PowerShell Integration with VMware® View™ 5.0

TECHNICAL WHITE PAPER

vmware®

Table of Contents

Introduction

VMware View

VMware® View™ is a best-in-class enterprise desktop virtualization platform. VMware View separates the personal desktop environment from the physical system by moving desktops to a datacenter, where users can access them using a client-server computing model. VMware View delivers a rich set of features required for any enterprise deployment by providing a robust platform for hosting virtual desktops from VMware vSphere™.

Windows PowerShell

Windows PowerShell is Microsoft's command line shell and scripting language. PowerShell is built on the Microsoft .NET Framework and helps in system administration. By providing full access to COM (Component Object Model) and WMI (Windows Management Instrumentation), PowerShell enables administrators to perform administrative tasks on both local and remote Windows systems.

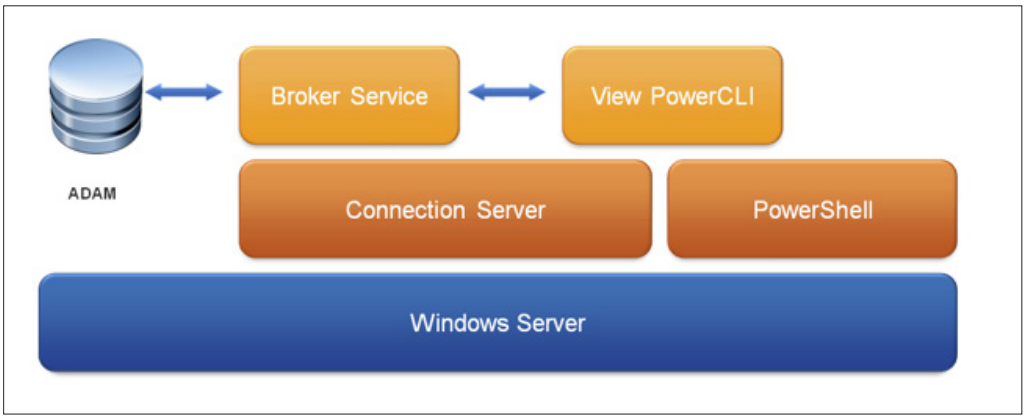
Administrators can manage the computers in the enterprise and perform administrative tasks from the command line using built-in `cmdlets`, which are specialized .NET classes. Unlike most other command line interfaces, PowerShell commands have been standardized using a verb-noun naming convention known as a `cmdlet`. This convention provides a clear description of the `cmdlet`, and enables access to different Windows components like the registry, file system, services, processes, and others. There are sufficient `cmdlets` to support most administrative activities.

This technical paper covers the integration of VMware View PowerCLI with Windows PowerShell and VMware vSphere PowerCLI.

Architecture

Cmdlet dll

VMware View PowerCLI `cmdlets` are provided by a dll, which is installed as part of the VMware View Connection Server (`Server\bin\PowershellServiceCmdlets.dll`) under the VMware Connection Server installation). This `dll`, once registered with PowerShell, allows all VMware View PowerCLI `cmdlets` to be run on the VMware Connection Server.



Communication with Broker

VMware View PowerCLI `cmdlets` send requests to the VMware Connection Server, using RPC-like calls to a service running under the VMware View Connection Broker service. As a result, the VMware View PowerCLI `cmdlets` must be run on a VMware View Standard or Replica Connection Server.

However, using the Windows Management Framework (which includes PowerShell 2.0 and Windows Remote Management), PowerShell `cmdlets` can be remotely invoked from another host. Note that in a remote PowerShell session, you would need to run `add-snapin.ps1` from the VMware Connection Server `extras` directory to load the VMware View PowerCLI `cmdlets`.

