Administrator's Guide

VMware Infrastructure Toolkit (for Windows) 1.0

VI Toolkit (for Windows) Administrator"s Guide
Revision: 20080627
Item: EN-000058-00

You can 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.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Book

This book, the *VI Toolkit (for Windows) Administrator's Guide,* provides information about using the VMware Infrastructure Toolkit (for Windows) cmdlets (pronounced "commandlets") set that ships with the VI Toolkit (for Windows) for managing, monitoring, automating, and handling life-cycle operations for VMware Infrastructure components—virtual machines, datacenters, storage, networks, and so on.

Revision History

This guide is revised with each release of the product or when necessary. A revised version can contain minor or major changes. Table 1 summarizes the significant changes in each version of this guide.

Table 1. Revision History

Revision	Description
20080313	First version of the documentation for the VI Toolkit (for Windows) 1.0.

To view the most current version of this guide, see the VMware Web site documentation page at:

http://www.vmware.com/support/pubs

Intended Audience

This book is intended for administrators with different levels of familiarity with VMware Infrastructure administration and Windows PowerShell. There are two categories of users for the VI Toolkit (for Windows):

- **Basic** administrators can use PowerShell commands included in the VI Toolkit (for Windows) to manage their VMware virtual infrastructure from the command line.
- **Advanced** administrators can develop PowerShell scripts that may be reused by other administrators or integrated into other applications.

NOTE All VI Toolkit (for Windows) users are expected to be familiar with the details of VMware Infrastructure administration and the Windows operating system. Solution developers are expected to be familiar with the .NET infrastructure and the VIM object model as well.

Document Feedback

VMware welcomes your suggestions for improving our documentation. Send your feedback to:

docfeedback@vmware.com

Technical Support and Education Resources

The following sections describe the technical support resources available to you. To access the current versions of other VMware manuals, go to:

http://www.vmware.com/support/pubs

Online Support

You can submit questions or post comments to the Developer Community: SDKs and APIs forum, which is monitored by VMware technical support and product teams. To access the forum, go to:

http://communities.vmware.com/community/developer

Support Offerings

Find out how VMware support offerings can help meet your business needs. Go to:

http://www.vmware.com/support/services

VMware Education Services

VMware courses offer extensive hands-on labs, case study examples, and course materials designed to be used as on-the-job reference tools. For more information about VMware Education Services, go to:

http://mylearn1.vmware.com/mgrreg/index.cfm

Getting Started with VI Toolkit (for Windows) 1

The VI Toolkit (for Windows) provides easy-to-use C# and PowerShell interface to VMware Infrastructure APIs. It ships with a number of cmdlets that you can use to perform various administration tasks on VMware infrastructure components. This chapter explains how to get started using the VI Toolkit cmdlets.

This chapter covers the following topics:

Introduction to the VI Toolkit Cmdlets

Microsoft PowerShell is both a command-line and scripting environment, designed for Windows. It leverages the .NET object model and provides administrators with management and automation capabilities. Working with PowerShell, like with any other console environment, is done by typing commands. In PowerShell commands are called cmdlets, which term we will use throughout this guide.

The VI Toolkit (for Windows) 1.0 ships with 126 cmdlets:

- 124 PowerShell-based cmdlets.
- 2 .NET cmdlets for use through PowerShell—the so called Web Service Access Cmdlets. See "Web Service Access Cmdlets" on page 18.

VI Toolkit cmdlets are created to answer the specific needs of the VMware Infrastructure administration and management. The VI Toolkit cmdlets are found in the `VMware.VimAutomation.Core` snapin.

Command-Line Syntax

VI Toolkit command-line syntax is the same as generic PowerShell syntax.

PowerShell cmdlets use a consistent verb-noun structure, where the verb specifies the action and the noun specifies the object to operate on. PowerShell cmdlets follow consistent naming patterns, which makes it easy to figure out how to construct a command if you know the object you want to work with.

