td>58</td>
</tr>
<tr>
<td>get_connected_users() method</td>
<td>57</td>
</tr>
<tr>
<td>get_current_state() method</td>
<td>57</td>
</tr>
<tr>
<td>get_guest_info() method</td>
<td>57, 67</td>
</tr>
<tr>
<td>get_heartbeats() method</td>
<td>58</td>
</tr>
<tr>
<td>get_hostname() method</td>
<td>51</td>
</tr>
<tr>
<td>get_id() method</td>
<td>60, 62</td>
</tr>
<tr>
<td>get_last_error() method</td>
<td>52, 54, 59, 100</td>
</tr>
<tr>
<td>get_password() method</td>
<td>51</td>
</tr>
<tr>
<td>get_pending_question() method</td>
<td>58</td>
</tr>
<tr>
<td>get_product_info() method</td>
<td>58, 64, 65</td>
</tr>
<tr>
<td>get_remote_connections() method</td>
<td>60</td>
</tr>
<tr>
<td>get_resource() method</td>
<td>53, 60</td>
</tr>
<tr>
<td>get_text() method</td>
<td>58, 62</td>
</tr>
<tr>
<td>get_tools_last_active() method</td>
<td>58, 60</td>
</tr>
<tr>
<td>get_uptime() method</td>
<td>60</td>
</tr>
<tr>
<td>get_username() method</td>
<td>51</td>
</tr>
<tr>
<td>is_connected() method</td>
<td>52, 54</td>
</tr>
<tr>
<td>register_vm() method</td>
<td>52</td>
</tr>
<tr>
<td>registered_vm_names() method</td>
<td>50, 52</td>
</tr>
<tr>
<td>reset() method</td>
<td>55</td>
</tr>
<tr>
<td>sample scripts</td>
<td>13, 69–92</td>
</tr>
<tr>
<td>server operations</td>
<td>109</td>
</tr>
<tr>
<td>virtual machine operations</td>
<td>110–114</td>
</tr>
<tr>
<td>use with Linux operating system</td>
<td>7</td>
</tr>
<tr>
<td>use with Windows operating system</td>
<td>7</td>
</tr>
<tr>
<td>VmPlatform</td>
<td>31</td>
</tr>
<tr>
<td>VmPowerOpMode</td>
<td>24–25, 29</td>
</tr>
<tr>
<td>VmProdInfo_Platform</td>
<td>31</td>
</tr>
<tr>
<td>VmProdInfo_Product</td>
<td>31</td>
</tr>
<tr>
<td>VmProdInfoType</td>
<td>22, 30</td>
</tr>
<tr>
<td>VmProduct</td>
<td>31</td>
</tr>
<tr>
<td>VmQuestion</td>
<td>27, 28, 100</td>
</tr>
<tr>
<td>VmQuestion.Choices</td>
<td>20</td>
</tr>
<tr>
<td>VmServerCtl</td>
<td>15, 18–19, 100</td>
</tr>
<tr>
<td>VmServerCtl.Connect() method</td>
<td>17, 19, 24</td>
</tr>
<tr>
<td>VmServerCtl.RegisteredVmNames</td>
<td>20</td>
</tr>
<tr>
<td>VMware Tools</td>
<td>22, 23, 32–34, 58, 60, 66–68, 113</td>
</tr>
<tr>
<td>VMware::VmPerl::ConnectParams</td>
<td>49, 51</td>
</tr>
<tr>
<td>VMware::VmPerl::Question</td>
<td>49, 58, 59, 62</td>
</tr>
<tr>
<td>VMware::VmPerl::Server</td>
<td>49, 52–53</td>
</tr>
<tr>
<td>vmware-cmd</td>
<td>107</td>
</tr>
<tr>
<td>options</td>
<td></td>
</tr>
<tr>
<td>server operations</td>
<td>109</td>
</tr>
<tr>
<td>virtual machine operations</td>
<td>110–114</td>
</tr>
<tr>
<td>vmware-control</td>
<td>107</td>
</tr>
</tbody>
</table>
W
waiting for user input 27, 28, 40–43, 49, 58, 59, 62, 82–85, 103, 105–106, 113
Web proxy 102
Windows operating system
installing Scripting APIs on 11–12
Windows Script File 38, 40, 43, 47