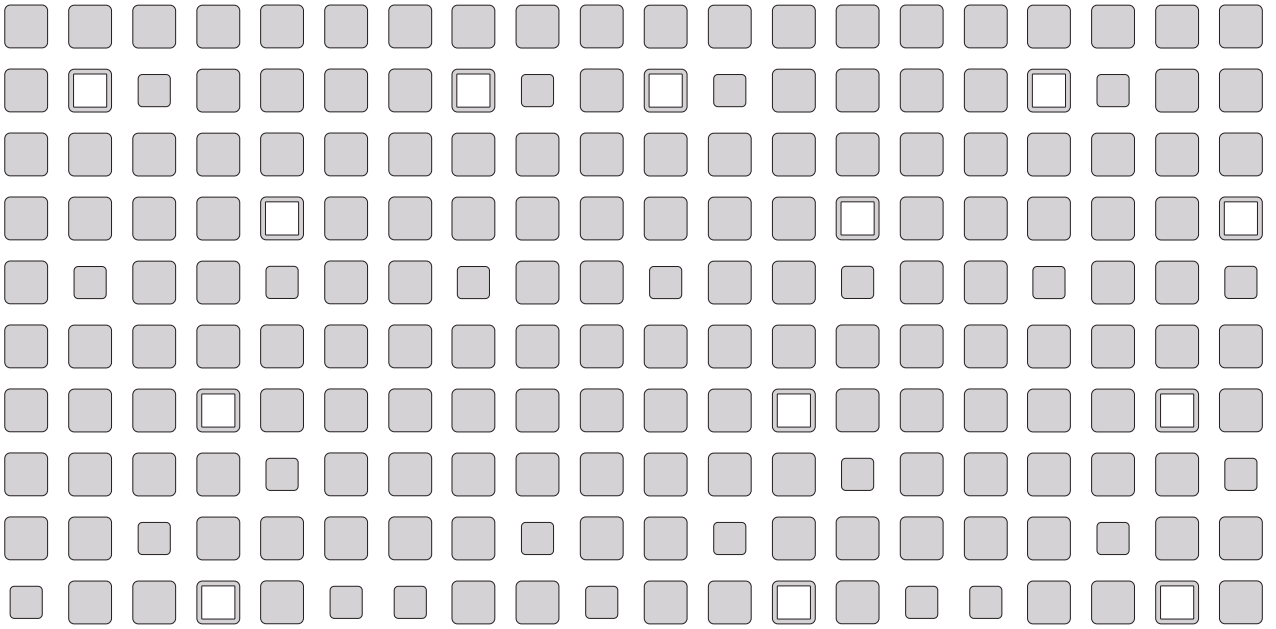
VERSION 2.1.5

VMware Scripting API

User's Manual



vmware®

VMware, Inc.

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

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

Revision 20050214 Item: AP1-ENG-Q304-005

Table of Contents

CHAPTER 1

Introduction

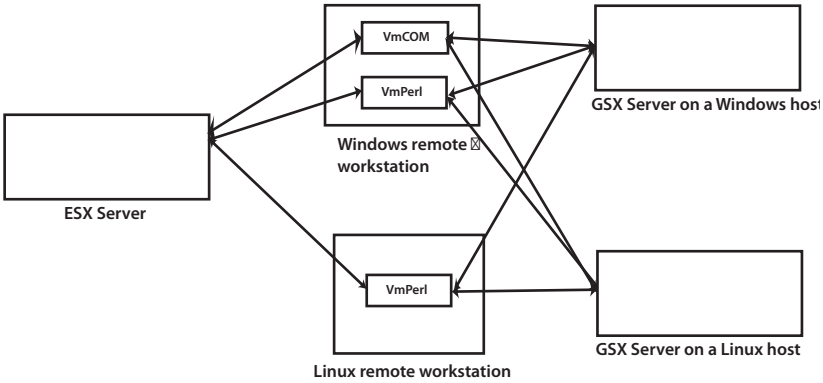
This release of VMware™ scripting APIs version 2.1.5 comprises two components: VmCOM and VmPerl.

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

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

Introducing the VMware Scripting APIs

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



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

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

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

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

Supported Products

We support the installation of the VmCOM and VmPerl Scripting APIs on VMware™ GSX Server™ 2.x, GSX Server 3.x and VMware ESX Server™ 2.x. For more information see:

- *www.vmware.com/support/gsx2/doc* (GSX Server 2.0)
- *www.vmware.com/support/gsx25/doc* (GSX Server 2.5)
- *www.vmware.com/support/gsx3/doc* (GSX Server 3.x)
- *www.vmware.com/support/esx2/doc* (ESX Server 2.0 and ESX Server 2.0.1)
- *www.vmware.com/support/esx21/doc* (ESX Server 2.1)
- *www.vmware.com/support/esx25/doc* (ESX Server 2.5)

Intended Audience

This manual is written for programmers that are familiar with either the Perl language or the Component Object Model (COM) interface for programming languages. Readers of this manual should be comfortable with developing system administration and system monitoring programs and general debugging techniques. In addition, developers who use this manual should be familiar with the operation and management of VMware GSX Server, the host operating system used for this application, and VMware ESX Server.

Getting Support from VMware

See *www.vmware.com/support/developer* or full details on the VMware Scripting APIs support policy.

Using the VMware Scripting APIs

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

Note: We support a maximum of two scripting connections to a virtual machine at a time.

Each VmCOM object and Perl module is described in the following chapters and includes the methods, the properties, and their usage. In addition, sample scripts and lists of error codes are provided. For VmCOM sample scripts, see Using Sample VmCOM Programs on page 35 and for VmPerl scripts, see Using Sample VmPerl Scripts on page 69. For the list of error codes, see Error Codes on page 100.

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

Installing the VMware Scripting API

GSX Server

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

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

ESX Server

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

The links are:

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

Installing the VMware Scripting API on a Windows Machine

You have a choice of installing either the VmCOM or the VmPerl Scripting API.

1. Choose **Start** > **Run** and browse to the directory where you saved the downloaded installer file (the name is similar to `VMware-VmPERLAPI-<xxxx>.exe` or `VMware-VmCOMAPI-<xxxx>.exe`, where `<xxxx>` is a series of numbers representing the version and build numbers).
2. The installer starts. Click **Next**.
3. Acknowledge the end user license agreement (EULA). Select **Yes, I accept the terms in the license agreement**, then click **Next**.
4. Choose the directory in which to install the Scripting API. To install it in a directory other than the default, click **Change** and browse to your directory of choice. If the directory does not exist, the installer creates it for you. Click **Next**.

Note: Windows and the Microsoft Installer limit the path length to 255 characters for a path to a folder on a local drive and 240 characters for a path to a folder on a mapped or shared drive. If the path to the Scripting API program folder exceeds this limit, an error message appears. You must select or enter a shorter path.

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

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

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

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

Installing VmPerl Scripting API on a Linux Machine

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

1. Copy the VmPerl package to the machine on which you want to run the VMware Scripting API.
2. In a terminal window, become root so you can carry out the installation.
3. Untar the package

   ```
   tar xzf VMware-VmPERLAPI-v.v.v-####.tar.gz
   ```

   where `v.v.v` is the specific version number and `####` is the build number.
4. Change to the directory where you expanded the package.

   ```
   cd vmware-api-distrib
   ```
5. Run the install script.

   ```
   ./vmware-install.pl
   ```
6. Press Enter to read the end user license agreement (EULA). You may page through it by pressing the space bar. If the `Do you accept?` prompt doesn't appear, press `Q` to get to the next prompt.
7. Choose the directory to install the VmPerl executable files or accept the default location.

This directory includes the uninstall script for the VmPerl API.

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

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

This completes the VmPerl API installation.

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

```
<executable_files_directory>/vmware-uninstall-api.pl
```

CHAPTER 2

Using VmCOM

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

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

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

Unlike the `VmServerCtl` object, `VmCtl.Connect()` also takes a string specifying the configuration file name of the virtual machine that will be connected.

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

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

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

VmCOM Objects

The VmCOM component provides the following objects:

- VmConnectParams
- VmServerCtl
- VmCollection
- VmCtl
- VmQuestion

VmConnectParams

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

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

Property Name	Property Type	Access Type	Description
Hostname	string	GET/PUT	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.
Port	integer	GET/PUT	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.
Username	string	GET/PUT	Retrieves and sets the name of a user on the server.
Password	string	GET/PUT	Retrieves and sets the user's password on the server.

VmServerCtl

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

Property

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

Note: The Resource property applies only to ESX Server.

Property Name	Property Type	Access Type	Description
RegisteredVmNames	string	GET	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.
Resource(<variable_name>) **Note:** This property applies only to ESX Server.	string	GET/PUT	Accesses the value of a ESX Server system resource variable identified by the string <variable_name>.The property throws an error if it accesses an undefined system variable. See VMware ESX Server System Resource Variables on page 120 for a list of server system variables.

Methods

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

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

Method	Description
object.Connect(<params>)	The `Connect()` method connects the object to a VMware GSX Server or VMware ESX Server where `params` is a `VmConnectParams` object that specifies the system and user information. There is no method to disconnect from a server. To reconnect to a different server, destroy the `VmServerCtl` object, create a new one, then call its `Connect()` method. The total number of connected `VmCtl` and `VmServerCtl` objects cannot exceed 62. 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.
object.RegisterVm(<vmName>)	The `RegisterVm` method registers a virtual machine on a server where `vmName` is a string specifying the virtual machine's configuration file name.
object.UnregisterVm(<vmName>)	The `UnRegisterVm` method unregisters a virtual machine from a server where `vmName` is a string specifying the virtual machine's configuration file name.