All command categories take parameters and arguments. A parameter starts with a hyphen and is used to control the behavior of the command. An argument is a data value consumed by the command.

A simple PowerShell command looks like the following:

```
command –parameter1 –parameter2 argument1 argument2
```

Launching VI Toolkit (for Windows) Console

To launch the VI Toolkit (for Windows) Console :

- From the **Start** menu, click **Programs** > **VMware** > **VMware Infrastructure Toolkit > VMware VI Toolkit.**

The script configuration file `Initialize-VIToolkitEnvironment.ps1` is loaded automatically. This file is located in the `Scripts` folder in the VI Toolkit (for Windows) installation directory. Administrators can edit and extend the script to define cmdlets aliases, configure the environment, or set VI Toolkit start up actions.

NOTE Instead of launching the VI Toolkit console, administrators can also access the VI Toolkit (for Windows) snapin directly from other tools, like PowerShell Plus or PowerGUI, by running:

```
Add-PSSnapin VMware.VimAutomation.Core
```

In this case, the `Initialize-VIToolkitEnvironment.ps1` script configuration file is not started automatically. To load it, type its name in the console window without specifying the path:

```
Initialize-VIToolkitEnvironment.ps1
```

Loading the file provides access to VI Toolkit (for Windows) cmdlets aliases, like `Get-VC`, `Get-ESX`, and to other configuration settings.

List All VI Toolkit (for Windows) Cmdlets

If you are new to the VI Toolkit (for Windows), one thing you want to know is what cmdlets are available to you. To get a list of all VI Toolkit cmdlets, use the `Get-Command` cmdlet with the `-PSSnapin` parameter in the following way:

```
Get-Command -PSSnapin VMware.VimAutomation.Core
```

The VI Toolkit cmdlets are listed in the console window as one long, scrolling topic. You can view them a single page at a time by piping the results of the `Get-Command` cmdlet to the `more` option in the following way:

```
Get-Command -PSSnapin VMware.VimAutomation.Core | more
```

Displaying Help for Any Cmdlet

You can get help for a specific cmdlet by supplying the `Get-Help` command in the VI Toolkit Console. For example, for information on the Add-VMhost cmdlet, run the Get-Help command as follows:

```
Get-Help Add-VMHost
```

For more detailed information, add the `-full` parameter:

```
Get-Help Add-VMHost -full
```

Alternatively, you can use the `help` alias with any cmdlet:

```
help Add-VMHost
```

To view detailed help information page by page, pipe the `help` cmdlet to the `more` cmdlet:

```
help Add-VMHost -full | more
```

Connecting to a Server

To run specific VI Toolkit cmdlets and perform administration or monitoring tasks, first establish a connection to an ESX Server or VirtualCenter host.

In the VI Toolkit console window, type the following cmdlet:

```
Connect-VIServer -Server <Server_Address> -Protocol <http_or_https> -User <user_name> -Password
<user_password>
```

You should provide specific values for the server address of the target VirtualCenter or ESX Server host, the protocol (http or https), as well as the user name and password required for log in. For example:

```
Connect-VIServer -Server 192.168.10.10 -Protocol http -User admin -Password sck9p84
```

After a connection is established, you are ready to run the VI Toolkit cmdlets.

For example, you can start a specific virtual machine using the following cmdlet:

```
Get-VM -name <virtual_machine_name> | Start-VM
```

For example, to run a virtual machine named MyVM, run:

```
Get-VM -name MyVM | Start-VM
```

Additional Sets of Cmdlets

Currently, VMware provides only one additional cmdlet set named VMware Update Manager – PowerShell Library. It contains thirteen cmdlets, which enable users to download updates, create baselines, and scan as well as remediate virtual machines and hosts through the VI Toolkit (for Windows) console. To find more information and download VMware Update Manager – PowerShell Library 1.0, visit http://vmware.com/go/powershell.

