Chapter 10

Automating Tasks

In Chapter 9, "Understanding WMI, Scripting, and Hyper-V," you were introduced to Windows Management Instrumentation (WMI) and scripting concepts that are important to understanding when effectively automating Hyper-V administrative tasks. The chapter used some relatively simple automation tasks and code examples.

The focus of this chapter is to show you how to accomplish more complex automation tasks for managing a Hyper-V virtualization environment. The examples in this chapter are exclusive to Hyper-V, unlike the Windows PowerShell code generated by (and used with) System Center Virtual Machine Manager (SCVMM) discussed in Chapter 11, "Using System Center Virtual Machine Manager 2008."

This chapter will discuss common administrative areas as well as automation and scripting examples. Scripting samples are primarily written in Windows PowerShell and rely heavily on a prewritten library of functions available on the Internet.

Common administrative tasks are often categorized in the following groups, which are the topics we'll cover in this chapter:

- Building on the work of others
- Provisioning
- Configuration management
- Managing access
- Migration
- Backup and recovery
- Collecting and monitoring data

Building on the Work of Others

Chapter 9 introduced the Hyper-V WMI provider and namespace. The script examples in Chapter 9 were fairly simple. Although they're useful, they don't accomplish much. Writing useful scripts can take a great deal of time and effort. Developing an understanding of the right WMI classes to access and how best to use them can be laborious. Using the insights of others and building on their efforts is therefore an attractive approach to efficient Hyper-V automation. SCVMM (covered in Chapter 12, "Protecting Virtualized Environments with System Center Data Protection Manager") is the best way to benefit from the expertise of others, but you can also use other approaches.

Many of the examples in this chapter use an evolving, prewritten library of Windows PowerShell Hyper-V functions. The library was created by James O'Neill and is available from CodePlex (`www.codeplex.com`).

TIP O'Neill has a wonderful blog at `http://blogs.technet.com/jamesone/` where he goes into great detail about Windows, virtualization, motor racing, and other topics.

CodePlex is Microsoft's website for open source project hosting, and it's home to numerous useful development projects. You can find code for other Hyper-V–focused initiatives on CodePlex, but James' library is among the most complete currently available anywhere; it's used widely for Hyper-V automation. His `HyperV.PS1` management library uses all the same WMI calls discussed in Chapter 9. The difference is that the calls in the library are surrounded by carefully written Windows PowerShell code. You can access the library for Hyper-V by navigating to `www.codeplex.com/PSHyperv`.

Original Hyper-V Library

There are multiple versions of the library, with some dating back to mid-2008. Each version is bundled in a `.zip` file that you can find by clicking the page's Downloads tab (there may also be separate documentation available). An older iteration of the library, 0.95a Beta, released in August 2008, is available as a single `.ps1` function library via this link:

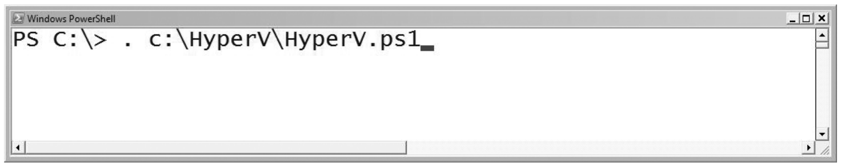
```
http://pshyperv.codeplex.com/releases/view/16422
```

NOTE You may wonder why we would show you a vintage version of the library. We still often use this older version of the library when we work with customers, because it is compatible with PowerShell 1.0, it's simple to load using the conventions we use every day, and it's all contained in one `.ps1` file. Nearly every function or filter in the library includes helpful examples within the source code to demonstrate its value. Browsing through the source can provide you with a wealth of ideas about how to use the library, as well as fantastic examples of how to write great Windows PowerShell code.

Inside the `.zip` file for the August 2008 version, you'll find `HyperV.PS1`, which is the library of functions O'Neill has created for Windows Server 2008, PowerShell 1.0, and Hyper-V.

Loading the `HyperV.PS1` library of functions is straightforward, assuming the execution policy is set and you follow proper calling conventions (see Figure 10.1).

FIGURE 10.1
Calling the library



```
PS C:\> . c:\HyperV\HyperV.ps1
```

NOTE Remember that you load prewritten Windows PowerShell libraries of functions in a similar way to running a script (specifying the path), except that the call to the library must be proceeded by an additional period. Also recall that you must set the execution policy in Windows PowerShell to allow for scripts to execute. See Chapter 9 for more information.

When the library loads, it lists the functions available for use (see Figure 10.2).

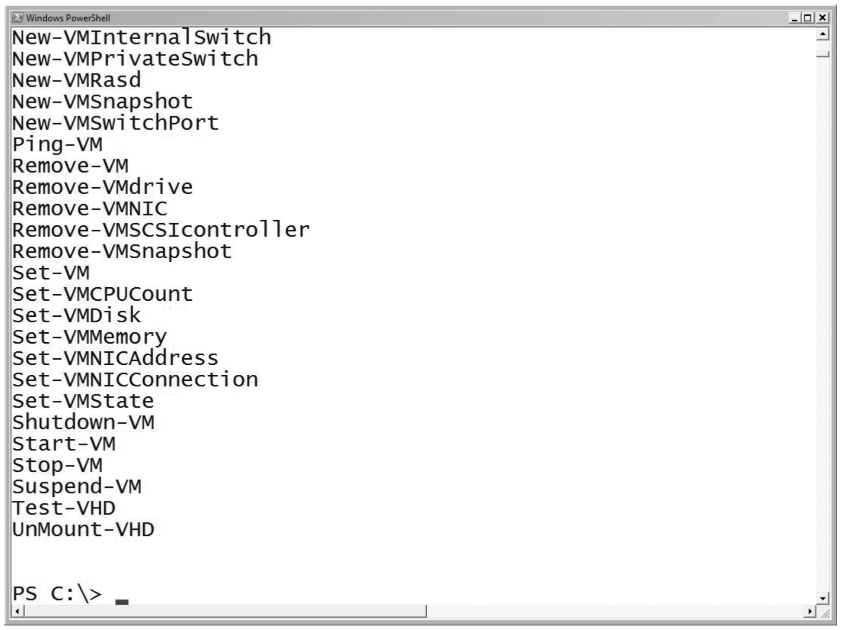


FIGURE 10.2
Functions in the 0.95a Beta library

```
New-VMInternalSwitch
New-VMPrivateSwitch
New-VMRasd
New-VMSnapshot
New-VMSwitchPort
Ping-VM
Remove-VM
Remove-VMdrive
Remove-VMNIC
Remove-VMSCSIcontroller
Remove-VMSnapshot
Set-VM
Set-VMCPUCount
Set-VMDisk
Set-VMMemory
Set-VMNICAddress
Set-VMNICConnection
Set-VMState
Shutdown-VM
Start-VM
Stop-VM
Suspend-VM
Test-VHD
UnMount-VHD


PS C:\> _
```

New R2 Library

Newer versions of the library have been revised to work exclusively with PowerShell 2.0. They include numerous fixes and additional functionality. We'll work through the chapter using the latest available library ("R2 Gold" from January 18, 2010) available here:

```
http://pshyperv.codeplex.com/releases/view/38769
```

Newer versions of the library don't load the same way as the early releases. O'Neill has chopped up the library into smaller files that now can be loaded as modules. Unzipping the contents of the install `.zip` file into a directory reveals a large collection of files rather than a single `.ps1` file (see Figure 10.3). Don't let all these files scare you! They're (mostly) there to make things easier.

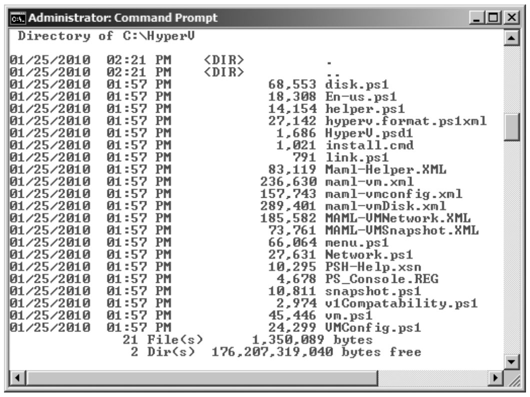
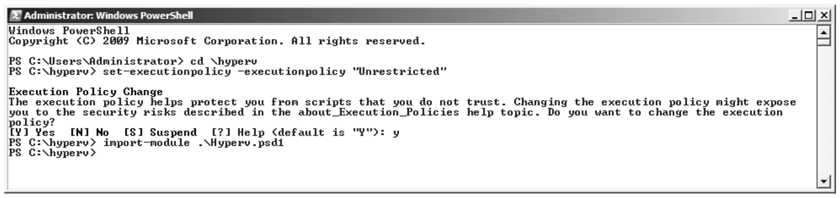


FIGURE 10.3
Files in ZIP download

```
 Directory of C:\HyperV

01/25/2010  02:21 PM    <DIR>          .
01/25/2010  02:21 PM    <DIR>          ..
01/25/2010  01:57 PM            68,553 disk.ps1
01/25/2010  01:57 PM            18,308 En-us.ps1
01/25/2010  01:57 PM            14,154 helper.ps1
01/25/2010  01:57 PM            27,142 hyperv.format.ps1xml
01/25/2010  01:57 PM             1,686 HyperV.psd1
01/25/2010  01:57 PM             1,021 install.cmd
01/25/2010  01:57 PM               791 link.ps1
01/25/2010  01:57 PM            83,119 Maml-Helper.XML
01/25/2010  01:57 PM           236,630 maml-vm.xml
01/25/2010  01:57 PM           157,743 maml-vmconfig.xml
01/25/2010  01:57 PM           289,401 maml-vmDisk.xml
01/25/2010  01:57 PM           185,582 MAML-VMNetwork.XML
01/25/2010  01:57 PM            73,761 MAML-VMSnapshot.XML
01/25/2010  01:57 PM            66,064 menu.ps1
01/25/2010  01:57 PM            27,631 Network.ps1
01/25/2010  01:57 PM            10,295 PSH-Help.xsn
01/25/2010  01:57 PM             4,678 PS_Console.REG
01/25/2010  01:57 PM            10,811 snapshot.ps1
01/25/2010  01:57 PM             2,974 v1Compatability.ps1
01/25/2010  01:57 PM            45,446 vm.ps1
01/25/2010  01:57 PM            24,299 VMConfig.ps1
              21 File(s)      1,350,089 bytes
               2 Dir(s)  176,207,319,040 bytes free
```

To load the library, start PowerShell, and be certain you have administrative rights and that the Execution Policy is set (we often execute a `set-executionpolicy –executionpolicy "Unrestricted"` command to be certain we do not have any issues!). Loading the library can be as simple as typing **`import-module .\HyperV`** (see Figure 10.4). Note that we ran the `import-module` command while our PowerShell session was in the directory containing the library files.

FIGURE 10.4
Loading the `HyperV` module



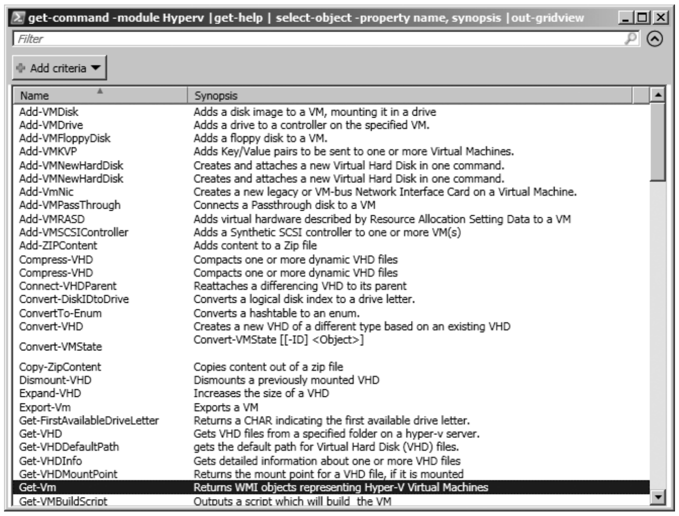
```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> cd \hyperv
PS C:\hyperv> set-executionpolicy -executionpolicy "Unrestricted"

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic. Do you want to change the execution
policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y
PS C:\hyperv> import-module .\Hyperv.psd1
PS C:\hyperv>
```

NOTE Even with the `Unrestricted` option, you may encounter security warning prompts that need to be answered for each `.ps1` and `.ps1xml` file in the module. If these prompts are not answered with R, then the modules do not get imported, and the subsequent commands may fail. To prevent this, use Windows Explorer to access the properties of each file in the library, and choose to "unblock" access.

Using a newer version of the library has advantages, such as the integrated help you can get with a fully developed, imported PowerShell module. Unlike older versions of the library, you can access a list of cmdlets using `get-command` and specifying `–module HyperV`. The `help` and `get-help` cmdlets also work with the newer libraries, so documentation is always handy. You can even combine it with other fancy PowerShell cmdlets (assuming you have installed the Windows PowerShell Integrated Scripting Environment [ISE] feature) like `out-gridview` to generate a sortable, filterable module help reference, as shown in Figure 10.5.

```
get-command -module HyperV | get-help | select-object -property name, synopsis | out-gridview
```

FIGURE 10.5
`out-gridView` list of functions


get-command -module Hyperv |get-help | select-object -property name, synopsis |out-gridview
Filter
Add criteria

Name	Synopsis
Add-VMDisk	Adds a disk image to a VM, mounting it in a drive
Add-VMDrive	Adds a drive to a controller on the specified VM.
Add-VMFloppyDisk	Adds a floppy disk to a VM.
Add-VMKVP	Adds Key/Value pairs to be sent to one or more Virtual Machines.
Add-VMNewHardDisk	Creates and attaches a new Virtual Hard Disk in one command.
Add-VMNewHardDisk	Creates and attaches a new Virtual Hard Disk in one command.
Add-VmNic	Creates a new legacy or VM-bus Network Interface Card on a Virtual Machine.
Add-VMPassThrough	Connects a Passthrough disk to a VM
Add-VMRASD	Adds virtual hardware described by Resource Allocation Setting Data to a VM
Add-VMSCSIController	Adds a Synthetic SCSI controller to one or more VM(s)
Add-ZIPContent	Adds content to a Zip file
Compress-VHD	Compacts one or more dynamic VHD files
Compress-VHD	Compacts one or more dynamic VHD files
Connect-VHDParent	Reattaches a differencing VHD to its parent
Convert-DiskIDtoDrive	Converts a logical disk index to a drive letter.
ConvertTo-Enum	Converts a hashtable to an enum.
Convert-VHD	Creates a new VHD of a different type based on an existing VHD
Convert-VMState	Convert-VMState [[-ID] <Object>]
Copy-ZipContent	Copies content out of a zip file
Dismount-VHD	Dismounts a previously mounted VHD
Expand-VHD	Increases the size of a VHD
Export-Vm	Exports a VM
Get-FirstAvailableDriveLetter	Returns a CHAR indicating the first available drive letter.
Get-VHD	Gets VHD files from a specified folder on a hyper-v server.
Get-VHDDefaultPath	gets the default path for Virtual Hard Disk (VHD) files.
Get-VHDInfo	Gets detailed information about one or more VHD files
Get-VHDMountPoint	Returns the mount point for a VHD file, if it is mounted
Get-Vm	Returns WMI objects representing Hyper-V Virtual Machines
Get-VMBuildScript	Outputs a script which will build the VM

Other Ways to Load the Library

The library is fantastic, but the documentation and some of the tools (like the `install.cmd` file) may not be 100 percent accurate or applicable to your environment. You may want to run (and investigate) the `install.cmd` file that comes in the package in some situations to update the registry as well as to enable PowerShell. We've chosen not to use this file and instead import the library as demonstrated here.

Loading the library as show earlier in this section is the easiest way we've found to get started, but feel free to experiment. After the installation is smoothed out, we often create a shortcut on the desktop of our Hyper-V hosts (if the library is set up) to start PowerShell and load the library automatically. We typically use the following command line (in a `.bat` file):

```
start powershell -noexit -command "import-module c:\hyperv\hyperv"
```

You can add the parameters to a PowerShell shortcut on your desktop instead of using a batch file without the `start` command. Note that this particular command-line example assumes that the library files have all been unzipped to the `C:\HyperV` directory.

Regardless of the version of the library you use and how you load it, the predefined Windows PowerShell functions included in the library are the underpinning of *all* the useful scripts in this chapter. The library is going to save you lots of time and effort if you want to automate Hyper-V–related tasks without the burden of other, more costly tools. We'll now show how to automate the creation of virtual machines.

Provisioning

Creating new virtual machine (VM) instances is the first big, useful automation task to conquer. With a good VM provisioning process, you can quickly and reliably create new VMs in minutes.

