VMware™ Scripting API

User’s Manual

Version 2.0
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>7</td>
</tr>
<tr>
<td>VMware Scripting APIs</td>
<td>8</td>
</tr>
<tr>
<td>Supported Products</td>
<td>9</td>
</tr>
<tr>
<td>Intended Audience</td>
<td>9</td>
</tr>
<tr>
<td>Getting Support from VMware</td>
<td>9</td>
</tr>
<tr>
<td>Using the VMware Scripting APIs</td>
<td>9</td>
</tr>
<tr>
<td>Installing the VMware Scripting API</td>
<td>9</td>
</tr>
<tr>
<td>Installing the VMware Scripting API on a Windows Machine</td>
<td>10</td>
</tr>
<tr>
<td>Installing VmPerl Scripting API on a Linux Machine</td>
<td>11</td>
</tr>
<tr>
<td>Using VmCOM</td>
<td>13</td>
</tr>
<tr>
<td>What is VmCOM?</td>
<td>14</td>
</tr>
<tr>
<td>VmCOM Objects</td>
<td>15</td>
</tr>
<tr>
<td>VmConnectParams</td>
<td>15</td>
</tr>
<tr>
<td>VmServerCtl</td>
<td>16</td>
</tr>
<tr>
<td>Property</td>
<td>16</td>
</tr>
<tr>
<td>Methods</td>
<td>16</td>
</tr>
<tr>
<td>VmCollection</td>
<td>16</td>
</tr>
<tr>
<td>VmCtl</td>
<td>17</td>
</tr>
<tr>
<td>Properties</td>
<td>17</td>
</tr>
<tr>
<td>Methods</td>
<td>19</td>
</tr>
<tr>
<td>VmQuestion</td>
<td>22</td>
</tr>
<tr>
<td>Symbolic Constant Enumerations</td>
<td>22</td>
</tr>
<tr>
<td>VmExecutionState</td>
<td>22</td>
</tr>
<tr>
<td>VmPowerOpMode</td>
<td>23</td>
</tr>
<tr>
<td>VmProdInfoType</td>
<td>24</td>
</tr>
<tr>
<td>VmProduct</td>
<td>24</td>
</tr>
<tr>
<td>VmPlatform</td>
<td>24</td>
</tr>
<tr>
<td>Using VmCOM to Pass User-Defined Information Between a Running Guest</td>
<td>25</td>
</tr>
<tr>
<td>Operating System and a Script</td>
<td>25</td>
</tr>
<tr>
<td>GuestInfo Variables</td>
<td>25</td>
</tr>
<tr>
<td>Sending Information Set in a VmCOM Script to the Guest</td>
<td>26</td>
</tr>
<tr>
<td>Operating System</td>
<td>26</td>
</tr>
<tr>
<td>Sending Information Set in the Guest Operating System to a VmCOM Script</td>
<td>27</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Using Sample VmCOM Programs</td>
<td>29</td>
</tr>
<tr>
<td>Sample VmCOM Programs</td>
<td>30</td>
</tr>
<tr>
<td>Copyright Information</td>
<td>30</td>
</tr>
<tr>
<td>MiniMUI Visual Basic Sample Program</td>
<td>31</td>
</tr>
<tr>
<td>JScript and VBScript Sample Programs</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Using VmPerl</td>
<td>43</td>
</tr>
<tr>
<td>VmPerl Modules</td>
<td>44</td>
</tr>
<tr>
<td>VMware::VmPerl::ConnectParams</td>
<td>45</td>
</tr>
<tr>
<td>VMware::VmPerl::Server</td>
<td>46</td>
</tr>
<tr>
<td>VMware::VmPerl::VM</td>
<td>47</td>
</tr>
<tr>
<td>Additional Information on get_tools_last_active</td>
<td>51</td>
</tr>
<tr>
<td>VMware::VmPerl::Question</td>
<td>52</td>
</tr>
<tr>
<td>Symbolic Constants</td>
<td>52</td>
</tr>
<tr>
<td>VM_EXECUTION_STATE_&lt;XXX&gt; Values</td>
<td>52</td>
</tr>
<tr>
<td>VM_POWEROP_MODE_&lt;XXX&gt; Values</td>
<td>53</td>
</tr>
<tr>
<td>Infotype Values</td>
<td>54</td>
</tr>
<tr>
<td>VM_PRODINFO_PRODUCT_&lt;XXX&gt; Values</td>
<td>54</td>
</tr>
<tr>
<td>VM_PRODINFO_PLATFORM_&lt;XXX&gt; Values</td>
<td>54</td>
</tr>
<tr>
<td>Using VmPerl to Pass User-Defined Information Between a Running Guest</td>
<td></td>
</tr>
<tr>
<td>Operating System and a Script</td>
<td>55</td>
</tr>
<tr>
<td>GuestInfo Variables</td>
<td>55</td>
</tr>
<tr>
<td>Sending Information Set in a VmPerl Script to the Guest Operating System</td>
<td>56</td>
</tr>
<tr>
<td>Sending Information Set in the Guest Operating System to a VmPerl Script</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Using Sample VmPerl Scripts</td>
<td>59</td>
</tr>
<tr>
<td>Sample Perl Scripts</td>
<td>60</td>
</tr>
<tr>
<td>Copyright Information</td>
<td>60</td>
</tr>
<tr>
<td>Listing the Virtual Machines on the Server</td>
<td>61</td>
</tr>
<tr>
<td>Starting All Virtual Machines on a Server</td>
<td>63</td>
</tr>
<tr>
<td>Checking a Virtual Machine's Power Status</td>
<td>65</td>
</tr>
<tr>
<td>Monitoring a Virtual Machine's Heartbeat</td>
<td>67</td>
</tr>
<tr>
<td>Answering Questions Posed by a Virtual Machine</td>
<td>70</td>
</tr>
<tr>
<td>Suspending a Virtual Machine</td>
<td>74</td>
</tr>
<tr>
<td>Setting a Virtual Machine's IP Address Configuration Variable</td>
<td>75</td>
</tr>
<tr>
<td>Getting a Virtual Machine's IP Address</td>
<td>77</td>
</tr>
</tbody>
</table>
## Error Codes and Event Logging

