A PowerShell Library for Hyper-V

Author: James O'Neill (http://www.blogs.technet.com/jamesone)

This Document, the PowerShell scripts with which it is supplied, and updates to them can be obtained from http://www.codeplex.com/psHyperV

Table of Contents

PowerShell Verbs and nouns

As you probably know PowerShell CMDlets use a standard Verb-Noun format, and try to use the same Verbs consistently. Where possible the Library uses the standard Verbs.

Standard PowerShell verbs used

- Get- Returns one or more specified objects.
- New- Creates an free standing object – for example new-VHD creates a new disk image file, without mounting it in a drive
- Add- Connects an existing or (more commonly) new object to something – for example, add-VMdisk mounts a disk image into a drive
- Remove Disconnects or deletes something
- Rename Changes the name of an object
- Set Changes one or more properties of something
- Test Checks something
- Update Changes something to match newer settings
- Start, Stop, Suspend Changes things between running and non-running states.
- Convert Changes an object to an equivalent object in a different form
- Copy Creates an additional instance of an item in another location
- Export Writes something to one or more files suitable for importing
- Import Creates something from one or more files
- Out Outputs an object

Non Standard PowerShell Verbs used

- Choose The verb “Select”, is used in PowerShell (and SQL) to mean “Return a subset of data based on the provided criteria”. This is different from “Ask for user input to indentify rows”
- Shutdown In Strict PowerShell syntax STOP is the preferred verb, but Hyper-V can Stop a VM (power it off) or use an integration component to tell the Guest OS to initiate a Shutdown.
- Apply In the sense of applying a snapshot, we could use “Import-Snapshot”, but snapshots are imported as part of importing a VM, so this would probably cause confusion
- Compact / Expand In the sense of making a VHD smaller or larger. Expand could be done with SET or as part of the existing Convert-VHD (indeed Compact could be a switch on Convert) but this syntax seems stilted
- Merge In strict Power shell syntax Join is the preferred verb but that has the sense of two independent things being brought together. A differencing VHD file is joined to its parent. Merging combines two things which are joined.
- Mount UnMount In strict terms these should probably be named “Add-VHDHostDrive” and “Remove-VHDHostDrive”, but this seems unwieldy and likely to confuse
- Ping This is used as a verb like the familiar ping command.

Summary of functions provided

Noun \ Verb	Add	Choose	Get	New	Remove	Set	Test
VFD				*			
VHD				*			*
VHDDefaultPath			*				
VHDInfo			*				
VM		*	*	*	*	*	
VMBackupScript			*				
VMConnectSession				*			
VMCPUCount			*			*	
VMDisk	*		*			*	
VMDiskController			*				
VMDrive	*				*		
VMExternalEthernet		*					
VMExternalSwitch				*			
VMFloppyDisk	*		*		*		
VMHeartBeat							*
VMHost			*				
VMInternalSwitch				*			
VMJPEG			*				
VMKVP	*		*		*		
VMMemory			*			*	
VMNewHardDisk	*						
VMNic	*	*	*		*		
VMNICAddress						*	
VMNICConnection						*	
VMNICPort			*				
VMNICSwitch			*				
VMPrivateSwitch				*			
VMProcessor			*				
VMSCSIcontroller	*				*		
VMSerialPort			*			*	
VMSettingData			*				
VMSnapshot		*	*	*	*		
VMSnapshotTree			*				
VMState			*			*	
VMSwitch		*	*				
VMSwitchPort				*			
WMIJob							*

The next few pages include a listing, of all the functions in the library grouped by their purpose (noun)

Admin functions not specific to Hyper-V

- Test-Admin — Tests if the Powershell session is privileged
- Get-ScriptPath — Returns the path to the script
- Choose-List — Allows the user to select a item from a list view
- Choose-Tree — Allows the user to select a item from a tree view
- Out-Tree — Prevents items in a tree view
- New-zip — Creates a new .ZIP archive file
- Add-ZIPContent — Adds files to a .ZIP archive file
- Copy-ZIPContent — Copies files to from .ZIP archive file
- Get-ZIPContent — Gives a lists of items in a ZIP archive file

Conversion functions

- Convert-DiskIDtoDrive — Converts a drive ID number into it's associated drive letters
- Convert-VMState — Converts th number for a VM state into the state's name

Functions for Hyper-V Servers.