VMware View PowerCLI Integration

VMware View PowerCLI Prerequisites

The following are the prerequisites for VMware View PowerCLI execution:

- VMware View Connection Server or VMware View Replica Server, installed and configured appropriately
- Windows PowerShell installed (minimum supported version level v1)
- Microsoft .NET Framework 2.0 SP1 or higher installed
- Optional: VMware vSphere PowerCLI

Using VMware View PowerCLI

- Launch VMware View PowerCLI using **Start>All Programs>VMware>View PowerCLI**. This will launch a PowerShell command prompt.
- You may get an error stating that **Execution of scripts is disabled on this system**. If so, execute the command `Set-ExecutionPolicy AllSigned`. Then close and relaunch VMware View PowerCLI. You may be prompted to accept the VMware code signing certificate
- You will now have a PowerShell console with the VMware View PowerCLI `cmdlets` loaded
- Get-Help can be used to get a full definition for each `cmdlet`
- In order to use the View `cmdlets`, the user must be a View Administrator with full privileges

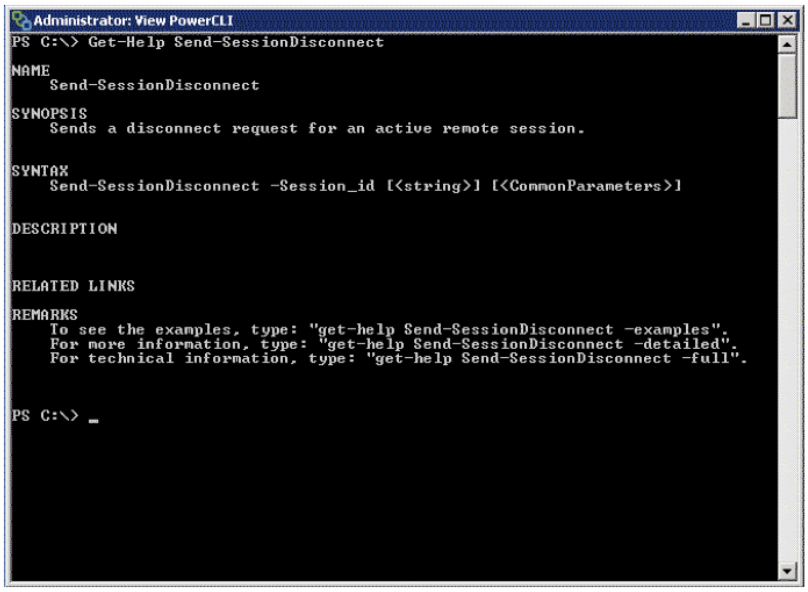


Figure 1: Get-Help Send-SessionDisconnect executed in VMware View PowerCLI command prompt

VMware View PowerCLI cmdlets

Add-AutomaticLinkedClonePool	Update-AutomaticLinkedClonePool
Add-AutomaticPool	Update-AutomaticPool
Get-ComposerDomain	Get-ConnectionBroker
Update-ConnectionBroker	Get-DesktopPhysicalMachine
Get-DesktopVM	Send-VMReset
Get-EventReportList	Get-GlobalSetting
Update-GlobalSetting	Get-License
Set-License	Send-LinkedCloneRebalance
Send-LinkedCloneRecompose	Send-LinkedCloneRefresh
Get-LocalSession	Send-LocalSessionRollback
Add-ManualPool	Update-ManualPool
Add-ManualUnmanagedPool	Update-ManualUnmanagedPool
Get-Monitor	Get-Pool
Remove-Pool	Add-PoolEntitlement
Get-PoolEntitlement	Remove-PoolEntitlement
Get-ProfileDisk	Get-RemoteSession
Send-SessionDisconnect	Send-SessionLogoff
Get-TerminalServer	Add-TerminalServerPool
Update-TerminalServerPool	Get-User
Remove-UserOwnership	Update-UserOwnership
Add-ViewVC	Get-ViewVC
Remove-ViewVC	Update-ViewVC

vSphere PowerCLI Integration

VMware View PowerCLI `cmdlets` can be used along with VMware vSphere PowerCLI `cmdlets`. If VMware vSphere PowerCLI is installed on the VMware Connection Server, VMware vSphere `cmdlets` will also be loaded when launching VMware View PowerCLI.

Examples of VMware View PowerCLI and VMware vSphere PowerCLI Integration

View PowerCLI supports piping of, and use of, the ID/Name fields from virtual machine and VMware Virtual Infrastructure Server objects as obtained from `Get-VM` and `Connect-VIServer` (or `$global:DefaultVIServer`) respectively.