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Codes and Event Logging</td>
<td>81</td>
</tr>
<tr>
<td>Error Codes</td>
<td>82</td>
</tr>
<tr>
<td>Error Handling for the VmCOM Library</td>
<td>82</td>
</tr>
<tr>
<td>Error Handling for the VmPerl Library</td>
<td>82</td>
</tr>
<tr>
<td>Common VmCOM and VmPerl Errors</td>
<td>83</td>
</tr>
<tr>
<td>Event Logging</td>
<td>84</td>
</tr>
<tr>
<td>Using the Event Viewer</td>
<td>85</td>
</tr>
<tr>
<td>Reading the Event Log</td>
<td>86</td>
</tr>
</tbody>
</table>

## Appendix A: vmware-cmd Utility

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the vmware-cmd Utility</td>
<td>89</td>
</tr>
<tr>
<td>Options</td>
<td>90</td>
</tr>
<tr>
<td>vmware-cmd Operations on a Server</td>
<td>90</td>
</tr>
<tr>
<td>vmware-cmd Operations on a Virtual Machine</td>
<td>91</td>
</tr>
<tr>
<td>&lt;powerop_mode&gt; Values</td>
<td>93</td>
</tr>
<tr>
<td>vmware-cmd Utility Examples</td>
<td>93</td>
</tr>
</tbody>
</table>

## Index

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>97</td>
</tr>
</tbody>
</table>
Introduction
VMware Scripting APIs

This release of VMware™ scripting APIs version 2.0 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.

You may install the Scripting APIs on the GSX Server host and on remote workstations.

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. Refer to the VMware GSX Server User’s Manual for complete information on system requirements at www.vmware.com/support/gsx25/doc.

**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 and the host operating system used for this application. For more information on VMware GSX Server refer to the VMware GSX Server User’s Manual at www.vmware.com/support/gsx25/doc.

**Getting Support from VMware**

See www.vmware.com/support/developer for 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.

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 Sample VmCOM Programs on page 26 and for VmPerl scripts, see Sample Perl Scripts on page 56. For the list of error codes, see Error Codes on page 78.

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

**Installing the VMware Scripting API**

You have the option of installing the VMware Scripting API on your server when you installed the software. For complete information on installing GSX Server, see www.vmware.com/support/gsx25/doc/install_gsx.html.

However, if you want to run VMware Scripting APIs from a machine other than the server, you need to install VmCOM or VmPerl on that machine. Your administrator will provide you with the appropriate
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 Sample VmCOM Programs on page 26 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 Sample Perl Scripts on page 56 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.
Introduction

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.
   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 Sample Perl Scripts on page 56 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
Using VmCOM

What is VmCOM?

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

A \texttt{VmServerCtl} object represents a server and exports server-level services, such as virtual machine enumeration and registration. A \texttt{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 \texttt{VmServerCtl} or \texttt{VmCtl} object by calling its \texttt{Connect()} method before accessing any other method.

The \texttt{Connect()} method requires a \texttt{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 \texttt{VmServerCtl} object, \texttt{VmCtl.Connect()} also takes a string specifying the configuration file name of the virtual machine that will be connected.

Once a \texttt{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 \texttt{VmServerCtl.RegisteredVmNames} property. This property returns a collection object named \texttt{VmCollection}. The collection's elements comprise virtual machine configuration file names and you can enumerate these elements using, for example, the \texttt{for each} syntax in Visual Basic. If you know the configuration file name of a specific virtual machine, you can connect the \texttt{VmCtl} object directly without using a \texttt{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 \texttt{Project > References}, and enable the check box for \texttt{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.
Using VmCOM

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 host with GSX Server, then the connection 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>
Using VmCOM

VmServerCtl

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

Property

The RegisteredVmNames read-only (GET) property 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.

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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object.Connect(&lt;params&gt;)</td>
<td>The Connect() method connects the object to a VMware GSX Server or a VMware ESX Server where params is a VmConnectParams object that specifies the system and user information. The Connect() method fails with error code vmErr_INSUFFICIENT_RESOURCES if this limit is reached. In order to connect new objects, destroy one or more connected VmCtl or VmServerCtl objects. For example, you can do this by setting an object to Nothing in Visual Basic.</td>
</tr>
<tr>
<td>object.RegisterVm(&lt;vmName&gt;)</td>
<td>The RegisterVm method registers a virtual machine on a server where vmName is a string specifying the virtual machine's configuration file name.</td>
</tr>
<tr>
<td>object.UnregisterVm(&lt;vmName&gt;)</td>
<td>The UnRegisterVm method unregisters a virtual machine from a server where vmName 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).
Using VmCOM

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)</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).