- Get-VMHost — Gets a list of Hosts running Hyper-V from Active directory

Functions for checking WMI jobs created through hyper-v

- Test-WMIJob — Checks on the state of job – can wait until it completes

Virtual Machine

- Choose-VM — Allows the user to select VMs from a list
- Get-VM — Retuns WMI objects representing VMs
- New-VM — Creates a new VM
- Set-VM — Sers the properties of an Existing VM
- Remove-VM — Deletes a VM
- *Export-VM* — Invokes Hyper-Vs export process
- *Import-VM* — Invokes Hyper-Vs import process
- *Ping-VM* — Ping FQDNs found via KVP exchange integration component
- *Shutdown-VM* — Shuts down a VM's OS cleanly via shutdown integration compoent
- *Start-VM* — Starts a VM or restarts a saved one
- *Stop-VM* — Powers down a VM without asking the OS to shutdown first
- *Suspend-VM* — Puts the VM into a saved state
- Set-VMState — Called by Start, stop and suspend VM set the requested states
- Get-VmBackupScript — Gets a script for backing up one or more VMs
- Get-VMByMACaddress — Discovers a VM from its MAC address
- Get-VMJPEG — Gets a JPEG image of the currect VM screen
- Test-VMHeartBeat — Tests the responses from the heartbeat integration component
- Get-VMSettingData — Returns the Setting data object for the VM
- New-VMConnectSession — Launches a VM connect session to a VM.

VM Memory

- Get-VMMemory — Returns the amount of memory assigned to a VM
- Set-VMMemory — Sets the amount of memory assigned to a VM

VM Processors

- Get-VMCPUCount — Gets the number and weighting of Processors assigned to VMs
- Set-VMCPUCount — Sets the number of Virtual Processors assigned to VMs
- Get-VMProcessor — Gets active Virtual Processors and their load data

VM Disk Controllers

- Get-VMDiskController — Gets an IDE or SCSI controller (IDE controllers can't be changed)
- Add-VMSCSIcontroller — Adds a synthetic SCSI controller to a VM
- Remove-VMSCSIcontroller — Removes a synthetic SCSI controller from a VM

Hard disk and DVD drive objects

- Add-VMDRIVE — Connects a Hard disk drive or DVD drive to a controller
- Remove-VMDRIVE — removes a Hard disk drive or DVD drive from a contoller
- Get-VMDriveByController — Gets the drives attached to a controller

Hardisk and DVD disk Image object

- Add-VMDisk — Mounts a DVD or Hard disk image into a Drive
- Get-VMDisk — Gets a list of Virtual disk images in use
- Set-VMDisk — changes the disk image attached to a drive
- Get-VMDiskByDrive — Gets the VM disk image attached to a drive
- Add-VMNewHardDisk — Attaches a new virtual hard disk image to a new drive

Virtual Hard disk image files

- Get-VhdDefaultPath — Gets the Default path Used by Hyper-v for VHD files
- Get-VHDInfo — Gets information about a VHD, such as its parent, size on disk
- New-VHD — Creates a new VHD file
- Test-VHD — Tests that a VHD can be mounted, and any parent is mountable
- *Compact-VHD* — Compacts a dynamic VHD file to save space on the host
- *Convert-VHD* — Changes a VHD from one type to another
- *Expand-VHD* — Expands the size of a virtual hard disk
- *Merge-VHD* — Merges the a child VHD with its parent to form a new disk
- *Mount-VHD* — Makes the VHD file appear as a local disk on the host
- *UnMount-VHD* — Reverses the mount process so the VHD can be used by a VM

VM Floppy Disk

- New-VFD — Creates a new floppy disk image file
- Add-VMFloppyDisk — Mounts the Floppy disk in the floppy drive of a VM
- Get-VMFloppyDisk — Returns information about the mounted floppy disk
- Remove-VMFloppyDisk — Ejects any floppy disk from the drive of a VM