VmCollection

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

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

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

Property Name	Property Type	Access Type	Description
Count	integer	GET	Gets the number of elements in the collection.
Item(<index_as_integer>)	string	GET	Gets the element at the specified index.

VmCtl

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

Properties

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

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

Property Name	Property Type	Access Type	Description
ExecutionState	VmExecutionState	GET	Current state of the virtual machine; powered on, powered off, suspended, or stuck. For more information on VmExecutionState, see VmExecutionState on page 29.
PendingQuestion	VmQuestion	GET	Returns a VmQuestion object if the virtual machine is currently in the vmExecutionState_Stuck state. Otherwise, an error is thrown
GuestInfo(keyName)	string	GET/PUT	Accesses a shared variable identified by the string `keyName`. For additional information, see Using VmCOM to Pass User-Defined Information Between a Running Guest Operating System and a Script on page 32.
Config(keyName)	string	GET/PUT	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.

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

Property Name	Property Type	Access Type	Description
Capabilities **Note:** This property applies only to ESX Server.	integer	GET	Returns the access permissions for the current user. This number is a bit vector, where 4=read, 2=write, and 1=execute. For a user with all three permissions, a value of 7 is returned when this property is used in a script.
RemoteConnections **Note:** This property applies only to ESX Server.	integer	GET	Returns the number of remotely connected users. This value includes the number of remote consoles, Scripting APIs, and Web-based management interface connections to the virtual machine.

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

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

Methods

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

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

Except where noted otherwise, these methods are synchronous; the method does not return until it finishes its operation, fails or times out. Most operations time out after 2 minutes, except for power operations, which time out after 4 minutes.

Note: Two methods, `object.AddRedo` and `object.Commit`, that are listed in the following table, apply only to ESX Server. Similarly, the `object.SetRunAsUser` and `object.RunAsUser` methods apply only to GSX Server.

Method	Description
object.Connect(<params>, <vmName>)	The `Connect()` method establishes a connection with a virtual machine where `params` is a `VmConnectParams` object that specifies the system and user information and `vmName` is a string specifying the virtual machine's configuration file name. You should use this as the first method invoked on a `VmCtl` object. You must first activate the `VmCtl` object by calling its `Connect()` 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 `VmCtl` object, create a new one, then call its `Connect()` method. The total number of connected `VmCtl` and `VmServerCtl` objects cannot exceed 62. 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.
object.Start(<mode>)	The `Start()` method powers on a previously powered-off virtual machine or resumes a suspended virtual machine, where `mode` is a `VmPowerOpMode` object that specifies the Start operation's behavior. For more information, see VmPowerOpMode on page 29. If the virtual machine is powered off, then it is powered on. If it is suspended, this method resumes the virtual machine. If the virtual machine is in any other state, the `Start()` method fails and throws an error.
object.Stop(<mode>)	The `Stop()` method shuts down and powers off a virtual machine where `mode` is a `VmPowerOpMode` object that specifies the Stop operation's behavior. For more information, see VmPowerOpMode on page 29. This method always fails if the virtual machine is not in the `vmExecutionState_On` state.
object.Reset(<mode>)	The `Reset()` method shuts down, then reboots a virtual machine where `mode` is a `VmPowerOpMode` object that specifies the operation's behavior. For more information, see VmPowerOpMode on page 29. This method always fails if the virtual machine is not in the `vmExecutionState_On` state.

Method	Description
object.Suspend(<mode>)	The `Suspend()` method suspends a virtual machine where `mode` is a `VmPowerOpMode` object that specifies the Suspend operation's behavior. It saves the current state of the virtual machine to a suspend file. For more information, see VmPowerOpMode on page 29. This method always fails if the virtual machine is not in the `vmExecutionState_On` state.
object.AddRedo(<diskDevName>) **Note:** This method applies only to ESX Server.	This method adds a redo log to a running virtual SCSI disk specified by <diskDevName>, that is associated with the virtual machine specified by the `VmCtl` object. Changes made to the virtual disk accumulate in the new redo log. This disk must be a ESX Server virtual disk stored on a VMFS volume. The virtual disk can be in persistent, undoable or append mode. The redo log for a virtual disk in persistent mode uses the file name of the virtual disk with `.REDO` appended to it (for example, if the disk is called, `vm.dsk`, the redo log is called `vm.dsk.REDO`). A virtual disk in undoable or append mode already has a redo log associated with it, so the new redo log you create is called `vm.dsk.REDO.REDO`, whose parent is the existing redo log, `vm.dsk.REDO`. This method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk already has two redo logs associated with it. If you add a redo log using the `AddRedo()` method, but do not commit your changes with the `Commit()` method, then the redo is automatically committed when the virtual machine is powered off.

Method	Description
object.Commit(<diskDevName>, <level>, <freeze>, <wait>) **Note:** This method applies only to ESX Server.	This method commits the changes in a redo log to a running virtual SCSI disk specified by <diskDevName> that is associated with the virtual machine specified by `$vm`. <level> can be `0` or `1`. When <level> is `0`, there can be one or two redo logs associated with the disk. If <level> is `0`, then the top-most redo log (the redo log being modified) is committed to its parent. For example, if there is currently only the disk `vm.dsk` with a single redo log `vm.dsk.REDO`, then the changes in `vm.dsk.REDO` are committed to `vm.dsk`. If a second REDO log `vm.dsk.REDO.REDO` has been added, then the changes in `vm.dsk.REDO.REDO` are committed to `vm.dsk.REDO`. <level> can be `1` only when there are two redo logs associated with the disk, `vm.dsk.REDO` and `vm.dsk.REDO.REDO`. When <level> is `1`, the changes in the next-to-top REDO log, `vm.dsk.REDO`, are committed to `vm.dsk`. In this case, the virtual machine is not frozen while the redo log is being committed. Also, when the log is committed, `vm.dsk.REDO.REDO` is renamed to `vm.dsk.REDO`. <freeze> can be `0` or `1`. If <freeze> is `0`, then the virtual machine is not frozen when changes are committed, though it runs more slowly. If <freeze> is `1`, then the virtual machine is frozen until the commit operation finishes. If <level> is `0`, then the virtual machine must be frozen when changes are committed and <freeze> is ignored. <wait> can be `0` or `1`. If <wait> is `0`, then the method returns as soon as the commit begins. If <wait> is `1`, then the method does not return until the commit completes. The method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk currently has no redo logs.

Method	Description
object.AnswerQuestion(<question>, <choice>)	The `AnswerQuestion()` method replies to a question where `question` is a `VmQuestion` object that represents the question that requires an answer and `choice` represents the index of the selected answer to the question. The index is an integer and the first choice's index is always 1 (one). The second choice's index is 2, and so on. When a virtual machine is in the `vmExecutionState_Stuck` state and requires user input to continue, use this method to answer the current question or dismiss the current error message. First, get a `VmQuestion` object from `VmCtl.PendingQuestion`. You can retrieve the possible choices and their respective indices from the `VmQuestion.Choices` property. Then, use the `AnswerQuestion` method to answer the question.

object.ConnectDevice(<devName>)	The `ConnectDevice()` method sets a virtual device to the connected state where `devName` is a string that identifies the virtual device you want to connect. The virtual machine must be powered on for this method to succeed, otherwise a vmErr_BADSTATE error is returned.
	Use the `Config()` property to set configuration parameters relevant to the virtual device before calling the `ConnectDevice()` method. The following code example illustrates connecting a virtual drive to a CD image file:
	`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.
object.SetRunAsUser(<myuseraccount>, <mypassword>) **Note:** This method applies only to GSX Server 3.1.	Runs the virtual machine as the user specified by the <myuseraccount> and <mypassword>.
object.RunAsUser **Note:** This method applies only to GSX Server 3.1.	Returns the name of the user running the virtual machine.

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:

Property Name	Property Type	Access Type	Description
Text	string	GET	Gets the question text.
Choices	string	GET	Gets a `VmCollection` of strings representing a list of possible answers to the question.
Id	integer	GET	Gets an integer used internally by the VmCOM component to identify the question.

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:

VmExecutionState Values	Description
vmExecutionState_On	The virtual machine is powered on.
vmExecutionState_Off	The virtual machine is powered off.
vmExecutionState_Suspended	The virtual machine is suspended.
vmExecutionState_Stuck	The virtual machine requires user input. The user must answer a question or dismiss an error.
vmExecutionState_Unknown	The virtual machine is in an unknown state.

VmPowerOpMode

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

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

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

VmPowerOpMode Values	Description
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 — Starts or resumes a virtual machine without running any scripts; a standard power on or resume. Stop, reset or suspend — Immediately and unconditionally powers off, resets, or suspends the virtual machine.
vmPowerOpMode_TrySoft	First attempts to perform the power transition operation with vmPowerOpMode_Soft. If this fails, the same operation is performed with vmPowerOpMode_Hard.

VmProdInfoType

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

VmProdInfoType Values	Description
vmProdInfo_Product	The VMware product type is returned as VmProduct. For more information on VmProduct, see the following section.
vmProdInfo_Platform	The host platform type is returned as VmPlatform. For more information on VmPlatform, see VmPlatform on page 31.
vmProdInfo_Build	The product's build number.
vmProdInfo_Version_Major	The product's major version number.