Creating a Bare-Bones VM

We'll use the VM provisioning process to help make the point that using O'Neill's library of functions simplifies creating your own scripts. The following Windows PowerShell code creates a new VM instance on the local server with the display name of New VM:

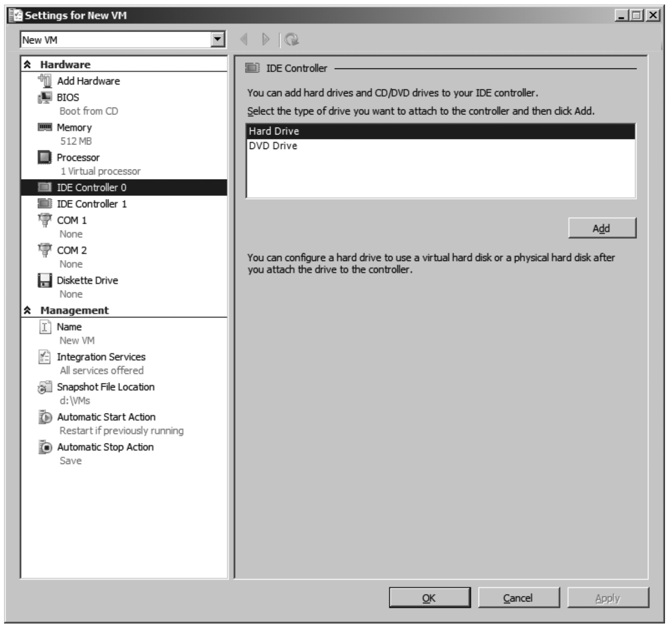
```
# Set the display name of the VM
$New_VM_Name = "New VM"
$VM_Service = GWMI -namespace root\virtualization ↵
Msvm_VirtualSystemManagementService
$NewVM = $VM_Service.DefineVirtualSystem()

# Parse the result and find the created VM
$resultID = $NewVM.DefinedSystem.Split('=')[2]
$resultID = $resultid.split('"')[1]
$VM = GWMI -namespace root\virtualization Msvm_ComputerSystem |
  where {$_.Name -match "$resultID"}

$VMSettingData = GWMI -namespace root\virtualization ↵
```

```
    Msvm_VirtualSystemSettingData -filter "SystemName = `'$($VM.Name)`'"

    # Set the display name of the VM
    $VMSettingData.ElementName = $New_VM_Name
    $VM_Service.ModifyVirtualSystem($VM.__PATH, $VMSettingData.psbase.getText(1))
```

Not all that much code is shown, but then again, it doesn't do much. You create a new VM on the local physical system and give it the intended display name. By contrast, the following single line of code accomplishes the same task using any version of the library:

```
    $myVM = (New-VM "New VM")
```

Using the library vastly simplifies VM management tasks by reducing the amount of code you need to write. In either case, you set defaults for the number of processors (one) and the amount of RAM (512 MB), but you do little else. The VM defined at this point is similar to a bare-bones PC kit (see Figure 10.6). It has only a virtual case, a power supply, a motherboard, limited RAM, and a single processor. The VM has no hard disks, no CD/DVD, and no network interface cards (NICs). You must attach and configure all these resources before you can use the VM.



BIOS: Boot from CD
Memory: 512 MB
Processor: 1 Virtual processor
COM 1: None
COM 2: None
Diskette Drive: None
Name: New VM
Integration Services: All services offered
Snapshot File Location: d:\VMs
Automatic Start Action: Restart if previously running
Automatic Stop Action: Save

You can add hard drives and CD/DVD drives to your IDE controller.

Select the type of drive you want to attach to the controller and then click Add.

You can configure a hard drive to use a virtual hard disk or a physical hard disk after you attach the drive to the controller.

FIGURE 10.6
Virtual settings after creating a basic VM

Defining a usable VM means more effort. You must write additional code to perform the following actions:

- Change the amount of RAM
- Alter the number of virtual CPUs
- Add NICs

- Connect NICs to a particular virtual switch
- Create a virtual hard disk (VHD) file
- Add hard drive(s) attached to VHD file(s) or pass-through disk
- Add CD/DVD drive(s)
- Mount CD/DVD(s)
- All other provisioning tasks (you get the point—changing startup actions, BIOS boot order, and so on)

Each of these actions requires you to weave more code into the basic provisioning process. The complexity of a script that must be handcrafted and maintained to handle all provisioning processes can be substantial. Writing a script of this magnitude is similar to running a marathon—not everyone has the capacity or even wants the challenge (especially if a free ride is available, like O'Neill's library!).

The following Windows PowerShell sample (using any version of the library) completes the common VM provisioning tasks mentioned and starts the new VM, with the execution shown in Figure 10.7:

```
$New_VM_Name  = "New VM"
$New_VHD_Name = "c:\VHDs\$($New_VM_Name).VHD"

$myVM = (New-VM $New_VM_Name)
Set-VMMemory   $myVM 1024MB
Set-VMCPUCount $myVM 2
Add-VMDrive    $myVM 0 0
New-VHD        $New_VHD_Name 20GB -wait
Add-VMDisk     $myVM 0 0 $New_VHD_Name
Add-VMDrive    $myVM 1 0 –dvd
Add-VMDisk     $myVM 1 0 "C:\ISOs\2008R2.iso" –dvd
Add-VMNIC      $myVM -virtualSwitch "Public"

Start-VM $myVM
```

You can immediately see the results of this compact and complete script in the Hyper-V console. Each customized element of the VM's settings are reflected in the settings (see Figure 10.8).

NOTE All the code examples shown have been tested with the latest versions of the library. Most of the code shown will work with either version of the library that we introduced earlier in the chapter. Everything shown was specifically tested with the newer R2 Gold library from January 18, 2010, using PowerShell 2.0.

Setting BIOS Options, Startup/Shutdown, and Additional Elements

You can set BIOS options (such as boot order) and startup/shutdown actions for a VM using the `set-VM` function:

```
set-vm $myVM -bootorder @(3,2,0,1)
```

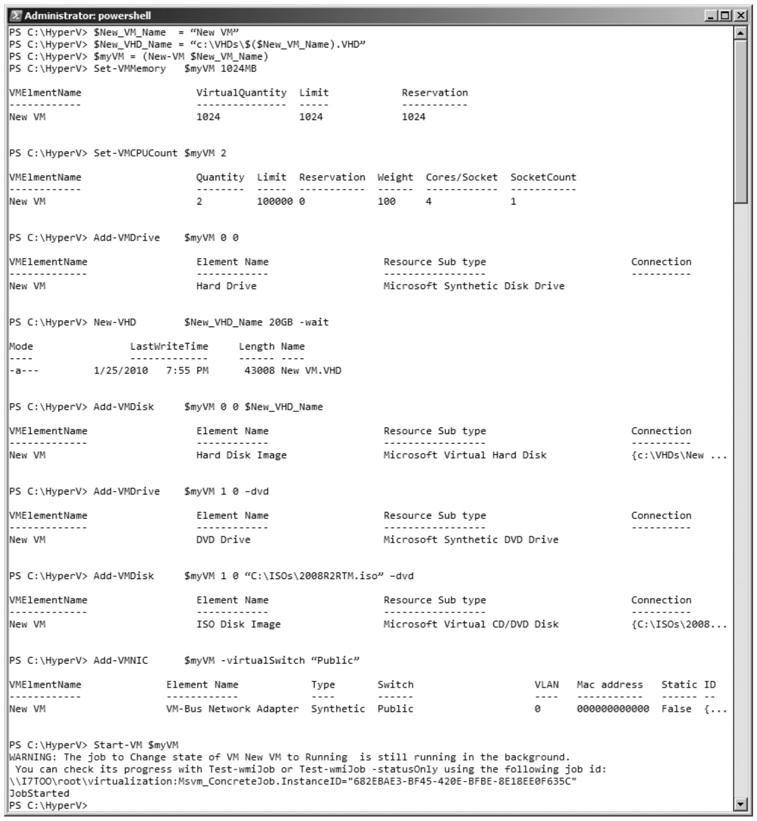


FIGURE 10.7
Script execution

```
PS C:\HyperV> $New_VM_Name  = “New VM”
PS C:\HyperV> $New_VHD_Name = “c:\VHDs\$($New_VM_Name).VHD”
PS C:\HyperV> $myVM = (New-VM $New_VM_Name)
PS C:\HyperV> Set-VMMemory   $myVM 1024MB

VMElmentName                VirtualQuantity  Limit           Reservation
------------                ---------------  -----           -----------
New VM                      1024             1024            1024


PS C:\HyperV> Set-VMCPUCount $myVM 2

VMElmentName                Quantity  Limit  Reservation  Weight  Cores/Socket  SocketCount
------------                --------  -----  -----------  ------  ------------  -----------
New VM                      2         100000 0            100     4             1


PS C:\HyperV> Add-VMDrive    $myVM 0 0

VMElementName               Element Name                Resource Sub type                     Connection
-------------               ------------                -----------------                     ----------
New VM                      Hard Drive                  Microsoft Synthetic Disk Drive


PS C:\HyperV> New-VHD       $New_VHD_Name 20GB -wait

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---         1/25/2010   7:55 PM      43008 New VM.VHD


PS C:\HyperV> Add-VMDisk     $myVM 0 0 $New_VHD_Name

VMElementName               Element Name                Resource Sub type                     Connection
-------------               ------------                -----------------                     ----------
New VM                      Hard Disk Image             Microsoft Virtual Hard Disk           {c:\VHDs\New ...


PS C:\HyperV> Add-VMDrive    $myVM 1 0 -dvd

VMElementName               Element Name                Resource Sub type                     Connection
-------------               ------------                -----------------                     ----------
New VM                      DVD Drive                   Microsoft Synthetic DVD Drive


PS C:\HyperV> Add-VMDisk     $myVM 1 0 “C:\ISOs\2008R2RTM.iso” -dvd

VMElementName               Element Name                Resource Sub type                     Connection
-------------               ------------                -----------------                     ----------
New VM                      ISO Disk Image              Microsoft Virtual CD/DVD Disk         {C:\ISOs\2008...


PS C:\HyperV> Add-VMNIC      $myVM -virtualSwitch “Public”

VMElmentName            Element Name           Type       Switch                 VLAN   Mac address   Static ID
------------            ------------           ----       ------                 ----   -----------   ------ --
New VM                  VM-Bus Network Adapter Synthetic  Public                 0      000000000000  False  {...


PS C:\HyperV> Start-VM $myVM
WARNING: The job to Change state of VM New VM to Running  is still running in the background.
 You can check its progress with Test-wmiJob or Test-wmiJob -statusOnly using the following job id:
\\I7TOO\root\virtualization:Msvm_ConcreteJob.InstanceID="682EBAE3-BF45-420E-BFBE-8E18EE0F635C"
JobStarted
PS C:\HyperV>
```

You can find friendly names for boot media (rather than numbers) in the definition of the `$BootMedia` global variable, as shown in Figure 10.9.

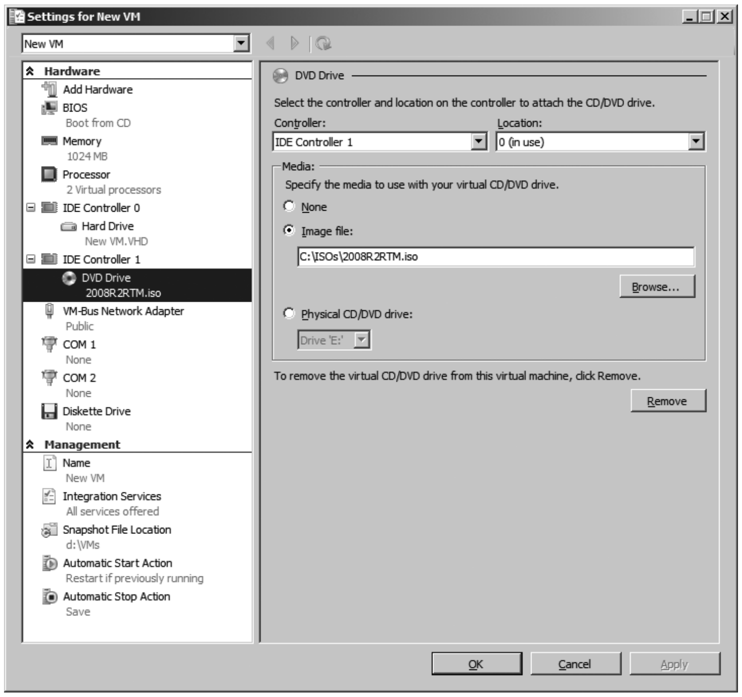
Additional global variables exist to clarify the codes behind the VM state as well as startup, shutdown, and recovery actions. To set the default startup action for a VM to always start, use either of the following lines (note that this code works only with the R2 Gold library):

```
set-vm -VM $myVM -AutoStartup $StartupAction["AlwaysStartup"]
set-vm -VM $myVM -autoStartup 2
```

You can add error handling and management to the earlier basic provisioning script (checks to ensure the ISO file exists, disk space is sufficient, the Public network switch is defined, and each step of the process completes successfully), but it may not be necessary in all situations.

NOTE You may notice that we call all the useful tools in the library functions, when in actuality many of them are defined as filters. Functions and filters are essentially the same thing (filters are a subset of functions). They're both blocks of code that process data. The difference is in how they process data that is piped into them from other functions. Through the evolution of the library, many functions have been converted to filters to add support for piped input, and some filters have changed to functions. Rather than split hairs and keep track, we'll continue to call everything a *function*.

FIGURE 10.8
VM settings after creating a complete VM


Settings for New VM

New VM

Hardware
- Add Hardware
- BIOS: Boot from CD
- Memory: 1024 MB
- Processor: 2 Virtual processors
- IDE Controller 0
  - Hard Drive: New VM.VHD
- IDE Controller 1
  - DVD Drive: 2008R2RTM.iso
- VM-Bus Network Adapter: Public
- COM 1: None
- COM 2: None
- Diskette Drive: None

Management
- Name: New VM
- Integration Services: All services offered
- Snapshot File Location: d:\VMs
- Automatic Start Action: Restart if previously running
- Automatic Stop Action: Save

DVD Drive

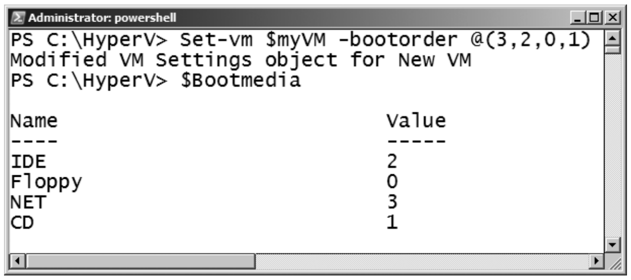
Select the controller and location on the controller to attach the CD/DVD drive.

Controller: IDE Controller 1
Location: 0 (in use)

Media:
Specify the media to use with your virtual CD/DVD drive.
None
[✓ Image file:] C:\ISOs\2008R2RTM.iso
Physical CD/DVD drive: Drive 'E:'

To remove the virtual CD/DVD drive from this virtual machine, click Remove.

FIGURE 10.9
Script execution


Administrator: powershell

```
PS C:\HyperV> Set-vm $myVM -bootorder @(3,2,0,1)
Modified VM Settings object for New VM
PS C:\HyperV> $Bootmedia

Name                           Value
----                           -----
IDE                            2
Floppy                         0
NET                            3
CD                             1
```

Remote Virtual Machine Provisioning

We haven't explicitly mentioned this in the chapter yet, but most functions in the library have been constructed to be executed against a remote server by specifying the `-server` *`<hostname>`* argument. Remotely managing servers is key, because Windows PowerShell doesn't run on the more compact Core installations of Windows Server 2008 (but of course PowerShell does run on Core for R2 as well as Hyper-V Server 2008 R2). Calling the `new-VM` function and specifying a remote host (if successful) populates `$myVM` with information about a new VM created on that remote host:

```
$New_VM_Name  = "New VM"
$myVM = (New-VM $New_VM_Name -server "RemoteHost")
```

You can see the results of successful remote calls to a system named hypernode2 in Figure 10.10. Any code in the examples that use `$myVM` to set VM settings (`set-VMmemory`, `set-VMCPUcount`), add resources (`add-VMdrive`, `add-VMdisk`, `add-VMNIC`), or in other ways affect the VM (`start-VM`) should work properly remotely.

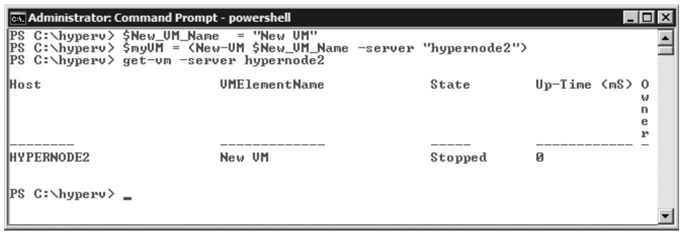


FIGURE 10.10 Call to remote server

```
Administrator: Command Prompt - powershell
PS C:\hyperv> $New_VM_Name  = "New VM"
PS C:\hyperv> $myVM = (New-VM $New_VM_Name -server "hypernode2")
PS C:\hyperv> get-vm -server hypernode2

Host                      VMElementName             State        Up-Time (mS) O
                                                                              w
                                                                              n
                                                                              e
                                                                              r
--------                  -------------             -----        ------------ -
HYPERNODE2                New VM                    Stopped      0

PS C:\hyperv> _
```

NOTE When a variable containing the description of a VM is passed to any of the functions that affect the VM, the function automatically contacts the remote server. If no VM parameter is passed to a function or if convenience dictates that you access the VM by name, you need to specify the `–server` parameter.

Some functions, such as `new-VHD` (called to create a new VHD to be later attached to the VM), don't rely on the VM information found in `$myVM`. Calls to these functions must also include the `-server` argument:

```
$New_VM_Name  = "New VM"
$New_VHD_Name = "c:\VHDs\$($New_VM_Name).VHD"
$Target_Host = "RemoteHost"


$myVM = (New-VM $New_VM_Name -server $Target_Host)
Set-VMMemory   $myVM 1024MB
Set-VMCPUCount $myVM 2
Add-VMDrive    $myVM 0 0
New-VHD        $New_VHD_Name 20GB -wait -Server $Target_Host
Add-VMDisk     $myVM 0 0 $New_VHD_Name
Add-VMDrive    $myVM 1 0 –dvd
Add-VMDisk     $myVM 1 0 "C:\ISOs\XP_Pro.iso" –dvd
Add-VMNIC      $myVM -virtualSwitch "Public"

Start-VM $myVM
```

Precreating Generic VHDs

In the previous provisioning example, the VM created has an ISO image file *inserted* in its virtual CD/DVD drive from which you can install Windows Server 2008 R2. Automating the insertion of an installation disc for a vanilla operating system is a convenient way to build VMs, but installation still requires considerable time and manual intervention (clicking through).