Resource Allocation Settings data objets

- New-VMRasd — Creates objects used by other functions to describe VM components

Serial Port Objects

- Get-VMSerialPort — Returns information about serial
- Set-VMSerialPort — Maps Serial ports on a VM to Named Pipes

Ethernet Cards

Add-VMNic	Adds a Nic to a VM
Choose-VMNic	Allows the user to select a NIC attached to a virtual machine
Get-VMNic	Returns information about network cards
Remove-VMNic	Removes a NIC from a VM
Set-VMNICAddress	Sets the MAC address for a NIC
Set-VMNICSwitch	Connects a NIC to a virtual Switch
Get-VMNicport	Gets the Switch port attached to a NIC
Get-VMnicSwitch	Returns the Switch a NIC is connected to

Ethernet Swtiches

Choose-VMExternalEthernet	Allows the user to choose a host network card
New-VMExternalSwitch	Creates a Virtual Switch connect to a host network card
New-VMInternalSwitch	Creates a Virtual Switch accessible to VMs and the Host
New-VMPrivateSwitch	Creates a Virtual Switch accessible to VMs
Choose-VMSwitch	Allows the user to choose a virtual switch
Get-VMSwitch	Returns information about virtual switches
New-VMSwitchPort	Defines a new port on a virtual switch

Key value Pairs

Get-VMKVP	Gets Key/value pair information sent to the Host by Virtual Machines
Add-VMKVP	Adds a Key/value pair to the list sent to VMs by the host
Remove-VMKVP	Removes a Key/value pair from the list sent to VMs by the host

Snapshots

Apply-VMSnapshot	Applies a snapshot to a VM
Choose-VMSnapshot	Allows the user to choose a snapshot
Get-VMSnapshot	Gets information about snapshots
Get-VMSnapshotTree	Formats the view of Snapshots as a tree
New-VMSnapshot	Creates a new snapshot of a VM
Remove-VMSnapshot	Deletes a snapshot or tree of snapshots
Rename-VMSnapshot	Renames a snapshot
Update-VMSnapshot	Deletes a snapshot, and creates a new one with same name

Detailed Explanation of the functions

Choose-List

Parameters

- InputObject
- Property
- Multiple [Switch]

This function allows the user to choose items from a list. Input object contains one or more objects, and Property contains one or more property names to be displayed. If only one object is passed it is returned; otherwise the user is prompted to make a selection, if –multiple is specified more than one item can be selected. This function is called by all the other choose- functions.

Out-Tree

Parameters

- Items
- StartAt
- Path — Default : "Path"
- Parent — Default : "Parent"
- Label — Default : "Label"
- Indent — Default : 0

This function outputs information in a tree format. Items contains a collection of objects, each of which must have properties which act as the path, the parent, and the label for the (the names of these are passed in Path and Parent). StartAt contains the item which should act as the root of the tree to be output.
The logic works as follows. The function outputs the label for the start item, indented to the number of spaces specified in the indent parameter. It increments the value of Indent, and finds the direct children of the starting item (those members of the Items collection whose *parent* field matches the *path* field of the starting item). If there are any children the function calls itself recursively for each child with the incremented value of Indent, and the child as the Starting item. That will out put the child's name (indented to form a tree view), and then process *it's* children. This function is used to by Get-VMSnapshotTree to output a formatted list.

Choose-Tree

Parameters

- Items
- StartAt
- Path — Default : "Path"
- Parent — Default : "Parent"
- Label — Default : "Label"
- indent — Default : 0
- [Switch]multiple

This function combines the out-tree function with the choose function. This is used by the Choose-VMSnapshot function

Convert-DiskIDtoDrive

Parameters

- DiskIndex

This function converts a Disk ID to a drive letter using the Win32_logicaldisktoPartition WMI object ; the Mount-VHD function discovers a logical disk ID and uses this to return the associated drive letter(s)

Example : `Convert-DiskIDtoDrive 2`

Returns the letter(s) associated with drive 2.