Basic Cmdlet Usage 2

This chapter explores the basics of the VI Toolkit cmdlets usage.

The chapter discusses the following topics:

- "PowerShell Cmdlet Usage" on page 11
- "Examples of Basic Usage of the VI Toolkit Cmdlets" on page 12

NOTE This chapter does not discuss PowerShell basics. You are expected to have knowledge of PowerShell and its command-line and scripting conventions

PowerShell Cmdlet Usage

In this section, some of the cmdlets syntax and usage basic concepts are described.

Pipelines

A pipeline is a series of commands separated by the pipe operator |. Each command in the pipeline receives an object from the previous command, performs some operation on it, and then passes it along to the next command in the pipeline. Objects are output from the pipeline as soon as they become available. You can type a pipeline on a single line, or spread it across multiple lines. You can cycle backwards through command history using the up arrow, so it is easier to repeat pipelines if you type them on a single line.

Wildcards

PowerShell has a number of pattern matching operators called wildcards, which work on strings. For example, to display all files with a `.txt` extension, run:

```
dir *.txt
```

In this example, the asterisk * operator matches any combination of characters.

Wildcard patterns allow you to specify character ranges as well. For example, to display all files that start with the letter S or T and have a `.txt` extension, run:

```
dir [st]*.txt
```

You can use the question mark ? wildcard to match any single character within a sequence of characters. For example, to display all `.txt` files whose names consist of 'string' and one more character in the end, run:

```
dir string?.txt
```

All wildcard expressions can be used with the VI Toolkit cmdlets.

Common Parameters

The Windows PowerShell engine implements a set of reserved parameter names, referred to as common parameters. All PowerShell cmdlets, including. the VI Toolkit cmdlets, support them. Common parameters are: `Verbose`, `Debug`, `ErrorAction`, `ErrorVariable`, `OutVariable`, and `OutBuffer`. Respectively, the following aliases are reserved for these parameters: `vb`, `db`, `ea`, `ev`, `ov`, and `ob`.

In addition, there are two risk mitigation parameters in PowerShell: `WhatIf` and `Confirm`. `Whatif` is used when you want to see the effects of a command without executing it. `Confirm` is used when a cmdlet performs an operation that stops a program or service or deletes data.

For more details on the usage of common parameters, use the following command:

```
Get-Help about_CommonParameters
```

Examples of Basic Usage of the VI Toolkit Cmdlets

This section provides some examples of basic VI Toolkit cmdlets usage.

Example 2-1. Connecting to a Server

The following cmdlet establishes a connection to a local server and asks for credentials (user name and password), as they are not passed as parameters:

```
Connect-VIServer -Server <VI_server_address>
```

For example:

```
Connect-VIServer -Server esx3.example.com
```

Example 2-2. Basic Virtual Machines Operations

The following example shows how to retrieve information of available virtual machines and their operation system. It also demonstrates how to shut down a virtual machine guest OS and to power off the virtual machine using VI Toolkit (for Windows) cmdlets.

1 After establishing a connection to a server, list all virtual machines on the target system:

```
Get-VM
```

2 Save the name and the power state properties of the virtual machines in the `myPool` resource pool into a file named `myVMProperties.txt`:

```
$pool = Get-ResourcePool myPool

Get-VM -Location $pool | Select-Object Name, PowerState > myVMProperties.txt
```

3 Start the `MyVM` virtual machine:

```
Get-VM "myVM" | Start-VM
```

4 Retrieve information of the guest OS of the `myVM` virtual machine:

```
Get-VMGuest myVM | fc
```

5 Shutdown the OS of the `myVM` virtual machine:

```
Shutdown-VMGuest "myVM"
```

6 Power off the `myVM` virtual machine:

```
Stop-VM "myVM"
```

7 Move the virtual machine `myVM` from the `ABC` host to the `XYZ` host:

```
Get-VM -Name "myVM" -Location (Get-VMHost 'ABC') | Move-VM -Destination (Get-VMHost 'XYZ')
```

NOTE If the virtual machine to be moved across hosts is powered on, it must be located on a shared storage registered as a datastore on both the original and the new host.