Creating a generic VM for a given operating system instance is a more efficient way to create multiple VMs. Installing a particular operating system version/edition once completely (the x64 full installation of Windows Server 2008 R2 Enterprise Edition, for instance), followed by properly executing `SysPrep.exe`, can save time and effort for repeated installations. `SysPrep.exe` is Microsoft's system preparation utility, which you can use to *depersonalize* a configured operating system instance for widespread deployment (commonly using an imaging tool such as `ImageX` and/or an automated deployment tool like Windows Deployment Services). When you correctly execute the command, you reset key system elements so that you can configure them again in setup to ensure uniqueness and the appropriate personalization in your environment.

TIP `SysPrep.exe` is specific to each edition of Windows. For Windows Vista, Windows 7, Windows Server 2008, and Windows Server 2008 R2, it ships with the product and can be found in the `C:\Windows\system32\sysprep` directory.

You can run `SysPrep.exe` either by passing it various command-line arguments to guide behavior (see Figure 10.11) or by using a graphical interface (see Figure 10.12).

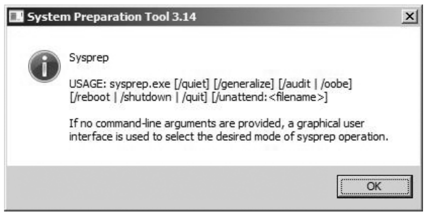


FIGURE 10.11
`SysPrep.exe` arguments

System Preparation Tool 3.14

Sysprep

USAGE: sysprep.exe [/quiet] [/generalize] [/audit | /oobe] [/reboot | /shutdown | /quit] [/unattend:<filename>]

If no command-line arguments are provided, a graphical user interface is used to select the desired mode of sysprep operation.

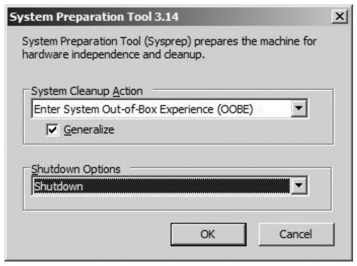


FIGURE 10.12
`SysPrep.exe` graphical interface

System Preparation Tool 3.14

System Preparation Tool (Sysprep) prepares the machine for hardware independence and cleanup.

System Cleanup Action: Enter System Out-of-Box Experience (OOBE)
[✓] Generalize

Shutdown Options: Shutdown

You should keep in mind that not all software can be preinstalled and configured before you run `SysPrep.exe`. The following list includes tasks you can perform before executing `SysPrep.exe`:

- Installing updated integration components (ICs), if not already included in the installation media
- Setting the time zone
- Configuring the patching option
- Applying patches from Windows Update
- Adding common features (PowerShell, for example)

The manual installation of Windows Server 2008 R2 Enterprise edition typically takes almost an hour when you apply patches and complete common configuration tasks. Duplicating a VHD file that has been Sysprepped and repersonalizing should take a small fraction of this time (80 to 90 percent less time is common, depending on system performance).

Automated installation (using answer files and scripts) can further reduce the install effort by automating domain joining, system renaming, and license activation, as well as the installation and configuration of application software and server roles.

TIP You can find detailed information and guidance about how to automate these tasks using `SysPrep.exe` online at `http://technet.microsoft.com` and other websites.

Here's a modified version of the VM provisioning script, including code to copy and register a preconfigured VHD file without an installation ISO or a new, blank VHD:

```
$New_VM_Name  = "New VM"
$New_VHD_Name = "c:\VHDs\$($New_VM_Name).VHD"
Copy "c:\SYSPREPed\Windows Server 2008R2.VHD" $New_VHD_Name

$myVM = (New-VM $New_VM_Name)
Set-VMMemory   $myVM 1024MB
Set-VMCPUCount $myVM 2
Add-VMDrive    $myVM 0 0
Add-VMDisk     $myVM 0 0 $New_VHD_Name
Add-VMDrive    $myVM 1 0 –dvd
Add-VMNIC      $myVM -virtualSwitch "Public"

Start-VM $myVM
```

Deprovisioning

You can automate the removal of VMs as well. You may think twice about creating scripts to remove VMs, because (operationally) simplifying the deletion of a VM presents risks. As with the Hyper-V Manager, removing a VM programmatically requires that the VM not be in a running state (in other words, it must be stopped or saved). Only the configuration of the VM is removed; associated VHD files are left behind and may also need to be deleted. You can remove a VM with one line of code that uses the function library:

```
Remove-VM "New VM"
```

You may also remove resources attached to a VM using functions from the library. Table 10.1 lists these destructive functions.

TABLE 10.1: Hyper-V Library Remove Functions

FUNCTION NAME	DESCRIPTION
`remove-VM`	Deletes the VM configuration
`remove-VMdrive`	Detaches a disk (VHD or pass-through)
`remove-VMfloppydisk`	Removes a floppy disk
`remove-VMKVP`	Removes key/value pair

Table 10.1: Hyper-V Library Remove Functions *(continued)*

Function Name	Description
`remove-VMNIC`	Removes a virtual NIC
`remove-VMSCSIcontroller`	Removes a virtual SCSI controller
`remove-VMRASD`	Removes virtual hardware described by Resource Allocation Setting Data
`remove-VMsnapshot`	Deletes snapshot
`remove-VMswitch`	Deletes virtual switch
`remove-VMswitchNIC`	Removes the parent (physical) NIC associated to a virtual switch

Physical Server Setup

Jumping ahead to show basic VM creation (as we did here) may seem like putting the cart before the horse. You want to ensure that the physical server is ready to accommodate VMs before you set up VMs. You have some common tasks to perform on a physical host, all of which you can automate for consistency. There is more value in automating some configuration tasks than others. For example, changing the default path for new VHDs and configuration files can be important but may not be entirely useful. If you configure failover clustering on a series of hosts, defaulting the settings to a particular volume may be useless, because these settings are typically ignored (VM configuration information and VHDs are often on unique shared storage unless a common file share, CSVs, or a cluster file system is used). Automating the creation of virtual network switches may be more important in a clustered environment, because virtual network switches across cluster nodes must be named consistently to ensure smooth operation.

Functions are available to create each of the three kinds of virtual network switches: *private*, *internal*, and *external*. Creating private and internal virtual switches programmatically is a relatively straightforward process:

```
New-VMInternalSwitch "VM and Host Network"
New-VMPrivateSwitch "VM ONLY Network"
```

Creating a virtual switch with external connectivity is a bit more complicated, because you must specify a physical network card:

```
New-VMExternalSwitch -virtualSwitchName "Wired Network" -ExternalEthernet
"Intel(R) PRO/1000 MT Desktop Adapter"
```

Knowing the name ahead of time for the desired physical NIC is important but not always practical. To simplify virtual switch creation, use the `choose-VMexternalethernet` function. This function queries the host operating system to discover Ethernet connections not already in use by Hyper-V. If more than one connection is found, you're prompted to select one, which is returned as the result:

```
choose-VMExternalEthernet |
New-VMExternalSwitch -virtualSwitchName "Wired Network""
```

Configuration Management

Discovering, managing, and maintaining the configurations of systems are core tasks in any well-managed infrastructure. Locating virtual hosts and VMs is a first step. Accessing and decoding configuration information for hosts and VMs is key to sustaining the health of the overall environment.

Discovery

The ease with which you can create VMs is both a blessing and a curse. The ability to create entire new virtual system instances with a few lines of code or clicks of a mouse means the traditional barriers to server deployment have dramatically changed. No longer do you need to purchase a new server for each new project. Now, a primary goal of server virtualization is often to reduce costs through server consolidation.

Once users begin to understand the speed with which you can create new servers, expectations for new servers increase. As you realize the promise of virtualization, enterprising users will create their own VMs in their own ways. Sometimes they will create systems without triggering processes to ensure that appropriate software licenses are ordered, backup capacity is reserved, or security audits are performed. The impact and cost implications of an unmanaged virtual environment can be enormous.

NOTE More than once, innovative users have built their own virtual test infrastructures, exposing (for example) Dynamic Host Configuration Protocol (DHCP) servers to production networks and interrupting business. (Let's just say we know a guy who did something really bad, and we'll leave it at that!)

Detecting Virtualization Hosts

You can locate installed Hyper-V hosts in a number of ways, including searching servers for running services (`vhdsvc`, `nvspwmi`, `vmms`), scanning volumes for files (including `.vhd` and `.vsv`), enumerating WMI namespaces, and using the power and efficiency of Active Directory (AD). You may not know that properly configured virtualization servers (those running Hyper-V and Virtual Server 2005) publish their binding information in AD as service connection point (SCP) objects.

TIP For more information about service connection points, go to the Service Publication page on the MSDN website at `http://msdn.microsoft.com/en-us/library/ms677950(VS.85).aspx`.

Querying AD for Hyper-V hosts is a great starting point for gathering a bounty of information about virtualization in an enterprise environment. The following sample VBScript generates a list and a count of Hyper-V hosts from the current domain:

```
' Adapted from Alex A. Kibkalo –
' (his is more complete) available from
' http://blogs.technet.com/vm/attachment/3048135.ashx

Set objSystemInfo = CreateObject("ADSystemInfo")
Set objRootDSE = GetObject("LDAP://rootDSE")
szDomainShortName  = objSystemInfo.DomainShortName
szDomainDN = objRootDSE.Get("defaultNamingContext")
```

```
Set oConnection = CreateObject("ADODB.Connection")
Set oCommand = CreateObject("ADODB.Command")
oConnection.Provider = ("ADsDSOObject")
oConnection.Open "Ads Provider"
oCommand.ActiveConnection = oConnection
oCommand.Properties("Page Size") = 99
oCommand.Properties("Searchscope") = &H2 'ADS_SCOPE_SUBTREE
oCommand.Properties("Chase Referrals") = &H60 'ADS_CHASE_REFERRALS_ALWAYS
oCommand.CommandText = "select distinguishedName from 'LDAP://" _
  & szDomainDN & "' " & _
  "where objectCategory='serviceConnectionPoint' and cn='Microsoft Hyper-V'"
Set oRecordSet = oCommand.Execute
oRecordSet.MoveFirst
Do Until oRecordSet.EOF
    szNodeName = oRecordSet.Fields("distinguishedName")
    ' Trim "CN=<szSCP>,CN="
    szNodeName = Mid(szNodeName, InStr(szNodeName, ",CN=") + 4)
    ' Trim the domain DN
    szNodeName = Left(szNodeName, InStr(szNodeName, ",") - 1)
    wscript.echo szNodeName
    oRecordSet.MoveNext
Loop
wscript.echo  "Domain: " & szDomainShortName & _
": " & oRecordSet.RecordCount & " hosts"
```

TIP John Howard posted a similar script on his virtualization blog at `http://blogs.technet.com/jhoward/`. The entry is dated June 30, 2008, and entitled "Hyper-V: Locate Hyper-V Enabled Servers In Your Domain."

Once again, O'Neill's library demonstrates its value by simplifying the task of searching AD for Hyper-V hosts. Using the `get-VMhost` function returns a list of registered Hyper-V servers in the current domain.

Enumerating Virtual Machines

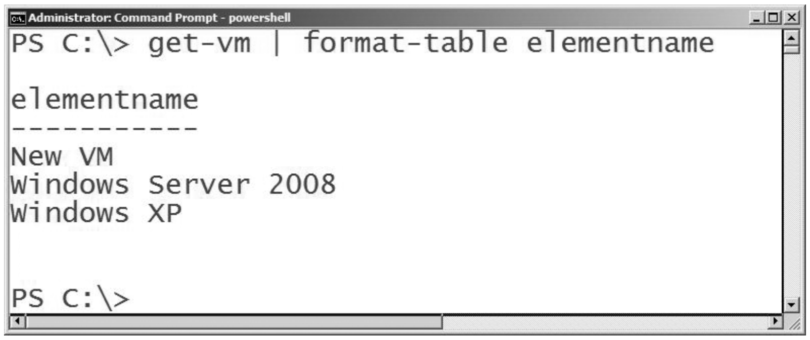
Creating a list of VMs on a particular host is a relatively simple process, as shown in Chapter 9. Creating such a list is even simpler using the functions included in the library (see Figure 10.13):

```
get-vm | format-table elementname
```

You can access all the externally viewable properties of a VM via the information available from the command `get-vm | FL *`.

Wrapping a call to `get-vm` with a loop that can access a list of virtualization hosts provides a useful building block for later automation. A simple text file with the name of each server on a single line can be the input (perhaps created from a query of Active Directory). For our purposes, we've created a file named `serverlist.txt`, which contains two server names:

```
HPDL380t
HPDL380b
```



Administrator: Command Prompt - powershell

```
PS C:\> get-vm | format-table elementname

elementname
-----------
New VM
Windows Server 2008
Windows XP


PS C:\>
```

FIGURE 10.13
List of local VMs

CUSTOMIZING THE OUTPUT OF LIBRARY COMMANDS

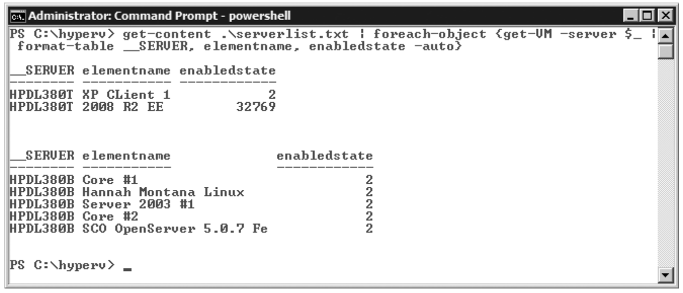
PowerShell allows you to use an XML file to set the default output format for different classes of objects. The early versions of the library didn't use this facility, so `get-VM` (for example) output a list of all the object properties, and the companion function `list-VM` provided formatted output. The newer versions of the library have an associated `Hyperv.Format.PS1XML` file that defines the default output format. In some cases, the XML file processes a property of an object and displays something that isn't available as a property—for example, it translates the value 2 in the `EnabledState` property to the text "running."

With the newer versions, if you want to see all the available properties from `get-VM`, you can pipe its output into `Format-List -Property *`; this can be shortened to `FL *`.

If you want to display output that is different from the default, you can either pipe the output of the function into `Format-Table` (which can be abbreviated to FT) or customize the `PS1.XML` file.

Iterating through the text file using `get-content` and `foreach-object` can generate a list of configured VMs (see Figure 10.14):

```
get-content .\serverlist.txt |
foreach-object {get-VM -server $_ |
format-table __SERVER, elementname, enabledstate -auto}
```



Administrator: Command Prompt - powershell

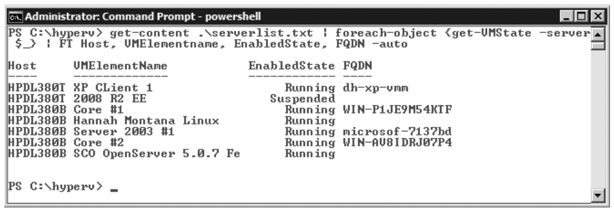
```
PS C:\hyperv> get-content .\serverlist.txt | foreach-object {get-VM -server $_ |
 format-table __SERVER, elementname, enabledstate -auto}

__SERVER elementname enabledstate
-------- ----------- ------------
HPDL380T XP CLient 1            2
HPDL380T 2008 R2 EE         32769



__SERVER elementname             enabledstate
-------- -----------             ------------
HPDL380B Core #1                            2
HPDL380B Hannah Montana Linux               2
HPDL380B Server 2003 #1                     2
HPDL380B Core #2                            2
HPDL380B SCO OpenServer 5.0.7 Fe            2


PS C:\hyperv> _
```

FIGURE 10.14
List of VMs using host input

NOTE Yes, you could pipeline the output from `get-VMhost` to `foreach-object`, but not all Hyper-V hosts may be online or accessible to you. In that case, exceptions are generated and/or additional error handling is required. You could also call `get-VM` and pass it all the host names (`get-VM -server hypernode1, hypernode2 HPDL380B | FT __SERVER, elementname, enabledstate -auto`) to achieve a similar result, but that may be more difficult to automate using a generated list of hosts.

