
What's New in the VI Toolkit v1.5?

Introduction

Hello again! It turns out that since you and I last met, VMware shipped a new version of the toolkit that we both know and love. Therefore, I thought I would whip up this "What's New" chapter to give you a proper introduction.

Version 1.5 was released on January 27th, 2009, and can be downloaded from the VI Toolkit landing page, at <http://vmware.com/go/powershell>. This chapter accompanies the official release notes which you can find on the same web page.

At a high level, here's what's new:

- 32 new cmdlets
- 25 new parameters to existing cmdlets
- Dozens of bugfixes
- Lots of new host server functionality is introduced, allowing you to configure things like NTP, SCSI LUNs, syslog, and firewall settings.
- A very cool new remote script execution engine which uses the guest tools service to interact with virtual machines.

Let's get right into the new stuff.

New Virtual Machine Cmdlets

Working with Virtual Machine Resource Configuration

The first set of cmdlets to discuss work similarly to **Get-ResourcePool** and **Set-ResourcePool** in that you can view and change the allocation of resources assigned to a virtual machine. The difference is that the **Get-VMResourceConfiguration** and **Set-VMResourceConfiguration** cmdlets work at the single virtual machine layer, rather than at a parent container layer as it does with resource pools.

Significant Parameters for Get-VMResourceConfiguration Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Server	VIServer[]	False	False
VM	VirtualMachine[]	False	True

The **Get-VMResourceConfiguration** cmdlet is pretty simple as you can see. The only thing to note is that you can supply the **VM** parameter on the pipeline.

Here is an example:

```
PS > $res = Get-VMResourceConfiguration MyVM
PS > $res

VirtualMachineId      : VirtualMachine-vm-3674
NumCpuShares          : 1000
CpuReservationMhz     : 0
CpuLimitMhz          : -1
CpuSharesLevel        : Normal
NumMemShares          : 5120
MemReservationMB      : 0
MemLimitMB            : -1
MemSharesLevel        : Normal
DiskResourceConfiguration : {2000}
HTCoreSharing         : Any
CpuAffinity           : NoAffinity
```

As you could probably guess, you can take the objects returned by **Get-VMResourceConfiguration** and pass it straight to the **Set-VMResourceConfiguration** cmdlet.

Significant Parameters for Set-VMResourceConfiguration Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Configuration	VMResourceConfiguration[]	True	True
HtCoreSharing	HtCoreSharing (nullable)	False	False
CpuAffinity	CpuAffinity (nullable)	False	False
CpuReservationMhz	Int64	False	False
CpuLimitMhz	Int64	False	False
CpuSharesLevel	CpuSharesLevel (nullable)	False	False
NumCpuShares	Int32	False	False
MemReservationMB	Int64	False	False
MemLimitMB	Int64	False	False

MemSharesLevel	MemSharesLevel (nullable)	False	False
NumMemShares	Int32	False	False
Disk	HardDisk[]	False	False
NumDiskShares	Int32	False	False
DiskSharesLevel	DiskSharesLevel (nullable)	False	False

Many of the parameters work like the **Set-ResourcePool** cmdlet, so I won't mention them again. Let's look at some of the new parameters in a little more detail.

- **Configuration:** This is the only mandatory parameter, and it is used to specify the resource configuration object which is to be modified.
- **HtCoreSharing:** If hyperthreading is enabled on your host server's CPU, then this parameter affects whether a virtual machine's virtual CPU will share its time with other VMs using the same physical core. If hyperthreading is disabled or not supported, then this parameter has no effect. Acceptable values are **Any**, **None**, or **Internal**. Or, since this parameter is nullable, you can pass **\$Null** as a value and that will cause the setting to be inherited from resource pool.
- **CpuAffinity:** This parameter allows you to assign a virtual machine to run on one or more CPU cores. You can either set this to "NoAffinity", or you can specify cores as a string like "Cpu0". However, this will only work when specifying a single CPU. For some reason, VMware thought it would be funny to make you specify multiple CPUs using a bitmask. You'll find an example of this in the help file for the cmdlet.
- There are three parameters which are related to storage resources: **Disk**, **NumDiskShares**, and **DiskSharesLevel**. These parameters work similarly as the ones for memory or CPU shares and levels, but for disk I/O.

Here is an example:

```
PS > $res | Set-VMResourceConfiguration -CpuAffinity Cpu0
```

VirtualMachineId	: VirtualMachine-vm-3674
NumCpuShares	: 1000
CpuReservationMhz	: 0
CpuLimitMhz	: -1
CpuSharesLevel	: Normal
NumMemShares	: 5120
MemReservationMB	: 0
MemLimitMB	: -1
MemSharesLevel	: Normal
DiskResourceConfiguration	: {2000}
HTCoreSharing	: Any
CpuAffinity	: Cpu0

Invoking Scripts on a Remote Virtual Machine

The next cmdlet, **Invoke-VMScript**, is really pretty cool. It allows you to invoke a command from your toolkit session and have it execute on a remote guest OS. This is performed via a link between the host server and the VM tools service that most of you will have running on every virtual machine.

System Requirements for Invoke-VMScript Remoting

Note that in order for this cmdlet to work, the VM tools service must be up-to-date and in the running state. Also, PowerShell must be installed in the guest OS. Yes, this means that for now, you can only use this cmdlet to execute scripts against Windows guest operating systems. One last thing to note is that your user ID in VirtualCenter must have the **VirtualMachine.Interaction.Console Interaction** privilege.