Example 2-3. Basic Virtual Machine Hosts Operations

The following example scenario illustrates some basic operations with virtual machine hosts, like adding a host to a VirtualCenter host, putting a host into maintenance mode, shutting down, and removing a host from the VirtualCenter Server.

1 List all hosts on the target VMware Infrastructure server you have established a connection with:

```
Get-VMHost
```

2 Add a standalone virtual machine host to the VirtualCenter host:

```
Add-VMHost -Name myHost -Location (Get-Datacenter Main) -User root -Password pass
```

3 Activate maintanence mode for the `myHost` virtual machine host:

```
Get-VMHost myHost | Set-VMHost -State maintenance
```

4 Remove the `myHost` virtual machine host from maintenance mode:

```
Get-VMHost myHost | Set-VMhost -state connected
```

5 Shut down the `myHost` virtual machine host

```
$host = Get-VMHost myHost

$hv = Get-View $h.ID

$hv.ShutdownHost_Task($true)
```

6 Remove the `myHost` virtual machine host without confirmation:

```
Remove-VMHost $host -Confirm:$false
```

Advanced Cmdlet Usage 3

This chapter provides examples of advanced usage of the VI Toolkit cmdlets.

The chapter discusses these topics:

Examples of Advanced Cmdlet Usage

This section contains examples of using the VI Toolkit cmdlets, the Web Service Access cmdlets, and the inventory provider functionality for retrieving and managing VMware Infrastructure objects.

Using the VI Toolkit Cmdlets

The following examples illustrate how to use advanced functionality provided by the VI Toolkit cmdlets:

Example 3-1. Creating VI Objects

The following example illustrates common methods for creating folders, datacenters, clusters, resource pools and virtual machines using VI Toolkit cmdlets.

1 Establish a connection to a VirtualCenter server using the `Connect-VIServer` command:

```
Connect-VIServer -Server <Server_Address>
```

For example:

```
Connect-VIServer -Server 192.168.10.10
```

When prompted, provide the administrator's user name and password to authenticate access on the server.

2 Get the inventory root folder and create a new folder called `USA` in it:

```
$usaFolder = Get-Folder -NoRecursion | New-Folder -Name USA
```

Note that the information about the location of the new folder is specified through the pipeline.

3 Create a new datacenter called `OrlandoDC` in the `USA` folder:

```
New-Datacenter -Location $usaFolder -Name OrlandoDC
```

4 Create a folder called `Engineering` under `OrlandoDC`:

```
Get-Datacenter OrlandoDC | New-Folder -Name Engineering
$engineeringFolder = Get-Folder -Name engineering
```

NOTE Search in PowerShell is not case-sensitive.

5 Create a new cluster `ResearchAndDevelopmentCluster` in the `Engineering` folder:

```
New–Cluster –Location $engineeringFolder –Name ResearchAndDevelopmentCluster –DrsEnabled
–DrsMode FullyAutomated
```

NOTE Drs (distributed resource scheduler) is a facility that allows automatic allocation of cluster resources.

6 Add a host in the cluster using the `Add–VMHost` command, which prompts you for credentials:

```
$hostInCluster1 = Add–VMHost –Name <host IP address> –Location ( Get–Cluster
ResearchAndDevelopmentCluster )
```

The parentheses interpolate the object returned by the `Get–Cluster` command into `Location` parameter.