You can extend or alter this basic *host loop* to gather additional useful information beyond those pieces of data exposed by `get-VM` (which uses the `MSVM_ComputerSystem` class shown in Chapter 9). Other VM interrogation functions such as `get-VMstate` can access additional information and handle common formatting tasks (see Figure 10.15).

```
get-content .\serverlist.txt | foreach-object {get-VMState -server $_}
```

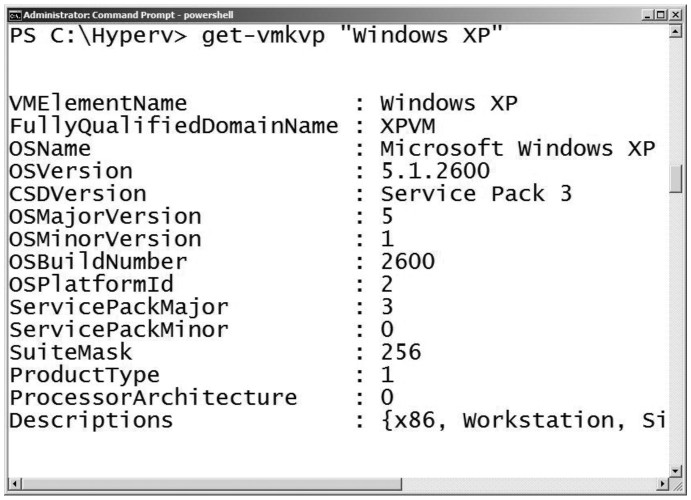


```
PS C:\hyperv> get-content .\serverlist.txt | foreach-object {get-VMState -server
 $_} | FT Host, VMElementname, EnabledState, FQDN -auto

Host     VMElementName             EnabledState FQDN
----     -------------             ------------ ----
HPDL380T XP CLient 1                    Running dh-xp-vmm
HPDL380T 2008 R2 EE                   Suspended
HPDL380B Core #1                        Running WIN-P1JE9M54KTF
HPDL380B Hannah Montana Linux           Running
HPDL380B Server 2003 #1                 Running microsof-7137bd
HPDL380B Core #2                        Running WIN-AV8IDRJ07P4
HPDL380B SCO OpenServer 5.0.7 Fe        Running


PS C:\hyperv> _
```

FIGURE 10.15
Using `get-VMstate` with a host input file

`get-VMstate` decodes the numeric VM state and displays it in a more understandable manner. The function also can access the fully qualified domain name (FQDN) of running VMs with installed ICs (which is super useful!). Fully *enlightened* VMs (those with installed ICs) *that are running* can communicate key information about the installed operating system to the parent partition via the ICs. The `get-VMKVP` function exposes more of these available attributes. Figure 10.16 shows the output from the August 2008 version of the library.

```
get-vmkvp "Core #1"
```

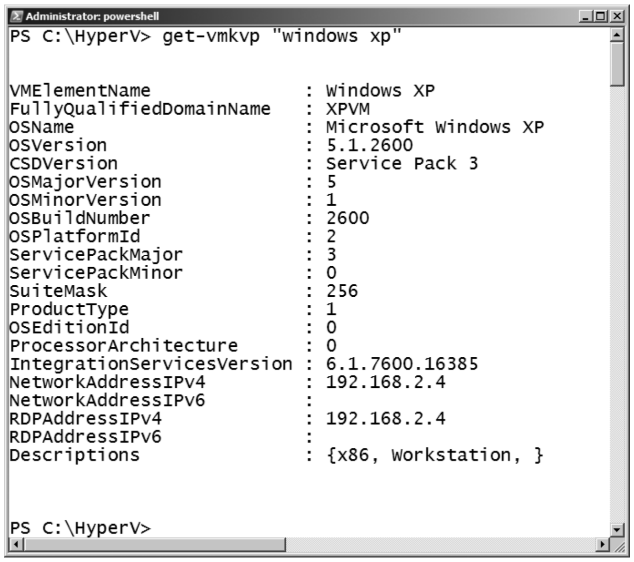


```
PS C:\Hyperv> get-vmkvp "Windows XP"


VMElementName              : Windows XP
FullyQualifiedDomainName : XPVM
OSName                     : Microsoft Windows XP
OSVersion                  : 5.1.2600
CSDVersion                 : Service Pack 3
OSMajorVersion             : 5
OSMinorVersion             : 1
OSBuildNumber              : 2600
OSPlatformId               : 2
ServicePackMajor           : 3
ServicePackMinor           : 0
SuiteMask                  : 256
ProductType                : 1
ProcessorArchitecture      : 0
Descriptions               : {x86, Workstation, Si
```

FIGURE 10.16
The `get-VMKVP` output from the 0.95a Beta version of `HyperV.PS1`

The R2 Gold version of the library returns more information, as shown in Figure 10.17.

FIGURE 10.17
The get-VMKVP output from the R2 Gold version of the Hyper-V library



```
PS C:\HyperV> get-vmkvp "windows xp"


VMElementName                : Windows XP
FullyQualifiedDomainName     : XPVM
OSName                       : Microsoft Windows XP
OSVersion                    : 5.1.2600
CSDVersion                   : Service Pack 3
OSMajorVersion               : 5
OSMinorVersion               : 1
OSBuildNumber                : 2600
OSPlatformId                 : 2
ServicePackMajor             : 3
ServicePackMinor             : 0
SuiteMask                    : 256
ProductType                  : 1
OSEditionId                  : 0
ProcessorArchitecture        : 0
IntegrationServicesVersion : 6.1.7600.16385
NetworkAddressIPv4           : 192.168.2.4
NetworkAddressIPv6           :
RDPAddressIPv4               : 192.168.2.4
RDPAddressIPv6               :
Descriptions                 : {x86, Workstation, }


PS C:\HyperV>
```

TIP This Hyper-V PowerShell library is a work in progress and continues to evolve. To get the most value from the library, remember to periodically check www.codeplex.com for updates.

Collecting information such as the operating system version, service pack level, and FQDN without directly accessing a VM can be valuable when you're troubleshooting or auditing your environment. Combining the server loop with get-vmkvp is fairly straightforward (see Figure 10.18):

```
get-content .\serverlist.txt |
foreach-object {get-VM -server $_ | get-VMKVP} |
format-table FullyQualifiedDomainName, OSName, CSDVersion -auto
```

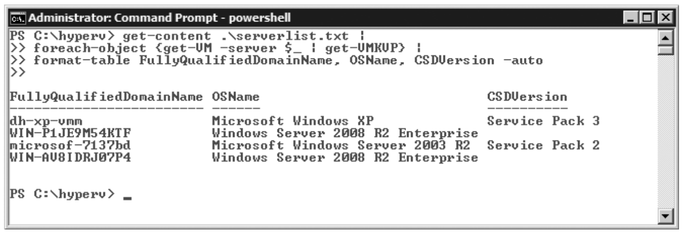
Windows PowerShell allows you to create output in a great many formats besides standard text, including comma-separated (CSV) and XML. Altering the format of your output can make the information easier for other tools and applications to use. You can create a CSV file of the information shown in Figure 10.18 by calling the export-CSV cmdlet:

```
get-content .\serverlist.txt |
foreach-object {get-VM -server $_ | get-VMKVP} |
export-csv -path c:\VMInfo.csv
```

CSV and XML files are handy formats for producing output to pass to other applications. Windows PowerShell version 2 includes a useful out-gridview cmdlet that you can use to view and manipulate data interactively (remember you have to install the Windows PowerShell

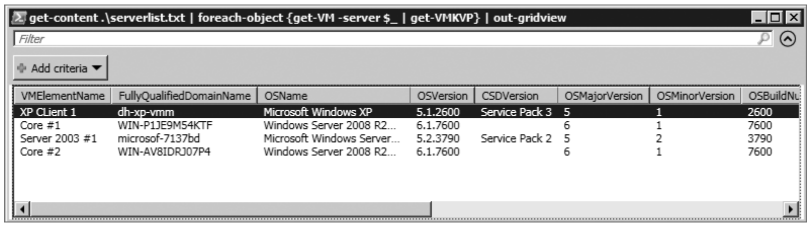
Integrated Scripting Environment [ISE] feature). The graphical interface lets you sort, search, and group data. Figure 10.19 shows output similar to Figure 10.18, but sent to `out-gridview` and grouped by operating system name. The full command would be something like this:

```
get-content .\serverlist.txt | foreach-object {get-VM -server $_ | get-VMKVP} |
out-gridview
```



```
Administrator: Command Prompt - powershell
PS C:\hyperv> get-content .\serverlist.txt |
>> foreach-object {get-VM -server $_ | get-VMKVP} |
>> format-table FullyQualifiedDomainName, OSName, CSDVersion -auto
>>

FullyQualifiedDomainName OSName                                 CSDVersion
------------------------ ------                                 ----------
dh-xp-vmm                Microsoft Windows XP                   Service Pack 3
WIN-P1JE9M54KTF          Windows Server 2008 R2 Enterprise
microsof-7137bd          Microsoft Windows Server 2003 R2       Service Pack 2
WIN-AV8IDRJ07P4          Windows Server 2008 R2 Enterprise


PS C:\hyperv> _
```

FIGURE 10.18
Looping with `Get-VMKVP`



get-content .\serverlist.txt | foreach-object {get-VM -server $_ | get-VMKVP} | out-gridview

VMElementName	FullyQualifiedDomainName	OSName	OSVersion	CSDVersion	OSMajorVersion	OSMinorVersion	OSBuildNu
XP CLient 1	dh-xp-vmm	Microsoft Windows XP	5.1.2600	Service Pack 3	5	1	2600
Core #1	WIN-P1JE9M54KTF	Windows Server 2008 R2...	6.1.7600		6	1	7600
Server 2003 #1	microsof-7137bd	Microsoft Windows Server...	5.2.3790	Service Pack 2	5	2	3790
Core #2	WIN-AV8IDRJ07P4	Windows Server 2008 R2...	6.1.7600		6	1	7600

FIGURE 10.19
`get-VMKVM` set to `out-gridview`

COLLECTING OTHER VIRTUAL MACHINE DETAILS

The library includes numerous `get` functions (and one `list` function). Table 10.2 provides a complete list of these functions and their results.

TABLE 10.2: `get` Functions

FUNCTION NAME	DESCRIPTION
`get-firstavailabledriveletter`	Gets the drive letter of the first available drive letter on the host
`get-VHD`	Retrieves VHD file information from a directory
`get-Vhddefaultpath`	Retrieves the default VHD path (parent specific)
`get-VHDinfo`	Retrieves information about a VHD file
`get-VHDmountpoint`	Returns a mount point of a VHD (if mounted)
`get-VM`	Accesses general VM information
`get-VMByMACaddress`	Retrieves VM information by Media Access Control (MAC) address
`get-VMclustergroup`	Shows the cluster group for a VM

TABLE 10.2: get Functions *(CONTINUED)*

FUNCTION NAME	DESCRIPTION
`get-VMCPUcount`	Retrieves the CPU count
`get-VMdisk`	Displays VM disk controller and drive information
`get-VMdiskbydrive`	Accesses a VM disk by drive
`get-VMdiskcontroller`	Accesses a VM disk by controller
`get-VMdrivebycontroller`	Accesses a VM drive by controller
`get-VMfloppydisk`	Displays VM floppy disks
`get-VMhost`	Queries AD for Hyper-V hosts
`get-VMintegrationcomponent`	Returns IC data for a VM
`get-VMKVP`	Returns key/value pairs for running VMs
`get-VMlivemigrationnetwork`	Gets the list of cluster networks
`get-VMmemory`	Displays the RAM allocated to a VM
`get-VMNIC`	Retrieves NIC information for a VM
`get-VMNICport`	Accesses network port information
`get-VMNICswitch`	Shows the switch connected to a virtual NIC
`get-VMNICVLAN`	Retrieves the VLAN ID for a NIC
`get-VMprocessor`	Returns CPU information for each VM
`get-VMserialport`	Retrieves VM serial port information
`get-VMsettingdata`	Gets active settings for a VM (BIOS, asset tag, other)
`get-VMsnapshot`	Accesses VM snapshot information
`get-VMsnapshottree`	Accesses VM snapshot information and shows it as a tree
`get-VMstate`	Alias for `get-VMsummary`
`get-VMsummary`	Retrieves/decodes the VM state and FQDN
`get-VMswitch`	Gets virtual switch information
`get-VMthumbnail`	Retrieves a JPEG image of a VM display (replace `get-VMJPEG`)
`get-ZIPcontent`	Returns information about the contents of a ZIP file

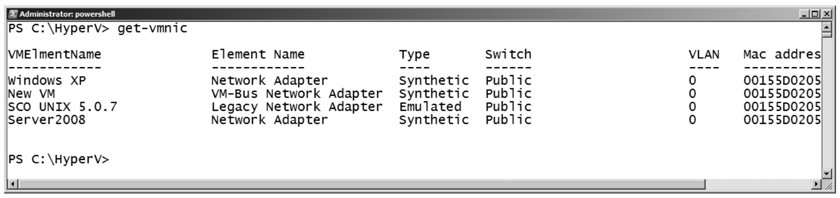
Get to know these functions. Experiment with them and read the examples included, and you'll gain a wealth of configuration knowledge.

TIP Real-life, useful examples abound for the `get` functions. For example, only one VM at a time can access a physical CD/DVD resource. A stopped VM may expect access to a drive when it starts and will fail to start if the drive is already being used. Adding a filter to the `get-VMdisk` function can help you find VMs that are using the physical CD/DVD: `get-VMdisk * | where {$_.diskpath -match "^IDE"} | select VMElementname`.

Creating Simple Reports

Windows PowerShell enables nearly limitless options for report generation. You may not have time or the PowerShell savvy to construct the ideal report. The supplied `get` functions do a fantastic job of exposing the components of Hyper-V and VMs, but sometimes the output is missing important information. The `get-VMNIC` cmdlet provides a great example of this point; it retrieves a list of configured VMs and their corresponding NICs with MAC address and the connected virtual network switch (see Figure 10.20):

```
get-vmnic
```



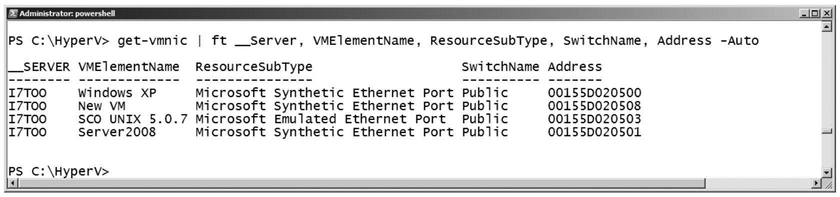
```
PS C:\HyperV> get-vmnic

VMElmentName             Element Name           Type       Switch                  VLAN   Mac addres
------------             ------------           ----       ------                  ----   ----------
Windows XP               Network Adapter        Synthetic  Public                  0      00155D0205
New VM                   VM-Bus Network Adapter Synthetic  Public                  0      00155D0205
SCO UNIX 5.0.7           Legacy Network Adapter Emulated   Public                  0      00155D0205
Server2008               Network Adapter        Synthetic  Public                  0      00155D0205


PS C:\HyperV>
```

FIGURE 10.20
`get-VMNIC` output

This function can be useful for collecting the MAC addresses for all VMs in your environment, but the default output lacks the name of the physical host. The server (host) name is available; you just have to ask for it, as in Figure 10.21.

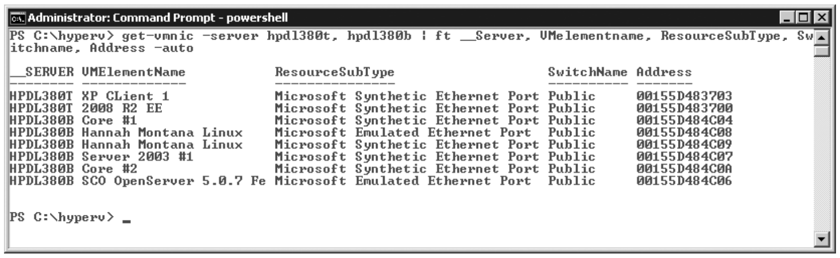
```
get-vmnic | ft __Server, VMElementName, ResourceSubType, SwitchName, Address -Auto
```

Remember that you can use the `-server` parameter to pass multiple server names to most of the functions in the library (as shown in Figure 10.22). This lets you see information on multiple hosts at once.



```
PS C:\HyperV> get-vmnic | ft __Server, VMElementName, ResourceSubType, SwitchName, Address -Auto

__SERVER VMElementName  ResourceSubType                  SwitchName Address
-------- -------------  ---------------                  ---------- -------
I7TOO    Windows XP     Microsoft Synthetic Ethernet Port Public     00155D020500
I7TOO    New VM         Microsoft Synthetic Ethernet Port Public     00155D020508
I7TOO    SCO UNIX 5.0.7 Microsoft Emulated Ethernet Port  Public     00155D020503
I7TOO    Server2008     Microsoft Synthetic Ethernet Port Public     00155D020501


PS C:\HyperV>
```

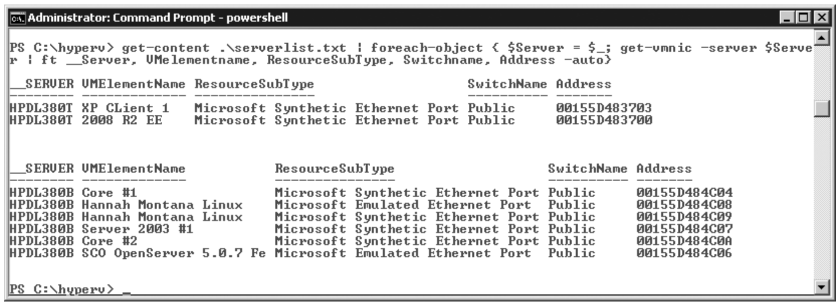
FIGURE 10.21
`get-VMNIC` including server (host) name



```
Administrator: Command Prompt - powershell
PS C:\hyperv> get-vmnic -server hpdl380t, hpdl380b | ft __Server, VMelementname, ResourceSubType, Sw
itchname, Address -auto

__SERVER VMElementName            ResourceSubType                  SwitchName Address
-------- -------------            ---------------                  ---------- -------
HPDL380T XP CLient 1              Microsoft Synthetic Ethernet Port Public     00155D483703
HPDL380T 2008 R2 EE               Microsoft Synthetic Ethernet Port Public     00155D483700
HPDL380B Core #1                  Microsoft Synthetic Ethernet Port Public     00155D484C04
HPDL380B Hannah Montana Linux     Microsoft Emulated Ethernet Port  Public     00155D484C08
HPDL380B Hannah Montana Linux     Microsoft Synthetic Ethernet Port Public     00155D484C09
HPDL380B Server 2003 #1           Microsoft Synthetic Ethernet Port Public     00155D484C07
HPDL380B Core #2                  Microsoft Synthetic Ethernet Port Public     00155D484C0A
HPDL380B SCO OpenServer 5.0.7 Fe Microsoft Emulated Ethernet Port  Public     00155D484C06


PS C:\hyperv> _
```