Passing VMs from Get-VM to VMware View PowerCLI cmdlets

```
Get-VM -name <VM name>| Add-ManualPool -pool_id <pool id> -vc_id (Get-ViewVC –serverName
<server>).vc_id
```

Registering a vCenter Server

Connect to and register a vCenter Server

```
Connect-VIServer <server> | Add-ViewVC –password <password>
```

Registering the currently connected vCenter Server

```
$global:DefaultVIServer | Add-ViewVC –password <password>
```

Using Other VMware vSphere Objects

VMware View PowerCLI references VMware vSphere objects by their path rather than their ID. The only exceptions for this are VMs and ESX hosts which are referred by IDs.. As the path is not readily available from VMware vSphere PowerCLI, this can pose a problem in referencing templates, datastores, resource pools, and so on when using a VMware vSphere `cmdlet`.

A path can be determined programmatically from a VMware vSphere inventory object based on its `ParentId` or `FolderId`, and functions to construct a path in this manner can be found in this paper in the Sample Scripts section, under "Inventory Path Manipulation." Below is an example using these functions to create an Automatic Pool (note that a datastore path is constructed using the cluster path obtained from a resource pool).

```
Get-ViewVC -serverName vc.mydom.int | Add-AutomaticPool -pool_id auto1 –displayName
'ADP1' -namePrefix "adp1-{n:fixed=4}" -vmFolderPath GetPath(Get-Folder "Guests")
-resourcePoolPath GetPath(Get-ResourcePool "Resources") -templatePath GetPath(Get-
Template "AutoTemplate") -dataStorePaths (GetDatastorePathFromResourcePool (Get-Datastore
"datatore1") (Get-ResourcePool "Resources") ) -customizationSpecName 'Windows 7 Variation
3' -minimumCount 4 -maximumCount 10
```

Advanced Usage

Integrating VMware View PowerCLI into Your Own Scripts

You can load VMware View PowerCLI `cmdlets` directly for those situations where PowerShell scripts won't be run from the VMware View PowerCLI console. You can load `cmdlets` by dot-sourcing the `add-snapin.ps1` script from the VMware Connection Server's `extras` directory. Add the following line at the start of a script using VMware View PowerCLI `cmdlets`:

```
. "<install directory>\Server\extras\PowerShell\add-snapin.ps1"
```

The VMware View Connection Server installation directory (noted above as `<install directory>`) will be under the path `C:\Program Files\VMware\VMware View\` by default.

Similar to launching from the Start Menu shortcut, this will also load the VMware vSphere PowerCLI `cmdlets` if installed.

Scheduling PowerShell Scripts

You can schedule a script to run at regular intervals. You will need to run the script from a batch file, which can be scheduled as you normally would in Windows. To call a PowerShell script from a batch file, you should execute the following command:

```
powershell -command "& 'X:\Path\to\your\script.ps1' "
```

Workflow with VMware View PowerCLI and VMware vSphere PowerCLI

In an enterprise, a typical workflow with VMware View PowerCLI and VMware vSphere PowerCLI integration consists of executing the following steps

• Add license

```
Set-License –key <View License Key>
```

<View License Key> will be set as the license key for VMware View

• Add VMware vCenter

```
Add-ViewVC -serverName <vCentername> -username <username> -password
<password>
```

The VC with <vCentername> hostname will be added as a VMware vCenter Server to VMware View

• Enable VMware View Composer

```
Add-ViewVC -serverName <vCentername> -username <username> -password <password>
-useComposer $true
```

VMware View Composer features will be enabled for the VMware vCenter Server, which will be used to create automaticLinkedClonePools.

• Create Manual Pool (using piping)

```
Get-ViewVC -serverName <IP/FQDN> | Get-DesktopVM -name <vm name> |
Add-ManualPool -pool_id <pool id>
```

This creates a manually provisioned pool of managed desktops. A manual pool provides access to an existing set of machines. Any type of machine that can install View Agent is supported.

• Create Automatic Pool (Full Clone)

```
Get-ViewVC -serverName <IP/FQDN> | Add-AutomaticPool -pool_id TestAutomaticPool
-namePrefix pad -templatePath /<datacenter>/vm/<template> -vmFolderPath /<datacenter>/
vm -resourcePoolPath /<datacenter>/host/<ESX>/Resources -dataStorePath /<datacenter>/
host/<ESX>/<datastore>
```

This creates an automatically provisioned full-clone pool. Desktop sources will be full virtual machines created from a VMware vCenter Server template.