7 Create a resource pool in the cluster's root resource pool:

```
$clusterRootRP = Get–ResourcePool –Location ( Get–Cluster ResearchAndDevelopmentCluster )
–Name Resources

New–ResourcePool –Location $clusterRootRP –Name DevelopmentResources
–CpuExpandableReservation $true –CpuReservationMhz 500 –CpuSharesLevel high
–MemExpandableReservation $true –MemReservationMB 500 –MemSharesLevel high
```

8 Create a virtual machine synchronously:

```
New–VM –Name DevelopmentVM1 –VMHost $hostInCluster1 –ResourcePool ( Get–ResourcePool
DevelopmentResources ) –DiskMB 4000 –MemoryMB 256
```

9 Create a virtual machine asynchronously:

```
$vmCreationTask = New–VM –Name DevelopmentVM2 –VMHost $hostInCluster1 –ResourcePool
(Get–ResourcePool DevelopmentResources) –DiskMB 4000 –MemoryMB 256 –RunAsync
```

The `–RunAsync` parameter specifies that the command will be executed asynchronously. This means that in contrast to a synchronous operation, you do not have to wait for the process to complete before supplying the next command in the command line.

Example 3-2. Creating Virtual Machines Using an XML Specifiication File

This example illustrates how to create virtual machines in accordance with the specification provided in an XML file.

Consider a `myVM.xml` file, with the following content:

```
<CreateVM>
    <VM>
        <Name>A</Name>
        <HDDCapacity>10000</HDDCapacity>
    </VM>
    <VM>
        <Name>B</Name>
        <HDDCapacity>10000</HDDCapacity>
    </VM>
</CreateVM>
```

1 To read the content of the `myVM.xml` file, run:

```
[xml]$s = Get–Content my.xml
```

2 To create the virtual machines, run:

```
$s.CreateVM.VM | where { New–VM –VMHost 10.23.113.41 –Name $_.Name –Disk MB $_.HDDCapacity}
```

Example 3-3. Using Virtual Machine Templates

A virtual machine template is a reusable image created from a virtual machine. The template, as a derivative of the source virtual machine, includes virtual hardware components, an installed guest operating system, and software applications.

This example illustrates how to create virtual machines templates and convert them to virtual machines. It uses the VMware Infrastructure objects created in the previous example.

1 Add an additional 2GB hard disk to the DevelopmentVM2 virtual machine:

```
$vmCreationTask | Wait-Task | New-HardDisk -CapacityKB ( 2 * 1024 * 1024 )
```

2 Create a template from the DevelopmentVM1 virtual machine:

```
New-Template -VM ( Get-VM DevelopmentVM1 ) -Name WebTemplate -Location (Get-Datacenter
orlandoDC )
```

A virtual machine template is a reusable image created from a virtual machine. The template, as a derivative of the source virtual machine, includes virtual hardware components, an installed guest operating system and software applications.

3 Convert this template for a use by a virtual machine named DevelopmentVM3:

```
Get-Template WebTemplate | Set-Template -ToVM -Name DevelopmentVM3
```

4 Create a template from the DevelopmentVM2 virtual machine:

```
New-Template -VM ( Get-VM DevelopmentVM2 ) -Name DatabaseTemplate -Location (Get-Datacenter
orlandoDC )
```

5 Convert this template to a virtual machine named DevelopmentVM4:

```
Get-Template DatabaseTemplate | Set-Template -ToVM -Name DevelopmentVM4
```

6 Move the virtual machines into the DevelopmentResources resource pool:

```
Get-VM DevelopmentVM? | Move-VM -Destination ( Get-ResourcePool DevelopmentResources )
```

Use ? as a wildcard to match just one symbol. The command returns all virtual machines whose names start with DevelopmentVM and have one more symbol at the end. In this example, DevelopmentVM1, DevelopmentVM2, DevelopmentVM3, and DevelopmentVM4 are retrieved and moved into the DevelopmentResources resource pool.

7 Start the virtual machines in the DevelopmentResources resource pool:

```
Get-ResourcePool DevelopmentResources | Get-VM | Start-VM
```

Example 3-4. Making Snapshots

A snapshot captures the memory, disk, and settings state of a virtual machine at a particular moment. When you revert to a snapshot, you return all these items to the state they were in at the time you took that snapshot.