VmProdInfoType Values	Description
vmProdInfo_Version_Minor	The product's minor version number.
vmProdInfo_Version_Revision	The product's revision number.

VmProduct

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

VmProduct Values	Description
vmProduct_WS	The product is VMware Workstation.
vmProduct_GSX	The product is VMware GSX Server
vmProduct_ESX	The product is VMware ESX Server.
vmProduct_UNKNOWN	The product type is unknown.

VmPlatform

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

VmPlatform Values	Description
vmPlatform_WINDOWS	The host platform is a Microsoft Windows operating system.
vmPlatorm_LINUX	The host platform is a Linux operating system.
vmPlatform_VMNIX	The host platform is the ESX Server console operating system.
vmPlatform_UNKNOWN	The host platform is unknown.

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

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

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

See your server documentation for more information about VMware Tools.

GuestInfo Variables

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

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

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

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

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

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

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

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

```
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:

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

In a Linux guest operating system:

```
/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:

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

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

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

In a Windows guest operating system:

```
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")
```

CHAPTER 3

Using 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:

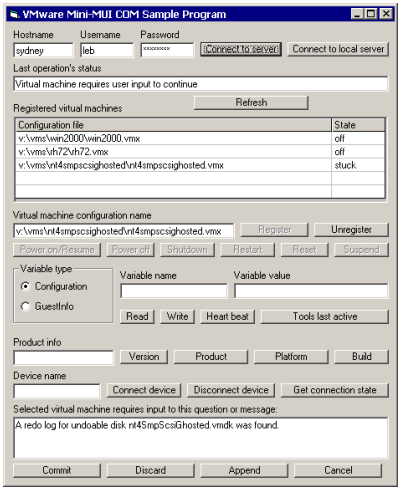
MiniMUI Visual Basic Sample Program

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

The source code demonstrates how to:

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

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



JScript and VBScript Sample Programs

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

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

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

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

where `<n>` is the sample program number.

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

JScript Sample Program 1

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

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

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

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

```
//  This script displays the virtual machines on the local server.
//  It prints the configuration file path and current execution
//  state of each VM. If a VM is in the stuck state, the current
//  question and its choices are also printed.
//
//  Instructions for Windows 2000 and later operating systems:
//
//  - save the contents of this file to a file named 'sample1.js'
//    unless it's already named that way
//
//  - there should be an accompanying file named 'sample1.wsf'
//    It is placed in the same directory as this file during
//    product installation. This file is responsible for setting
//    up the Windows Script Host environment and loading the
//    VmCOM type library, thereby enabling this script to
//    reference symbolic constants such as vmExecutionState_On
//
//  - in a command line window, type:
//    cscript //nologo sample1.wsf
//

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

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

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

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

   if (execStateString == "STUCK") {

      question = vm.PendingQuestion;
      str += "  pending question='" + question.text + "' choices=";

      choices = question.choices
      for (i = 1; i <= choices.count; i ++) {
         str += "[" + choices(i) + "] ";
      }
   }
   WScript.Echo(str);
```

```
}

function State2Str(vm) {
   switch (vm.ExecutionState) {
      case vmExecutionState_On:
         return "ON";
         break;
      case vmExecutionState_Off:
         return "OFF";
         break;
      case vmExecutionState_Suspended:
         return "SUSPENDED";
         break;
      case vmExecutionState_Stuck:
         return "STUCK";
         break;
      default:
         return "UNKNOWN";
         break;
   }
}
```

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

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

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

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

VBScript Sample Program 2

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

In addition, sample program 2 also illustrates how to handle a virtual machine that is waiting for input to a question (that is, the virtual machine is in the `vmExecutionState_Stuck` state). For example, if a virtual machine is configured with an undoable disk and a redo log is found, this

script automatically keeps the redo log during a shutdown operation or appends the redo log during a power-on operation.

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

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

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

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

```
'
'   - in a command line window, type:
'     cscript //nologo sample2.wsf
'


Set cp = CreateObject("VmCOM.VmConnectParams")
Set server = CreateObject("VmCOM.VmServerCtl")

server.Connect cp
Set vmCollection = server.RegisteredVmNames

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

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

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

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

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

```
   end if
   WScript.Echo(s)
next

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

The source for the sample program 2 accompanying Windows Script File is in the SampleScripts folder in the VmCOM Scripting API directory.

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

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

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

VBScript Sample Program 3

This VBScript sample program lists, then starts locally registered virtual machines that are not already running on a server. This script powers on powered-off virtual machines and resumes suspended virtual machines that have the line "autostart=true" in their configuration files.

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

You can use a script like the following to start selected virtual machines automatically after a server boots. However, this script must be configured as a service for it to run without requiring a login from a user.

Tools exist that allow any application, including a script, to run as a service. One example is the `instsrv` and `srvany` programs from the Microsoft Windows 2000 Resource Kit. If you use `srvany` to implement the service, then configure your service to launch the `cscript` program. Set the program's argument to the path of the script's `.wsf` file. Refer to the Microsoft Windows 2000 Resource Kit documentation for more details. If you choose to use a different tool, then refer to your specific tool's documentation to configure the script to run as a service.

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

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

```
'
'  VmCOM VBScript Sample Program 3
'  Copyright (c) 1998-2004 VMware, Inc.
'  .
'  .
'  .
'  This program is for educational purposes only.
'  It is not to be used in production environments.
'
'  Description:
'
'  This script gets a list of virtual machines registered on
'  the local server.  It attempts to power-on each VM that
'  is not already running and has a line in the config file:
'
'  autostart=true
'
'
'  Instructions for Windows 2000 and Windows XP host:
'
'  - save the contents of this file to a file named 'sample3.vbs'
'
'  - there should be an accompanying file named 'sample3.wsf'
'    It is placed in the same directory as this file during
'    product installation. This file is responsible for setting
'    up the Windows Script Host environment and loading the
'    VmCOM type library, thereby enabling this script to
'    reference symbolic constants such as vmExecutionState_On
'
'  - in a command line window, type:
'    cscript //nologo sample3.wsf
'
```

```
Set connect_params = CreateObject("VmCOM.VmConnectParams")

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

Set vm_server = CreateObject("VmCOM.VmServerCtl")

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

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

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

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

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

```
   vm.Connect connect_params, config

   if Err.Number <> 0 then
     WScript.Echo "Could not connect to VM " & config & ": " & Err.Description
      Err.Clear
   else
      ' Check that the VM should be started automatically
      auto_start = vm.Config("autostart")
      if Err.Number <> 0 then
         if Err.Number <> vmErr_NOPROPERTY then
              WScript.Echo "Could not read autostart variable: " &
Err.Number & ": " & Err.Description
         else
          WScript.Echo "This VM is not configured for autostart: " & config
         end if
         Err.Clear
      else
         if auto_start = "true" or auto_start = "TRUE" then
            ' Check that the VM is powered off

            power_state = vm.ExecutionState
            if Err.Number <> 0 then
                  WScript.Echo "Error getting execution state: " &
Err.Number & ": " & Err.Description
              Err.Clear
            else
                if power_state = vmExecutionState_Off or power_state =
vmExecutionState_Suspended then
                 WScript.Echo "Powering on " & config
                 vm.Start(vmPowerOpMode_Soft)
                 if Err.Number <> 0 then
                  WScript.Echo "Error powering on " & config & ": " & Err.Description
                    Err.Clear
                 else
                  ' Wait between starting up VMs to smooth out the load on the server
                     WScript.Sleep 5000
                 end if
               end if
            end if
          end if
      end if
   end if