<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 18.</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 21.</td>
</tr>
</tbody>
</table>
Using VmCOM

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Property Type</th>
<th>Access Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>ConfigFileName</td>
<td>string</td>
<td>GET</td>
<td>Returns the configuration file name for the virtual machine. This method fails if the VmCtl object is not connected.</td>
</tr>
<tr>
<td>Heartbeat</td>
<td>integer</td>
<td>GET</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 this service is not running.</td>
</tr>
<tr>
<td>ToolsLastActive</td>
<td>integer</td>
<td>GET</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 ToolsLastActive on page 15.</td>
</tr>
<tr>
<td>DeviceIsConnected(devName)</td>
<td>Boolean</td>
<td>GET</td>
<td>Returns True if the specified device is connected. Otherwise, False is returned.</td>
</tr>
<tr>
<td>ProductInfo(infoType)</td>
<td>integer, VmProduct or VmPlatform</td>
<td>GET</td>
<td>Returns an integer representing the value of the product information field specified by infoType, which is of type VmProdInfoType. See VmProdInfoType on page 20 for a list of valid types and return values.</td>
</tr>
</tbody>
</table>
Using VmCOM

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.
# Using VmCOM

<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 19. 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 19. 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 19. This method always fails if the virtual machine is not in the <code>vmExecutionState_On</code> state.</td>
</tr>
</tbody>
</table>
## Using VmCOM

<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 19. This method always fails if the virtual machine is not in the <code>vmExecutionState_On</code> state. If you attempt to suspend a virtual machine with more the 2GB of memory, the suspend operation will time fail after a time-out period.</td>
</tr>
<tr>
<td>object.AnswerQuestion(&lt;question&gt;, &lt;choice&gt;)</td>
<td>The <code>AnswerQuestion()</code> method replies to a question where <code>question</code> is a <code>VmQuestion</code> object that represents the question that requires an answer and <code>choice</code> 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 <code>vmExecutionState_Stuck</code> state and requires user input to continue, use this method to answer the current question or dismiss the current error message. First, get a <code>VmQuestion</code> object from <code>VmCtl.PendingQuestion</code>. You can retrieve the possible choices and their respective indices from the <code>VmQuestion.Choices</code> property. Then, use the <code>AnswerQuestion</code> method to answer the question.</td>
</tr>
<tr>
<td>object.ConnectDevice(&lt;devName&gt;)</td>
<td>The <code>ConnectDevice()</code> method sets a virtual device to the connected state where <code>devName</code> 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 <code>vmErr_BADSTATE</code> error is returned. Use the <code>Config()</code> property to set configuration parameters relevant to the virtual device before calling the <code>ConnectDevice()</code> method. The following code example illustrates connecting a virtual drive to a CD image file:</td>
</tr>
</tbody>
</table>

```javascript
vm.Config("ide1:0.devicetype") = "cdrom-image"
vm.Config("ide1:0.filename") = "/iso/foo.iso"
vm.ConnectDevice("ide1:0")
```
| object.DisconnectDevice(devName) | 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. |
Using VmCOM

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

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 these scripts, see [www.vmware.com/support/gsx25/doc/tools_gsx.html](http://www.vmware.com/support/gsx25/doc/tools_gsx.html). Refer to the section on executing scripts.

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

<table>
<thead>
<tr>
<th>VmPowerOpMode Values</th>
<th>Description</th>
</tr>
</thead>
</table>
| vmPowerOpMode_Soft         | **To succeed, soft power transitions require the current version of the VMware Tools service to be installed and running in the guest operating system.** 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.  
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.  
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. |
| vmPowerOpMode_Hard         | Start 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.                                                                                                                                                                                                 |
Using VmCOM

VmProdInfoType

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

<table>
<thead>
<tr>
<th>VmProdInfoType Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmProdInfo_Product</td>
<td>The VMware product type is returned as VmProduct. For more information on VmProduct, see the following section.</td>
</tr>
<tr>
<td>vmProdInfo_Platform</td>
<td>The host platform type is returned as VmPlatform. For more information on VmPlatform, see VmPlatform on page 20.</td>
</tr>
<tr>
<td>vmProdInfo_Build</td>
<td>The product's build number.</td>
</tr>
<tr>
<td>vmProdInfo_Version_Major</td>
<td>The product's major version number.</td>
</tr>
<tr>
<td>vmProdInfo_Version_Minor</td>
<td>The product's minor version number.</td>
</tr>
<tr>
<td>vmProdInfo_Version_Revision</td>
<td>The product's revision number.</td>
</tr>
</tbody>
</table>

VmProduct

The VmProduct symbol 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 symbol 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_VMNIX</td>
<td>The host platform is the ESX Server console operating system.</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 22 for an example of this procedure.


**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.

---

<table>
<thead>
<tr>
<th>VmPlatform Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vmPlatform_UNKNOWN</td>
<td>The host platform is unknown.</td>
</tr>
</tbody>
</table>
Using VmCOM

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”:

```vbnet
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:

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

In a Linux guest operating system:

```cmd
/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:

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

**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:

    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:

    str = vm.GuestInfo("name")
Using Sample VmCOM Programs
Using Sample VmCOM Programs

Sample VmCOM Programs

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) 1999-2003 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
Using Sample VmCOM Programs

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
Using Sample VmCOM Programs

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 13.

**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) 1999-2003 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.
```
Using Sample VmCOM Programs

// 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

var cp = WScript.CreateObject("VmCOM.VmConnectParams");
var server = WScript.CreateObject("VmCOM.VmServerCtl");
var vmCollection = server.RegisteredVmNames;

for (var j = 1; j <= vmCollection.count; j++) {

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

  var str = "config path=\" + vmName + \" OS=\" + vm.Config("guestOS") + \" state=\";
  var execStateString = State2Str(vm);
  str += execStateString;

  if (execStateString == "STUCK") {
    var question = vm.PendingQuestion;
    str += " pending question=\" + question.text + \" choices=\";

    var choices = question.choices;
    for (var i = 1; i <= choices.count; i++) {
      str += "[\" + choices[i] + \"] \";
    }
  }

  WScript.Echo(str);
}

function State2Str(vm) {
  // Implement the function to convert the state string to
  // a human-readable format.
Using Sample VmCOM Programs

```
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 `vm.ExecutionState_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
Using Sample VmCOM Programs

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 2.x.