FIGURE 10.22
`get-VMNIC` output including two servers

It may not always be easy to pass all the server names into a function at once. It might be more convenient for you to use a list of servers contained in a file (as shown in the "Enumerating Virtual Machines" section earlier). Using `get-content` and `foreach-object` can help you collect information from a larger number of targets, as shown in Figure 10.23, without too much extra code.



```
Administrator: Command Prompt - powershell

PS C:\hyperv> get-content .\serverlist.txt | foreach-object { $Server = $_; get-vmnic -server $Serve
r | ft __Server, VMelementname, ResourceSubType, Switchname, Address -auto}

__SERVER VMElementName ResourceSubType                   SwitchName Address
-------- ------------- ---------------                   ---------- -------
HPDL380T XP CLient 1   Microsoft Synthetic Ethernet Port Public     00155D483703
HPDL380T 2008 R2 EE    Microsoft Synthetic Ethernet Port Public     00155D483700



__SERVER VMElementName            ResourceSubType                  SwitchName Address
-------- -------------            ---------------                  ---------- -------
HPDL380B Core #1                  Microsoft Synthetic Ethernet Port Public     00155D484C04
HPDL380B Hannah Montana Linux     Microsoft Emulated Ethernet Port  Public     00155D484C08
HPDL380B Hannah Montana Linux     Microsoft Synthetic Ethernet Port Public     00155D484C09
HPDL380B Server 2003 #1           Microsoft Synthetic Ethernet Port Public     00155D484C07
HPDL380B Core #2                  Microsoft Synthetic Ethernet Port Public     00155D484C0A
HPDL380B SCO OpenServer 5.0.7 Fe Microsoft Emulated Ethernet Port  Public     00155D484C06


PS C:\hyperv> _
```

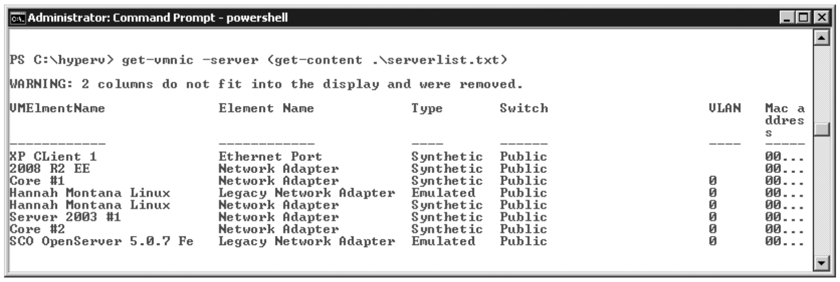
FIGURE 10.23
`get-VMNIC` using loop

You may have noticed that the output isn't presented in the most easy-to-read format. By calling `get-vmnic` multiple times, you get multiple tables of information returned (one for each host). A better way to use `get-content` is within the `get-vmnic`, rather than the other way around:

```
get-vmnic -server (get-content .\serverlist.txt)
```

This actually outputs a single table, as shown in Figure 10.24. You can add extra code to select just the columns you want, similar to the output in Figure 10.21, earlier in this chapter.

```
get-vmnic -server (get-content .\serverlist.txt) |
ft __Server, VMElementName, ResourceSubType,
SwitchName, Address -Auto
```



```
Administrator: Command Prompt - powershell

PS C:\hyperv> get-vmnic -server (get-content .\serverlist.txt)

WARNING: 2 columns do not fit into the display and were removed.

VMElmentName             Element Name           Type       Switch                 VLAN   Mac a
                                                                                         ddres
                                                                                         s
------------             ------------           ----       ------                 ----   -----
XP CLient 1              Ethernet Port          Synthetic  Public                        00...
2008 R2 EE               Network Adapter        Synthetic  Public                        00...
Core #1                  Network Adapter        Synthetic  Public                 0      00...
Hannah Montana Linux     Legacy Network Adapter Emulated   Public                 0      00...
Hannah Montana Linux     Network Adapter        Synthetic  Public                 0      00...
Server 2003 #1           Network Adapter        Synthetic  Public                 0      00...
Core #2                  Network Adapter        Synthetic  Public                 0      00...
SCO OpenServer 5.0.7 Fe  Legacy Network Adapter Emulated   Public                 0      00...
```

FIGURE 10.24
`get-VMNIC` output presented as a single table

We like to simply dump the output to `out-gridview` and select the data we're interested in by "selecting columns" in the GUI (Figure 10.25 shows the output):

```
get-vmnic -server (get-content .\serverlist.txt) |
select * | out-gridview
```

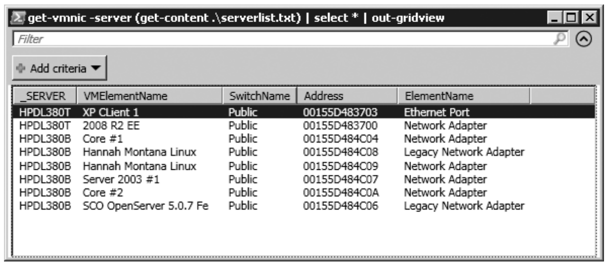


Figure 10.25 `get-VMNIC` information sent to `out-gridview`

get-vmnic -server (get-content .\serverlist.txt) | select * | out-gridview

_SERVER	VMElementName	SwitchName	Address	ElementName
HPDL380T	XP CLient 1	Public	00155D483703	Ethernet Port
HPDL380T	2008 R2 EE	Public	00155D483700	Network Adapter
HPDL380B	Core #1	Public	00155D484C04	Network Adapter
HPDL380B	Hannah Montana Linux	Public	00155D484C08	Legacy Network Adapter
HPDL380B	Hannah Montana Linux	Public	00155D484C09	Network Adapter
HPDL380B	Server 2003 #1	Public	00155D484C07	Network Adapter
HPDL380B	Core #2	Public	00155D484C0A	Network Adapter
HPDL380B	SCO OpenServer 5.0.7 Fe	Public	00155D484C06	Legacy Network Adapter

Regardless of your approach for creating usable scripts and reports, using the HyperV library will save you the time and effort of creating custom code. To make this point again, the following is the Windows PowerShell code that is roughly equivalent to the previous loop but does *not* use O'Neill's functions:

```
foreach ($s in (gc c:\serverlist.txt)) {
  $vms=gwmi -computer $s -namespace "root\virtualization" ↩
 msvm_computersystem -filter "name <> '$s'"
  foreach ($vm in $vms) {
    gwmi -computer $s -NameSpace "root\virtualization" ↩
 -query "Select * From MsVM_EmulatedEthernetPortSettingData ↩
 Where instanceId Like 'Microsoft:$($vm.name)%'"|
    select-object ↩
@{name="VM";expression={$vm.elementname}},↩
@{name="MACAddress";expression={$_.address}},↩
@{name="Server";expression={$_.__SERVER}},↩
@{name="Type";expression={$_.ResourceSubType}},↩
@{name="Network";expression={(gwmi -computer $s ↩
-NameSpace "root\virtualization" -Query "ASSOCIATORS OF ↩
{$(gwmi -computer $s -NameSpace ""root\virtualization"" ↩
-Query ""Select * From Msvm_SwitchPort where
__Path='$($_.connection[0].replace("""\""","""\\"""))'"")} ↩
where resultclass = Msvm_VirtualSwitch").elementname}}
    gwmi -computer $s -NameSpace "root\virtualization" ↩
-query "Select * From MsVM_SyntheticEthernetPortSettingData ↩
Where instanceId Like 'Microsoft:$($vm.name)%'"|select-object
@{name="VM";expression={$vm.elementname}},↩
@{name="MACAddress";expression={$_.address}},↩
@{name="Server";expression={$_.__SERVER}},↩
@{name="Type";expression={$_.ResourceSubType}},↩
@{name="Network";expression={(gwmi -computer $s ↩
-NameSpace "root\virtualization" -Query "ASSOCIATORS OF ⮐
{$(gwmi -computer $s -NameSpace ""root\virtualization"" ⮐
-Query ""Select * From Msvm_SwitchPort where
__Path='$($_.connection[0].replace("""\""","""\\"""))'"")} ⮐
where resultclass = Msvm_VirtualSwitch").elementname}}
    }
}
```

Managing the Virtual Environment

Creating VMs and collecting configuration information about your environment are only the first steps. Managing your virtual environment is critical to realizing the benefits of virtualization. You should strive to use enterprise system management tools if at all possible. The Microsoft System Center family of products provides comprehensive tools to manage physical and virtual environments and is covered in Chapters 11 through 13.

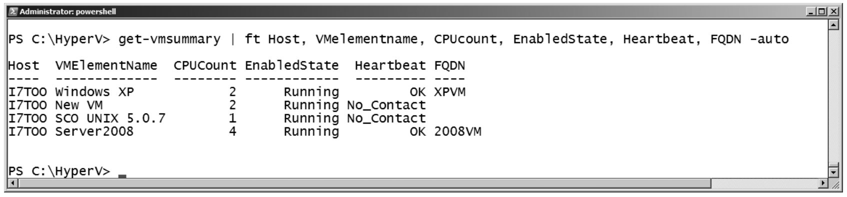
System management can mean many different things, such as managing system configuration, provisioning, performance, security policies, hardware configuration, storage, and even the power state of a system. For our purposes, we'll only discuss managing the system state (power state) for VMs and the management tasks for VHD files. The scripts shown earlier for provisioning VMs demonstrated how to alter the configuration of a VM (add/set resources). We'll review additional VM configuration tasks later, in the "Maintaining Virtual Systems" section of this chapter.

Managing State

Chapter 9 included verbose examples of how to show and alter the system state of VMs. In this chapter, Figures 10.14 and 10.15 illustrate accessing and viewing the state of all VMs on a group of hosts to demonstrate information discovery. The system/power state of a VM is represented by an integer value, discussed and shown in Table 9.5 in Chapter 9. You may not need to care much about this table, because the Hyper-V library discussed in this chapter understands the friendly names of these states. The codes and decodes are contained in the `$VMState` global variable, found near the beginning of the library. Calling `get-VMsummary` (or `get-VMstate`, which is the same) returns the state information of all VMs on the local host (see Figure 10.26):

```
get-vmsummary | ft Host, VMelementname,
CPUcount, EnabledState, Heartbeat, FQDN -auto
```

Figure 10.26
`get-VMsummary` output



```
PS C:\HyperV> get-vmsummary | ft Host, VMelementname, CPUcount, EnabledState, Heartbeat, FQDN -auto

Host  VMElementName  CPUCount EnabledState  Heartbeat FQDN
----  -------------  -------- ------------  --------- ----
I7TOO Windows XP            2      Running         OK XPVM
I7TOO New VM                2      Running No_Contact
I7TOO SCO UNIX 5.0.7        1      Running No_Contact
I7TOO Server2008            4      Running         OK 2008VM


PS C:\HyperV> _
```

To access the state of a single VM, use the `get-VMsummary` function and specify the friendly name of the VM (see Figure 10.27):

```
Get-VMSummary "New VM"
```

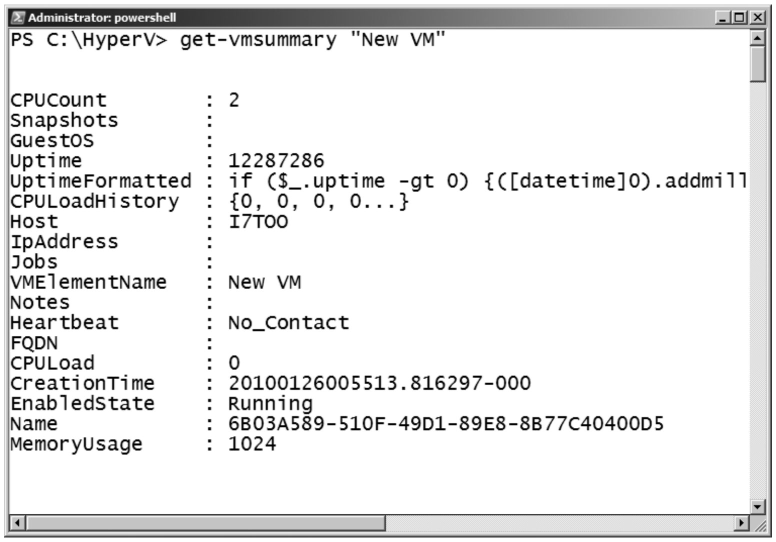


FIGURE 10.27
get-VMstate output for one VM

```
PS C:\HyperV> get-vmsummary "New VM"


CPUCount        : 2
Snapshots       :
GuestOS         :
Uptime          : 12287286
UptimeFormatted : if ($_.uptime -gt 0) {([datetime]0).addmill
CPULoadHistory  : {0, 0, 0, 0...}
Host            : I7TOO
IpAddress       :
Jobs            :
VMElementName   : New VM
Notes           :
Heartbeat       : No_Contact
FQDN            :
CPULoad         : 0
CreationTime    : 20100126005513.816297-000
EnabledState    : Running
Name            : 6B03A589-510F-49D1-89E8-8B77C40400D5
MemoryUsage     : 1024
```

NOTE In the beta version of the library, when you run `get-VMsummary` or `get-VMstate`, you'll see an error if a paused VM is found. A fix for this and other enhancements are in the R2 Gold version of the library.

Changing the state of a VM is just as simple as retrieving it. Table 10.3 shows state management functions.

TABLE 10.3: Hyper-V Library State Management Functions

FUNCTION NAME	DESCRIPTION
`get-VMstate` or `get-VMsummary`	Shows the state of VMs
`start-VM`	Turns on/resumes VMs
`stop-VM`	Turns off VMs
`suspend-VM`	Suspends VMs (pause)
`shutdown-VM`	Shuts down VMs via ICs
`set-VMState`	Specifies the desired VM state

You use `start-VM`, `stop-VM`, `suspend-VM`, and `shutdown-VM` the same way. To call each of these functions, specify the name of the target VM:

```
Start-VM "New VM"
```

As with many functions in the library, you may also specify a remote physical host:

```
Start-VM "New VM" -server HyperNode1
```

It's important to know the state a VM is in before you make a change request. For example, you can't transition from a saved (suspended) state to a paused state. If a VM is already running, a request to start it will fail (see Figure 10.28).

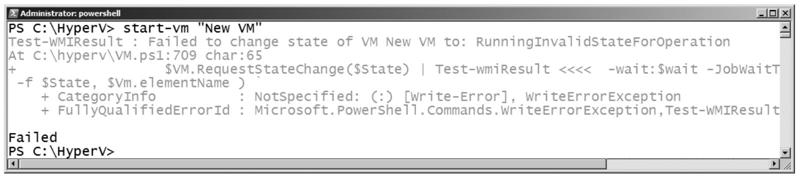


FIGURE 10.28 Trying to start a running VM

```
PS C:\HyperV> start-vm "New VM"
Test-WMIResult : Failed to change state of VM New VM to: RunningInvalidStateForOperation
At C:\hyperv\VM.ps1:709 char:65
+                   $VM.RequestStateChange($State) | Test-wmiResult <<<<  -wait:$wait -JobWaitT
 -f $State, $Vm.elementName )
    + CategoryInfo          : NotSpecified: (:) [Write-Error], WriteErrorException
    + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.WriteErrorException,Test-WMIResult

Failed
PS C:\HyperV>
```

You can change the state of all VMs on a host at once. For example, it could be valuable to put all VMs on a given host into a saved state for backup or while you perform maintenance on the physical system. Using an asterisk instead of the individual VM name with `suspend-VM` saves the state of all running or paused VMs (see Figure 10.29):

```
suspend-vm *
```

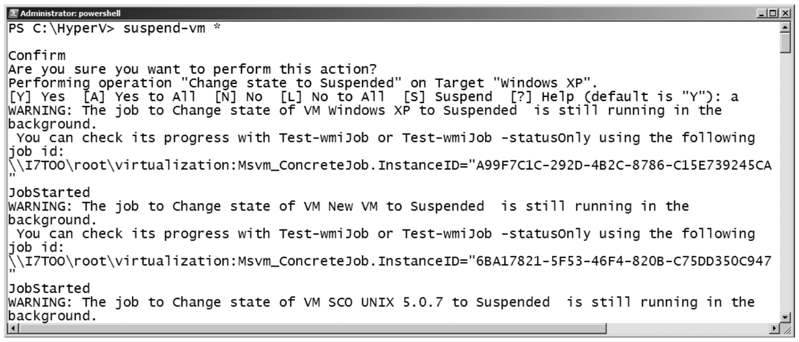


FIGURE 10.29 Suspending all running VMs

```
PS C:\HyperV> suspend-vm *

Confirm
Are you sure you want to perform this action?
Performing operation "Change state to Suspended" on Target "Windows XP".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): a
WARNING: The job to Change state of VM Windows XP to Suspended  is still running in the
background.
 You can check its progress with Test-wmiJob or Test-wmiJob -statusOnly using the following
job id:
\\I7TOO\root\virtualization:Msvm_ConcreteJob.InstanceID="A99F7C1C-292D-4B2C-8786-C15E739245CA
"
JobStarted
WARNING: The job to Change state of VM New VM to Suspended  is still running in the
background.
 You can check its progress with Test-wmiJob or Test-wmiJob -statusOnly using the following
job id:
\\I7TOO\root\virtualization:Msvm_ConcreteJob.InstanceID="6BA17821-5F53-46F4-820B-C75DD350C947
"
JobStarted
WARNING: The job to Change state of VM SCO UNIX 5.0.7 to Suspended  is still running in the
background.
```

Understanding the role of ICs is also important. They allow for a coordinated shutdown of a VM. You can facilitate an orderly power-down of a VM with installed ICs by using `shutdown-VM` (see Figure 10.30):

```
shutdown-VM "Windows XP"
```

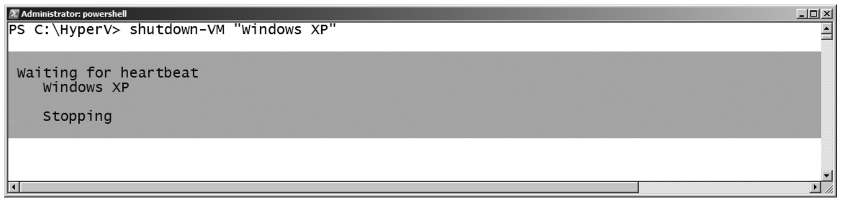
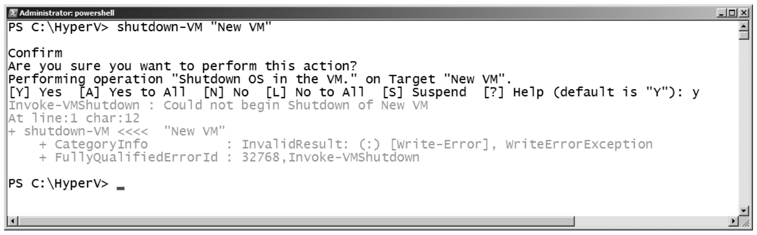