Test-WMIJob

Parameters

- JobID
- [Switch]Wait
- Description Default: "Job"

Many of the WMI functions called by the script start a background job and return an ID for it. This function tests the state of a job. If –Wait is specified and progress bar is displayed as long as the job is running. The function returns the status code of the job. It is used by any function which has a -WAIT switch

Test-Admin

Parameters

- [Switch]verbose

This function returns $True if the session has administrative privileges and false otherwise . The main reason for using it is for users who are in the administrators group (other than the built in Administrator account) on Windows Vista or Server 2008 will start Powershell without elevated permissions which are needed in some situations. Typical usage is shown in the example.

Example:

```
if (-not (test-admin)) {write-host -foregroundColor Red "This Powershell
session does not have elevated priviledges."}
```

Get-VhdDefaultPath

Parameters

- server Default : "."

This functions gets the default path for Virtual Hard Disk (VHD) files. By default this is "\users\public\documents\hyper-v\Virtual Hard Disks" folder.

New-VFD

Parameters

- [String]vfdPath (Mandatory)
- server Default : "."
- [Switch]wait

This function creates a new Virtual Floppy Disk (VFD) file. The VFD path is mandatory; if only a file name is specified (as in the first example) the default hard disk folder is used.

Example. The following are equivalent on a system where the default disk folder has not been changed.

```
New-VFD "Floppy.VFD"
New-VFD –VFDpath "C:\users\public\documents\hyper-v\Virtual Hard
Disks\Floppy.VFD"
```

New-VHD

Parameters

- [String]vhdPath (mandatory)
- [int64]size
- parentDisk
- server Default : "."
- [Switch]Fixed
- [Switch]wait

This function creates a virtual hard disk (VHD) file.
The path to the VHD file is required. If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server's default folder (not the current working directory) will be assumed.
To create a differencing disk the ParentDisk parameter must be specified and the size parameter is ignored and fixed parameters are then ignored.
If only a size is specified, then a dynamically expanding disk is created.
If a fixed disk is required the –Fixed switch and size must be specified
If the wait parameter is specified then the function will wait until the disk creation job completes which can take several minutes in the case of large fixed disks. If it is not specified the ID of the job is returned
***Example 1*:** `New-VHD "$(Get-VhdDefaultPath)\tenby.vhd" 20GB`
Expressly specifies the folder to contain a new VHD named Tenby.VHD as a 20GB dynamically expanding disk
Example 2: `New-VHD –vhdpath "tenby.vhd" –size 20GB -fixed –wait`
Creates a new VHD named Tenby.VHD as a 20GB fixed size disk in the default location and waits for completion
***Example 3*:** `$diskJob =: New-VHD "tenby.vhd" –parent "$(Get-VhdDefaultPath)\New-Core.vhd"`
Creates a new differencing disk and stores the ID of the job to create it in a variable

Mount-VHD

Parameters

- vhdPath (Accepted from the pipe)
- Partition
- Letter
- [Switch]offline

VHDpath is either
a System.io.fileinfo object which represents a VHD file or
a string which contains the name of a VHD file or
an array which contains these.
If no VHDPath is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server's default folder (not the current working directory) will be assumed.
If a valid file is provided then the VHD file at that location is mounted using the Image Management Service. Unless the –Offline switch is specified the function calls DiskPart.exe to bring the disk on line and set it read-write
Partition is the number of a Partition on the, and letter is a drive letter for the disk (if more than one character is passed, only the first is used). If these are specified then the partition identified will be assigned the letter specified.
Example 1: `dir "$(Get-VhdDefaultPath)\*.vhd" | Mount-Vhd –offline`
Gets all the VHD files in the default folder, and mounts them in an Offline state.
Example 2: `Mount-Vhd tenby`
Mounts tenby.VHD from the default folder, and brings the disk on line
Example 3: `Mount-Vhd –path "C:\users\public\documents\hyper-v\Virtual Hard Disks\tenby.vhd" –Partition 2 –letter H`
Mounts tenby.VHD, brings the disk on line and assigns drive H: to the Second partition.