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.

; VmCOM VBScript Sample Script (sample2)
; Copyright (c) 1999-2003 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
; - in a command line window, type:
;   cscript //nologo sample2.wsf
Using Sample VmCOM Programs

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 = " -- Automatic selection of " & q.choices(i) & "
            next
        end if
    end if

end if
Using Sample VmCOM Programs

```wscript
WScript.Echo(s)
next

function State2Str(vm)
    select case vm.ExecutionState
        case vm.ExecutionState_On
            State2Str = "ON"
        case vm.ExecutionState_Off
            State2Str = "OFF"
        case vm.ExecutionState_Suspended
            State2Str = "SUSPENDED"
        case vm.ExecutionState_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.

```xml
<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.
Using Sample VmCOM Programs

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) 1999-2003 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")
```
Using Sample VmCOM Programs

' 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
Using Sample VmCOM Programs

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
end if
Using Sample VmCOM Programs

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
VmPerl Modules

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 host with GSX Server, then the connections is 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>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, 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>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>$connectparams-&gt;get_username()</td>
<td>Gets or set the name of a user on the server.</td>
</tr>
<tr>
<td>$connectparams-&gt;get_password()</td>
<td>Gets or set the user’s password on the server.</td>
</tr>
</tbody>
</table>
Using VmPerl

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>
<tr>
<td>$server-&gt;register_vm($vm_name)</td>
<td>Registers a virtual machine on a server where $vm_name is a string specifying the virtual machine's configuration file name.</td>
</tr>
<tr>
<td>$server-&gt;unregister_vm($vm_name)</td>
<td>Unregisters a virtual machine from a server where $vm_name is a string specifying the virtual machine's configuration file name.</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$vm-&gt;start($mode)</code></td>
<td>Returns the defined value on success or <code>undef</code> (undefined value) on failure. Powers on a previously powered-off virtual machine or resumes a suspended virtual machine where <code>$mode</code> specifies the operation's behavior based on the value of the <code>VMware::VmPerl::VMPOWEROP_MODE_&lt;XXX&gt;</code> where <code>&lt;XXX&gt;</code> is HARD, SOFT, or TRYSOFT. If <code>$mode</code> is not specified, the default mode is <code>VMPOWEROP_MODE_SOFT</code>. For more information, see <code>VMPOWEROP_MODE_&lt;XXX&gt;</code> Values on page 49. <strong>Note:</strong> If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify <code>VMware::VmPerl::VMPOWEROP_MODE_HARD</code> 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><code>$vm-&gt;stop($mode)</code></td>
<td>Returns the defined value on success or <code>undef</code> (undefined value) on failure. Shuts down and powers off a virtual machine where <code>$mode</code> specifies the operation's behavior based on the value of the <code>VMware::VmPerl::VMPOWEROP_MODE_&lt;XXX&gt;</code> where <code>&lt;XXX&gt;</code> is HARD, SOFT, or TRYSOFT. If <code>$mode</code> is not specified, the default mode is <code>VMPOWEROP_MODE_SOFT</code>. For more information, see <code>VMPOWEROP_MODE_&lt;XXX&gt;</code> Values on page 49. <strong>Note:</strong> If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify <code>VMware::VmPerl::VMPOWEROP_MODE_HARD</code> as the mode or the operation will fail. This method always fails if the virtual machine is not in the <code>VM_EXECUTION_STATE_ON</code> state.</td>
</tr>
<tr>
<td><code>$vm-&gt;reset($mode)</code></td>
<td>Returns the defined value on success or <code>undef</code> (undefined value) on failure. Shuts down, then reboots a virtual machine where <code>$mode</code> specifies the operation's behavior based on the value of the <code>VMware::VmPerl::VMPOWEROP_MODE_&lt;XXX&gt;</code> where <code>&lt;XXX&gt;</code> is HARD, SOFT, or TRYSOFT. If <code>$mode</code> is not specified, the default mode is <code>VMPOWEROP_MODE_SOFT</code>. See <code>VMPOWEROP_MODE_&lt;XXX&gt;</code> Values on page 49. <strong>Note:</strong> If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify <code>VMware::VmPerl::VMPOWEROP_MODE_HARD</code> as the mode or the operation will fail. This method always fails if the virtual machine is not in the <code>VM_EXECUTION_STATE_ON</code> state.</td>
</tr>
</tbody>
</table>
**Using VmPerl**

<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 $mode is not specified, the default mode is VM::POWEROP_MODE_SOFT. For more information, see VM::POWEROP_MODE_&lt;XXX&gt; Values on page 49. <strong>Note:</strong> If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware:VmPerl::VM::POWEROP_MODE_HARD as the mode or the operation will fail. This method always fails if the virtual machine is not in the VMware:VmPerl::VM::EXECUTION_STATE_ON state. If you attempt to suspend a virtual machine with more the 2GB of memory, the suspend operation will time fail after a time-out period.</td>
</tr>
<tr>
<td>$vm-&gt;get_execution_state()</td>
<td>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_&lt;XXX&gt; Values on page 48.</td>
</tr>
<tr>
<td>$vm-&gt;get_guest_info($key_name)</td>
<td>It accesses a shared variable identified by the string $key_name. If you write a GuestInfo variable by using the set_guest_info() method, the new value is written into memory and is discarded when the virtual machine process terminates. For additional information, see Using VmPerl to Pass User-Defined Information Between a Running Guest Operating System and a Script on page 51.</td>
</tr>
<tr>
<td>$vm-&gt;set_guest_info($key_name, $value)</td>