FIGURE 10.30 Shutting down a VM with ICs

```
PS C:\HyperV> shutdown-VM "Windows XP"


 Waiting for heartbeat
    Windows XP

    Stopping
```

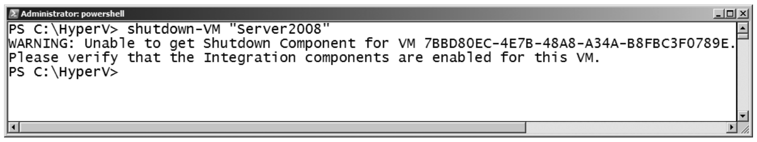
A VM without installed and running ICs can't take advantage of shutdown integration. An unenlightened VM must be shut down from within the VM or via another means (perhaps simply turned off). Currently, only supported versions of Windows with installed ICs support integrated shutdown. For the example in Figure 10.31, `new-VM` is a VM with no operating system (or ICs) installed. Attempting to shut down this VM using the HyperV library fails.



```
PS C:\HyperV> shutdown-VM "New VM"

Confirm
Are you sure you want to perform this action?
Performing operation "Shutdown OS in the VM." on Target "New VM".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
Invoke-VMShutdown : Could not begin Shutdown of New VM
At line:1 char:12
+ shutdown-VM <<<<  "New VM"
    + CategoryInfo          : InvalidResult: (:) [Write-Error], WriteErrorException
    + FullyQualifiedErrorId : 32768,Invoke-VMShutdown

PS C:\HyperV> _
```

Figure 10.31
Failed shutdown request

You also can't shut down VMs with ICs if the ICs aren't available. For example, if VMs are in a saved (suspended) state, the ICs are unavailable (see Figure 10.32).



```
PS C:\HyperV> shutdown-VM "Server2008"
WARNING: Unable to get Shutdown Component for VM 7BBD80EC-4E7B-48A8-A34A-B8FBC3F0789E.
Please verify that the Integration components are enabled for this VM.
PS C:\HyperV>
```

Figure 10.32
Failed shutdown—all VMs are suspended

Managing VHDs

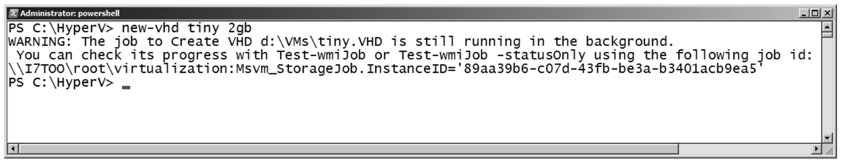
The common container for storing a VM-accessible disk is the VHD file. You can create, change, test, and compact these disks while they aren't in use by a VM. Table 10.4 lists common VHD management functions.

Table 10.4: Common Storage Management Functions in the Hyper-V Library

Function Name	Description
`new-VHD`	Creates a new VHD file
`compress-VHD`	Compacts a VHD file
`convert-VHD`	Changes to/from fixed or dynamic; creates new VHD
`expand-VHD`	Increases the size of a VHD
`get-VHDInfo`	Retrieves information about a VHD
`merge-VHD`	Merges a child with a parent disk (untested at time of writing)
`mount-VHD`	Mounts a VHD on a host for access
`unmount-VHD`	Unmounts a VHD from a host
`test-VHD`	Validates the integrity of a VHD file

You learned how to create a new VHD file using `new-VHD` as part of VM provisioning (see "Creating a Bare-Bones VM" earlier in this chapter). As you're likely aware, you can create VHD files with either a static (fixed) or dynamic size; dynamic is the default. To create a new VHD file, you must supply a name and the desired size (see Figure 10.33) in the default directory for virtual hard disks. The following code creates a dynamic VHD named `tiny` with a size of 2 GB:

```
new-vhd tiny 2gb
```



```
PS C:\HyperV> new-vhd tiny 2gb
WARNING: The job to Create VHD d:\VMs\tiny.VHD is still running in the background.
 You can check its progress with Test-wmiJob or Test-wmiJob -statusOnly using the following job id:
\\I7TOO\root\virtualization:Msvm_StorageJob.InstanceID='89aa39b6-c07d-43fb-be3a-b3401acb9ea5'
PS C:\HyperV> _
```

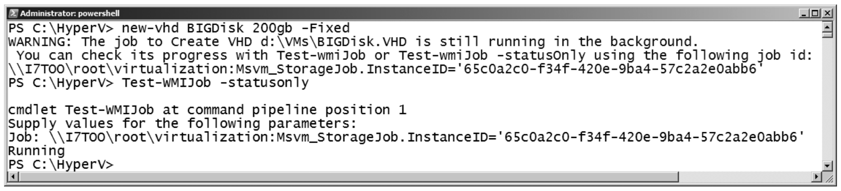
FIGURE 10.33 Using the `new-VHD` function

You may notice that the call to `new-VHD` spawns a job that runs in the background. Some Hyper-V administrative tasks (like VHD creation) can take a long time. In the case of `new-VHD`, you can opt to have your script wait for the task to complete by using the `-wait` parameter:

```
New-VHD "big" 20GB -fixed -wait
```

You can periodically check the status of the WMI job by using the included `test-WMIjob` function (see Figure 10.34):

```
new-vhd BIGDisk 200gb -Fixed
Test-WMIJob $Diskjob
```



```
PS C:\HyperV> new-vhd BIGDisk 200gb -Fixed
WARNING: The job to Create VHD d:\VMs\BIGDisk.VHD is still running in the background.
 You can check its progress with Test-wmiJob or Test-wmiJob -statusOnly using the following job id:
\\I7TOO\root\virtualization:Msvm_StorageJob.InstanceID='65c0a2c0-f34f-420e-9ba4-57c2a2e0abb6'
PS C:\HyperV> Test-WMIJob -statusonly

cmdlet Test-WMIJob at command pipeline position 1
Supply values for the following parameters:
Job: \\I7TOO\root\virtualization:Msvm_StorageJob.InstanceID='65c0a2c0-f34f-420e-9ba4-57c2a2e0abb6'
Running
PS C:\HyperV>
```

FIGURE 10.34 Using the `test-WMIjob` function

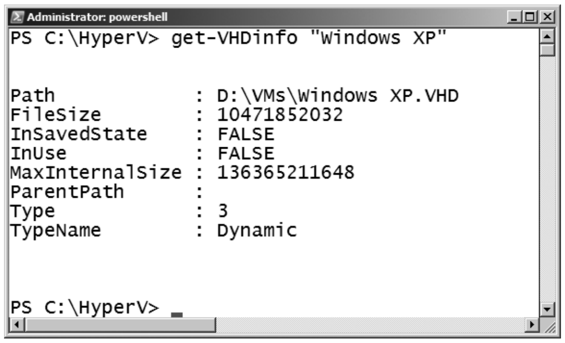
`get-VHDinfo` can provide basic information about a VHD file, including the actual file size, the maximum internal size, the type, and whether it's in use at a given time (see Figure 10.35):

```
get-vhdinfo "Windows XP"
```

Just like `new-VHD`, `get-VHDinfo` defaults to the Hyper-V virtual hard disk directory.

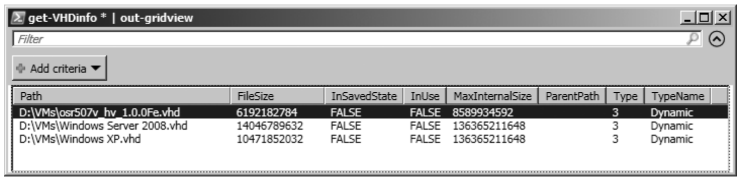
Monitoring storage used by VHDs can be a critical management function, particularly when you're using dynamic VHDs. Unexpected VHD growth on a shared physical disk can lead to performance issues for all VMs homed there. You can create a tiny and useful VHD storage report like the one in Figure 10.36:

```
get-vhdinfo *| out-gridview
```



```
PS C:\HyperV> get-VHDinfo "Windows XP"


Path            : D:\VMs\Windows XP.VHD
FileSize        : 10471852032
InSavedState    : FALSE
InUse           : FALSE
MaxInternalSize : 136365211648
ParentPath      :
Type            : 3
TypeName        : Dynamic



PS C:\HyperV> _
```

FIGURE 10.35
`get-VHDinfo` retrieves basic information.



Path	FileSize	InSavedState	InUse	MaxInternalSize	ParentPath	Type	TypeName
D:\VMs\osr507v_hv_1.0.0Fe.vhd	6192182784	FALSE	FALSE	8589934592		3	Dynamic
D:\VMs\Windows Server 2008.vhd	14046789632	FALSE	FALSE	136365211648		3	Dynamic
D:\VMs\Windows XP.vhd	10471852032	FALSE	FALSE	136365211648		3	Dynamic

FIGURE 10.36
VHD size report

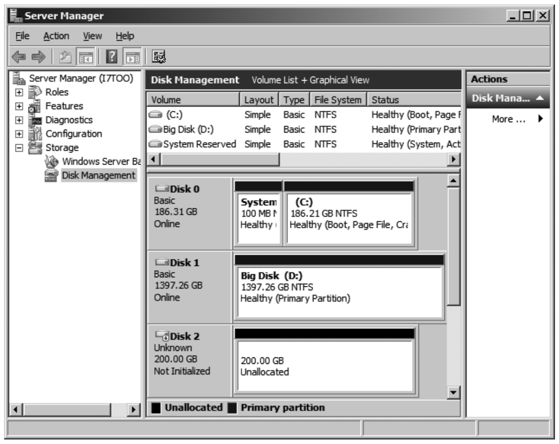
If your VHDs reside somewhere other than in the default virtual hard disk directory, you can get a similar report by stringing together the path and additional formatting information:

```
dir "d:\VMs\*.vhd" | get-vhdinfo *| out-gridview
```

It's useful to access the contents of a VHD file from the virtualization host as if it were a locally attached drive. Being able to add files to or remove files from a VHD without starting a VM can save you time and can facilitate offline maintenance. You can use `mount-VHD` and `unmount-VHD` to simplify the mounting of local VHD files:

```
Mount-VHD HUGE
```

The `mount-VHD` function doesn't return much useful information in the PowerShell user interface, but you can see the VHD listed as a volume within Server Manager's Disk Management (Figure 10.37).



Volume	Layout	Type	File System	Status
(C:)	Simple	Basic	NTFS	Healthy (Boot, Page F
Big Disk (D:)	Simple	Basic	NTFS	Healthy (Primary Part
System Reserved	Simple	Basic	NTFS	Healthy (System, Act

Disk 0
Basic
186.31 GB
Online

System
100 MB N
Healthy (

(C:)
186.21 GB NTFS
Healthy (Boot, Page File, Cra

Disk 1
Basic
1397.26 GB
Online

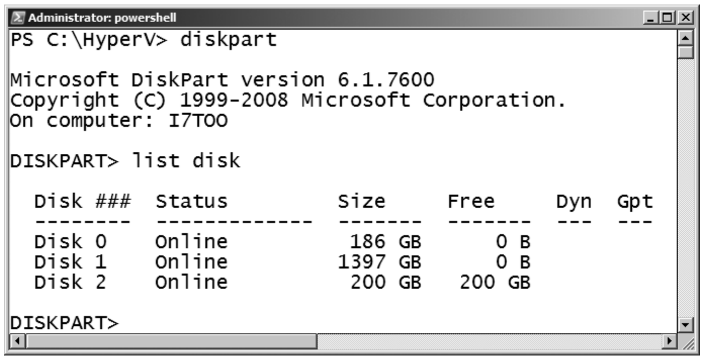
Big Disk (D:)
1397.26 GB NTFS
Healthy (Primary Partition)

Disk 2
Unknown
200.00 GB
Not Initialized

200.00 GB
Unallocated

FIGURE 10.37
Mounted VHD in Server Manager

This mounted VHD wasn't formatted as part of the creation process (it hadn't yet been exposed to an operating system installation process). After mounting, you can locate the VHD on the host either by using the Computer Management console or by using the `diskpart` command (see Figure 10.38). Once you identify the volume, you can then perform additional storage tasks from the host.



```
PS C:\HyperV> diskpart

Microsoft DiskPart version 6.1.7600
Copyright (C) 1999-2008 Microsoft Corporation.
On computer: I7TOO

DISKPART> list disk

  Disk ###  Status         Size     Free     Dyn  Gpt
  --------  -------------  -------  -------  ---  ---
  Disk 0    Online          186 GB      0 B
  Disk 1    Online         1397 GB      0 B
  Disk 2    Online          200 GB   200 GB

DISKPART>
```

FIGURE 10.38 Using `diskpart` to see information about a mounted VHD

NOTE The capability to mount a VHD within the parent partition is so useful that it was added to Server Manager in Windows Server 2008 R2. The functions `dismount-VHD`, `compress-VHD`, `convert-VHD`, `expand-VHD`, `test-VHD`, and `merge-VHD` all help you manage and alter your VHDs in ways consistent with their names. For more information about how to use them, you can review the examples contained in the Hyper-V library.

Maintaining Virtual Systems

Your virtual systems depend on you to keep them properly maintained. Automating required configuration changes, installing software updates, and sometimes rolling back changes are all tasks necessary to keep physical systems and VMs running efficiently.

Configuration Changes

Business and technical pressures may require you to alter the configurations of existing VMs. Perhaps the applications on a given VM require more RAM or CPU resources. Maybe a physical NIC is experiencing a high network load, and some VM traffic must be offloaded to a new interface. Earlier in the chapter, we reviewed functions for automating configuration changes for VMs. Table 10.5 lists functions commonly used from the library to define or alter the configuration of a VM.

TABLE 10.5: HyperV Library VM Configuration Management Functions

FUNCTION NAME	DESCRIPTION
`add-VMdisk`	Adds a disk (VHD or ISO) to a defined drive
`add-VMdrive`	Adds a drive to a defined controller

Table 10.5: HyperV Library VM Configuration Management Functions *(continued)*

Function Name	Description
`add-VMKVP`	Adds key/value pairs to be sent to a VM
`add-VMfloppydisk`	Adds a floppy disk
`add-VMnewharddisk`	Creates a new VHD and attaches it to a VM
`add-VMNIC`	Adds a NIC to a VM
`add-VMpassthrough`	Connects pass-through disk to a VM
`add-VMRASD`	Adds hardware described by virtual Resource Allocation Setting Data
`add-VMSCSIcontroller`	Adds a synthetic SCSI controller
`add-ZIPcontent`	Adds content to a ZIP file (extra bonus!)
`new-VFD`	Creates a new virtual floppy drive
`new-VHD`	Creates a new VHD file
`new-VM`	Creates a new VM
`new-VMswitchport`	Creates a new virtual switch port
`set-VM`	Sets the BIOS boot order and startup/shutdown actions
`set-VMCPUcount`	Sets the CPU count (1–4)
`set-VMdisk`	Changes the configuration of an existing disk
`set-VMintegrationcomponent`	Enables/disables ICs for one or more VMs
`set-VMmemory`	Sets the amount of RAM
`set-VMNICaddress`	Sets the MAC address for a virtual NIC
`set-VMNICswitch`	Connects a NIC to a virtual switch
`set-VMNICVLAN`	Sets VLAN ID for a NIC
`set-VMRASD`	Changes virtual hardware described by Resource Allocation Setting Data to a VM
`set-VMSerialPort`	Connects serial port to a named pipe
`set-VMNICSwitch`	Connects NIC to a virtual switch

Patching

You can update the software components of a running VM the same way you do for a physical system. You can also patch VMs while they're offline using the Offline Virtual Machine Servicing Tool, discussed briefly in Chapter 4, "Utilizing Virtualization Best Practices." You can use `mount-VHD` to access the disk for offline VMs and prestage software for later installation.

Snapshots

Hyper-V can create point-in-time VM snapshots. You can use retained snapshot information to revert a VM to a known state in the past. Table 10.6 lists the snapshot-related functions found in the library.

Table 10.6: HyperV Library Snapshot Management Functions

Function Name	Description
`merge-VHD`	Merges VHDs from snapshots
`restore-VMsnapshot`	Reverts to a previous snapshot
`choose-VMsnapshot`	Selects a snapshot
`get-VMsnapshot`	Accesses VM snapshot information
`get-VMsnapshottree`	Accesses VM snapshot information and shows it as a tree
`new-VMsnapshot`	Creates a new snapshot
`remove-VMsnapshot`	Deletes a snapshot
`rename-VMsnapshot`	Changes the name of an existing snapshot
`update-VMsnapshot`	Creates a new snapshot using an existing name

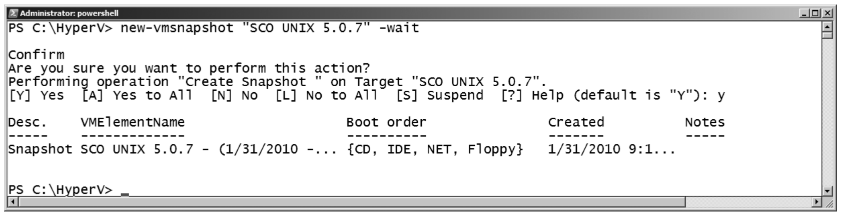
NOTE Snapshots aren't backups. We hate snapshots! They are very useful for development and testing purposes (to roll back changes), but they aren't suitable for all situations and should never ever be used in production scenarios.