Significant Parameters for Invoke-VMScript Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
ScriptText	String	True	False
VM	VirtualMachine[]	True	True
HostCredential	PSCredential	False	False
HostUser	String	False	False
HostPassword	SecureString	False	False
GuestCredential	PSCredential	False	False
GuestUser	String	False	False
GuestPassword	SecureString	False	False
ToolsWaitSecs	Int32	False	False
Server	VIserver[]	False	False

- The **ScriptText** parameter is a string containing the actual command you wish to execute on the remote system. This command will execute in a PowerShell session, so you can use any non-interactive PowerShell commands that you like.
- You must specify credentials for both the host server which runs the VM, and the guest where the script is executed. I'm told that a future version of this cmdlet will reduce or even eliminate these requirements, but for now, it is indeed mandatory. If you don't supply any credentials, you will be prompted. The credentials can be supplied either as a username and password pair, or with a PSCredential object as supplied by the **Get-Credential** cmdlet.

Here is my example:

```
PS > $gcred = Get-Credential
PS > $hcred = Get-Credential
PS > $vm = Get-VM win2k8dc3
PS > Invoke-VMScript -VM $vm -GuestCredential $gcred -HostCredential $hcred -ScriptText @"
>> Get-Process
>> ipconfig /all
>> "@
>>
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
530	10	2476	3804	119	1.59	496	csrss
83	7	1940	2516	112	2.80	540	csrss
203	8	2204	3444	116	4.56	2532	csrss

The output continues on, showing every process, followed by the IP configuration information. In my opinion, a comprehensive IT management solution, virtual or otherwise, should include a means of executing remote processes and then acting on the results. Using the **Invoke-VMScript** cmdlet is a great way to gain that capability and a

very worthy addition to your automation toolbox. Note that the object returned from **Invoke-VMScript** is a string array.

New Host Server Cmdlets

Configuring NTP Services

Thankfully, VMware has created a set of cmdlets which allow you to configure the NTP (time sync) service on your host servers. The cmdlets are: **Add-VMHostNtpServer**, **Get-VMHostNtpServer**, and **Remove-VMHostNtpServer**. Let's check out the parameters.

ESX Version Requirement for NTP Cmdlets

Note that the NTP cmdlets depend on an API method which is only supported in ESX or ESXi version 3.5 and above.

Significant Parameters for Add-VMHostNtpServer Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
NtpServer	String[]	True	False
VMHost	VMHost[]	False	True
Server	VIserver[]	False	False

Significant Parameters for Get-VMHostNtpServer Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
VMHost	VMHost[]	False	True
Server	VIserver[]	False	False

Significant Parameters for Remove-VMHostNtpServer Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
NtpServer	String[]	True	False
VMHost	VMHost[]	False	True
Server	VIserver[]	False	False

Here are some examples:

```
PS > $vmhost = Get-VMHost esx1*
PS > $vmhost | Add-VMHostNtpServer 10.2.11.20
10.2.11.20
PS > Get-VMHostNtpServer -VMHost $vmhost
10.2.11.20
PS > $vmhost | Remove-VMHostNtpServer 10.2.11.20
```

Configuring SCSI LUNs

There are four new cmdlets for reading information about and configuring your storage subsystem: **Get-ScsiLun**, **Set-ScsiLun**, **Get-ScsiLunPath**, and **Set-ScsiLunPath**.

Significant Parameters for Get-ScsiLun Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
CanonicalName	String[]	False	False
VmHost	VMHost[]	False	True
Key	String[]	False	False
LunType	String[]	False	False
Server	VIserver[]	False	False

The parameters **CanonicalName**, **Key**, and **LunType** are all filters that accept wildcards. There is a list of acceptable values for **LunType** in the cmdlet help. You will probably find “disk” the most useful.

Here is an example:

```
PS > $vmhost = Get-VMHost esx1.halr9000.com
PS > $lun = Get-ScsiLun -VmHost $vmhost
PS > $lun.Length
4
PS > $lun[0] | Format-List

CanonicalName      : naa.600a0b800026f332000006e0485265ac
Key                : key-vim.host.ScsiDisk-020000000600a0b800026f332000006e0485265ac31383134...
LunType            : disk
Model              : 1814          FASST
SerialNumber       : unavailable
Vendor             : IBM
ConsoleDeviceName  : /vmfs/devices/disks/naa.600a0b800026f332000006e0485265ac
CapacityMB         : 347472
MultipathPolicy    : Fixed
CommandsToSwitchPath :
BlocksToSwitchPath :
HostId             : HostSystem-host-15

PS > Get-ScsiLun -VmHost $vmhost -Key *1342020

CanonicalName ConsoleDeviceName LunType CapacityMB MultipathPolicy
ame
-----
naa.600... /vmfs/devices/disks/naa.600... disk 347472 Fixed
naa.600... /vmfs/devices/disks/naa.600... disk 347472 Fixed
```

Of course, once “gotten”, an object can be set. The **Set-ScsiLun** cmdlet allows you to configure multipath policy, preferred paths, and a few other things.

Significant Parameters for Set-ScsiLun Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
MultipathPolicy	MultipathPolicy (nullable)	False	True
PreferredPath	ScsiLunPath	False	True

ScsiLun	ScsiLun	True	True
CommandsToSwitchPath	Int32	False	False
BlocksToSwitchPath	Int32	False	False
NoCommandsSwitch	Boolean	False	False
NoBlocksSwitch	Boolean	False	False

The **MultiPathPolicy** parameter will accept these values:

- Fixed
- RoundRobin
- MostRecentlyUsed

Use the objects returned from the **Get-ScsiLun** cmdlet as input to the **ScsiLun** parameter.

The Pipeline Is Not Always Good

There's an amusing note in the help file for this and perhaps a couple of other cmdlets. It says that the option to supply a **MultiPathPolicy** or **PreferredPath** value on the pipeline is deprecated. In other words, don't get used to doing things that way with these specific parameters, as they plan on removing the functionality in a future version. Why did VMware bother with the message? This was probably due to a late stage architectural change that was too dangerous to actually change in the code at that point in the release cycle. I happen to agree with the change. Using a parts-of-speech analogy in line with the PowerShell Verb-Noun syntax, it does not make much sense to send what amounts to an adjective over the pipeline.

Once you have a device, you can use the **Get-ScsiLunPath** cmdlet to read the path details.

Significant Parameters for Get-ScsiLunPath Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
ScsiLun	ScsiLun[]	True	True

Here are some examples:

```
PS > $path = $lun[0] | Get-ScsiLunPath
PS > $path | Format-List
```

```
LunPath           : fc.2000001b322b7d48:2100001b322b7d48-fc.200400a0b826f332:200400a0b826f33
                  3-naa.600a0b800026f332000006e0485265ac
SanId             : 20:04:00:A0:B8:26:F3:33
State            : Active
Preferred        : False
ScsiCanonicalName : naa.600a0b800026f332000006e0485265ac

LunPath           : fc.2001001b322b7d48:2101001b322b7d48-fc.200400a0b826f332:200400a0b826f33
                  4-naa.600a0b800026f332000006e0485265ac
SanId             : 20:04:00:A0:B8:26:F3:34
State            : Standby
Preferred        : False
```

```
ScsiCanonicalName : naa.600a0b800026f332000006e0485265ac
```

```
PS > $lun | Get-ScsiLunPath | Where-Object { $_.State -eq 'Active' }
```

LunPath	SanID	State	Preferred
fc.2000...	20:04:00:A0:B8:26:F3:33	Active	False
sas.500...		Active	False
fc.2000...	20:04:00:A0:B8:26:F3:33	Active	False
fc.2000...	20:04:00:A0:B8:26:F3:33	Active	False

Use the **Set-ScsiLunPath** cmdlet to configure active and standby paths, and to select a preferred path.

Significant Parameters for Set-ScsiLunPath Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Active	Boolean	False	False
ScsiLunPath	ScsiLunPath	True	True
Preferred	Switch	False	False

Here is an example:

```
PS > $lun[1] | Get-ScsiLunPath
```

LunPath	SanID	State	Preferred
sas.500...		Active	False

```
PS > $lun[1] | Get-ScsiLunPath | Set-ScsiLunPath -Preferred
```

LunPath	SanID	State	Preferred
sas.500...		Active	True

It's actually kind of silly to set a preferred path on a system with just one path, but hey, I left my good SAN at the office today.

Working with Advanced Host Server Settings

VMware has included four new cmdlets which work with some of the more advanced settings. These settings are the sorts of things you don't touch unless you have to, but when you do, it's an emergency. What better time for a script that can orchestrate the changes across your entire infrastructure?

The first cmdlet to cover is **Get-VMHostAdvancedConfiguration**. This cmdlet returns a hashtable which contains all the items in the Advanced Settings window of the VI Client.

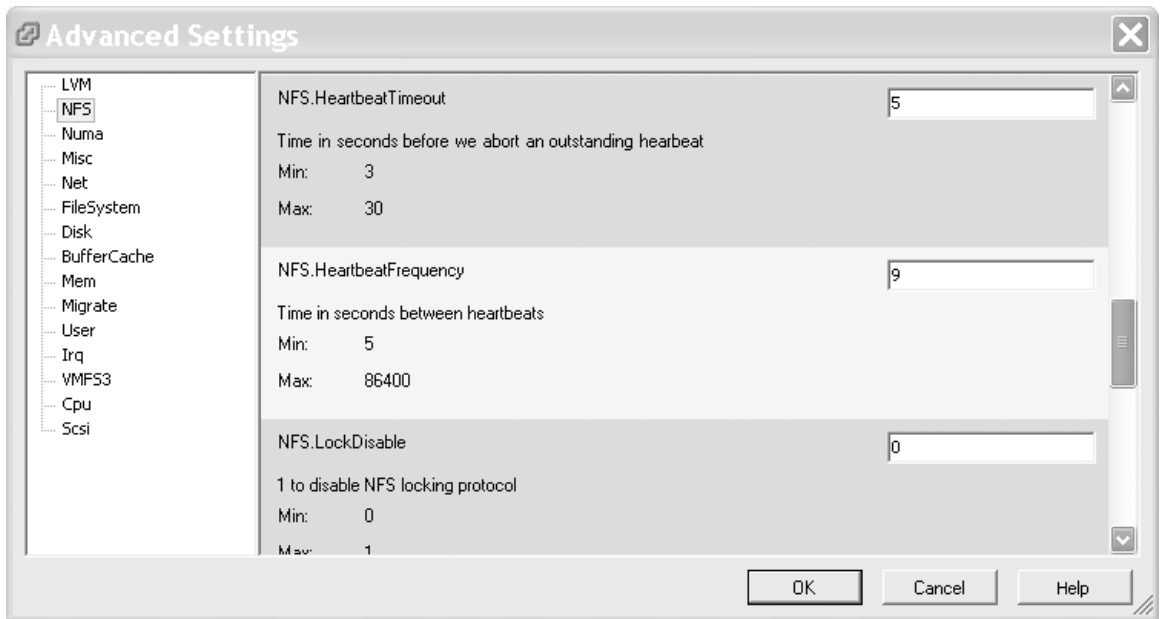


Figure 8-1 Screenshot of Advanced Settings dialog box

Significant Parameters for Get-VMHostAdvancedConfiguration Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Name	String[]	False	False
VMHost	VMHost[]	False	True
Server	VIserver[]	False	True

Here is an example:

```
PS > $vmhost = get-vmhost esx1.halr9000.com
PS > $conf = Get-VMHostAdvancedConfiguration -VMHost $vmhost
PS > $conf.Keys | Select-String nfs
```

```
NFS.HeartbeatFrequency
NFS.HeartbeatMaxFailures
NFS.HeartbeatTimeout
NFS.DiskFileLockUpdateFreq
NFS.VolumeRemountFrequency
NFS.ReceiveBufferSize
NFS.LockRenewMaxFailureNumber
NFS.SendBufferSize
NFS.HeartbeatDelta
NFS.IndirectSend
NFS.LockDisable
NFS.MaxVolumes
NFS.LockUpdateTimeout
```

```
PS > $conf["NFS.HeartbeatTimeout"]
5
```

You can optionally supply a **Name** parameter (wildcards accepted) to filter the results.

```
PS > Get-VMHostAdvancedConfiguration -VMHost $vmhost -Name net.max*
```

Name	Value
-----	-----
Net.MaxNetifRxQueueLen	100
Net.MaxBeaconsAtOnce	100
Net.MaxPageInQueueLen	500
Net.MaxNetifTxQueueLen	100
Net.MaxPortRxQueueLen	50
Net.MaxBeaconVlans	100

Of course the **Set-VMHostAdvancedConfiguration** cmdlet is used to effect changes. This cmdlet has two parameter sets. The first is used to change one setting by its name. But if you want to apply many settings, you can do so with the aptly-named **HashtableParameterSet**. It was so named because you pass it a hashtable (like the one generated by **Get-VMHostAdvancedConfiguration**), that will hold multiple names and multiple values for the configuration items.

Significant Parameters for Set-VMHostAdvancedConfiguration Cmdlet

ParameterSet: NameValueParameterSet

Name	Type	IsMandatory	Pipeline
-----	-----	-----	-----
Name	String	False	False
Value	Object	False	False
VMHost	VMHost[]	False	True
Server	VIserver[]	False	False

ParameterSet: HashtableParameterSet

Name	Type	IsMandatory	Pipeline
-----	-----	-----	-----
NameValue	Hashtable	False	True
VMHost	VMHost[]	False	True
Server	VIserver[]	False	False

It is super easy to change a setting across multiple hosts as this example shows:

```
PS > Get-VMHost | Set-VMHostAdvancedConfiguration -Name NFS.HeartbeatTimeout -Value 5
```

Name	Value
-----	-----
NFS.HeartbeatTimeout	5
NFS.HeartbeatTimeout	5
NFS.HeartbeatTimeout	5

The help file for this cmdlet includes a really clever example that grabs several settings from one host and applies them to another in two short lines. Be sure to check it out.

There are two cmdlets in the new toolkit used to read the available diagnostic partitions and if there are multiple, set which one is active. These storage partitions are used to store core dumps for debugging and technical support. The new cmdlets are **Get-VMHostDiagnosticPartition**, and **Set-VMHostDiagnosticPartition**.

Significant Parameters for Get-VMHostDiagnosticPartition Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
VMHost	VMHost[]	False	True
All	Boolean	False	False
Server	VIserver[]	False	False

Here is a quick example:

```
PS > $vmhost | Get-VMHostDiagnosticPartition
```

CanonicalName	Active	DiagnosticType	SlotCount	StorageType
vmhba0:0:0	True	singleHost	1	directAttached

And the “setter”:

Significant Parameters for Set-VMHostDiagnosticPartition Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Active	Boolean	True	False
VMHostDiagnosticPartition	VMHostDiagnosticPartition[]	True	True

Now, we’ll talk about configuring syslog.

Configuring Syslog

If you run a good-sized Unix/Linux shop, chances are high you are currently aggregating the syslog files on your servers to a central syslog repository. There are two new cmdlets which expose settings which you can use to configure your host servers with the same centralized logging as your other Unix systems.

Compatibility Warning: ESXi Only!

For some reason I don’t quite understand, ESXi can do this syslog stuff, yet ESX cannot. The error you receive is “SysLogServer is not supported for host”.

The first cmdlet to mention is **Get-VMHostSysLogServer**. It returns a simple object with the host and port of your current syslog settings.

Significant Parameters for Get-VMHostSysLogServer Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
VMHost	VMHost[]	False	True
Server	VIserver[]	False	False

Here is an example:

```
PS > Get-VMHostSysLogServer
```

Host	Port
----	----

As you will see in a moment, it's very easy to change this setting using the **Set-VMHostSysLogServer** cmdlet.

Significant Parameters for Set-VMHostSysLogServer Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
SysLogServer	NamedIPEndPoint	False	True
VMHost	VMHost[]	False	True
SysLogServerPort	Int32	False	False
Server	VIServer[]	False	False

And the aforementioned example:

```
C:\> Set-VMHostSysLogServer -SysLogServer 172.16.14.12 -SysLogServerPort 514
```

Host	Port
172.16.14.12	514

Not a whole lot to talk about there, it's very straightforward—provided you are using ESXi.

Configuring Firewall Rules

The previous version of the toolkit didn't include anything to work with firewall rules. You may recall that I went over some techniques in the book to use the host server managed object and get around this gap. With version 1.5 of the toolkit, you'll no longer have to worry about that. There are four new cmdlets which deal with firewall rules:

- **Get-VMHostFirewallDefaultPolicy**
- **Set-VMHostFirewallDefaultPolicy**
- **Get-VMHostFirewallException**
- **Set-VMHostFirewallException**

First, we will talk about the default firewall policy. The first two cmdlets deal with an "all or nothing" firewall setting which frankly, isn't that great of an idea, security-wise. I'm more of a pragmatist when it comes to security though, so I'm sure there's a good reason or two to use these options which you will see explained below.

Significant Parameters for Get-VMHostFirewallDefaultPolicy Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
VMHost	VMHost[]	False	True
Server	VIServer[]	False	False

The parameters need no introduction, so let's go straight to an example:

```
PS > Get-VMHost | Get-VMHostFirewallDefaultPolicy
```

VMHostId	IncomingEnabled	OutgoingEnabled	Client
HostSystem-host-1612	False	True	VMware.VimAutomatio...
HostSystem-host-190	False	False	VMware.VimAutomatio...
HostSystem-host-195	False	False	VMware.VimAutomatio...

The **VMHostId** field is a string representation of the managed object reference of your host servers. Why VMware didn't put the actual host name here, I don't know. Hopefully, they will take my advice and change that accordingly. I also don't think much of the **Client** field. After a brief discussion with VMware, they informed me that the Client field is only for internal use and it should've been hidden. Expect it to go away in the future.

But the big deals here are the **IncomingEnabled** and **OutgoingEnabled** fields. If either is set to \$True, then network traffic will flow unimpeded. That's right, the firewall will be turned off.

Here is the cmdlet used to set these two policies:

Significant Parameters for Set-VMHostFirewallDefaultPolicy Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
AllowIncoming	Boolean	False	False
AllowOutgoing	Boolean	False	False
Policy	VMHostFirewallDefaultPolicy[]	True	True

As the **AllowIncoming** and **AllowOutgoing** parameters are Boolean, this means you must explicitly specify **\$true** or **\$false**.

Here is an example of turning incoming traffic on and then off:

```
PS > $vmhost = Get-VMHost esx1.halr9000.com
PS > $defpol = Get-VMHostFirewallDefaultPolicy -VMHost $vmhost
PS > Set-VMHostFirewallDefaultPolicy -Policy $defpol -AllowIncoming:$true
```

VMHostId	IncomingEnabled	OutgoingEnabled	Client
HostSystem-host-1612	True	True	VMware.VimAutomatio...

```
PS > Set-VMHostFirewallDefaultPolicy -Policy $defpol -AllowIncoming:$false
```

VMHostId	IncomingEnabled	OutgoingEnabled	Client
HostSystem-host-1612	False	True	VMware.VimAutomatio...

Now, let's talk about the cmdlets used to actually poke holes in the firewall: **Get-VMHostFirewallException**, and **Set-VMHostFirewallException**.

Significant Parameters for Get-VMHostFirewallException Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Name	String[]	False	False
VMHost	VMHost[]	False	True
Port	Int32[]	False	False
Enabled	Boolean	False	False
Server	VIserver[]	False	False

Name, **Port**, and **Enabled** parameters will let you filter by those criteria. Here are a couple of examples:

```
PS > $vmhost = Get-VMHost esx1*
PS > Get-VMHostFirewallException -VMHost $vmhost -Port 22
```

Name	Enabled	IncomingPorts	OutgoingPorts	Protocols	ServiceRunning
NFS Client	False		111, 2049, ...	UDP, TCP	
NIS Client	False		111, 0-65535	UDP, TCP	
SSH Client	False		22	TCP	
SSH Server	True	22		TCP	True

```
PS > Get-VMHostFirewallException -VMHost $vmhost -Enabled:$true
```

Name	Enabled	IncomingPorts	OutgoingPorts	Protocols	ServiceRunning
CIM Secure Server	True	5989		TCP	
CIM SLP	True	427	427	UDP, TCP	
VMware License Cl...	True		27000, 27010	TCP	
VCB	True		443, 902	TCP	
FTP Client	True		21	TCP	
NTP Client	True		123	UDP	True
SSH Server	True	22		TCP	True
VMware VirtualCen...	True		902	UDP	

Here is the cmdlet used to alter the firewall rules. The syntax is rather simple as you can see below:

Significant Parameters for Set-VMHostFirewallException Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Enabled	Boolean	True	False
Exception	VMHostFirewallException[]	True	True

Here's an example which will allow you to enable a network service by poking a hole in the firewall across multiple hosts:

```
PS > Get-VMHost | Get-VMHostFirewallException "SSH Client" |
>> Set-VMHostFirewallException -Enabled:$true
>>
```

Name	Enabled	IncomingPorts	OutgoingPorts	Protocols	ServiceRunning
SSH Client	True		22	TCP	
SSH Client	True		22	TCP	
SSH Client	True		22	TCP	

Now we'll move on to some new cluster cmdlets.

New DRS Cmdlets

You may have noticed when reading about version 1 of the toolkit that there was no easy way to configure Dynamic Resource Schedule (DRS) affinity rules with version 1 of the VI Toolkit. Now, with version 1.5, you have cmdlets to get and set, as well as create and remove DRS rules.

Significant Parameters for New-DrsRule Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Name	String	True	False
Cluster	Cluster[]	True	True
Enabled	Boolean	False	False
KeepTogether	Boolean	True	False
VM	VirtualMachine[]	True	False
RunAsync	Boolean	False	False
Server	VIserver[]	False	True

Note that the **Cluster** and **VM** parameters are mandatory, as is **KeepTogether**. The **KeepTogether** Boolean parameter determines whether the VMs you specify are kept on the same host server (affinity) or spread apart (anti-affinity) by the DRS service.

Here's an example showing how to create a rule to keep four VMs together. I'm using simple array notation (a comma) on the **Get-VM** cmdlet to retrieve multiple objects.