• Create Automatic Pool (Linked Clone)

```
Get-ViewVC -serverName <IP/FQDN> | Add-AutomaticLinkedClonePool -pool_id
TestAutomaticLinkedClonePool -namePrefix pad -parentVMPath /<datacenter>/vm/<parentVM>
-parentSnapshotPath /<Snapshot> -vmFolderPath /<datacenter>/vm -resourcePoolPath
/<datacenter>/host/<ESX>/Resources -dataStoreSpecs /<datacenter>/host/<ESX>/<datastore>
-composer_ad_id <composer ad id>
```

This creates an automatically provisioned linked-clone pool. VMware View Composer linked clones share the same base image and use less storage space than full virtual machines. Linked-clone pools are created from a snapshot.

Sample Scripts

These scripts are intended as internal tools, or as reference materials for scripting. To avoid any false new lines created by copying the text from this document, you can use the attached scripts instead. The scripts can be executed using dot-source notation:

```
. ./<ScriptFile>.ps1
```

Add or Remove Datastores in Automatic Pools

```
#
# AddRemoveDatastores.ps1
# Function(s) for managing lists of datastores in automatic pools
#

####   Add a datastore to an automatic full-clone pool
####   Parameters:
####      $Pool : pool_id of pool to be updated
####      $NewDatastore : path to datastore to be added
function AddDatastoreToAutomaticPool
{ param ($Pool, $NewDatastore)
       $PoolSettings = (Get-Pool -pool_id $Pool)
       $datastores = $PoolSettings.datastorePaths + ";$NewDatastore"
       Update-AutomaticPool -pool_id $Pool -datastorePaths $datastores
}

####   Remove a datastore from an automatic full-clone pool
####   Parameters:
####      $Pool : pool_id of pool to be updated
####      $DatastoreToRemove : path to datastore to be removed
function RemoveDatastoreFromAutomaticPool
{ param ($Pool, $DatastoreToRemove)
       $PoolSettings = (Get-Pool -pool_id $Pool)
       $currentdatastores = $PoolSettings.datastorePaths
       $datastores = ""
       foreach ($path in $currentdatastores.split(";"))
              {
              if(-not ($path -eq $DatastoreToRemove))
                     { $datastores = $datastores + "$path;"}
              }
       Update-AutomaticPool -pool_id $Pool -datastorePaths $datastores
}

####   Add a datastore to a linked clone pool
####   Parameters:
####      $Pool : pool_id of pool to be updated
####      $NewDatastore : path to datastore to be added
function AddDatastoreToLinkedClonePool
{ param ($Pool, $NewDatastoreSpec)
       $PoolSettings = (Get-Pool -pool_id $Pool)
       $datastores = $PoolSettings.datastoreSpecs + ";$NewDatastoreSpec"
       Update-AutomaticLinkedClonePool -pool_id $Pool -datastoreSpecs $datastores
}
```

```
####   Remove a datastore from a linked clone pool
####   Parameters:
####      $Pool : pool_id of pool to be updated
####  $DatastoreToRemove : path to datastore to be removed
function RemoveDatastoreFromLinkedClonePool
{ param ($Pool, $DatastoreToRemove)
       $PoolSettings = (Get-Pool -pool_id $Pool)
       $currentdatastores = $PoolSettings.datastoreSpecs
       $datastores = ""
       foreach ($spec in $currentdatastores.split(";"))
              {
              $path = $spec.split("]")[1]
              $pathToRemove = $DatastoreToRemove.split("]")[1]
              if(-not $pathToRemove)
                     {
                     $pathToRemove = $DatastoreToRemove
                     }
              if(-not ($path -eq $pathToRemove))
                     {
                     $datastores = $datastores + "$spec;"
                     }
              }
       Update-AutomaticLinkedClonePool -pool_id $Pool -datastoreSpecs $datastores
}
```