next
```

The source for the sample program 3 accompanying Windows Script File is in the SampleScripts folder in the VmCOM Scripting API directory.

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

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

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

CHAPTER 4

Using VmPerl

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

The VmPerl API consists of four modules or packages:

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

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

A VMware::VmPerl::Server object represents a server and exports server-level services, such as virtual machine enumeration and registration. A VMware::VmPerl::VM object represents a virtual

machine on a particular server and provides virtual machine specific methods including power operations. You activate the VMware::VmPerl::Server or VMware::VmPerl::VM object by calling its `connect()` method before accessing any other method.

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

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

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

VMware::VmPerl::ConnectParams

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

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

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

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

VMware::VmPerl::Server

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

Method	Description
$server->connect($connectparams) Returns the defined value on success or `undef` (undefined value) on failure.	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.
$server->get_last_error() Returns the error code and descriptive string.	Gets details about the last error that occurred in an array of form `[$error_num, $error_string]`.
$server->is_connected() Returns the defined value on success or `undef` (undefined value) on failure (if the server is not connected or if there is a failure). You can use `$vm->get_last_error` to determine if an error occurred or if the server is not connected.	Use this method to determine whether or not a connection exists to the server specified by `$server`.

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

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

Method	Description
$server->registered_vm_names() Returns a list of virtual machine configuration file names, an empty list (if no virtual machines are registered or if there is a failure). You can use `$vm->get_last_error` to determine if an error occurred or there are no registered virtual machines.	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.
$server->register_vm($vm_name) Returns the defined value on success or `undef` (undefined value) on failure.	Registers a virtual machine on a server where `$vm_name` is a string specifying the virtual machine's configuration file name.

Method	Description
$server->unregister_vm($vm_name) Returns the defined value on success or `undef` (undefined value) on failure.	Unregisters a virtual machine from a server where `$vm_name` is a string specifying the virtual machine's configuration file name.
$server->get_resource ("system.<variable_name>") Returns the defined value on success or `undef` (undefined value) on failure. $server->set_resource ("system.<variable_name>, <value>") Returns the defined value on success or `undef` (undefined value) on failure. **Note:** These methods apply only to ESX Server.	Gets or sets the value of the ESX Server system resource variable specified by system.<variable_name>. For a list of ESX Server system variables, see VMware ESX Server System Resource Variables on page 120.

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.

Method	Description
$vm->connect($connectparams, $vm_name) Returns the defined value on success or `undef` (undefined value) on failure.	Establishes a connection with a virtual machine using the specified parameters where `$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.
$vm->get_last_error() Returns the error code and descriptive string.	Gets details about the last error that occurred in an array of form `[$error_num, $error_string]`.
$vm->is_connected() Returns the defined value on success or `undef` (undefined value) on failure (if the virtual machine is not connected or if there is a failure). You can use `$vm->get_last_error` to determine if an error occurred or if the virtual machine is not 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()`.

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

Method	Description
$vm->start($mode) Returns the defined value on success or `undef` (undefined value) on failure.	Powers on a previously powered-off virtual machine or resumes a suspended virtual machine where `$mode` specifies the operation's behavior based on the value of the VMware::VmPerl::VM_POWEROP_MODE_<XXX> where <XXX> is HARD, SOFT, or TRYSOFT. If `$mode` is not specified, the default mode is VM_POWEROP_MODE_SOFT. For more information, see VM_POWEROP_MODE_<XXX> Values on page 63. **Note:** If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM_POWEROP_MODE_HARD as the mode or the operation will fail. f 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 `start()` method fails and throws an error.
$vm->stop($mode) Returns the defined value on success or `undef` (undefined value) on failure.	Shuts down and powers off a virtual machine where `$mode` specifies the operation's behavior based on the value of the VMware::VmPerl::VM_POWEROP_MODE_<XXX> where <XXX> is HARD, SOFT, or TRYSOFT. If `$mode` is not specified, the default mode is VM_POWEROP_MODE_SOFT. For more information, see VM_POWEROP_MODE_<XXX> Values on page 63. **Note:** If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM_POWEROP_MODE_HARD as the mode or the operation will fail. This method always fails if the virtual machine is not in the VM_EXECUTION_STATE_ON state.
$vm->reset($mode) Returns the defined value on success or `undef` (undefined value) on failure.	Shuts down, then reboots a virtual machine where `$mode` specifies the operation's behavior based on the value of the VMware::VmPerl::VM_POWEROP_MODE_<XXX> where <XXX> is HARD, SOFT, or TRYSOFT. If `$mode` is not specified, the default mode is VM_POWEROP_MODE_SOFT. See VM_POWEROP_MODE_<XXX> Values on page 63. **Note:** If you are connecting to GSX Server 1.x or ESX Server 1.x, then you must specify VMware::VmPerl::VM_POWEROP_MODE_HARD as the mode or the operation will fail. This method always fails if the virtual machine is not in the VM_EXECUTION_STATE_ON state.

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

Method	Description
$vm->commit($disk, $level, $freeze, $wait) Returns the defined value on success or `undef` (undefined value) on failure. **Note:** This method applies only to ESX Server.	This method commits the changes in a redo log to a running virtual SCSI disk specified by `$disk` that is associated with the virtual machine specified by `$vm`. `$level` can be `0` or `1`. When `$level` is `0`, there can be one or two redo logs associated with the disk. If `$level` is `0`, then the top-most redo log (the redo log being modified) is committed to its parent. For example, if there is currently only the disk `vm.dsk` with a single redo log `vm.dsk.REDO`, then the changes in `vm.dsk.REDO` are committed to `vm.dsk`. If a second REDO log `vm.dsk.REDO.REDO` has been added, then the changes in `vm.dsk.REDO.REDO` are committed to `vm.dsk.REDO`. `$level` can be `1` only when there are two redo logs associated with the disk, `vm.dsk.REDO` and `vm.dsk.REDO.REDO`. When `$level` is `1`, the changes in the next-to-top REDO log, `vm.dsk.REDO`, are committed to `vm.dsk`. In this case, the virtual machine is not frozen while the redo log is being committed. Also, when the log is committed, `vm.dsk.REDO.REDO` is renamed to `vm.dsk.REDO`. `$freeze` can be `0` or `1`. If `$freeze` is `0`, then the virtual machine is not frozen when changes are committed, though it runs more slowly. If `$freeze` is `1`, then the virtual machine is frozen until the commit operation finishes. If `$level` is `0`, then the virtual machine must be frozen when changes are committed and `$freeze` is ignored. `$wait` can be `0` or `1`. If `$wait` is `0`, then the method returns as soon as the commit begins. If `$wait` is `1`, then the method does not return until the commit completes. The method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk currently has no redo logs.
$vm->get_connected_users() Returns the defined value on success or an empty list `undef` (undefined value) on failure.	Returns a list of local and remote connected users, and their IP addresses. This list includes remote console connections, API connections, and Web-based management interface connections to the specified virtual machine.
$vm->get_execution_state() Returns the defined value on success or `undef` (undefined value) on failure.	Returns the virtual machine's current state: powered on, powered off, suspended, or stuck. For a list of the execution states, see VM_EXECUTION_STATE_<XXX> Values on page 63.
$vm->get_guest_info($key_name) Returns the defined value on success or `undef` (undefined value) on failure. $vm->set_guest_info($key_name, $value) Returns the defined value on success or `undef` (undefined value) on failure.	It accesses a shared variable identified by the string `$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 66.

Method	Description
$vm->get_config_file_name() Returns the defined value on success or `undef` (undefined value) on failure.	Returns a string containing the configuration file name for the virtual machine. This method fails if the VMware::VmPerl::VM object is not connected.
$vm->get_config($key_name) Returns the defined value on success or `undef` (undefined value) on failure. $vm->set_config($key_name, $value) Returns the defined value on success or `undef` (undefined value) on failure.	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.
$vm->get_product_info($infotype) Returns the defined value on success or `undef` (undefined value) on failure.	Gets information about the product. For additional information, see Infotype Values on page 64.
$vm->get_heartbeat() Returns the defined value on success or `undef` (undefined value) on failure.	Returns the current heartbeat count generated by the VMware Tools service running in the guest operating system. The count is initialized to zero when the virtual machine is powered on. The heartbeat count is typically incremented at least once per second when the VMware Tools service is running under light load conditions. The count stays constant if the service is not running.
$vm->get_tools_last_active() Returns the defined value on success or `undef` (undefined value) on failure.	Returns an integer indicating how much time has passed, in seconds, since the last heartbeat was detected from the VMware Tools service. This value is initialized to zero when the virtual machine powers on. It stays at zero until the first heartbeat is detected, after which the value is always greater than zero until the virtual machine is power-cycled again. For additional information, see Additional Information on get_tools_last_active on page 60.
$vm->get_pending_question() Returns the defined value on success or `undef` (undefined value) on failure.	Returns a Vmware::VmPerl::Question object if the virtual machine is currently in the VM_EXECUTION_STATE_STUCK state. Use $question->get_text() to retrieve the actual question text. For additional information, see VMware::VmPerl::Question on page 62.

Method	Description
$vm->answer_question($question, $choice) Returns the defined value on success or `undef` (undefined value) on failure.	Replies to a question where `$question` represents the question and `$choice` represents the index of the selected answer to the question. The index is a number associated with an answer. The first choice's index is always 0. The second choice's index is 1, and so on. Use this method to answer the current question or dismiss the current error message when a virtual machine is in the VM_EXECUTION_STATE_STUCK state and requires user input to continue. First, get a VMware::VmPerl::Question object from the VMware::VmPerl::VM object's `get_pending_question()` method. You can retrieve the possible choices and their respective indices from the VMware::VmPerl::Question object's `get_choices()` method. Then, use the `answer_question()` method to answer the question.
$vm->device_is_connected($dev_name) Returns the defined value on success or false on failure (if the device is not connected or if there is a failure). You can use `$vm->get_last_error` to determine if an error occurred or if the device is not connected.	Determines the connection state where `$dev_name` identifies the virtual device.

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

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

Method	Description
$vm->get_resource ("<variable_name>") Returns the defined value on success or `undef` (undefined value) on failure. $vm->set_resource ("<variable_name>, <value>") Returns `1` on success or `undef` (undefined value) on failure. **Note:** These methods apply only to ESX Server.	Gets or sets the value of the virtual machine resource variable specified by <variable_name>. For a list of virtual machine resource variables, see Virtual Machine Resource Variables for ESX Server on page 124.
$vm->get_uptime()	Accesses the uptime of the guest operating system on the virtual machine.
$vm->get_id() **Note:** This method applies only to ESX Server.	Returns the unique (world) ID for a running virtual machine.
$vm->get_pid()	Returns the process ID of a running virtual machine.
$vm->get_capabilities() **Note:** This method applies only to ESX Server.	Returns the access permissions for the current user. This number is a bit vector, where 4=read, 2=write, and 1=execute. For a user with all three permissions, a value of 7 is returned when this property is used in a script.
$vm->get_remote_connections() **Note:** This method applies only to ESX Server.	Returns the number of remotely connected users. This value includes the number of remote consoles, Scripting APIs, and Web-based management interface connections to the virtual machine.
$vm->set_runas_user($user, $password) **Note:** This method applies only to GSX Server 3.1.	Runs the virtual machine as the user specified by the $user and $password.
$vm->get_runas_user() **Note:** This method applies only to GSX Server 3.1.	Returns the name of the user running the virtual machine.

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:

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

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.

Method	Description
$question->get_text() Returns the defined value on success or `undef` (undefined value) on failure.	Gets the question text.
$question->get_choices() Returns the defined value on success or `undef` (undefined value) on failure.	Gets an array of strings representing a list of possible answers to the question.
$question->get_id() Returns the defined value on success or `undef` (undefined value) on failure.	Gets an integer used internally by VmPerl to identify the question.

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:

Execution_state Values	Description
VM_EXECUTION_STATE_ON	The virtual machine is powered on.
VM_EXECUTION_STATE_OFF	The virtual machine is powered off.
VM_EXECUTION_STATE_SUSPENDED	The virtual machine is suspended.
VM_EXECUTION_STATE_STUCK	The virtual machine requires user input. The user must answer a question or dismiss an error.
VM_EXECUTION_STATE_UNKNOWN	The virtual machine is in an unknown state.

VM_POWEROP_MODE_<XXX> Values

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

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

The possible values are listed in the following table:

Powerop_mode Values	Description
VM_POWEROP_MODE_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, 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.
VM_POWEROP_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.
VM_POWEROP_MODE_TRYSOFT	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.

Infotype Values

`$infotype` specifies the product information for the `get_product_info()` method.

Infotype Values	Description
VM_PRODINFO_PRODUCT	The VMware product is returned as VmProduct. For more information on VmProduct, see the following section.
VM_PRODINFO_PLATFORM	The host's operating system is returned as VmPlatform. For more information on VmPlatform, see VM_PRODINFO_PLATFORM_<XXX> Values on page 65.
VM_PRODINFO_BUILD	The product's build number.
VM_PRODINFO_VERSION_MAJOR	The product's major version number.

Infotype Values	Description
VM_PRODINFO_VERSION_MINOR	The product's minor version number.
VM_PRODINFO_VERSION_REVISION	The product's revision number.

VM_PRODINFO_PRODUCT_<XXX> Values

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

VM_PRODINFO_PRODUCT Values	Description
VM_PRODUCT_WS	The product is VMware Workstation.
VM_PRODUCT_GSX	The product is VMware GSX Server.
VM_PRODUCT_ESX	The product is VMware ESX Server.
VM_PRODUCT_UNKNOWN	The product is unknown.

VM_PRODINFO_PLATFORM_<XXX> Values

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

VM_PRODINFO_PLATFORM Values	Description
VM_PLATFORM_WINDOWS	The platform is a Microsoft Windows operating system.
VM_PLATORM_LINUX	The platform is a Linux operating system.
VM_PLATFORM_VMNIX	The platform is the VMware Service Console.
VM_PLATFORM_UNKNOWN	The platform is unknown.

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

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

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

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

GuestInfo Variables

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

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

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

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

For an example showing how the VMware guest service can be invoked in a Perl script, see the sample Perl script to get the IP address of a guest operating system on Setting a Virtual Machine's IP Address Configuration Variable on page 88.

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

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

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

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

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

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

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

Retrieving the Information in the Guest Operating System

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

In a Windows guest operating system:

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

In a Linux guest operating system:

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

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

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

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

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

In a Windows guest operating system:

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

In a Linux guest operating system:

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

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

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

Retrieving Information in a VmPerl Script

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

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