The following example illustrates taking a snapshot of virtual machines and then reverting the virtual machines to it. It uses the objects created in Example 3-1.

1 Take a snapshot of all virtual machines in the DevelopmentResources resource pool:

```
Get-ResourcePool DevelopmentResources | Get-VM | New-Snapshot -Name InitialDevConfig
```

The -Location parameter takes arguments of the VIContainer type, on which Cluster, Datacenter, Folder, ResourcePool, and VMHost object types are based. Therefore, the -Location parameter can use arguments of all these types.

2 Revert all virtual machines in the DevelopmentResources resource pool to the InitialDevConfig snapshot:

```
$VMs = Get-VM -Location ( Get-ResourcePool DevelopmentResources )
foreach( $vm in $VMs ) { Set-VM -VM $vm -Snapshot ( Get-Snapshot -VM $vm -Name
InitialDevConfig ) }
```

Example 3-5. Removing VI Objects

In this example you will remove some of the VMware Infrastructure objects created in the previous examples.

1 Retrieve the virtual machines using the '?' wildcard symbol in the Name parameter and remove them along with their disks:

```
Get-VM DevelopmentVM? | Remove-VM -DeleteFromDisk
```

2 Remove the USA folder:

```
Remove-Folder -Folder ( Get-Folder usa )
```

By executing the Remove-Folder command, all child objects for the folder—datacenters, folders, clusters, hosts, resource pools, and so on—are also removed.

Web Service Access Cmdlets

The VI Toolkit (for Windows) 1.0 list of cmdlets includes two Web Service Access cmdlets:

- Get-View
- Get-VIObjectByVIView

They enable access to the programming model of the .NET Toolkit from PowerShell and can be used to initiate .NET Toolkit objects. Each object:

- Is a static copy of a server-side managed object and is not automatically updated when the object on the server changes.
- Includes properties and methods that correspond to the properties and operations of the server-side managed object. For more information about server-side object methods and properties, check the *VMware Infrastructure (VI) API Reference Guide* (http://www.vmware.com/support/pubs/sdk_pubs.html).

Using the Web Service Access cmdlets for low-level VMware Infrastructure management requires some knowledge of both PowerShell scripting and the VMware Infrastructure API.

Example 3-6. Filtering VI Objects

This example illustrates the use of the Get-View cmdlet in combination with a filter. The filter parameter is a HashTable containing one or more pairs of filter criteria. Each of the criteria consists of a property path and a value that represents a regular expression pattern used to match the property.

The filter in this example gets a list of the powered on virtual machines whose guest OS names contain "Windows XP." The Get-View cmdlet then initiates shutdown for each guest OS in the list."

1 Create a filter by the power state and the guest operating system name of the virtual machines:

```
$filter = @{"Runtime.PowerState" ="poweredOn"; "Config.GuestFullName" = "Windows XP"}
```

2 Get a list of the virtual machines using the created filter and call the ShutdownGuest method for each virtual machine in the list:

```
Get-View -ViewType "VirtualMachine" -Filter $filter | foreach{$_.ShutdownGuest()}
```

Example 3-7. Populating a View Object

This example illustrates how to populate a view object from an already retrieved managed object using the Get-View cmdlet.

1 Get the Development2 virtual machine using a filter by name and populates the view object.

```
$vm = Get-View -ViewType VirtualMachine -Filter @{"Name" = "DevelopmentVM2}
$hostView = Get-View -Id $vm.Runtime.Host
```

2 Retrieve runtime information:

```
$hostView.Summary.Runtime
```

Example 3-8. Updating the State of a Server-Side Object

This example illustrates how to update the state of server-side objects:

1 Get the Development2 virtual machine using a filter by name:

```
$vm = Get-View -ViewType VirtualMachine -Filter @{"Name" = "DevelopmentVM2}
$hostView = Get-View -Id $vm.Runtime.Host
```