Add or Remove Virtual Machines

```
#
# AddRemoveVMs.ps1
# Author: telliott
# Functions for adding and removing VMs in View Pools.
# WARNING: These functions manipulate the ADAM directory directly, which may cause
faults in an environment if used incorrectly.
# Use is at own risk.
#

####  Add a VM to a pool
####  Parameters:
####    $VMObject      :      Virtual Machine object for VM to be added (obtained from
Get-DesktopVM)
####    $pool_id       :      Name of pool to which VMObject should be added
####  Example Usage: AddVMToPool (Get-DesktopVM -Name <vmname>) <pool_id>
function AddVMToPool
{ param ($VMObject, $pool_id)
      if($VMObject.vm){
        if($VMObject.isInPool -eq "true"){
              Write-Error ("The specified VM (" + $VMObject.Name + ") is already
assigned to a pool.")
       } else {

              # Get the GUID for this VM's entry in ADAM, creating an entry if necessary
              $machine_id = $VMObject.machine_id
```

```
                if(-not $machine_id){
                        $machine_id = AddVMToADAM $VMObject
                }

                # Locate the server group for the pool to which this VM belongs
                $poolObject = [ADSI](“LDAP://localhost:389/cn=” + $pool_id + “,ou=Applica
tions,dc=vdi,dc=vmware,dc=int”)
                $serverGroup =          [ADSI](“LDAP://localhost:389/” + $poolObject.
get(“pae-Servers”))

                # Add the distinguished name of the VM’s ADAM entry to the server group
                if($serverGroup){
                                $machineName = $VMObject.Name
                                $serverGroupId = $serverGroup.get(“cn”)
                                Write-Output (“Adding $machineName to pool $pool_id”)

                                $machineList = & {
                                        trap [Exception] {
                                                continue;
                                        }
                                        $serverGroup.get(“pae-MemberDN”)
                                }

                                if($machineList){
                                        if($machineList.Count -gt 1){
                                                $machineListAL = New-Object System.
Collections.ArrayList(,$machineList)
                                        } else {
                                                $machineListAL = New-Object System.
Collections.ArrayList
                                                $null = $machineListAL.add($machineList)
                                        }
                                } else {
                                        $machineListAL = New-Object System.Collections.
ArrayList
                                }

                                $null = $machineListAL.Add(“CN=” + $machine_id + “,ou=Serve
rs,dc=vdi,dc=vmware,dc=int”)
                                $serverGroup.put(“pae-MemberDN”,$machineListAL.ToArray())
                                $serverGroup.SetInfo()
                        }
        }
      } else {
        Write-Error “The object passed as a parameter was not a valid VM object (a valid
object would be returned by Get-DesktopVM).”
      }
}

###     Create an entry for a managed VM in ADAM
###     The entry is created by generating a temporary Manual Pool, which is deleted
after the VM is registered.
###     Parameters:
```

```
###            $VM            :       DesktopVM object for VM to be added (can be
obtained from Get-DesktopVM).
###     Returns a machine_id for the registered VM.
function AddVMToADAM
{ param($VM)
        $temp_pool = [guid]::NewGuid()
        Add-ManualPool -pool_id $temp_pool -vm_id_list $VM.id -vc_id $VM.vc_id
        $VMUpdated = Get-DesktopVM -pool_id $temp_pool
        $null = RemoveVMFromPool $VMUpdated $false
        $null = Remove-Pool -pool_id $temp_pool -DeleteFromDisk $false
        $VMUpdated.machine_id
}

####  Add multiple VMs to a pool
####  Parameters:
####    $VMObject      :       Array of Virtual Machine objects to be added (obtained from
Get-DesktopVM)
####    $pool_id       :       Name of pool to which VMObject should be added
####  Example Usage: AddVMsToPool (Get-DesktopVM -Name agent*) <pool_id>
function AddVMsToPool
{ param ($VMs, $pool_id)
      foreach($VM in $VMs){
        AddVMToPool $VM $pool_id
      }
}

####   Remove a VM from its pool.
####   Parameters:
####      $VMObject : Virtual Machine object for machine to be removed (obtained from
Get-DesktopVM)
####    $deleteEntry: Should the ADAM entry for this VM also be deleted?
####   Example Usage: RemoveVMFromPool (Get-DesktopVM -Name <vmname>)
function RemoveVMFromPool
{ param ($VMObject, $deleteEntry=$true)
      $ldapBaseURL = "LDAP://localhost:389/"
      if($VMObject -and $VMObject.vm){
        if($VMObject.isInPool -eq "true"){
                $machineFull = [ADSI]($ldapBaseURL + "cn=" + $VMObject.machine_id + ",ou=
Servers,dc=vdi,dc=vmware,dc=int")

                if($machineFull){
                        $machineDN = $machineFull.get("distinguishedName")
                        $serverGroupDN = $machineFull.get("pae-MemberDNOf")
                        $serverGroup = [ADSI]($ldapBaseURL + $serverGroupDN)

                        if($serverGroup){
                                $machineName = $machineFull.get("pae-DisplayName")
                                $serverGroupId = $serverGroup.get("cn")
                                Write-Output ("Removing $machineName from server group
$serverGroupId")

                                $machineList = & {
                                        trap [Exception] {
                                                continue;
```

```
                                        }
                                        $serverGroup.get("pae-MemberDN")
                                }


                                if($machineList){
                                        if($machineList.Count -gt 1){
                                                $machineListAL = New-Object System.
Collections.ArrayList(,$machineList)
                                        } else {
                                                $machineListAL = New-Object System.
Collections.ArrayList
                                                $null = $machineListAL.add($machineList)
                                        }

                                        $null = $machineListAL.Remove($machineDN)

                                        $serverGroup.put("pae-MemberDN",$machineListAL.
ToArray())
                                        $serverGroup.SetInfo()

                                }

                        }

                        if($deleteEntry){
                                $servers = [ADSI]($ldapBaseURL + "ou=Servers,dc=vdi,dc=vmwa
re,dc=int")

                                $objMachine = $servers.Delete("pae-VM", "cn=" + $VMObject.
machine_id)
                                if($objMachine){
                                        $null = $objMachine.SetInfo()
                                }
                        }
                }
        } else {
                Write-Error "The specified machine is not part of a pool."
        }
      } else {
        Write-Error "The object passed as a parameter was not a valid VM object (a valid
object would be returned by Get-DesktopVM, for example)."
      }
}

####   Remove multiple VMs from their pools.
####   Parameters:
####      $VMObject : Array of Virtual Machine objects to be removed from pools
(obtained from Get-DesktopVM)
####   Example Usage: RemoveVMsFromPool (Get-DesktopVM -Name agent*)
function RemoveVMsFromPool
{ param ($VMs)
      foreach($VM in $VMs){
```

```
        RemoveVMFromPool $VM
      }
}

AddVMToPool (Get-DesktopVM -Name test_Win7) dfgdf
```