<td>Returns the defined value on success or <strong>undef</strong> (undefined value) on failure. It accesses a shared variable identified by the string $key_name. If you write a GuestInfo variable by using the set_guest_info() method, the new value is written into memory and is discarded when the virtual machine process terminates. For additional information, see Using VmPerl to Pass User-Defined Information Between a Running Guest Operating System and a Script on page 51.</td>
</tr>
<tr>
<td>$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 <strong>undef</strong> (undefined value) on failure. It 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>
</tbody>
</table>
### Using VmPerl

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$vm-&gt;get_product_info($infotype)</code></td>
<td>Returns the defined value on success or <code>undefined</code> on failure. For additional information, see <code>Infotype Values</code> on page 50.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_heartbeat()</code></td>
<td>Returns the defined value on success or <code>undefined</code> on failure. The current heartbeat count generated by the VMware Tools service running in the guest operating system. The count is initialized to zero when the virtual machine is powered on. The heartbeat count is typically incremented at least once per second when the VMware Tools service is running under light load conditions. The count stays constant if the service is not running.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_tools_last_active()</code></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 <code>Additional Information</code> on <code>get_tools_last_active</code> on page 47.</td>
</tr>
<tr>
<td><code>$vm-&gt;get_pending_question()</code></td>
<td>Returns a Vmware::VmPerl::VmQuestion object if the virtual machine is currently in the <code>VM_EXECUTION_STATE_STUCK</code> state. Otherwise, an error is thrown.</td>
</tr>
<tr>
<td><code>$vm-&gt;answer_question($question, $choice)</code></td>
<td>Replies to a question where <code>$question</code> represents the question and <code>$choice</code> represents the index of the selected answer to the question. The index is a number associated with an answer. The first choice's index is always 0. The second choice's index is 2, and so on. Use this method to answer the current question or dismiss the current error message when a virtual machine is in the <code>VM_EXECUTION_STATE_STUCK</code> state and requires user input to continue. First, get a Vmware::VmPerl::Question object from the Vmware::VmPerl::VM object's <code>get_pending_question()</code> method. You can retrieve the possible choices and their respective indices from the Vmware::VmPerl::Question object's <code>get_choices()</code> method. Then, use the <code>answer_question()</code> method to answer the question.</td>
</tr>
<tr>
<td><code>$vm-&gt;device_is_connected($dev_name)</code></td>
<td>Determines the connection state where <code>$dev_name</code> identifies the virtual device. You can use <code>$vm-&gt;get_last_error</code> to determine if an error occurred or if the device is not connected.</td>
</tr>
</tbody>
</table>
Using VmPerl

### 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>get_tools_last_active Method 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>
Using VmPerl

**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>$question-&gt;get_text()</td>
<td>Returns the defined value on success or <em>undef</em> (undefined value) on failure.</td>
</tr>
<tr>
<td>$question-&gt;get_choices()</td>
<td>Returns the defined value on success or <em>undef</em> (undefined value) on failure.</td>
</tr>
<tr>
<td>$question-&gt;get_id()</td>
<td>Returns the defined value on success or <em>undef</em> (undefined value) on failure.</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>
</tbody>
</table>
Using VmPerl

<table>
<thead>
<tr>
<th>Execution_state Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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 [www.vmware.com/support/gsx25/doc/tools_gsx.html](http://www.vmware.com/support/gsx25/doc/tools_gsx.html). Refer to the section on executing scripts.

The possible values are listed in the following table:

<table>
<thead>
<tr>
<th>Powerop_mode Values</th>
<th>Description</th>
</tr>
</thead>
</table>
| VM_POWEROP_MODE_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 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.  
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 Start() 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. |
| VMPOWEROP_MODE_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. |
Using VmPerl

### Infotype Values

$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 50.</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>
<tr>
<td>VM_PRODINFO_VERSION_MINOR</td>
<td>The product's minor version number.</td>
</tr>
<tr>
<td>VM_PRODINFO_VERSION_REVISION</td>
<td>The product's revision number.</td>
</tr>
</tbody>
</table>

#### VM_PRODINFO Producto <XXX> Values

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

<table>
<thead>
<tr>
<th>VM_PRODINFO Producto 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>
</tbody>
</table>

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.
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 52 for an example of this procedure.


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.

<table>
<thead>
<tr>
<th>VM_PRODINFO_PLATFORM Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM_PLATFORM_VMNIX</td>
<td>The platform is the ESX Server console operating system.</td>
</tr>
<tr>
<td>VM_PLATFORM_UNKNOWN</td>
<td>The platform is unknown.</td>
</tr>
</tbody>
</table>

www.vmware.com
Using VmPerl

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 71.

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”:

```perl
$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:

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

In a Linux guest operating system:

```cmd
/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:

```cmd
/etc/vmware-tools/vmware-guestd --cmd "info-get guestinfo.name"
```
Using VmPerl

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
Using Sample VmPerl Scripts

Sample Perl 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. They are located in the SampleScripts subdirectory in the VmPerl directory.

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.

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

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) 1999-2003 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.
Using Sample VmPerl Scripts

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) 1999-2003 VMware, Inc.
#
# 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 = (C:\Program Files\VMware\VmwVmware VMware VmPerl Scripting API\perl5\site_perl\5.005",}
Using Sample VmPerl Scripts

'C:\Program Files\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 "\nThe 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);
Using Sample VmPerl Scripts

# 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.

#!/usr/bin/perl -w
#
# Copyright (C) 1999-2003 VMware, Inc.
#
BEGIN {
  if ($^O eq "MSWin32") {
    @INC = ("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::Server;
use VMware::VmPerl::ConnectParams;
use strict;
Using Sample VmPerl Scripts

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.
            ...
Using Sample VmPerl Scripts

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 $config: Error ".
    "$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.
Using Sample VmPerl Scripts

#!/usr/bin/perl -w
#
# Copyright (C) 1999-2003 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\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;

# 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",}
Using Sample VmPerl Scripts

"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;

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.
Using Sample VmPerl Scripts

#!/usr/bin/perl -w
#
# Copyright (C) 1999-2003 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 );
Using Sample VmPerl Scripts

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\n";
}

# 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\nis not powered on. Exiting.\n";
}

# 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:\n     Error $error_number: $error_string\n";
  }
  if ($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).
  }
}
Using Sample VmPerl Scripts

```perl
# 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."
} 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) 1999-2003 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";
    }
Using Sample VmPerl Scripts

# 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();
    # Destroys the virtual machine object, thus disconnecting from the virtual machine
    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";
    }
}
Using Sample VmPerl Scripts

# 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\n";
    print "one of the answers below:\n";
    for (my $i = 0; $i <= $#choices; $i++) {
        print "\t" . ($i + 1) . ". $choices[$i]\n";
    }
    print "Final answer? ";
}

# 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;
    }
}
Using Sample VmPerl Scripts

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.

```perl
#!/usr/bin/perl -w
#
# Copyright (C) 1999-2003 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\Vmperl\perl5\site_perl\5.005',
      'C:\Program Files\VMware\Vmperl\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 73.

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 51.

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) 1999-2003 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
    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.
Using Sample VmPerl Scripts

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;
}

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 \"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 71), that sets a virtual machine's IP address configuration variable in the guest operating system.
Using Sample VmPerl Scripts

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 51.

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) 1999-2003 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'};
  }
}

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

if (@ARGV ne 1  &&  @ARGV ne 4) {
  print "Usage $0: <path_to_config_file> [<server> <user> <password>]\n";
  exit(1);
}

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

# 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 $cfg_path is: \n$ip\n";

# Destroys the virtual machine object, thus disconnecting from the virtual machine.
undef $vm;
Error Codes and Event Logging
**Error Codes and Event Logging**

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 79.

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.
Error Codes and Event Logging

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 Connect() 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>
</tbody>
</table>
Error Codes and Event Logging

<table>
<thead>
<tr>
<th>VmCOM Error Code</th>
<th>VmPerl Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>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. For more information on using a Web proxy, see <a href="http://www.vmware.com/support/gsx25/doc/consles_gsx.html">www.vmware.com/support/gsx25/doc/consles_gsx.html</a>.</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
Error Codes and Event Logging

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.

   ![Event Viewer 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.
Error Codes and Event Logging

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.

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.
Error Codes and Event Logging

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.
Appendix A: vmware-cmd Utility
Appendix A: vmware-cmd Utility

Using the vmware-cmd Utility

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.

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 GSX 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).

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.
Appendix A: vmware-cmd Utility

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.

### Server Operation

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vmware-cmd -l</code> Lists the virtual machines on the local server. Unlike the other server operations, this option does not require the <code>-s</code> option.</td>
</tr>
<tr>
<td><code>vmware-cmd -s register &lt;vm-cfg-path&gt;</code> Registers a virtual machine specified by <code>&lt;vm-cfg-path&gt;</code> on the server.</td>
</tr>
<tr>
<td><code>vmware-cmd -s unregister &lt;vm-cfg-path&gt;</code> Unregisters a virtual machine specified by <code>&lt;vm-cfg-path&gt;</code> on the server.</td>
</tr>
</tbody>
</table>

### Virtual Machine Operation

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; getstate</code> 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> 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 89</code>.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; stop &lt;powerop_mode&gt;</code> 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 89</code>.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; reset &lt;powerop_mode&gt;</code> 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 89</code>.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; suspend &lt;powerop_mode&gt;</code> 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 89</code>.</td>
</tr>
</tbody>
</table>
## Appendix A: vmware-cmd Utility

<table>
<thead>
<tr>
<th>Virtual Machine Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; setconfig &lt;variable&gt;</code> <code>&lt;value&gt;</code></td>
<td>Sets a configuration variable for the virtual machine connected to the remote console.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; getconfig &lt;variable&gt;</code> <code>&lt;value&gt;</code></td>
<td>Retrieves the value for a configuration variable for the virtual machine connected to the remote console.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; setguestinfo &lt;variable&gt;</code> <code>&lt;value&gt;</code></td>
<td>Writes a GuestInfo variable into memory. The variable is discarded when the virtual machine process terminates.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; getguestinfo &lt;variable&gt;</code></td>
<td>Retrieves the value for a GuestInfo variable.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; getproductinfo &lt;prodinfo&gt;</code></td>
<td>Returns information about the product, where <code>&lt;prodinfo&gt;</code> 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).</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; connectdevice &lt;device_name&gt;</code></td>
<td>Connects the specified virtual device to a virtual machine.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; disconnectdevice &lt;device_name&gt;</code></td>
<td>Disconnects the specified virtual device from the virtual machine.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; getconfigfile</code></td>
<td>Returns a string containing the configuration file name for the virtual machine. This method fails if the virtual machine is not connected.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; getheartbeat</code></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 this service is not running.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; gettoolslastactive</code></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.</td>
</tr>
<tr>
<td><code>vmware-cmd &lt;vm-cfg-path&gt; answer</code></td>
<td>Prompts the user to answer a question for a virtual machine waiting for user input.</td>
</tr>
</tbody>
</table>
Appendix A: vmware-cmd Utility

<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
To succeed, soft power operations require the current version of VMware Tools to be installed and running in the guest operating system. | 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.cfg getstate
```

where /home/vmware/win2000.cfg is the path to the virtual machine’s configuration file.
Appendix A: vmware-cmd Utility

In a Windows guest operating system:

```bash
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:

```bash
vmware-cmd -v /home/vmware/win2000.cfg start
```

where `-v` indicates the verbose option, `/home/vmware/win2000.cfg` 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:

```bash
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.