```
$vm->get_guest_info('name')
```

CHAPTER 5

Using Sample VmPerl Scripts

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

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

The sample scripts illustrate:

- Listing the Virtual Machines on the Server
- Starting All Virtual Machines on a Server
- Checking a Virtual Machine's Power Status
- Monitoring a Virtual Machine's Heartbeat
- Answering Questions Posed by a Virtual Machine
- Suspending a Virtual Machine
- Setting a Virtual Machine's IP Address Configuration Variable
- Getting a Virtual Machine's IP Address

- Adding a Redo Log to a Virtual Disk (ESX Server only)
- Committing a Redo Log to a Virtual Disk without Freezing the Virtual Machine (ESX Server only)

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

Copyright Information

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

Listing the Virtual Machines on the Server

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

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

```
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
# .
# .
# .

#
# enumerate.pl
#
# This script lists all of the registered virtual machines
# on the server specified by hostname.
#
# usage:
#   enumerate.pl <hostname> <user> <password>
#

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

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

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

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

# Create a new VMware::VmPerl::Server to connect to the server
# To connect to the remote server, use the following line:
```

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

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

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


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

print "\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);

# 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) 1998-2004 VMware, Inc.
# .
# .
# .
#
# startallvms.pl
#
# This script powers on all VMs on the system that are not
# already running.
#
# usage:
#   startallvms.pl <hostname> <user> <password>
#

BEGIN {
   if ($^O eq "MSWin32") {
      @INC = (
         # Set the path to your VmPerl Scripting directory if different
        'C:\Program Files\VMware\VMware VmPerl Scripting API\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;

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

# Change this to your port if it is different.
my $port = 902;
```

```
# Create a ConnectParams object
my $connect_params =
   VMware::VmPerl::ConnectParams::new($server_name,$port,$user,$passwd);

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

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

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


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

my $config;

foreach $config (@list) {

   my $vm = VMware::VmPerl::VM::new();

   # Connect to the VM, using the same ConnectParams object.
   if (!$vm->connect($connect_params, $config)) {
      my ($error_number, $error_string) = $server->get_last_error();
      print STDERR "Could not connect to VM $config: Error $error_number: ".
                   "$error_string\n";
   } else {
      # Only power on VMs with the config setting autostart = "true"
      my $autostart = $vm->get_config("autostart");

      if($autostart && $autostart =~ /true/i) {

         # Only try this for VMs that are powered off or suspended.
         my $power_state = $vm->get_execution_state();

         if (!defined($power_state)) {
            my ($error_number, $error_string) = $server->get_last_error();
           print STDERR "Could not get execution state of VM $config: Error ".
```

```
                        "$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*.

```
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
# .
# .
# .
#
# status.pl
#
# This script returns the current power status (on, off, suspended) of the
# virtual machine specified by config on the server defined by hostname.
#
# usage:
#   status.pl <path_to_config_file> [<server> <user> <password>]
#
# If server, user and password are not given, connect to the local server
# as the current user.
#

BEGIN {
   if ($^O eq "MSWin32") {
      @INC = (
         # Set the path to your VmPerl Scripting directory if different
        'C:\Program Files\VMware\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",
   "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*.

```
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
# .
# .
# .
#
# hbcheck.pl
#
# You can use this script to check the virtual machine specified by
# ConfigToCheck for a heartbeat within a certain interval in seconds.
# If no heartbeat is received within the specified Interval, then this
# script will forcefully shutdown ConfigToCheck, and start ConfigToStart.
#
# usage:
#   hbcheck.pl <ConfigToCheck> <ConfigToStart> [Interval]
#

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

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

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

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

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

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

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

# Initialize the object for the virtual machine we want to check.
my $vm = VMware::VmPerl::VM::new();
if (!$vm->connect($connect_params, $cfg_to_check)) {
   my ($error_number, $error_string) = $vm->get_last_error();
   die "Could not connect to virtual machine at $cfg_to_check:\n" .
       "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).
      # 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:\n" .
             "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:n" .
             "Error $error_number: $error_string\n";
      }
      undef $vm_to_start;
      die "Lost heartbeat of $cfg_to_check,\npowered on $cfg_to_start.\n";
   } else {
     # Wait $interval seconds before checking for the virtual machine's heartbeat.
      print "Got heartbeat count $hb\n";
      sleep ($interval);
   }
   $last_hb = $hb;
}
```

Answering Questions Posed by a Virtual Machine

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

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

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

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

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

```
# Read in command line options.
my $cfg = shift or die "Usage: $0 <config-file>\n";


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

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

# Check the power state of the virtual machine.  If it's stuck, get the
# question and list the possible responses.
my $state = $vm->get_execution_state();
if (!defined($state)) {
   my ($err, $errstr) = $vm->get_last_error();
  # 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";
   }
}

# 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;
   }
}
```

Suspending a Virtual Machine

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

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

```
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
# .
# .
# .
#
# suspend.pl
#
# This script suspends to disk the virtual machine specified by config on
# the server defined by hostname.
#
# usage:
#   suspend.pl hostname user password config

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

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

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

my ($cfg_path, $server_name, $user, $passwd) = @ARGV;
# Use the default port of 902.  Change this if your port is different.
my $port = 902;

# Connect to the local host on the default port as yourself.
```

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

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

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

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

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

Setting a Virtual Machine's IP Address Configuration Variable

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

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

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

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

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

```
my($err);

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

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

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

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

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

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

      $myip = $iparr[0];
   }

   return $myip;
}

# Stores the IP address in the guestinfo name space.
sub set_ip_variable {
   if ($^O eq "MSWin32") {
      # Please ensure that VMwareService is in your path.
      # VMwareService needs double quotes around the command.
```

```
      my $cmd = "VMwareService -cmd " . '"' . "info-set guestinfo.ip $ip" . '"';
      system($cmd);
   }
   else {
      # Please ensure that vmware-guestd is found in the path used below
      system("/etc/vmware/vmware-guestd --cmd 'info-set guestinfo.ip $ip'");
   }
   return $?;
}
```