Inventory Path Manipulation

```
#
# PathManipulation.ps1
# Author: telliott
# Function(s) for manipulation/generation of vSphere inventory paths.
#

####    Get the full vSphere path to an Inventory object from vSphere PowerCLI
####    Parameters:
####            $InventoryObject Individual vSphere PowerCLI object
####    Example Usage:
####            GetPath(Get-VM -name myVM)
####            GetPath(Get-ResourcePool | Select -first 1)
function GetPath($InventoryObject)
{

        if($InventoryObject -and $InventoryObject.Id)
        {
                $path = ""
                # Recursively move up through the inventory hierarchy, by parent or folder
                if($InventoryObject.ParentId)
                {
                        $path = GetPath (Get-Inventory -Id $InventoryObject.ParentId)
                }
                elseif ($InventoryObject.FolderId)
                {
                        $path = GetPath (Get-Folder -Id $InventoryObject.FolderId)
                }
                # Skip the "Datacenters" folder at the root
                if(-not $InventoryObject.isChildTypeDatacenter)
                {
                        # Add this object to the path
                        $path = $path + "/" + $InventoryObject.Name
                }
                $path
        }
}

####    Construct a View-friendly path to a datastore on a specific cluster (defined by a
Resource Pool)
####    Parameters:
####            $Datastore Datastore object
####            $ResourcePool Resource Pool in desired cluster
####    Example Usage:
####            GetDatastorePathFromResourcePool (Get-Datastore "datastore1") (Get-
ResourcePool "Resources")
```

```
function GetDatastorePathFromResourcePool($Datastore,$ResourcePool)
{
        $ClusterPath = GetPath(Get-Inventory -Id $ResourcePool.ParentId)
        $path = $ClusterPath + "/" + $Datastore.Name
        $path
}
```