```
PS > $c = Get-Cluster "test drs cluster"
PS > $vm = Get-VM -Name "jelly", "porkchops", "applesauce", "peanutbutter"
PS > New-DrsRule -Name "Food VMs" -Cluster $c -VM $vm -KeepTogether:$true
```

Name	Enabled	KeepTogether	VMIDs
Food VMs	True	True	{VirtualMachine-vm-3674, VirtualMachine-vm...

Now, let's look at the **Get-DrsRule** cmdlet and the object it returns.

Significant Parameters for Get-DrsRule Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Name	String[]	False	False
Cluster	Cluster[]	False	True
VM	VirtualMachine[]	False	True
Server	VIserver[]	False	False

You must specify a Cluster name or object to successfully find DRS rules. Here is an example:

```
PS > $c | Get-DrsRule | Format-List

Name           : Food VMs
Enabled        : True
ClusterId      : ClusterComputeResource-domain-c1273
```

```
KeepTogether : True
VMIDs       : {VirtualMachine-vm-3674, VirtualMachine-vm-3666, VirtualMachine-vm-3668, Virtu
             alMachine-vm-3672}
```

While it may not be the most convenient thing in the world that the cmdlet returns cluster and VM IDs rather than names, it's actually not hard to convert these into the respective entities by using the **ID** property on the **Get-VM** and **Get-Cluster** cmdlets.

Here, I'll show you:

```
PS > $rule = $c | Get-DrsRule
PS > $rule.VMIDs
VirtualMachine-vm-3674
VirtualMachine-vm-3666
VirtualMachine-vm-3668
VirtualMachine-vm-3672
PS > $rule.VMIDs[0]
VirtualMachine-vm-3674
PS > Get-VM -Id $rule.VMIDs[0]
```

Name	PowerState	Num CPUs	Memory (MB)
-----	-----	-----	-----
applesauce	PoweredOn	1	1024

Now I'll show you how to use the **Set-DrsRule** cmdlet to make changes to an existing rule.

Significant Parameters for Set-DrsRule Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
-----	-----	-----	-----
Enabled	Boolean (Nullable)	False	False
Rule	DrsVMAffinityRule[]	True	True
Name	String	False	False
VM	VirtualMachine[]	False	False
RunAsync	Boolean	False	False

This example will disable the previously created rule:

```
PS > $rule | Set-DrsRule -Enabled:$false
```

Name	Enabled	KeepTogether	VMIDs
-----	-----	-----	-----
Food VMs	False	True	{VirtualMachine-vm-3674, VirtualMachine-vm...

Lastly, the **Remove-DrsRule** cmdlet is used to—you guessed it, blow them away.

Significant Parameters for Remove-DrsRule Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
-----	-----	-----	-----
Rule	DrsVMAffinityRule[]	True	True
RunAsync	Boolean	False	False

Here's a quick example:

```
PS > $rule | Remove-DrsRule
```

```
Perform operation?
```

```
Perform operation 'Removing Drs rule' on Drs rule 'Food VMs'?
```

```
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"):
```

```
PS >
```

If you are going to be removing rules non-interactively, don't forget to specify "-Confirm:\$false" so that the prompt won't break your script.

New Statistics Cmdlets

There are four new cmdlets that work with statistics intervals, and one which reads the types of stat counters available on an inventory item. A statistics interval simply refers to a period of time over which you can collect (or "roll-up") a performance counter.

Before we go into the cmdlets, be aware that the ability to create new intervals is a feature which was deprecated in vCenter version 2.5. Therefore, the `New-StatInterval` and `Remove-StatInterval` cmdlets are somewhat limited in usefulness, unless you happen to be running vCenter version 2.0.

By default, the stat intervals are:

- Day
- Week
- Month
- Year

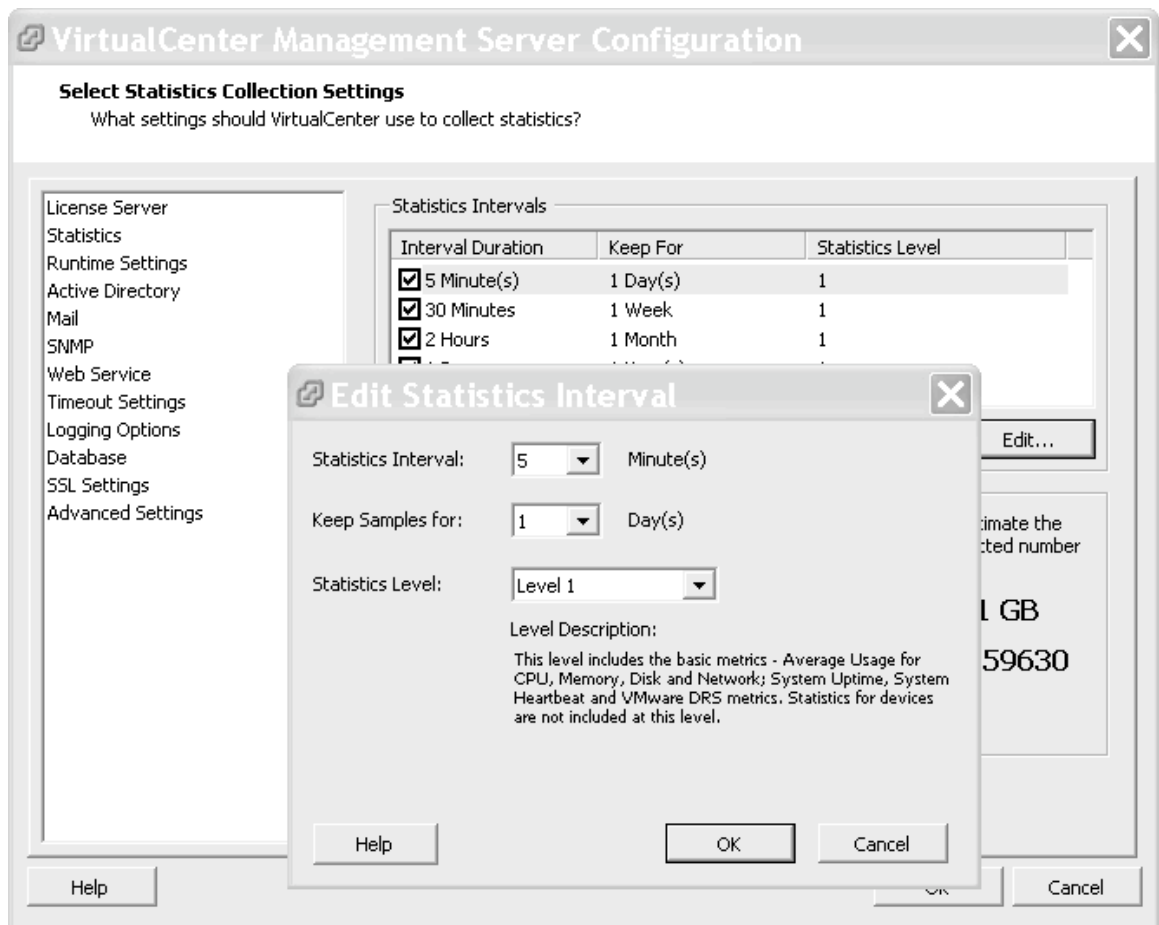


Figure 8-2 Edit Statistics Interval dialog box

Note that if you are running VirtualCenter Server version 2.0 (no later), you can create new stat intervals using the **New-StatInterval** cmdlet.

Significant Parameters for New-StatInterval Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Name	String	True	False
SamplingPeriodSecs	Int32	True	False
StorageTimeSecs	Int32	True	False
Server	VIServer[]	False	True

As mentioned above, this cmdlet is only useful if you are running a previous version of vCenter. Let's move on to **Get-StatInterval**.

Significant Parameters for Get-StatInterval Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
------	------	-------------	----------

Name	Type	IsMandatory	Pipeline
Name	String[]	False	False
SamplingPeriodSecs	Int32[]	False	False
Server	VIserver[]	False	True

Run the cmdlet all by itself to display the current set of stat intervals:

```
PS > Get-StatInterval
```

Name	SamplingPeriodSecs	StorageTimeSecs	Client
Past Day	300	86400	VMware.VimAutomatio...
Past Week	1800	604800	VMware.VimAutomatio...
Past Month	7200	2592000	VMware.VimAutomatio...
Past Year	86400	31536000	VMware.VimAutomatio...

Or specify a sampling period to retrieve just one interval. This example grabs the 5 minute interval:

```
PS > Get-StatInterval -SamplingPeriodSecs 300
```

Name	SamplingPeriodSecs	StorageTimeSecs	Client
Past Day	300	86400	VMware.VimAutomatio...

Pass the stat interval objects to the **Set-StatInterval** cmdlet to make changes.

Significant Parameters for Set-StatInterval Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
SamplingPeriodSecs	Int32	False	False
StorageTimeSecs	Int32	False	False
Interval	StatInterval[]	True	True
Server	VIserver[]	False	True

- The **SamplingPeriodSecs** parameter determines how often a counter using this interval is checked.
- Use the **StorageTimeSecs** parameter to specify how long the recorded value is stored in the vCenter database.
- The **Interval** parameter is mandatory and refers to the stat interval to be changed. Obtain these by using the **Get-StatInterval** cmdlet.

The last stat interval cmdlet is **Remove-StatInterval**, and you can probably guess what it does.

Significant Parameters for Remove-StatInterval Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Interval	StatInterval[]	True	True
Server	VIserver[]	False	True

As with most remove cmdlets, this only has one important parameter: **Interval**.

The **Get-StatType** cmdlet will help you when using the **Get-Stat** cmdlet to collect statistical data. It will return the name of the stat types, also known as performance counters, which are applicable to a specified inventory item.

Significant Parameters for Get-StatType Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
-----	-----	-----	-----
Name	String[]	False	False
Entity	InventoryItem[]	True	True
Start	DateTime	False	False
Finish	DateTime	False	False
Interval	StatInterval[]	False	False
Server	VIServer[]	False	False

The **Entity** parameter is required, and can be one of these objects: VM, host server, cluster, or resource pool.

Here are a couple of examples:

```
PS > Get-StatType -Entity (Get-VM myvm)
cpu.usage.average
cpu.usagemhz.average
mem.usage.average
disk.usage.average
net.usage.average
sys.uptime.latest
sys.heartbeat.summation
```