UnMount-VHD

Parameters

- vhdPath (Accepted from the Pipe)

VHDpath is either
a System.io.fileinfo object which represents a VHD file or

a string which contains the name of a VHD File or
an array which contains these.
If no VHDPath is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server's default folder (not the current working directory) will be assumed.
If a valid file is provided then the VHD file at that location is dismounted using the Image Management Service.
Example 1: `UnMount-Vhd  tenby.vhd`
Example 2: `dir "(Get-VhdDefaultPath)\*.vhd" | UnMount-Vhd`

Compact-VHD

Parameters

- vhdPath (Accepted from the Pipe)
- Server Default : "."
- [switch]wait

VHDpath is either
a System.io.fileinfo object which represents a VHD file or
a string which contains the name of a VHD file or
an array which contains these.
If no VHDPath is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server's default folder (not the current working directory) will be assumed.
If a valid file is provided then the function attempts to compact a dynamic VHD file at that location using the Image Management Service.
If the wait switch is specified the function will display a progress indicator until the job to compact the VHD completes.
Example: Compact-Vhd (get-VHDdefaultPath) +"\tenby.vhd"

Get-VHDInfo

Parameters

- vhdPath(Accepted from the Pipe)
- server Default : "."

VHDpath is either
a System.io.fileinfo object which represents a VHD file or
a string which contains the name of a VHD File or
an array which contains these.
If no VHDPath is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server's default folder (not the current working directory) will be assumed.
If a valid file is provided, the function calls the GetVirtualHardDiskInfo method of the Image mangement service: this function returns a block of XML which describes the hard disk, and the function converts this into an object. In Server 2008 hyper-V this object will have the following properties

- VHDPath the path to the file,
- FileSize The size of the file on disk
- InSavedState Indicates whether the disk is associated with a VM in a saved state
- InUse Indicates whether the disk is mounted
- MaxInternalSize the size as seen by the virtual machine

- ParentPath — the parent for a differencing disk (empty for other disks)
- Type — 2=”Fixed”,3=”Dynamic”,4=”Differencing”

Example 1 `cd (Get-VhdDefaultPath) ; dir *.vhd | get-vhdinfo`
Moves to the default folder for VHDs gets all the VHD files and passes them into Get VHDInfo
Example 2 `(Get-VHDInfo 'C:\Users\Public\Documents\Microsoft Hyper-V\Virtual Hard Disks\Core.vhd').parentPath`
Returns the parent path of a single differencing disk
Example 3 `Get-VMDisk “core%” | forEach {Get-VHDInfo $_.Diskpath} | measure-object -Sum filesize`
Gets all the disks on virtual machines with names beginning with CORE, gets the disk info for each one, and calculates the sum their file sizes.

Test-VHD

Parameters

- vhdPath — (Accepted from the Pipe)
- server — Default : "."

VHDpath is either
a System.io.fileinfo object which represents a VHD file or
a string which contains the name of a VHD File or
an array which contains these.
If no VHDPath is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server’s default folder (not the current working directory) will be assumed.
If a valid file is provided the function tests to see if it can be opened in read-only mode. If the disk is a differencing disk this tests to see if the parent can be opened as well (and if the parent has a parent that is tested until a non-differencing disk is reached)
The function will produce an error if the disk is in use by a running VM
Example `Get-VMDisk | %{$_.DiskPath} | where {$_.endswith(".vhd")} | Test-VHD.`
Gets all disks attached to all VMs, takes only the DiskPath property , and if it ends with .VHD pipes into test VHD.

Expand-VHD

Parameters

- vhdPath — (Accepted from the Pipe)
- [int64]size
- server — Default : "."
- [Switch]wait

VHDpath is either
a System.io.fileinfo object which represents a VHD file or
a string which contains the name of a VHD File or
an array which contains these.
If no VHDPath is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server’s default folder (not the current working directory) will be assumed.
If a valid file is provided the function attempts to extend the VHD at that location using the Image Management Service to the value specified in the size parameter
If the wait switch is specified the function will display a progress indicator until the job completes.