Poll Pool Usage

```
#
# PollPoolUsage.ps1
# Author: telliott
# Functions for polling current usage of pools, resizing them if they are at capacity.
#

####   Check usage for all pools, warning or resizing if necessary
####   Parameters:
####   $increment : size by which to increase at-capacity automatic pool (defaults to 5)
function PollAllPoolsUsage
{ param ($increment)
        if(-not $increment)
        {
                $increment = 5
        }
        # Retrieve all pool objects and check each one individually
        $pools = Get-Pool
        foreach ($pool in $pools)
        {
                PollPoolUsage $pool $increment
        }
}

####   Check current usage of a pool
####   Output a warning if all desktops for this pool are in use
####   Increase the maximum size for such pools if possible
####
####   Parameters:
####   $Pool : Pool object representing the pool to be checked
####   $increment : size by which to increase at-capacity automatic pools
function PollPoolUsage
{ param ($Pool, $increment)
        ## Get list of remote & local sessions for specified pool (will not output errors)
        $remotes = Get-RemoteSession -pool_id $Pool.pool_id -ErrorAction SilentlyContinue
        $locals = Get-LocalSession -pool_id $Pool.pool_id -ErrorAction SilentlyContinue

        ## Count number of remote and local sessions in list
        $remotecount = 0
        $localcount = 0
        if($remotes)
        {
                $remotecount = ([Object[]]($remotes)).Count
        }
        if($locals)
```

```
        {
                $localcount = ([Object[]]($locals)).Count
        }
        $totalcount = $localcount + $remotecount

        # Determine total or maximum number of desktops for this pool
        $maxdesktops = 0
        if($Pool.deliveryModel -eq “Provisioned”)
        {
                $maxdesktops = $Pool.maximumCount
        }
        else
        {
                $maxdesktops = $Pool.machineDNs.split(“;”).Count
        }

        Write-Output (“====  “ + $Pool.pool_id + “ ====”)
        Write-Output (“Remote session count: “ + $remotecount)
        Write-Output (“Local session count: “ + $localcount)
        Write-Output (“Total session count: “ + $totalcount)
        Write-Output (“Maximum desktops: “ + $maxdesktops)

        ## If a pool is using all its desktops, increase the maximum size or output a
warning
        ## Linked Clone and Automatic Pools can be re-sized.
        ## Automatic Pools have desktopSource == “VC” && deliveryModel == “Provisioned”
        ## Linked Clone Pools have desktopSource == “SVI” && deliveryModel ==
“Provisioned”
        if($maxdesktops -eq $totalcount)
        {
                if($Pool.deliveryModel -eq “Provisioned”)
                {
                        $newmaximum = [int]$Pool.maximumCount + [int]$increment
                        if($Pool.desktopSource -eq “VC”)
                        {
                                Update-AutomaticPool -pool_id $Pool.pool_id -maximumCount
$newmaximum
                        }
                        elseif ($Pool.desktopSource -eq “SVI”)
                        {
                        Update-AutomaticLinkedClonePool -pool_id $Pool.pool_id
-maximumCount $newmaximum
                        }
                        Write-Output (“Pool “ + $Pool.pool_id + “ is using 100% of its
desktops. Maximum VMs increased to “ + $newmaximum)
                }
                else
                {
                Write-Output (“Pool “ + $Pool.pool_id + “ is using 100% of its desktops.
Consider increasing its capacity.”)
                }
        }
}
```

Basic Troubleshooting

"**View Server Connect FAILED**" error when running a VMware View `cmdlet`

This error means that the VMware View PowerCLI `cmdlets` are unable to connect to the VMware View Connection Server. Check that the VMware View Connection Server service is running in the Windows Services console.

"**NoQueueManager**" error when running a VMware View `cmdlet`

The VMware View Connection Server service may still be starting up. Wait a few minutes and then try again. Should the problem persist, try restarting the VMware View Connection Server Windows service.

About the Authors

Tina de Benedictis, Senior Technical Marketing Manager in End-User Computing at VMware, updated this document with script corrections, deletion of known issues, and attachment of the script files.

The original version of this document was written by Tom Elliott, formerly a Senior MTS at VMware; Raghavendra Babu, QE Manager at VMware; and N P Rao, QE Engineer at VMware.

vmware®

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 www.vmware.com