Getting a Virtual Machine's IP Address

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

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

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

```
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
# .
# .
# .
#
# getguestip.pl
#
# This script returns the value of the guest_info variable 'ip' set by
# the guest OS in a virtual machine on a given server.
#
# usage:
#   getguestip.pl <path_to_config_file> [<server> <user> <password>]
#

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

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;

# 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;
```

Adding a Redo Log to a Virtual Disk (ESX Server only)

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

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

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

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

Note: This script applies only to ESX Server.

```
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
# .
# .
# .
#
# addredo.pl
#
# This script takes a specification of a server name, user, password,
# and the path for the config file of a virtual machine on that
# server.  It then displays the disks in the virtual machine
# configuration, allows the user to choose a disk and then adds a redo
# log to that disk. The user can also specify the disk directly as the
# fifth argument on the command line.
#
# usage:
#   addredo.pl server user password config [virtual disk]
#

BEGIN {
   if ($^O eq "MSWin32") {
      @INC = ("./5.00503/lib",
           "./5.00503/lib/MSWin32-x86/auto",
           "./5.00503/lib/MSWin32-x86",
           "./site/5.00503/lib",
           "./site/5.00503/lib/MSWin32-x86/auto",
```

```
            "./site/5.00503/lib/MSWin32-x86");
   } else {
      push(@INC,
           ("/usr/lib/perl5/site_perl/5.005/i386-linux",
            "/usr/lib/perl5/5.00503",
            "."));
   }
}

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

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

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

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

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

$vm->disconnect();
```

Committing a Redo Log to a Virtual Disk without Freezing the Virtual Machine (ESX Server only)

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

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

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

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

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

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

Note: This script applies only to ESX Server.

```
#!/usr/bin/perl -w
#
# Copyright (C) 1998-2004 VMware, Inc.
# .
# .
# .
#
# commitnext.pl
#
# This script takes a specification of a server name, user, password,
# and the path for the config file of a virtual machine on that
# server.  It then displays the disks in the virtual machine
```

```
# configuration and allows the user to choose a disk.  It then commits
# the next-to-top redo log (there must be at least two redo logs) of
# that disk to its. The virtual machine is not frozen during the committing
# process, but the script waits until the commit finishes.  The user can
# also specify the disk directly as the fifth argument on the command line.
#
# usage:
#  commitnext.pl server user password config [virtual disk]
#

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

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

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

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

# Open up a VM object for the virtual machine associated with the
# specified config file.
my $params = VMware::VmPerl::ConnectParams::new($serverName, $port, $user, $passwd);
my $vm = VMware::VmPerl::VM::new();
my $err = $vm->connect($params, $cfg);
if (!defined($err)) {
   my ($errorNumber, $errorString) = $vm->get_last_error();
```

```
   die  "Cannot connect to vm: Error $errorNumber: $errorString\n";
}

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

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

$vm->disconnect();
```

CHAPTER 6

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 101.

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

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

Error Handling for the VmPerl Library

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

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

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

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

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

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

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

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

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

```
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.

VmCOM Error Code	VmPerl Error Code	Description
vmErr_BADSTATE	VM_E_BADSTATE	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.
vmErr_BADVERSION	VM_E_BADVERSION	The version of the VmCOM component/VmPerl module and the VMware server product are incompatible.
vmErr_DISCONNECT	VM_E_DISCONNECT	The network connection to the virtual machine was lost.
vmErr_INSUFFICIENT_RESOURCES	VM_E_INSUFFICIENT_RESOURCES	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.
vmErr_INVALIDARGS	VM_E_INVALIDARGS	The specified arguments are not valid for this operation.
vmErr_INVALIDVM	VM_E_INVALIDVM	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.
vmErr_NEEDINPUT	VM_E_NEEDINPUT	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.
vmErr_NETFAIL	VM_E_NETFAIL	A network failure or misconfiguration prevented the operation from completing.
vmErr_NOACCESS	VM_E_NOACCESS	The operation could not be completed because of an access violation (a permissions problem).
vmErr_NOMEM	VM_E_NOMEM	Your system has run out of memory. Shut down some processes to free up memory.
vmErr_NOPROPERTY	VM_E_NOPROPERTY	The requested variable or property name does not exist.

VmCOM Error Code	VmPerl Error Code	Description
vmErr_NOTCONNECTED	VM_E_NOTCONNECTED	An operation was attempted on a disconnected virtual machine. Connect the virtual machine before performing this operation.
vmErr_NOTSUPPORTED	VM_E_NOTSUPPORTED	The attempted operation is not supported by your version of VMware server.
vmErr_PROXYFAIL	VM_E_PROXYFAIL	The Scripting API could not connect to the server because of a proxy failure. You see this error only if you have configured your remote workstation to use a Web proxy. For more information on using a Web proxy, see your VMware product documentation.
vmErr_TIMEOUT	VM_E_TIMEOUT	There is no response to the request (the operation timed out).
vmErr_UNSPECIFIED	VM_E_UNSPECIFIED	An unspecified error has occurred.
vmErr_VMBUSY	VM_E_VMBUSY	You attempted to connect to a virtual machine that is under the control of a local console running on the server.
vmErr_VMEXISTS	VM_E_VMEXISTS	You attempted to register a virtual machine that is already registered.
vmErr_VMINITFAILED	VM_E_VMINITFAILED	The virtual machine process could not be started on the server.

Event Logging

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

- Power transitions

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

- Messages

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

- Message answers

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

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

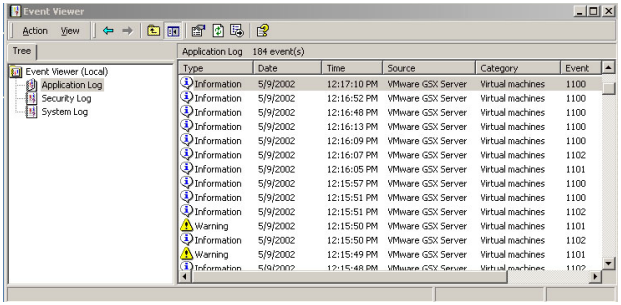
1. Change directories to the VMware GSX Server program directory. The default location is `C:\Program Files\VMware\VMware GSX Server`.
2. Edit the `config.ini` file with a text editor of your choice. Add one or more of the following configuration variables. Each configuration variable turns off event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

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

Using the Event Viewer

1. Open the **Event Viewer** application. This application is typically in the Administrative Tools folder. Refer to your operating system's documentation for additional information on this application.
2. Open the **Application Log** file.

   The Event Viewer is displayed as shown in the following image.

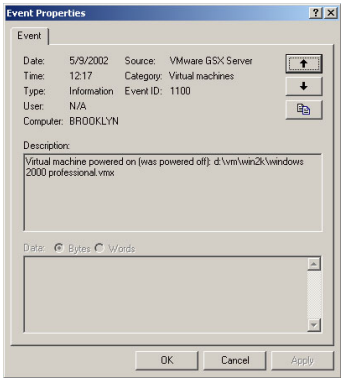


You can use the filtering feature in Event Viewer to see selected events on a virtual machine. All virtual machine events are stored in the "Virtual Machines" category. By contrast, all `serverd` and `authd` events are stored in the default "None" category.

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

Event ID	Event Type
1100	Power transition events
1101	Message events
1102	Message answer events

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.
```