```
PS > Get-VMHost esx1* | Get-StatType
cpu.usage.average
cpu.usagemhz.average
mem.usage.average
disk.usage.average
net.usage.average
sys.uptime.latest
clusterServices.cpubfairness.latest
clusterServices.memfairness.latest
```

New Credential Store Cmdlets

I can sense the question forming in your mind. “What is a credential store?” That’s what you’re thinking, right? Well, I’ll tell you. A credential store is a small XML file stored in your Windows AppData directory (by default) which contains a host name, a user name, and an obfuscated password. Default ACLs on this file ensure that only the user account which created the credential store items can read them.

Before I go any further, let me say that unless you often work with VMware’s Remote CLI (RCLI) toolkit, which is based on Perl, you will never need the credential store cmdlets. In fact, I recommend against using them. Instead, please refer to an earlier chapter where I taught you how to save PSCredential objects for use with unattended scripts. These new cmdlets, which I’ll show you below, serve a similar purpose, yet do so in a much less secure fashion.

First up is the **New-VICredentialStoreItem** cmdlet. This is used to create credential store items which are stored in the aforementioned XML file.

Significant Parameters for New-VICredentialStoreItem Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Host	String	True	False
User	String	True	False
Password	String	False	False
File	String	False	False

Here is an example of creating a credential store item:

```
PS > New-VICredentialStoreItem -Host eazye -User root -Password password
```

Host	User	File
eazye	root	

Yes, you have to type the password in the clear. Lame, I know. A while ago, I wrote (or found—it's been a while, my apologies to the possible original author) a function which helps get around issues like this called Read-HostMasked. Here is the source for that function:

```
function Read-HostMasked([string]$prompt="Password") {
    $password = Read-Host -AsSecureString $prompt
    $BSTR = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($password)
    $password = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)
    [System.Runtime.InteropServices.Marshal]::ZeroFreeBSTR($BSTR)
    return $password
}
```

Load this function in the current scope (dot-source it), and then you can use it this way:

```
PS > New-VICredentialStoreItem -Host esx1 -User root -Password (Read-HostMasked)
Password: *****
```

I just can't stand typing passwords in the clear, so this function comes in handy. Also, don't forget that PowerShell will persist your command history—including passwords.

Ok, back to VMware! Let's talk about the **Get-VICredentialStoreItem** cmdlet.

Significant Parameters for Get-VICredentialStoreItem Cmdlet

ParameterSet: Default

Name	Type	IsMandatory	Pipeline
Host	String	False	False
User	String	False	False
File	String	False	False

You can optionally use the **Host**, **User** and **File** parameters to filter the results. Otherwise, you just get the entire contents of the XML file, converted to a custom object, as you see below:

```
PS > Get-VICredentialStoreItem
```

Host	User	File
-----	----	----
esx1	root	

Just for grins, let's have a look at the XML file itself:

```
PS > Get-Content "$($env:appdata)\vmware\credstore\vicredentials.xml"
<?xml version="1.0" encoding="UTF-8"?>
<viCredentials>
  <version>1.0</version>
  <passwordEntry>
    <host>esx1</host>
    <username>root</username>
    <password>tKqjvq+pzLpaK9ARAFddjJ023xn3Cfz48c5cprXzkjchlm9TnUCmfGKz0VnYKbgYuIWIqnPkFefT16Ak
uGoxmkTt3bX+rUP+sRIU8Bx7JtCdZrjNzT3b0y7s/nsTFDETVLAPxg3bt513SQtnFGXSDfY1nRjAZdcKaI0di2W3jA=</
password>
  </passwordEntry>
```

And of course, there's a **Remove-VICredentialStoreItem** cmdlet.

Significant Parameters for Remove-VICredentialStoreItem Cmdlet

ParameterSet: ByCredentialItemObject

Name	Type	IsMandatory	Pipeline
-----	----	-----	-----
CredentialStoreItem	VICredentialStoreItem[]	True	True

ParameterSet: ByFilters

Name	Type	IsMandatory	Pipeline
-----	----	-----	-----
Host	String	False	False
User	String	False	False
File	String	False	False

The single required parameter in the first parameter set is the item to be removed. As you can see, there is a second parameter set which allows for filtering by **Host**, **User**, or **File**.

Here is an example:

```
PS > Get-VICredentialStoreItem | Remove-VICredentialStoreItem
```

Confirm

Are you sure you want to perform this action?

Performing operation "Remove-VICredentialStoreItem" on Target "Remove credential store item for host 'esx1' and username 'root?'".

[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"):

If I had multiple items, the cmdlet would present a confirmation prompt for each one before deleting them.

Changes and Fixes to Existing Cmdlets

There's no sense repeating the release notes here, but I want to highlight a few changes to existing cmdlets which I feel are noteworthy.

- **New-Cluster**, **New-VM**, **Set-Cluster**, and **Set-VM** cmdlets now have parameters which allow you to configure HA and DRS policies.
- **New-Cluster**, **Set-Cluster**, and **Set-VMHost** gain the ability to configure whether a VM swapfile resides with the VM disk files, or in a host server's custom swapfile datastore.
- **New-VMHostAccount** now has a **GrantShellAccess** parameter. That's definitely a nice little addition, as the steps necessary to do the same in a script are relatively long.
- My favorite bugfix is that **New-VM** will now allow you to clone multiple VMs asynchronously.

Support for vSphere and vCenter

By the time you read this, you might have your hands on VI4 and the newly rebranded products vSphere and vCenter. Version 1.5 of the toolkit is meant to be used with VI version 3.5 and that is the company line you'll get from VMware. Unofficially, you ought to be able to do most if not all of what you want to get done with this version of the toolkit with ESX 3.0.x and up, as well as VirtualCenter Server 2.x and up. On the other hand, the new VI4 products will be shipping with a new version of the toolkit which, of course, may have enhanced functionality, so you will want to upgrade to that version when it is available. VMware is very good about maintaining backwards compatibility (provided you avoid the deprecated stuff), so this should not create any problems with your older scripts.