Example

```
Expand-VHD 'C:\users\Public\Documents\Microsoft Hyper-V\Virtual Hard Disks\Tenby.vhd' 22gb
```

Expands the named disk to 22GB in size

Merge-VHD

Parameters

- vhdPath (Mandatory)
- [String]DestPath (Mandatory)
- server Default : "."
- [Switch]wait

VHDpath is either
a Single System.io.fileinfo object which represents a VHD file or
a single string which contains the name of a VHD File
DestPath is a single string.
If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server's default folder (not the current working directory) will be assumed.
If a valid file is provided the function attempts to merge the differencing specified by VHDpath into its ancestor specified by DestPath using the Image Management Service
If the wait switch is specified the function will display a progress indicator until the job completes.

Convert-VHD

Parameters

- vhdPath (Mandatory)
- [String]DestPath (Mandatory)
- [int]type (Mandatory)
- server Default : "."
- [Switch]wait

VHDpath is either
a Single System.io.fileinfo object which represents a VHD file or
a single string which contains the name of a VHD File
DestPath is a single string.
If the .VHD file extension is omitted it will be added and if the path only a file name the VHD the server's default folder (not the current working directory) will be assumed.
The new type of disk is specified as a number, and these numbers are store in the variable $DiskType as follows : "Fixed"=2; "Dynamic"=3; "Differencing"=4; "PhysicalDrive"=5
So, for example $DiskType.Fixed can be used in a script.
If a valid file is provided the function attempts to convert the disk specified by VHDPath to a new type with the result stored in the location specified by DestPath. disk with its parent and produce a new disk at that location specified by DestPath using the Image Management Service
If the wait switch is specified the function will display a progress indicator until the job completes.

Example 1 :

```
convert-Vhd "core" "temp" -type disktype.dynamic
```

Converts core.vhd in the default VHD folder to a fixed disk named temp.vhd in the same folder

Example 2 :

```
Convert-vhd "$(Get-VhdDefaultPath )\Temp.vhd" F:\backups\MyDisk.VHD `
            -type $Disktype.fixed
```

Explicitly specifies the default folder, and converts temp.vhd to a fixed sized VHD named MyDisk.VHD on a different drive.

Example 3 : `pushd (Get-VhdDefaultPath) ; dir *.vhd | get-vhdinfo | where-object {$_.type -eq 3} | foreach {convert-vhd($_.path) '.\temp.vhd' -type 2 -wait" ; del ($_.path) ; ren temp.vhd($_.path)} ; popd`

Moves to the default folder for VHD files, gets a list of VHD files , reduces the list to dynamic disks, and for each dynamic disk converts the file to a fixed disk waiting for the operation to complete then deletes the original disk, and renames temp.VHD to the name of the original disk

Get-VMHost

Parameters

- Domain Default : Root Domain Naming Context of current forest

Queries Active directory for servers which have been registered as having the hyper-v role installed.

Example get-vmhost "DC= ite,DC= contoso,DC= com" | foreach {$_;Get-vmstate –server $_}

Get-VM

Parameters

- [String]Name Default : "%"
- Server Default : "."
- [Switch]suspended
- [Switch]running
- [Switch]stopped

This function returns WMI objects (from the class MsVM_ComputerSystem) representing virtual machines (either by VM Name or by state (or both). The Name can take a wildcard; because WMI uses % for wildcards this should be a % sign, however to allow for users using * instead the function will convert * to %

If –suspended, -running or –Stopped are specified only machines in those states are returned. These switches can be combined – for example –suspended –running will return machines which are in either the running or suspended states

The accompanying format.ps1xml file formats MsVM_ComputerSystem objects to show the name of the Host where the VM is running, its element name (the name which is used to refer to it), its state, uptime and owner. Internally Hyper-V as a *name* property this contains a GUID which unique refers to a machine – there is nothing to prevent multiple machines using the same element name.

Example 1: `Get-VM`

Returns WMI MsVM_ComputerSystem objects for all Virtual Machines on the local server(n.b. Parent Partition is filtered out)

Example 2: `Get-VM "Windows 2008 Ent Full TS"`

Returns a single WMI MsVM_ComputerSystem object for the VM named "Server 2008 ENT Full TS"

Example 3 **:** `Get-VM "%2008%" -Server James-2008`

Returns WMI MsVM_ComputerSystem objects for VMs containing 2008 in their name on the server James-2008 (n.b. WQL Wild card is %, function converts * to %)

Choose-VM

Parameters

Server Default : "."