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

where `foo.cfg` 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
Appendix A: vmware-cmd Utility

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.
Symbols
$choice 50
$connectparams 44, 46, 47
$name 50, 51
 inflicted 50, 54
$key_name 49, 56–57
$mode 48
$question 50
$vm_name 47

A
answer_question() method 50
answering a question 22, 44,
50, 52, 70–73, 85, 87–88, 92
AnswerQuestion() method 21,
22, 82
API incompatible with server 83
authd 85

C
choice 21
Choices property 17, 21, 22
content
VmCOM 14
VmPerl 44
Config property 18, 21
config.ini file 85
ConfigFileName property 18
configuration file for virtual
machine 14, 16, 18, 20, 44, 46,
47, 49, 83, 92
configuration variable 18, 49,
75–77, 77–79, 92
Connect() method 14, 15, 16,
20, 82, 83
connect() method 44, 46, 47,
83
connect_device() method 51
ConnectDevice() method 21

D
device, connecting to 21, 51,
92
device, disconnecting from 21,
51, 92
device_is_connected() method
50
DevicesConnected property 18
devName 21
DHCP lease 23, 53
disconnect_device() method
51
DisconnectDevice() method 21
disconnected virtual machine
84
disconnecting from a device 21,
51, 92

e
error condition requiring user
input 22, 44, 50, 52, 85, 87–88
error handling 82
error, VmCOM 83–84
vmErr_BADSTATE 21, 83
vmErr_BADVERSION 83
vmErr_DISCONNECT 82, 83
vmErr_INSUFFICIENT_RESOURCES 83
vmErr_INVALIDARGS 83
vmErr_INVALIDVM 83
vmErr_NEEDEDINPUT 82, 83

vmErr_NETFAIL 83
vmErr_NOACCESS 84
vmErr_NOMEM 84
vmErr_NOPROPERTY 84
vmErr_NOTCONNECTED 82, 84
vmErr_NOTSUPPORTED 84
vmErr_PROXYFAIL 84
vmErr_TIMEOUT 84
vmErr_UNSPECIFIED 84
vmErr_VMBUSY 84
vmErr_VMEXIT 84
vmErr_VMEXIT 84

error, VmPerl 83–84
VM_E_BADSTATE 51, 83
VM_E_BADVERSION 83
VM_E_DISCONNECT 83
VM_E_INSUFFICIENT_RESOURCES 83
VM_E_INVALIDARGS 83
VM_E_INVALIDVM 83
VM_E_NEEDEDINPUT 83
VM_E_NETFAIL 83
VM_E_NOACCESS 84
VM_E_NOMEM 84
VM_E_NOPROPERTY 84
VM_E_NOTCONNECTED 84
VM_E_NOTSUPPORTED 84
VM_E_PROXYFAIL 84
VM_E_TIMEOUT 84
VM_E_UNSPECIFIED 84
VM_E_VMBUSY 84
VM_E_VMEXIT 84
VM_E_VMEXIT 84

event ID 86
Event Viewer 84–88
ExecutionState property 17

G
get_choices() method 50, 52
get_config() method 49
get_config_file_name() method 49
get_execution_state() method
49
Index

get_guest_info() method 49, 57
get_heartbeat() method 50
get_hostname() method 52
get_id() method 50
get_last_error() method 46, 47, 50, 82–83
get_password() method 45
get_pending_question() method 50
get_port() method 45
get_product_info() method 50, 54
get_text() method 52
get_username() method 50
get_guest_info() method
guest operating system 19, 25–27, 51, 55–57
GuestInfo property
GuestInfo variable 25–27, 55–57, 92
H
hard power transition 23, 53, 93
heartbeat 18, 50, 67–70, 92
Heartbeat property 18
host platform 24, 54, 92
hostname 15, 45
I
Id property 22
index 50
infoType 18
input, requiring 21, 22, 44, 50, 52, 70–73, 85, 87–88, 92
installation
VmCOM 10
VmPerl 10, 11
instrv 37
insufficient memory 84
insufficient resources 16, 20, 46, 47, 83
invalid power transition 83
is_connected() method 46, 47
ISupportErrorInfo 82
Item property 17
J
JScript 32, 32–34
K
keyName 17, 26–27
L
limits 16, 20, 46, 47, 83
Linux operating system
list of virtual machines 14, 32–34, 34–37, 37–41, 44, 61–63, 63–65, 91
M
memory 21, 49, 84
memory size 18, 49
memory, values stored in 18, 49
messages 85
method, VmCOM
AnswerQuestion() 21, 22, 82
Connect() 14, 15, 16, 20, 82, 83
ConnectDevice() 21
RegisterVm() 16
Reset() 20
Start() 20
Stop() 20
Suspend() 21
UnregisterVm() 16
method, VmPerl
answer_question() 50
connect() 44, 46, 47, 83
connect_device() 51
device_is_connected() 50
disconnect_device() 51
get_choices() 50, 52
get_config() 49
get_config_file_name() 49
get_execution_state() 49
guest_info() 49, 57
get_hostname() 45
get_id() 52
get_last_error() 46, 47, 50, 82–83
get_password() 45
guest_info() 50, 54
guest_info() 52
get_tools_last_active() 50, 51
get_username() 45
is_connected() 46, 47
register_vm() 46
registered_vm_names() 44, 46
reset() 48
set_config() 49, 51
set_guest_info() 49, 56
start() 48
stop() 48
suspend() 49
unregister_vm() 46