Creating a snapshot is a straightforward process of calling `new-VMsnaphot` and specifying or passing the target VMs to be snapped (see Figure 10.39):

```
new-vmsnapshot "SCO UNIX 5.0.7" -wait
```

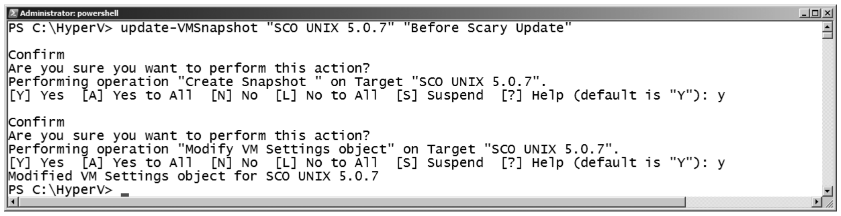
A more useful function for creating new snapshots may be `update-VMsnapshot`. This function creates a new snapshot and alters the displayed name to something of your choosing (see Figure 10.40 and Figure 10.41):

```
Update-VMSnapshot "SCO UNIX 5.0.7" "Before Scary Update"
```



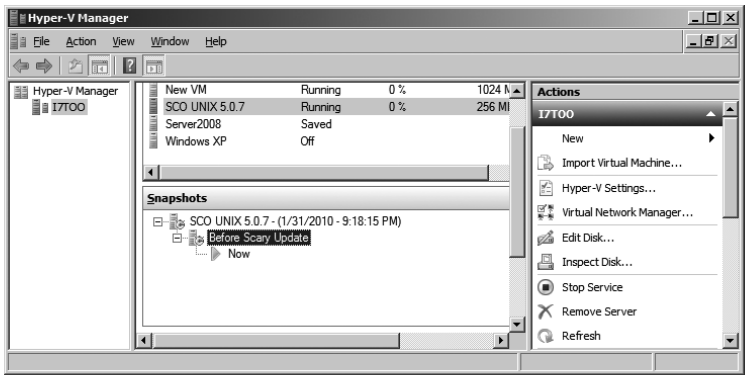
```
PS C:\HyperV> new-vmsnapshot "SCO UNIX 5.0.7" -wait

Confirm
Are you sure you want to perform this action?
Performing operation "Create Snapshot " on Target "SCO UNIX 5.0.7".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

Desc.    VMElementName                  Boot order              Created         Notes
-----    -------------                  ----------              -------         -----
Snapshot SCO UNIX 5.0.7 - (1/31/2010 -... {CD, IDE, NET, Floppy}   1/31/2010 9:1...


PS C:\HyperV> _
```

FIGURE 10.39
New snapshot



```
PS C:\HyperV> update-VMSnapshot "SCO UNIX 5.0.7" "Before Scary Update"

Confirm
Are you sure you want to perform this action?
Performing operation "Create Snapshot " on Target "SCO UNIX 5.0.7".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

Confirm
Are you sure you want to perform this action?
Performing operation "Modify VM Settings object" on Target "SCO UNIX 5.0.7".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
Modified VM Settings object for SCO UNIX 5.0.7
PS C:\HyperV> _
```

FIGURE 10.40
New snapshot with specified name



```
New VM          Running   0 %   1024 M
SCO UNIX 5.0.7  Running   0 %   256 MI
Server2008      Saved
Windows XP      Off
```

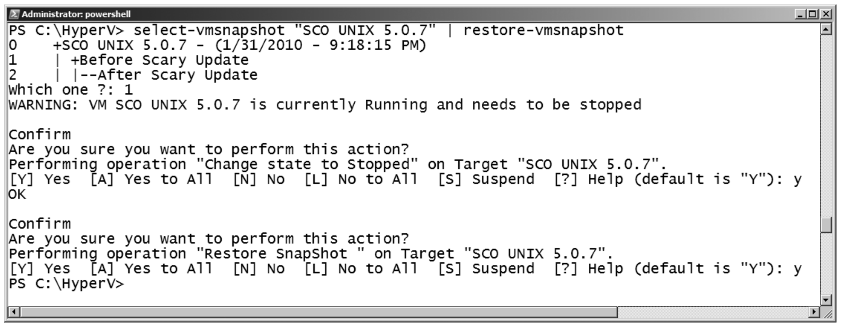
Snapshots

```
SCO UNIX 5.0.7 - (1/31/2010 - 9:18:15 PM)
  Before Scary Update
    Now
```

FIGURE 10.41
New snapshot In Hyper-V GUI

Reverting and applying snapshots is also simplified with the library when you use `select-VMsnapshot` in conjunction with `restore-VMsnapshot` (see Figure 10.42):

```
select-vmsnapshot "SCO UNIX 5.0.7" | Restore-VMSnapshot
```



```
PS C:\HyperV> select-vmsnapshot "SCO UNIX 5.0.7" | restore-vmsnapshot
0    +SCO UNIX 5.0.7 - (1/31/2010 - 9:18:15 PM)
1    | +Before Scary Update
2    | |--After Scary Update
Which one ?: 1
WARNING: VM SCO UNIX 5.0.7 is currently Running and needs to be stopped

Confirm
Are you sure you want to perform this action?
Performing operation "Change state to Stopped" on Target "SCO UNIX 5.0.7".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
OK

Confirm
Are you sure you want to perform this action?
Performing operation "Restore SnapShot " on Target "SCO UNIX 5.0.7".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y
PS C:\HyperV>
```

FIGURE 10.42
Restoring and applying snapshots

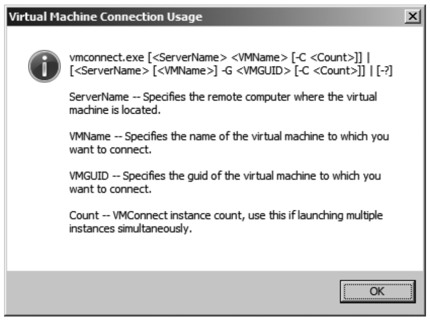
Managing Access

Controlling access to VMs is an important task for IT managers. When you encapsulate an entire system into a single file, you run into new security challenges. Poorly secured network links to a physical host or shaky backup processes can quickly bypass locked data-center doors. If you have bad security practices, entire virtual systems can be pilfered without detection.

Virtualization enables a new mechanism for system access: remote desktop interaction. You can remotely view and interact with the console of a VM. Before virtualization, server consoles could be secured in a data center or computer room unless remote-access tools (IP-based keyboard/video/mouse [KVM] or lights-out/remote console hardware) were employed. Virtualization can create a security opportunity for these formerly inaccessible system consoles.

Securing access to VM consoles is just as important as other means of securing virtual systems. Virtual hosting of desktops also requires careful access control management for VMs. Chapter 5, "Securing Hyper-V," discusses access management for individual VMs. After you set up proper security for users or administrative access, you can start remote display sessions to a particular VM by calling `VMConnect.exe`. The `VMConnect` client is typically found in `C:\Program Files\Hyper-V`, and it has its own set of command-line options, which are shown in Figure 10.43.

Figure 10.43 `VMConnect` parameters



```
vmconnect.exe [<ServerName> <VMName> [-C <Count>]] |
[<ServerName> [<VMName>] -G <VMGUID> [-C <Count>]] | [-?]
```

ServerName -- Specifies the remote computer where the virtual machine is located.

VMName -- Specifies the name of the virtual machine to which you want to connect.

VMGUID -- Specifies the guid of the virtual machine to which you want to connect.

Count -- VMConnect instance count, use this if launching multiple instances simultaneously.

You can create a remote access session without calling `VMConnect.exe` directly by using the `new-VMconnectsession` function.

Migration

In many situations, you need to migrate a VM from one physical server to another. Hardware failures, capacity limitations, and maintenance are all reasons to relocate a VM. You can automate the move of a VM between systems in a number of ways. Common methods include importing/exporting (discussed in Chapter 6, "Migrating Virtual Machines"), using failover clustering (covered in Chapter 8, "High Availability)," performing a simple file copy, or undertaking a virtual to virtual (V2V) migration. SCVMM 2008 and SCVMM 2008 R2 support the automation of VM migration better than any other solution; we'll cover it in Chapter 11.

Simple File Copy

VM information lives in files. Why not copy or move the files that define a VM from one host to another? Finding all the necessary files and ensuring they're properly migrated can be a complicated task. Guaranteeing the VM is in a movable state (off or saved) is important, as is handling host-specific dependencies such as the migration of virtual network resources and security settings.

Copying all the files from one host to another doesn't work without careful coordination. The export and import capabilities exposed through the Hyper-V Manager handle these checks fairly well, and they're a supported way to migrate VMs. SCVMM also has a supported move process as well as V2V capability.

Still, you may choose to move a VM using an entirely unsupported copy process. For more information about how to do this, review the `diskshadow` backup and recovery process detailed in Chapter 7, "Backing Up and Recovering VMs." Using `xcopy` parameters can address the specifics of file security issues, but you're still likely to run into issues.

Export/Import

Exporting a VM from one host and importing it on another is perhaps the simplest and cleanest migration method to automate without SCVMM. Exporting requires that a VM be either off or in a saved state. The HyperV library supports the exporting and importing of VMs by using the `export-VM` and `import-VM` functions. Calling `export-VM` requires a reference to the VM to be exported as well as the export path, with optional parameters that include the name of the physical server, whether to export state (VHDs), and the ability to wait for the process to complete:

```
export-VM "New VM" c:\exports -server localhost -copystate -wait
```

The `import-VM` function only requires the path to the exported VM as a parameter, with optional parameters similar to those of `export-VM`:

```
import-VM "C:\exports\New VM" -wait
```

SCVMM's ability to move VMs between hosts masks the storage and security dependency of Hyper-V export and is a more suitable approach for automating migrations in many environments.

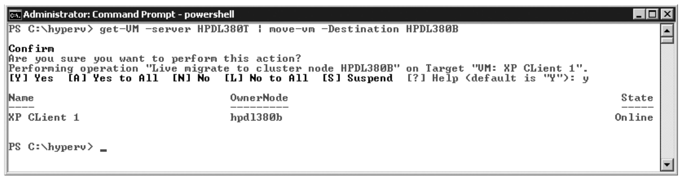
Failover Clustering

Failover clustering facilitates the migration of VMs from one physical host to another with limited or no perceptible downtime, but it requires preconfiguration. Hyper-V clusters are presented in Chapter 8. Hyper-V failover clustering (Quick Migration and Live Migration) requires identically configured physical hosts as well as shared storage. You typically automate cluster management tasks—creation, configuration, and workload migration—using `cluster.exe` or via PowerShell. Clustering tasks can also be performed using the failover-clustering WMI provider (`root\MSCluster`). SCVMM provides cluster management capabilities for Hyper-V that you can automate using its set of Windows PowerShell cmdlets.

The R2 Gold version of the library includes failover cluster management functions, such as `get-VMclustergroup`, `get-VMlivemigrationnetwork`, `move-VM`, `select-clustersharedvolume`, `select-VMlivemigrationnetwork`, and `sync-VMclusterconfig`. A simple, real-world example of the cluster management functions is the common requirement to migrate all VMs from a failover cluster node. In our test cluster, we have one VM still running on a system (HPDL380T) that we need to migrate in order to perform hardware maintenance. The following command migrates all clustered VMs from HPDL380T to the other node in the cluster (HPDL380B), as shown in Figure 10.44:

```
Get-VM -server HPDL380T | move-vm -Destination HPDL380B
```

FIGURE 10.44
Moving all VMs from a node



```
PS C:\hyperv> get-VM -server HPDL380T | move-vm -Destination HPDL380B

Confirm
Are you sure you want to perform this action?
Performing operation "Live migrate to cluster node HPDL380B" on Target "VM: XP CLient 1".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): y

Name                                OwnerNode                                         State
----                                ---------                                         -----
XP CLient 1                         hpdl380b                                         Online


PS C:\hyperv> _
```

Virtual to Virtual Migration

Virtual to virtual (V2V) migrations are similar to physical to virtual (P2V) migrations; they're discussed in Chapter 6. Automating V2V migrations is a tricky process and is largely unnecessary if you're moving a VM from one host to another. You can also automate V2V migrations with SCVMM cmdlets.

Backup and Recovery

In Chapter 7, we covered backup and recovery without the use of enterprise tools. System Center Data Protection Manager (DPM) is the best option for enterprise-class backup with Hyper-V; we discuss that process in Chapter 12.

Earlier versions of the library included a `diskshadow` script-generating function (`get-VMbackupscript`). Calling the function generated a `diskshadow` script similar to the one used in Chapter 7. It has been removed from the R2 Gold version of the library, so you should look to the processes in Chapter 7 or Chapter 12 for more sustainable backup and recovery processes.

Collecting and Monitoring Data

Monitoring how your virtual environment performs is key to ensuring smooth operation. You've seen how to locate virtual hosts, enumerate their child VMs, and access configuration information. Visibility into the health and performance of each VM is also important.

Enterprise tools like those found in the Microsoft System Center family are the best solution for collecting, analyzing, and monitoring health and performance. System Center Operations Manager (SCOM), which can be connected to SCVMM, can serve as a repository for historical performance data for your entire Microsoft-centric computing environment. SCOM 2007 R2 also has the capability to monitor systems running common Linux and UNIX operating systems, so monitoring non-Microsoft-centric VMs is also possible. It is challenging to do comprehensive monitoring if you don't have access to System Center tools.

Viewing the Desktop

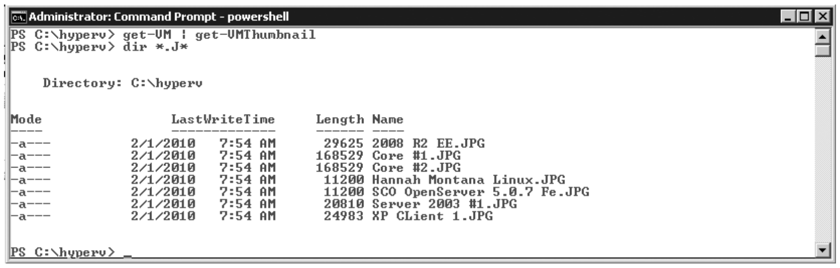
Before you go too far down the path of data collection, you have to please corporate management. Data centers and operations rooms often have banks of monitors filled with color images including graphs, charts, maps, and system consoles. A dirty little secret of many of these rooms is that certain screens are simply for show—some of the big, blinking displays exist only to create the appearance of a well-monitored environment.

Another truth of large data centers is that operators (personnel who work regularly in the computer room) often need to see what is on the screen of a particular system. It's also a reality that these employees aren't always trusted to interact with these same systems for regulatory reasons or by management mandate.

The virtualization provider allows you to request a JPEG picture of a VM's desktop. You can capture and display these images in any number of ways, and they can be useful to operators and administrators for auditing/monitoring purposes (or handy to show on a large display). The `get-VMthumbnail` function creates a JPEG file of a running VM with a name based on the VM's display (element) name and writes it to the current directory (as shown in Figure 10.45):

```
Get-VM | Get-VMThumbnail
```

FIGURE 10.45
Using `get-VMthumbnail` to create images of VM desktops



```
PS C:\hyperv> get-VM | get-VMThumbnail
PS C:\hyperv> dir *.J*


    Directory: C:\hyperv


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---          2/1/2010   7:54 AM      29625 2008 R2 EE.JPG
-a---          2/1/2010   7:54 AM     168529 Core #1.JPG
-a---          2/1/2010   7:54 AM     168529 Core #2.JPG
-a---          2/1/2010   7:54 AM      11200 Hannah Montana Linux.JPG
-a---          2/1/2010   7:54 AM      11200 SCO OpenServer 5.0.7 Fe.JPG
-a---          2/1/2010   7:54 AM      20810 Server 2003 #1.JPG
-a---          2/1/2010   7:54 AM      24983 XP CLient 1.JPG


PS C:\hyperv> _
```

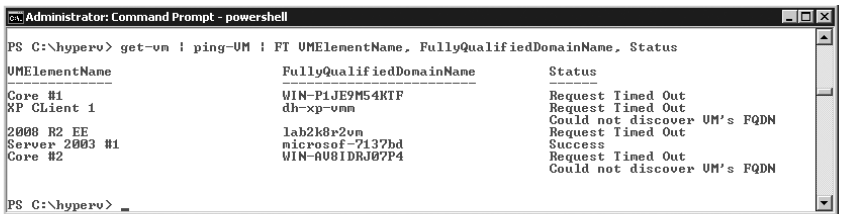
You can post the generated image files to a website, save them into SharePoint, or easily view them using Windows Explorer. Regularly capturing an image of the desktop can be useful for troubleshooting or compliance purposes.

Testing for Service

Ping is often the first monitoring tool most administrators use to check the health of a system. It's not a comprehensive test, but it can show that a particular system's TCP/IP stack is accessible under normal circumstances, as well as point out environmental issues on a network (name resolution, routing, firewall settings, or latency challenges). The `ping-VM` function makes it convenient to ping configured VMs (see Figure 10.46):

```
get-vm | ping-VM |
FT VMElementName, FullyQualifiedDomainName, Status -auto
```

FIGURE 10.46
Using `ping-VM` to check the status of configured VMss



```
PS C:\hyperv> get-vm | ping-VM | FT VMElementName, FullyQualifiedDomainName, Status

VMElementName                     FullyQualifiedDomainName          Status
-------------                     ------------------------          ------
Core #1                           WIN-P1JE9M54KTF                   Request Timed Out
XP CLient 1                       dh-xp-vmm                         Request Timed Out
                                                                    Could not discover VM's FQDN
2008 R2 EE                        lab2k8r2vm                        Request Timed Out
Server 2003 #1                    microsof-7137bd                   Success
Core #2                           WIN-AV8IDRJ07P4                   Request Timed Out
                                                                    Could not discover VM's FQDN


PS C:\hyperv> _
```

TIP Firewall status for contemporary versions of Microsoft Windows may not allow a response from Ping, so this function may not provide much value in a secure environment.