[Switch]multiple

This function allows the user to select a VM from those on one or more servers It does this by passing the results of a call to the Get-VM function into the choose-list function. Usually the choice will be stored in a variable or piped into another command.

Example 1: `Choose-vm -multiple`

Example 2: `Choose-vm -Server James-2008,Jackie-2008`

Get-VMkvp

Parameters

- vm (Accepted from the Pipe)
- Server Default : "."

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If nothing is passed via the pipe or as a parameter all VMs are tried.
This function looks for the Key value pair exchange component for the requested VM. This contains an XML description of the Key value pairs which the function parses and returns as an object.
In Server 2008 Hyper-V the fields returned are

- VMElementName (not part of the XML but for ease of Processing)
- FullyQualifiedDomainName The FQDN that the guest calls itself
- OS Name The full text description of the OS
- OS Version The “all in one” version of the OS, major, Minor, and build
- Os Major Version The Major Version of the OS (before the .)
- Os Minor Version The minor Version of the os (After the .)
- Os BuildNumber The build of the OS
- CSDVersion The Corrective Service Disk (Service Pack) description
- Service Pack Major The major Version of the SP
- Service Pack Minor The Minor Version of the SP
- OSPlatformID 2 =”Windows NT based” 1=”Windows 9x based”
- Product type 1=”Workstation”, 2=“Domain Controller”, 3=“Server”
- Processor architecture 0=”X86”,6=”IA-64”,9=”X64”
- SuiteMask A bitmapped field where 1="Small Business"; 2="Enterprise"; 4="BackOffice"; 8="Communications"; 16="Terminal"; 32="Small Business Restricted"; 64="Embedded NT"; 128="Data Center"; 256="Single User"; 512="Personal"; 1024="Blade"
- Descriptions The result of expanding ProductType, Processor Architecture.

Example 1: `(Get-VMKVP "Windows 2008 Ent Full TS").OSName`
Returns the OS that VM is running, for example "Windows Server (R) 2008 Enterprise"
Example 2: `Get-vmkvp –VM “%” -server “james-2008”`
Returns the Key Value pairs sent back by all the VMs on the Server James-2008
Example 3: `Get-Vm -running | get-VMKVP`
Returns the Key Value pairs for running VMs on the local Server

Test-vmheartBeat

Parameters

- vm (Accepted from the Pipe)
- TimeOut
- Server Default : "."

Checks with heartbeat integration component to see if the specified machine isrunning.
VM is either

a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the TimeOut Parameter is passed the function waits up to that number of seconds for a heart beat to be found.
Example `start-vm "London DC" ; Test-vmheartBeat "London DC" -Timeout 300; start-vm "London SQL"`
Starts the VM named London DC and waits up to 5 minutes for its heartbeat, then starts VM "London SQL"

Ping-VM

Parameters

- vm (Accepted from the Pipe)
- Server Default : "."

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
This function attempts to send an ICMP ping request to the specified VM(s). To do this, it checks to see if the machine is running and attempts to retrieve its fully qualified domain name from Key-value pair exchange. It only attempts to ping the machine if it's FQDN is found. It returns an object (with 4 properties.

- VMName – the name of the VM
- FullyQualifiedDomainName – Its FQDN returned by from the key-value pairs
- NetworkAddress – the IP address resolved during the ping process
- Status – the result of the ping

Example 1: `Ping-VM "Tenby" -server james2008`
Pings the VM named Tenby, on the server named James2008
Example 2: `get-vm -r | foreach-object {if ((Ping-VM_).statusCode -ne 0) {"$($_.elementname) is inaccessible"} }`
Gets all running VMs, for each one it pings the VM and if it is not accessible, displays a message

Get-VMState

Parameters

- [string]Name Default : "%"
- Server Default : "."

This function returns object (with 6 properties.)