MiniMUI Visual Basic project 10, 31
N
network failure 83
network port 45
no response 84
not enough memory 84
P
passing information between script and guest operating system 25–27, 55–57
password 15, 45
PendingQuestion property 17, 21, 22, 82
permission 84
platform 24, 54, 92
platform information 24
port 15, 45, 90
power status of a virtual machine 17, 49, 65–67, 91
power transition 23, 53, 84, 86
hard 23, 53, 93
invalid 83
soft 23, 53, 93
trysoft 23, 54, 93
powering off a virtual machine 20, 23, 48, 53, 91, 93
Index

powering on a virtual machine 20, 23, 48, 53, 91, 93
product information 18, 24, 50, 54, 92
ProductInfo property 18
property
Choices 17, 21, 22
Config 18, 21
ConfigFileName 18
Count 17
DevicesConnected 18
ExecutionState 17
GuestInfo 17
Heartbeat 18
Id 22
Item 17
PendingQuestion 17, 21, 22, 82
ProductInfo 18
RegisteredVmNames 16
Text 22
ToolsLastActive 18, 19
proxy 84
proxy failure 84
Q
question 44, 50, 52, 70–73, 85, 87–88, 92
R
reconnect to a virtual machine 20
redo log 34
register_vm() method 46
registeredVmNames() method 44, 46
RegisteredVmNames property 16
registering virtual machine 46, 84, 91
RegisterVm() method 16
Reset() method 20
reset() method 48
resetting a virtual machine 20, 23, 48, 53, 91, 93
resuming a suspended machine 20, 48, 91, 93
S
Sample scripts, VmCOM 10, 30–41
connecting to server and listing virtual machines 32–34, 34–37
listing and starting virtual machines 37–41
sample scripts, VmPerl 60–79
answering question for stuck virtual machine 70–73
determining power status 65–67
listing and starting virtual machines 63–65
listing virtual machines 61–63
monitoring virtual machine heartbeat 67–70
retrieving a configuration variable 77–79
setting a configuration variable 75–77
suspending a virtual machine 74–75
sample scripts, VmPerl 11
script 25–27, 55–57
security 15, 45, 84
server
connecting to 14, 16, 20, 32–34, 34–37, 44, 46, 47
incompatible with API 83
security 15, 45
virtual machines on 14
VmServerCtl 14, 16, 82
VMware::VmPerl::Server 44, 46
served 85
set_config() method 49, 51
set_guest_info() method 49, 56
shared variables 25–27, 55–57
simultaneous connections 16, 20, 46, 47
soft power transition 23, 53, 93
svrany 37
Start() method 20
start() method 48
starting a virtual machine 20, 23, 48, 53, 91, 93
state of virtual machine 17, 49, 91
Stop() method 20
stop() method 48
stopping a virtual machine 20, 23, 48, 53, 91, 93
string
$key_name 49
keyName 17
Suspend() method 21
suspend() method 49
suspended machine, resuming 20, 48, 91, 93
suspending a virtual machine 21, 23, 49, 53, 74–75, 91, 93
T
TCP port 15, 45
Text property 22
time out 84
time out during suspension 21, 49
ToolsLastActive property 18, 19
trysoft power transition 23, 54, 93
U
undoable disk 34
uninstalling VmPerl 11
unregister_vm() method 46
UnregisterVm() method 16
user_input 21, 22, 44, 50, 52, 70–73, 85, 87–88, 92
user name 15, 45, 90
V
variable 18, 25–27, 49, 55–57, 92
VBScript 32, 34–37, 37–41
virtual device 18, 21, 50, 51, 92
virtual machine 44, 47–51, 55
configuration file 14, 16, 18, 20, 44, 46, 47, 49, 83, 92

www.vmware.com
Index

get_id() method 52
get_last_error() method 46, 47, 50, 82-83
get_password() method 45
get_pending_question() method 50
get_port() method 45
get_product_info() method 50, 54
get_text() method 52
get_tools_last_active() method 50, 51
get_username() method 45
is_connected() method 46, 47
register_vm() method 46
registered_vm_names() method 44, 46
reset() method 48
sample scripts 11, 60-79
set_config() method 49, 51
set_guest_info() method 49, 56
start() method 48
stop() method 48
suspend() method 49
unregister_vm() method 46
use with Linux operating system 8
use with Windows operating system 8
VmPlatform 24
VmPowerOpMode 20-21, 23
vmProdInfo_Platform 24
vmProdInfo_Product 24
VmProdInfoType 18, 24
VmProduct 24
VmQuestion 21, 22, 82
VmQuestion.Choices 17
VmServerCtl 14, 16, 82
VmServerCtl.RegisteredVmNames 17
VMware Tools 18, 19, 25-27, 50, 51, 55-57, 92
VMware::VmPerl::Connect-
Params 44, 45
VMware::VmPerl::Question 44, 50, 52
VMware::VmPerl::Server 44, 46
VMware::VmPerl::VM 44, 47-51, 55
vmware-cmd
options 90
server operations 90-91
virtual machine operations 91-92
vmware-control 90
W
waiting for user input 21, 22, 34-37, 44, 50, 52, 70-73, 85, 87-88, 92
Web proxy 84
Windows operating system 10
installing Scripting APIs on
Windows Script File 32, 34, 37, 41