2 Print the current power state:

```
$vm.Runtime.PowerState
```

3 Change the power state of the virtual machine:

```
If ($vm.Runtime.PowerState –ne “PoweredOn”) {
    $vm.PowerOnVM($vm.Runtime.Host)
} else {
    $vm.PowerOffVM()
}
```

4 Print the value of `$vm` power state (the power state is still not updated because the virtual machine property values are not updated automatically):

```
$vm.Runtime.PowerState
```

5 Update the view:

```
$vm.UpdateViewData()
```

6 Show the actual power state of the virtual machine:

```
$vm.Runtime.PowerState
```

Mixed Usage of VI Toolkit and Web Service Access Cmdlets

To get more advantages of the usability and functionality of the VI Toolkit cmdlets and the Web Service Access cmdlets you can use them together.

Example 3-9. Rebooting a virtual machine host

1 Use the `Get–VMHost` cmdlet to get a virtual machine host by its name, and pass the result to the `Get–View` cmdlet to retrieve the host view:

```
$hostView = Get–VMHost –Name "host.domain.com" | Get–View
```

“`host.domain.name`” is the IP address or the DNS name of the ESX Server host as shown in the inventory.

2 Call the reboot method of the host view object to reboot the host:

```
$hostView.RebootHost()
```

Example 3-10. Modilfying the CPU levels of a virtual machine

This example shows how to modify the CPU levels of a virtual machine using combination of the `Get–View` and `Get–VIObjectByVIView` cmdlets.

1 Retrieve the `Development2` virtual machine, shut down it, and pass the result to the `Get–View` cmdlet to retrieve the virtual machine view object:

```
$vmView = Get–VM  DevelopmentVM2 | Stop–VM | Get–View
```

2 Create a `VirtualMachineConfigSpec` object to modify the virtual machine CPU levels and call the `ReconfigVM` method of the virtual machine view managed object.

```
$spec = new–object VMware.Vim.VirtualMachineConfigSpec;
$spec.CPUAllocation = New–Object VMware.Vim.ResourceAllocationInfo;
$spec.CpuAllocation.Shares = New–Object VMware.Vim.SharesInfo;
$spec.CpuAllocation.Shares.Level = "normal";
$spec.CpuAllocation.Limit = –1;
$vmView .ReconfigVM_Task($spec)
```

3 Get a virtual machine object by using the `Get–VIObjectByVIView` cmdlet and start the virtual machine.

```
$vm =  Get–VIObjectByVIView $vmView  | Start–VM
```

The Inventory Provider

The Inventory Provider (`VimInventory`) is designed to expose a raw inventory view of the inventory items from a server. It enables interactive navigation and file-style management of the VMware Infrastructure inventory. By creating a PowerShell drive based on a managed object (such as a datacenter), you obtain a view of its contents and the relationships between the items. In addition, you are able to manipulate objects (move, rename or delete them) by running commands from the VI Toolkit console.

Example 3-11. Basic Functions of the Inventory Provider

The following example illustrates some basic functions of the inventory provider:

1 Establish a connection to the server using the `Connect–VIServer` command:

```
Connect–VIServer –Server <Server_Address>
```

For example:

```
Connect–VIServer –Server 192.168.10.10
```

When prompted, provide the administrator user name and password to authenticate access on the server.

2 Get the root folder of the server:

```
$root = Get–Folder –NoRecursion
```

3 Create a PowerShell drive named `VI`, based on the server root folder. You can use the built-in `New–psDrive` cmdlet.

```
New–PSDrive –Location $root –Name vi –PSProvider VimInventory –Root '\'
```

NOTE In this release, a single backslash is the required value for the `–Root` parameter.

A different way to create a inventory drive is to use the `New–VIInventoryDrive` command, which is an alias of `New–PSDrive.` It creates a new inventory drive using the `Name` and `Location` parameters. For example:

```
Get–Folder –NoRecursion | New–VIInventoryDrive –Name vi
```

4 Access the new drive by running the following command:

```
cd vi:
```

To list the drive content, run

```
dir
```

`dir` is an alias of the `Get–ChildItem` cmdlet.