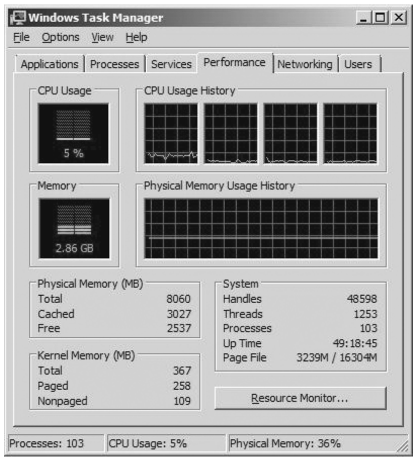
You can also use `test-VMheartbeat` and `get-VMKVP` to verify that a VM is running and functioning (`get-VMKVP` was discussed earlier in the chapter). You can think of `test-VMheartbeat` as a sort of a ping to the ICs running in a VM. If the heartbeat component included in the ICs is functioning, the test is successful. O'Neill has included a timeout parameter for managing the startup sequence of VMs. Using `test-VMheartbeat`, you can stagger an environment's power-up until key network services are ready:

```
start-vm "TestDC"
Test-vmheartBeat "TestDC" -Timeout 300
start-vm "TestExchange"
```

TIP Staggering/delaying the startup of a VM is such a common need that O'Neill recently added more options to the `start-VM` function. You can alternately use `start-VM "Test-DC" - wait -Heartbeat 300` to achieve the same result.

Accessing Processor Performance Data

You may have noticed that the memory used by VMs is reflected on the Windows Task Manager's Performance tab. Every time you start a VM, the amount of memory used by the physical system increases, which is reflected by the Windows Task Manager. This isn't true for processor (CPU) utilization: the CPU load of child VMs isn't reflected in the Windows Task Manager of the host/parent. Figure 10.47 shows the CPU and memory usage history of a quad-core system.



CPU Usage: 5 %
Memory: 2.86 GB

```
Physical Memory (MB)            System
Total        8060               Handles              48598
Cached       3027               Threads               1253
Free         2537               Processes              103
                                Up Time           49:18:45
Kernel Memory (MB)              Page File    3239M / 16304M
Total         367
Paged         258
Nonpaged      109

Processes: 103    CPU Usage: 5%    Physical Memory: 36%
```

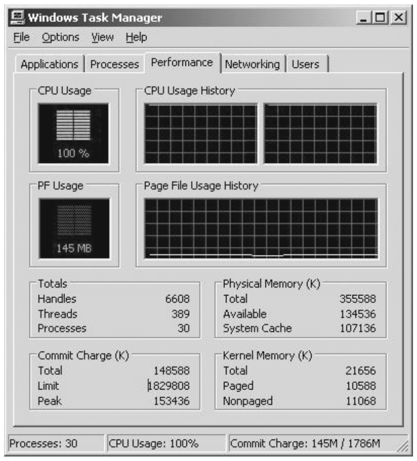
FIGURE 10.47
Host's Task Manager: low load

The Task Manager reflects relatively low CPU usage. But the system is the host for a two-processor Server 2003 VM running at more than 90 percent processor load (Figure 10.48 is the Task Manager from within the VM).

You could query each individual VM remotely to access and retrieve the CPU load, but that wouldn't give you an accurate view of the actual load on the host. It would also require network access to the VM as well as an appropriate level of security.

You can access information about the performance of individual VM virtual processors through the parent using the virtualization provider and the `MSVM_Processor` class, as reflected by the WMI query in Figure 10.49:

```
GWMI –Class MSVM_Processor –Namespace root\virtualization |
ft SystemName,LoadPercentage -auto
```



100 %

145 MB

Totals
Handles 6608
Threads 389
Processes 30

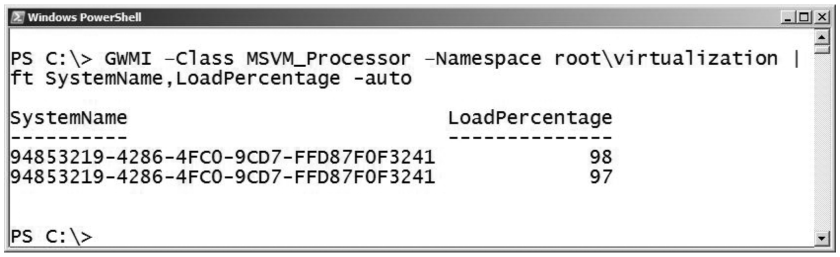
Physical Memory (K)
Total 355588
Available 134536
System Cache 107136

Commit Charge (K)
Total 148588
Limit 1829808
Peak 153436

Kernel Memory (K)
Total 21656
Paged 10588
Nonpaged 11068

Processes: 30 | CPU Usage: 100% | Commit Charge: 145M / 1786M

FIGURE 10.48
VM: high CPU load



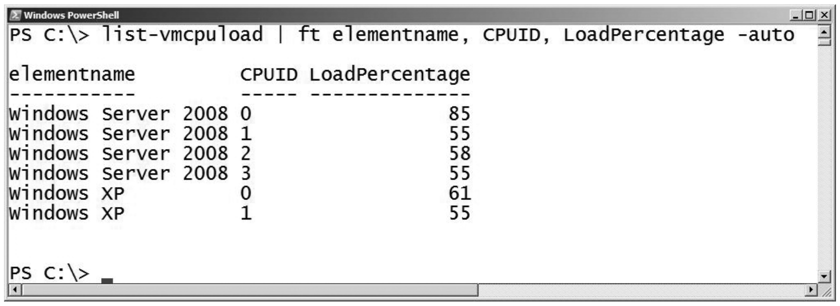
```
PS C:\> GWMI -Class MSVM_Processor -Namespace root\virtualization |
ft SystemName,LoadPercentage -auto

SystemName                           LoadPercentage
----------                           --------------
94853219-4286-4FC0-9CD7-FFD87F0F3241             98
94853219-4286-4FC0-9CD7-FFD87F0F3241             97


PS C:\>
```

FIGURE 10.49
CPU load-percentage query

The query shows the CPU load percentage and the unfriendly name of the associated VM. `HyperV.PS1` didn't include any processor performance-related functions, but O'Neill has enhanced newer versions of the library with the `get-VMprocessor` function (we'll show you that later). The following is a sample Windows PowerShell function that collects the virtual processor information from a local host and ties in the VM name:

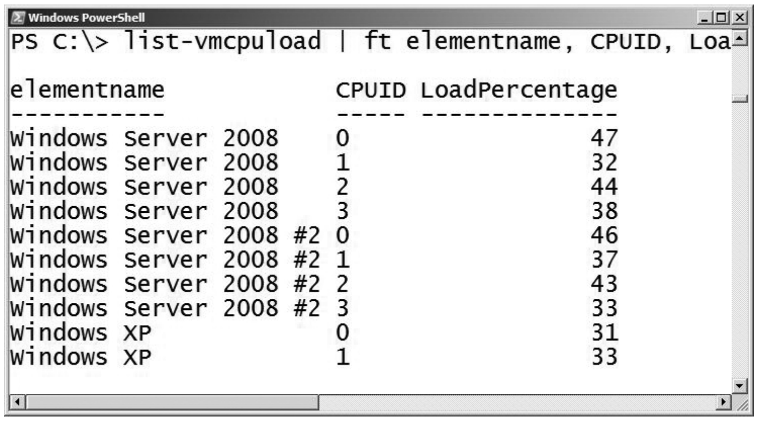
```
function List-VMCPULoad
{Param ($server=".")
   $Procs= GWMI -computerName $server -Namespace ↩
root\virtualization -Class MSVM_Processor
   foreach ($Proc in $Procs) {
     GWMI -computerName $server -Namespace ↩
root\virtualization -Query ↩
"Select * From MSVM_ComputerSystem Where Name = '$($Proc.SystemName)'" |
     add-member -passthru noteproperty "Load%" $Proc.LoadPercentage |
     add-member -passthru noteproperty "CPUID" $Proc.deviceid.split("\")[-1]
  }
}

list-vmcpuload | ft elementname, CPUID, Load% -auto
```

This function lists all individual running virtual processors on a system, showing each VM name. This may not be entirely useful in developing a clear picture of processor performance, because other processes on the physical host (including other VMs) can reduce the available compute cycles. This interference can artificially reduce the `LoadPercentage` value retrieved from `MSVM_Proccessor`. For example, two VMs are listed in Figure 10.50 (one with four cores and one with two) using about as much CPU as they're allowed (the six virtual cores are running on a host with only four cores). The individual `LoadPercentage` value for each virtual CPU can appear to be low (55 percent for one) because of the sharing of resources. The VMs themselves believe they're running at full steam, but `LoadPercentage` doesn't clearly reflect this. Adding an additional four-core VM under extreme CPU load makes this point more clearly, as shown in Figure 10.51.



```
PS C:\> list-vmcpuload | ft elementname, CPUID, LoadPercentage -auto

elementname         CPUID LoadPercentage
-----------         ----- --------------
Windows Server 2008 0                 85
Windows Server 2008 1                 55
Windows Server 2008 2                 58
Windows Server 2008 3                 55
Windows XP          0                 61
Windows XP          1                 55


PS C:\> _
```

FIGURE 10.50
Nice CPU load percentage query



```
PS C:\> list-vmcpuload | ft elementname, CPUID, Loa

elementname            CPUID LoadPercentage
-----------            ----- --------------
Windows Server 2008    0                 47
Windows Server 2008    1                 32
Windows Server 2008    2                 44
Windows Server 2008    3                 38
Windows Server 2008 #2 0                 46
Windows Server 2008 #2 1                 37
Windows Server 2008 #2 2                 43
Windows Server 2008 #2 3                 33
Windows XP             0                 31
Windows XP             1                 33
```

FIGURE 10.51
More load, lower percentage

With 10 virtual cores all taxed and competing for the power of the 4 physical cores (along with processes on the parent partition), the `LoadPercentage` value is reduced. Looking solely at the `LoadPercentage` of a single VM can mislead you into believing that a VM isn't low on processing power. To get the actual utilization of each physical processor, it's recommended that you don't use the virtualization provider but instead use the tried-and-true Common Information Model (CIM) version 2.

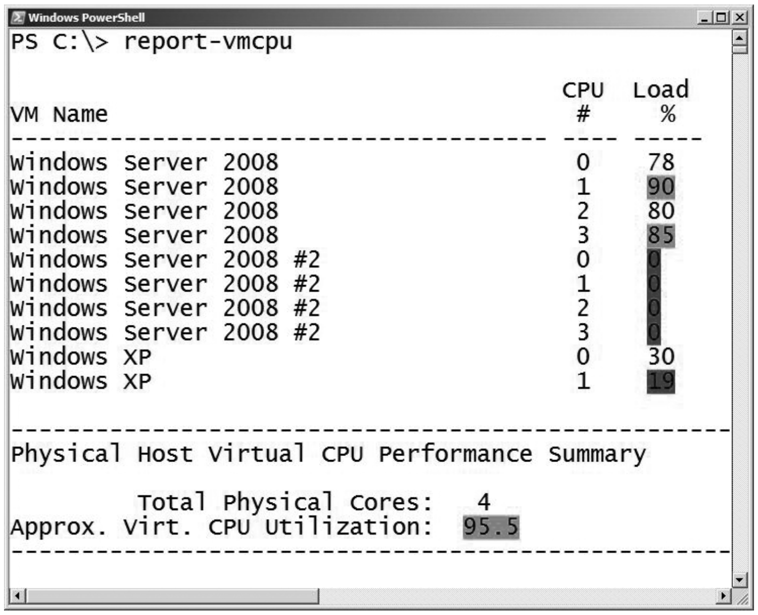
TIP Several good performance resources describe how to access counters and troubleshoot performance issues for Hyper-V, including "Measuring Performance on Hyper-V" (`http://msdn.microsoft.com/en-us/library/cc768535.aspx`) and an "All Topics Performance" post titled "How to Get Processor Utilization for Hyper-V via WMI" (`http://blogs.msdn.com/tvoellm/archive/2008/07/14/how-to-get-processor-utilization-for-hyper-v-via-wmi.aspx`). They're both informative and can help you create a comprehensive and accurate view of CPU utilization. They also have too much math and too many formulas for day-to-day use.

Accessing the WMI `CIMv2` class `Win32_PerfRawData_HVStats_HyperVHypervisorLogicalProcessor` is the recommended approach, but the formulas to derive processor utilization are a hassle. You can approximate the overall utilization by adding together the `loadPercentage` values from each virtual processor and dividing the total by the number of physical cores. The following function creates a useful CPU utilization report with color coding to connote high CPU load on individual virtual CPUs, as well as on the host system (see Figure 10.52):

```
function Report-VMCPU
{Param ($server=".")
   $LoadSum = 0
   $PCores = 0
   $VProcs= GWMI -computerName $server –Namespace ↩
root\virtualization -Class MSVM_Processor
   write-host "`n                                     CPU  Load"
   write-host "VM Name                             #     %"
   write-host "----------------------------------- ---- -----"
   foreach ($VProc in $VProcs) {
      $VM = GWMI -computerName $server -Namespace ↩
root\virtualization -Query ↩
"Select * From MSVM_ComputerSystem Where Name = '$($VProc.SystemName)'"
      $VMname = $VM.Elementname.PadRight(39," ")
      $VMCPU  = $VProc.deviceid.split("\")[-1]
      $VMCPULOAD = $VProc.LoadPercentage
      Write-Host "$VMname $VMCPU    "   -nonewline
      if ($VMCPULOAD -lt 30) {
         write-host $VMCPULOAD -backgroundcolor green
      }
      elseif ($VMCPULOAD -gt 80) {
         write-host $VMCPULOAD -backgroundcolor red
      }
      else {write-host $VMCPULOAD}
      $LoadSum = $LoadSum + $VMCPULOAD
   }
   $PProcs= GWMI -computerName $server –Namespace ↩
root\CIMV2 -Class Win32_Processor
   foreach ($PProc in $PProcs) {
      $PCores = $PCores + $PProc.NumberOfCores }
   $VLoad =  $LoadSum / $PCores
   write-host "`n-----------------------------------------------"
```

```
    Write-Host "Physical Host Virtual CPU Perf Summary" -nonewline
    if($Server -ne ".") {
       write-host " for $Server"} else {write-host " "}
    Write-Host "`n        Total Physical Cores: " $PCores
    Write-Host "Approx. Virt. CPU Utilization: " -nonewline
    if ($VLoad -lt 30) {write-host $VLoad -backgroundcolor green}
    elseif ($VLoad -gt 80) {write-host $VLoad -backgroundcolor red}
    else {write-host $VLoad}
    write-host "--------------------------------------------`n"
 }
```

FIGURE 10.52
Better CPU load report



```
PS C:\> report-vmcpu

                                        CPU  Load
VM Name                                  #     %
-------------------------------------- ---- -----
Windows Server 2008                      0    78
Windows Server 2008                      1    90
Windows Server 2008                      2    80
Windows Server 2008                      3    85
Windows Server 2008 #2                   0    0
Windows Server 2008 #2                   1    0
Windows Server 2008 #2                   2    0
Windows Server 2008 #2                   3    0
Windows XP                               0    30
Windows XP                               1    19

--------------------------------------------------
Physical Host Virtual CPU Performance Summary

         Total Physical Cores:   4
Approx. Virt. CPU Utilization:  95.5
--------------------------------------------------
```

The code may not reflect the acme of Windows PowerShell or mathematics elegance, but you can use the output to clearly show a high CPU load condition.

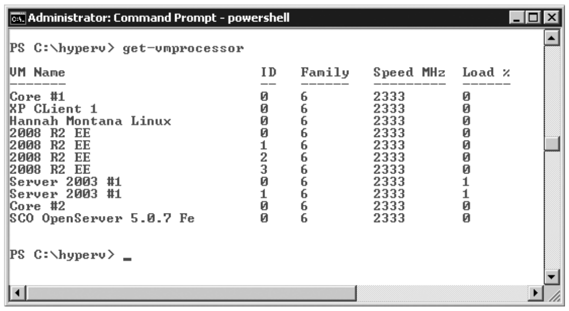
NOTE `MSVM_Processor` also includes `LoadPercentageHistory`, which is an array of recent measurements of `LoadHistory`.

As we mentioned earlier, James has included the `get-VMprocessor` function in later versions of the library, which generates a very similar report. Figure 10.53 shows the output of `get-VMprocessor` on a different system.

Performance Monitoring and PowerGadgets

The previous code sample produces some usable and ugly output. SoftwareFX sells a great set of inexpensive graphical tools, called PowerGadgets, that connect right into PowerShell. You can use PowerGadgets to quickly and easily create interactive tools using gauges, charts, graphs, and maps; you can then use these tools with Windows PowerShell to monitor and manage Hyper-V. Gadgets you create can even be added to the Vista or Windows 7 Sidebar. You can download an evaluation copy of PowerGadgets from the SoftwareFX website at `www.softwarefx.com`.

FIGURE 10.53 The `get-VMprocessor` output



```
PS C:\hyperv> get-vmprocessor

VM Name                         ID   Family   Speed MHz   Load %
-------                         --   ------   ---------   ------
Core #1                         0    6        2333        0
XP CLient 1                     0    6        2333        0
Hannah Montana Linux            0    6        2333        0
2008 R2 EE                      0    6        2333        0
2008 R2 EE                      1    6        2333        0
2008 R2 EE                      2    6        2333        0
2008 R2 EE                      3    6        2333        0
Server 2003 #1                  0    6        2333        1
Server 2003 #1                  1    6        2333        1
Core #2                         0    6        2333        0
SCO OpenServer 5.0.7 Fe         0    6        2333        0

PS C:\hyperv> _
```

Summary

The WMI provider combined with Windows PowerShell or another scripting language can help you automate virtually any Hyper-V administrative task. Building on the work and insight of others can save you time. The library maintained in `www.codeplex.com` is a useful resource. Learning basic tricks in Windows PowerShell can magnify your capabilities and the value of your Hyper-V environment.