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.
```

CHAPTER 7

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.

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

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

vmware-cmd Utility Options

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

Option	Description
-H	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.
-O	Specifies an alternative port. The default port number is 902.
-U	Specifies the username.
-P	Specifies the user's password.
-h	Prints a help message, listing the options for this utility.
-q	Turns on the quiet option with minimal output. The specified operation and arguments are not specified in the output.
-v	Turns on the verbose option.

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.

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

vmware-cmd Operations on a Virtual Machine

The syntax for this utility on a virtual machine is:

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

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

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

Virtual Machine Operation	Description
vmware-cmd <vm-cfg-path> getstate	Retrieves the execution state of a virtual machine: on, off, suspended, stuck (requires user input) or unknown.
vmware-cmd <vm-cfg-path> start <powerop_mode>	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 <powerop_mode>. If <powerop_mode> is not specified, the default behavior is soft. For more information, see <powerop_mode> Values on page 115.
vmware-cmd <vm-cfg-path> stop <powerop_mode>	Shuts down and powers off a virtual machine. Hard, soft or trysoft specifies the behavior of the power operation <powerop_mode>. If <powerop_mode> is not specified, the default behavior is soft. For more information, see <powerop_mode> Values on page 115.
vmware-cmd <vm-cfg-path> reset <powerop_mode>	Shuts down, then reboots a virtual machine. Hard, soft or trysoft specifies the behavior of the power operation <powerop_mode>. If <powerop_mode> is not specified, the default behavior is soft. For more information, see <powerop_mode> Values on page 115.
vmware-cmd <vm-cfg-path> suspend <powerop_mode>	Suspends a virtual machine. Hard, soft or trysoft specifies the behavior of the power operation <powerop_mode>. If <powerop_mode> is not specified, the default behavior is soft. For more information, see <powerop_mode> Values on page 115.

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

Virtual Machine Operation	Description
vmware-cmd <vm-cfg-path> commit <disk_device_name> <level> <freeze> <wait> **Note:** This operation applies only to ESX Server.	This method commits the changes in a redo log to a running virtual SCSI disk specified by <disk_device_name> that is associated with the virtual machine specified by <vm-cfg-path>. <level> can be `0` or `1`. When <level> is `0`, there can be one or two redo logs associated with the disk. If <level> is `0`, then the top-most redo log (the redo log being modified) is committed to its parent. For example, if there is currently only the disk `vm.dsk` with a single redo log `vm.dsk.REDO`, then the changes in `vm.dsk.REDO` are committed to `vm.dsk`. If a second REDO log `vm.dsk.REDO.REDO` has been added, then the changes in `vm.dsk.REDO.REDO` are committed to `vm.dsk.REDO`. <level> can be `1` only when there are two redo logs associated with the disk, `vm.dsk.REDO` and `vm.dsk.REDO.REDO`. When <level> is `1`, the changes in the next-to-top REDO log, `vm.dsk.REDO`, are committed to `vm.dsk`. In this case, the virtual machine is not frozen while the redo log is being committed. Also, when the log is committed, `vm.dsk.REDO.REDO` is renamed to `vm.dsk.REDO`. <freeze> can be `0` or `1`. If <freeze> is `0`, then the virtual machine is not frozen when changes are committed, though it runs more slowly. If <freeze> is `1`, then the virtual machine is frozen until the commit operation finishes. If <level> is `0`, then the virtual machine must be frozen when changes are committed and <freeze> is ignored. <wait> can be `0` or `1`. If <wait> is `0`, then the method returns as soon as the commit begins. If <wait> is `1`, then the method does not return until the commit completes. The method fails if the specified virtual disk does not exist, the specified virtual disk is in nonpersistent mode, an online commit is already in progress, or the virtual disk currently has no redo logs.
vmware-cmd <vm-cfg-path> setconfig <variable> <value>	Sets a configuration variable for the virtual machine connected to the remote console.
vmware-cmd <vm-cfg-path> getconfig <variable>	Retrieves the value for a configuration variable for the virtual machine connected to the remote console.
vmware-cmd <vm-cfg-path> setguestinfo <variable> <value>	Writes a GuestInfo variable into memory. The variable is discarded when the virtual machine process terminates.
vmware-cmd <vm-cfg-path> getguestinfo <variable>	Retrieves the value for a GuestInfo variable.

Virtual Machine Operation	Description
vmware-cmd <vm-cfg-path> getproductinfo <prodinfo>	Returns information about the product, where <prodinfo> is product, platform, build, majorversion (product's major version number), minorversion (product's minor version number) or revision. If product is specified, the return value is one of the following: ws (VMware Workstation), gsx (VMware GSX Server) esx (VMware ESX Server) or unknown (unknown product type). If platform is specified, the return value is one of the following: windows (Microsoft Windows), linux (Linux operating system) or unknown (unknown platform type).
vmware-cmd <vm-cfg-path> connectdevice <device_name>	Connects the specified virtual device to a virtual machine.
vmware-cmd <vm-cfg-path> disconnectdevice <device_name>	Disconnects the specified virtual device from the virtual machine.
vmware-cmd <vm-cfg-path> getconfigfile	Returns a string containing the configuration file name for the virtual machine. This method fails if the virtual machine is not connected.
vmware-cmd <vm-cfg-path> getheartbeat	Returns the current heartbeat count generated by the VMware Tools service running in the guest operating system. The count is initialized to zero when the virtual machine is powered on. The heartbeat count is typically incremented at least once per second when the VMware Tools service is running under light load conditions. The count stays constant if this service is not running.
vmware-cmd <vm-cfg-path> gettoolslastactive	Returns an integer indicating how much time has passed, in seconds, since the last heartbeat was detected from the VMware Tools service. This value is initialized to zero when the virtual machine powers on. It stays at zero until the first heartbeat is detected, after which the value is always greater than zero until the virtual machine is power-cycled again.
vmware-cmd <vm-cfg-path> answer	Prompts the user to answer a question for a virtual machine waiting for user input.
vmware-cmd getresource <vm-cfg-path> <variable_name> **Note:** This method applies only to ESX Server.	Gets the value of the virtual machine resource variable specified by <variable_name>. For a list of ESX Server virtual machine resource variables, see Virtual Machine Resource Variables for ESX Server on page 124.
vmware-cmd setresource <vm-cfg-path> <variable_name> <value> **Note:** This method applies only to ESX Server.	Sets the value of the virtual machine resource variable specified by <variable_name>. For a list of ESX Server virtual machine resource variables, see Virtual Machine Resource Variables for ESX Server on page 124.
vmware-cmd <vm-cfg-path> getuptime **Note:** This method applies only to ESX Server.	Accesses the uptime of the guest operating system on the virtual machine.

Virtual Machine Operation	Description
vmware-cmd <vm-cfg-path> getid **Note:** This method applies only to ESX Server.	Returns the unique (world) ID for a running virtual machine.
vmware-cmd <vm-cfg-path> getpid **Note:** This method applies only to ESX Server.	Returns the process ID of a running virtual machine.
vmware-cmd <vm-cfg-path> getcapabilities **Note:** This method applies only to ESX Server.	Returns the access permissions for the current user. This number is a bit vector, where 4=read, 2=write, and 1=execute. For a user with all three permissions, a value of 7 is returned when this property is used in a script.
vmware-cmd <vm-cfg-path> getremoteconnections **Note:** This method applies only to ESX Server.	Returns the number of remotely connected users. This value includes the number of remote consoles, Scripting APIs, and Web-based management interface connections to the virtual machine.
vmware-cmd <vm-cfg-path> setrunasuser <username> <password> **Note:** This method applies only to GSX Server 3.1.	Runs the virtual machine as the user specified by the <username> and <password>.
vmware-cmd <vm-cfg-path> getrunasuser **Note:** This method applies only to GSX Server 3.1.	Returns the name of the user running the virtual machine.

<powerop_mode> Values

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

Powerop_mode Values	Description
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.vmx getstate
```

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

In a Windows guest operating system:

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

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

Performing a Power Operation

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

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

In a Linux guest operating system:

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

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

Similarly, in a Windows guest operating system:

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

where `-q` indicates the quiet option (only the results of the operation are printed), `C:\home\vmware\win2000.vmx` is the path to the virtual machine's configuration file and `reset` is the power operation. This example specifies a hard reset so the virtual machine is immediately and unconditionally reset.

Setting a Configuration Variable

The following example illustrates setting a configuration variable in a Linux guest operating system.

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

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

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

Connecting a Device

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

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

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

where `D:\foo.vmx` is the virtual machine's configuration file and `ide1:0` is the device name.

CHAPTER 8

VMware ESX Server Resource Variables

This chapter includes ESX Server resource variables that you can set on ESX Server and on virtual machines running on ESX Server. You can get and set these resource variables through the VmCOM resource property and the VmPerl `get_resource` and `set_resource` methods.

VMware ESX Server System Resource Variables

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

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

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

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

Variable Name	Variable Type	Description
System Statistics		
system.sys.cosUptime	INT	VMware Service Console uptime, in seconds.
system.sys.vmkUptime	INT	VMkernel uptime, in seconds.
CPU Statistics		
system.cpu.number	INT	Number of CPUs on the ESX Server system. On hyperthreading-enabled systems, this number of CPUs reflects the number of logical CPUs (double the number of physical CPUs).
system.cpu.#	INT	Information about the CPU (specified by #).
system.cpu.#.idlesec	FLOAT	Idle time, in seconds, of the physical or logical CPU (specified by #).
system.cpu.#.usedsec	FLOAT	Time, in seconds, the physical or logical CPU (specified by #) is in use.
Memory Statistics		
system.mem.avail	INT	Amount of memory, in KB, available for all virtual machines.
system.mem.COS	INT	Amount of memory, in KB, allocated to the service console.
system.mem.COSavail	INT	Amount of free memory, in KB, available to the service console.
system.mem.active	INT	Amount of memory (subset of system.mem.size) that has been recently used.
system.mem.cosUsage	INT	Amount of memory, in KB, currently used by the service console. The sum of system.mem.COSavail and system.mem.cosUsage should equal the value for system.mem.COS.
system.mem.cpt-tgt	INT	Total amount of data, in KB, read from suspend files for all virtual machines.
system.mem.cptread	INT	Sum of cptread (KB read from suspend files) for all running virtual machines.

Variable Name	Variable Type	Description
system.mem.mctltgt	INT	Total amount of memory, in KB, that the `vmmemctl` balloon drivers should reclaim from all guest operating systems in running virtual machines. The value of system.mem.memctl should always approach this number.
system.mem.memctl	INT	Total amount of memory, in KB, that the `vmmemctl` balloon drivers have reclaimed from all guest operating systems in running virtual machines.
system.mem.overhd	INT	Sum of the current overhead memory, in KB, for all running virtual machines.
system.mem.ovhdmax	INT	Sum of the maximum overhead memory, in KB, for all running virtual machines.
system.mem.reservedMem **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Reserved memory, in KB, across all running virtual machines. This number is the total of user-set minimum memory sizes and the default minimum memory sizes (50% of the specified maximum) on all virtual machines.
system.mem.reservedSwap **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Reserved swap space, in KB, across all virtual machines. This value is equal to the difference between the `max` (maximum) memory size and `min` (minimum) memory size, for all virtual machines.
system.mem.shared	INT	Memory allocated to running virtual machines that is securely shared with other virtual machines.
system.mem.sharedCommon **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Total amount of memory, in MB, that's required for a single copy of shared pages in running virtual machines.
system.mem.sharedVM **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Total amount of shared memory, in MB, for running virtual machines. This is the same value as the sum of system.mem.shared and system.mem.sharedCommon.
system.mem.size	INT	Sum of the current memory size, in KB, for all running virtual machines.
system.mem.sizetgt	INT	Sum of the target memory size, in KB, for all running virtual machines.
system.mem.swapin **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Cumulative memory, in KB, swapped into memory since the last time ESX Server was booted.
system.mem.swapout **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Cumulative memory, in KB, swapped out to disk since the last time ESX Server was booted.
system.mem.swapped	INT	Sum of currently swapped memory, in KB, for all running virtual machines. It is the same value as when system.mem.swapin is subtracted from system.mem.swapout.