5 Navigate through your server inventory using the `cd` command with the full path to the host. For a fictional VMware Infrastructure inventory, it might look like the following:

```
cd Folder01\DataCenter01\host\Web\LiveHost01
```

6 List the content of the host using the `ls` command:

```
ls
```

`ls` is the UNIX style alias of the `Get–ChildItem` cmdlet.

This command will return the virtual machines and the root resource pool of the host. To view only the virtual machines of the host, run the following cmdlet:

```
Get–VM
```

When called within the inventory drive, `Get–VM` retrieves only the virtual machines on the current drive location.

7 Delete a virtual machine named LiveVm01:

```
del LiveVm01
```

8 Rename a virtual machine from LiveVm01New to LiveVm01:

```
ren LiveVm01New LiveVm01
```

9 Start all virtual machines whose names start with LiveVm:

```
dir LiveVm* | Start–VM
```

Note that the provider file-style operations work on the same inventory objects as the toolkit cmdlets.

The Datastore Provider

The Datastore Provider (`VimDatastore`) is designed to provide access to the contents of one or more datastores. The items in a datastore are files that contain configuration, virtual disk, and the other data associated with a virtual machine.All file operations are case-sensitive.

NOTE On VirtualCenter 2.0 and ESX Server 3.0, Datastore Provider supports only browsing and deleting items. On VirtualCenter 2.5 and ESX Server 3.5, all file operations are available, including moving, copying, and renaming items.

Example 3-12. Basic functions of the Datastore Provider

The following example illustrates some basic functions of the Datastore Provider:

1 Establish a connection to the server using the `Connect–VIServer` command:

```
Connect–VIServer –Server <Server_Address>
```

For example:

```
Connect–VIServer –Server 192.168.10.10
```

When prompted, provide the administrator user name and password to authenticate access on the server.

2 Get a datstore by its name and assign it to the `$datastore` variable.

```
$datastore = Get–Datastore Storage1
```

3 Create a PowerShell drive `ds:`, based on `$datastore`. You can use the built-in `New–PSDrive` cmdlet.

```
New–PSDrive –Location $datastore –Name ds –PSProvider VimDatastore –Root '\'
```

A different way to create a datastore drive is to use the `New–DatastoreDrive` command, which is an alias of `New–PSDrive`. It creates a new datastore drive using the `Name` and `Datastore` parameters. For example:

```
Get–Datastore Storage1 | New–DatastoreDrive –Name ds
```

4 Access the new drive by running the following command:

```
cd ds:
```

To list the drive content, run:

```
dir
```

`dir` is an alias of the `Get–ChildItem` cmdlet.

5 Navigate to a specific folder on the `ds:` drive, using the `cd` command:

```
cd <Folder_Name>
```

For a fictional VMware Infrastructure inventory, the command might look like the following:

```
cd VirtualMachines\XPVirtualMachine
```

6 List the files of the folder using the `ls` command:

```
ls
```

`ls` is the UNIX style alias of the `Get–ChildItem` cmdlet.

7 Rename a file using the `Rename–Item` cmdlet or its alias `ren`. For example, to change the name of the `vmware–3.log` file to `vmware–3old.log`, run the following command:

```
ren vmware–3.log vmware–3old.log
```

All file operations apply only on files in the current folder.

8 Delete a file using the `Remove-Item` cmdlet or its alias `del`. For example, to remove the `vmware-3old.log` file from the `XPVirtualMachine` folder, use the following command:

```
del ds:\VirtualMachines\XPVirtualMachine\vmware-2.log
```

9 Copy a file using the `Copy-Item` cmdlet or its alias `copy`. For example:

```
copy ds:\VirtualMachines\XPVirtualMachine\vmware-3old.log ds:\VirtualMachines\vmware-3.log
```