Variable Name	Variable Type	Description
system.mem.swaptgt	INT	For all running virtual machines, the target memory size, in KB, used for swapping to the VMFS swap files.
system.mem.sysCodeSize **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	VMkernel code size (memory used to store the executable instructions of the VMkernel).
system.mem.sysHeapSize **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Amount of memory allocated, in KB, to the VMkernel heap file.
system.mem.sysUsage **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Amount of overhead memory, in KB, currently used by the VMkernel.
system.mem.totalMem **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Total amount of memory, in KB, on ESX Server.
system.mem.totalSwap **Note:** This applies only to ESX Server 2.1.1 and higher.	INT	Total amount of swap space, in MB, on ESX Server.
system.mem.vmMaxSize	INT	Memory size of the largest virtual machine that can be powered on, if the minimum (`min`) size is unspecified.
system.mem.vmkUsage	INT	Amount of memory, in KB, currently used by the VMkernel.
system.mem.vmkernel	INT	Amount of memory, in MB, currently allocated to the VMkernel.
Disk Statistics		
system.disk.<HTL>.KBread	INT	Total data read (in KB) for the target SCSI device specified by <HTL>.
system.disk.<HTL>.KBwritten	INT	Total data written (in KB) for the target SCSI device specified by <HTL>.
system.disk.<HTL>.busResets	INT	Total number of bus resets for the target SCSI device specified by <HTL>. We convert bus and device resets to virtual disk files (excluding raw devices) into aborts, so if you compare counters from the virtual machine with this information, then they do not always match.
system.disk.<HTL>.cmds	INT	Total number of commands for the target SCSI device specified by <HTL>.
system.disk.<HTL>.cmdsAborted	INT	Total number of commands aborted for the target SCSI device specified by <HTL>. We convert bus and device resets to virtual disk files (excluding raw devices) into aborts, so if you compare counters from the virtual machine with this information, then they do not always match.
system.disk.<HTL>.reads	INT	Total number of reads for the target SCSI device specified by <HTL>.
system.disk.<HTL>.writes	INT	Total number of writes for the target SCSI device specified by <HTL>.

Variable Name	Variable Type	Description
Network Statistics		
system.net.<device>.totKBRx	INT	Total amount of data received (in KB) on the virtual switch attached to <device>. This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic.
system.net.<device>.totKBTx	INT	Total amount of data transmitted (in KB) on the virtual switch attached to <device>. This data includes both local (virtual machine to virtual machine) and remote (virtual machine to physical network) traffic.
system.net.<device>.totPktsRx	INT	Total number of packets received on the virtual switch attached to <device>. This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic. **Note:** In releases prior to ESX Server 2.1.1,there may be inconsistencies when non-unicast packets (broadcasts, multicasts, and unicasts to virtual NICs in promiscuous mode) are received.
system.net.<device>.totPktsTx	INT	Total number of packets transmitted on the virtual switch attached to <device>. This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic. **Note:** In releases prior to ESX Server 2.1.1,there may be inconsistencies when non-unicast packets are transmitted.
Miscellaneous Statistics		
system.worlds	string	Space-delimited set of IDs listing running virtual machines.
system.net.adapters	string	Space-delimited set of bonds and physical network adapters; for example, `bond0` or `vmnet_0`.
system.disk.HTL	string	Space-delimited set of disks specified as a set of host target LUN (HTL); for example, `vmhba0:0:0` or `vmhba1:0:1`.

Virtual Machine Resource Variables for ESX Server

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

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

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

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

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

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

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

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

Variable Name	Variable Type	Description
CPU Statistics		
cpu.number	INT	Number of CPUs configured for the virtual machine.
cpu.emin	INT	Effective `min` (minimum) percentage of a CPU that is assigned to the virtual machine on CPU x. cpu.emin is always greater than cpu.min. This emin value can change when the virtual machine powers on or off, and when any changes are made to CPU allocations for any running virtual machines.

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

Variable Name	Variable Type	Description
cpu.shares	INT Read-write	Number of CPU shares assigned to a virtual machine.
Memory Statistics		
mem.active	INT	Amount of memory, in KB, actively used by a virtual machine.
mem.cpt-tgt	INT	Amount of memory, in KB, that the virtual machine reads into physical memory from its suspend file. This matches the memory size of the virtual machine, or is `0` (zero), if the system is not swapping from the virtual machine's suspend file.
mem.cptread	INT	Size, in KB, of the suspend file (for a virtual machine) that has been read into memory. If the system memory is low, the suspend file is used as a special swap file for the VMkernel, when the virtual machine is resumed.
mem.mctltgt	INT	The VMkernel believes this is the size, in KB, of the `vmmemctl` balloon driver in the guest operating system for a virtual machine. This value should approach mem.memctl (next row in this table).
mem.memctl	INT	Amount of reclaimed memory, in KB, after running the `vmmemctl` module for a virtual machine.
mem.overhd	INT	Overhead memory, in KB, for a virtual machine.
mem.ovhdmax	INT	Maximum overhead memory, in KB, for a virtual machine.
mem.shared	INT	Amount of memory, in KB, that is shared through transparent page sharing either within a virtual machine or with other virtual machines running on the same server.
mem.size	INT	Size of the actual memory, in KB, for a virtual machine. This value should approach mem.sizetgt (next row in this table).
mem.sizetgt	INT	Target memory size, in KB, for a virtual machine.
mem.swapin	INT	KB swapped into memory since the last time the virtual machine was booted.
mem.swapout	INT	KB swapped out to disk since the last time the virtual machine was booted.
mem.swapped	INT	Amount of memory, in KB, swapped in and out to the VMFS partition swap file for a virtual machine. This value should approach mem.swaptgt (next row in this table).
mem.swaptgt	INT	Target memory size, in KB, to swap to the VMFS swap file for a virtual machine.
mem.affinity	string Read-write	Specifies memory affinity for a virtual machine to a single NUMA node.

Variable Name	Variable Type	Description
mem.max	INT Read-write	Size of the maximum memory for a virtual machine. (This is the amount of memory a virtual machine thinks it has.) In ESX Server 2.0, mem.max is in KB. In ESX Server 2.0.1 and higher, mem.max is in MB.
mem.min	INT Read-write	Size of the minimum memory for a virtual machine. In ESX Server 2.0, mem.min is in KB. In ESX Server 2.0.1 and higher, mem.min is in MB.
mem.shares	INT Read-write	Number of memory shares assigned to a virtual machine. You can use memory shares to calculate proportional server resource allocation between virtual machines.
Disk Statistics		
disk.HTL	string	Space-delimited set of disks specified as a set of host target LUN (HTL); for example, `vmhba0:0:0` or `vmhba1:0:1`.
disk.<HTL>.KBread	INT	Total data read (in KB) for the virtual machine on the target device specified by <HTL>.
disk.<HTL>.KBwritten	INT	Total data written (in KB) for the virtual machine on the target device specified by <HTL>.
disk.<HTL>.busResets	INT	Number of bus resets that occurred on the target device specified by <HTL> due to a command from the virtual machine.
disk.<HTL>.cmds	INT	Total number of commands made by the virtual machine on the target device specified by <HTL>.
disk.<HTL>.cmdsAborted	INT	Total number of commands made by the virtual machine that were aborted on the target device specified by <HTL>.
disk.<HTL>.reads	INT	Total number of disk reads for the virtual machine on the target device specified by <HTL>.
disk.<HTL>.writes	INT	Total number of disk writes for the virtual machine on the target device specified by <HTL>.
disk.<HTL>.shares	INT Read-write	Number of disk shares assigned to the virtual machine on the target device specified by <HTL>.
Network Statistics		
net.adapters	string	Space-delimited set of MAC addresses corresponding to virtual network adapters.
net.<mac>.totKBRx	INT	Total amount of data received (in KB) by the virtual adapter specified by <mac>. This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic.

Variable Name	Variable Type	Description
net.<mac>.totKBTx	INT	Total data transmitted (in KB) by the virtual adapter specified by <mac>. This data includes both local (virtual machine to virtual machine) and remote (virtual machine to physical network) traffic.
net.<mac>.totPktsRx	INT	Total number of packets by the virtual adapter specified by <mac>. This data includes both local (virtual machine to virtual machine) and remote (physical network to virtual machine) traffic.
net.<mac>.totPktsTx	INT	Total number of packets transmitted by the virtual adapter specified by <mac>. This data includes both local (virtual machine to virtual machine) and remote (virtual machine to physical network) traffic.
Other Virtual Machine Statistics		
worldid	INT	World ID for a virtual machine.
pid	INT	Process ID for a virtual machine.
uptime	INT	Uptime, in seconds, of a running virtual machine.

Using ESX Server Virtual Machine Resource Variables Efficiently

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

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

Using the Server Object

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

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

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

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

Reusing a Single Server Object

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

For example, in VmPerl:

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

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

# one-time teardown of server object
$server->disconnect();
```

Index