- Host The name of the server hosting the VM
- VMElementname The Name of the VM used by hyper-V
- State running, stopped etc
- FQDN the fully qualified domain name reported through Key-value pair exchange
- IPaddress discovered by pinging the FQDN
- Heartbeat Status reported by the Heartbeat component

Example 1: `Get-VMState -server "james-2008","jackie-2008"`
Example 2: `Get-VMState "Windows 2008 Ent Full TS"`

Convert-VMState

Parameters

- ID

MsVM wmiobjects use numbers to to indicate the state of a VM as follows
"Running"=2 ; "Stopped"=3 ; "Paused"=32768 ; "Suspended"=32769 ; "Starting"=32770 ; "Snapshotting"=32771 ; "Saving"=32773 ; "Stopping"=32774
This function converts these numbers to the associated text
Example `Convert-VMState 2`
Returns “Running”

Set-VMState

Parameters

- vm (Accepted from the Pipe)
- state
- Server Default : "."

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
The state is number (as in Convert-VMstate above), the variable $VmState contains a hash table which can be used to access these numbers. Changing the state of a VM returns the ID of a WMI job – this is returned as the result of the function
Example 1. `get-vm Core% | Set-VMState –state $vmstate.running`
Example 2. `set-vmstate "core","Tenby" $vmstate.running -Server "James-2008"`

Start-VM

Parameters

- vm (Accepted from the Pipe)
- [Switch]wait

This function as wrapper to call Set-VMState with the state $VMState.Running. If no VM parameter is passed then the function takes its value from the pipe. The value of VM is passed through to Set-vmState and needs to the follow the same rules. A job is created inside the Set-VMstate function and it’s ID is passed back to this function. If the wait switch is specified the function will display a progress indicator until the job completes.
Example 1: `Start-VM (choose-VM -server James-2008 -multiple)`
Starts the virtual machines on the server James-2008 which the user selects
Example 2: `get-vm | where-object {$_.EnabledState –eq $vmState.Suspended} | start-vm`
Selects machines which are in a suspended (saved) state, and starts them. Note that this could be simplified to Get-VM -Susp

Suspend-VM

Parameters

- vm (Accepted from the Pipe)
- [Switch]wait

This function as wrapper to call Set-VMState with the state $VMState.Suspended (which is refered to as “Saved” in the GUI). If no VM parameter is passed then the function takes its value from the pipe. The value of VM is passed through to Set-vmState and needs to the follow the same rules. A job is created inside the Set-VMstate function and it’s ID is passed back to this function. If the wait switch is specified the function will display a progress indicator until the job completes.

Example `get-vm -running | suspend-vm -wait ; shutdown -s -t 0`

Stop-VM

Parameters

- vm (Accepted from the Pipe)
- [Switch]wait

This function as wrapper to call Set-VMState with the state $VMState.Stopped (in the GUI this is described as “turning off” the VM. I.e. the Guest OS is not asked to shutdown cleanly) . If no VM parameter is passed then the function takes its value from the pipe. The value of VM is passed through to Set-vmState and needs to the follow the same rules. A job is created inside the Set-VMstate function and it’s ID is passed back to this function. If the wait switch is specified the function will display a progress indicator until the job completes.

Example : `Stop-VM (choose-VM -server James-2008 -multiple)`

Stops the virtual machines on the server James-2008 which the user selects

Shutdown-VM

Parameters

- vm (Accepted from the Pipe)
- Reason Default : "Scripted"
- Server Default : "."

This function calls the shutdown integration component to request that the guest OS performs an orderly shutdown. This depends on the integration components being present and enabled in the guest OS. The Integration component does not signal successful shutdown (you can test to see if the VM’s *enabledState* property changes from running to stopped)

The VM parameter contains either:

a MsVM_ComputerSystem WMI object which represents a Virtual Machine or

a string which contains a VM Element name on the specified server

an array which contains these.

If no VM is passed as a parameter, the function looks for input to be piped to it.

If an array is passed the function calls itself recursively for each item.

The Integration component allows a reason for the shutdown to be logged on the guest OS. By default this just puts the word “Scripted”.

Example 1: `shutdown-vm "Tenby" -server James-2008`

Requests the OS of the VM named Tenby on the Server named James-2008 to start an orderly shutdown.

Example 2 : `get-vm | where-object {$_.EnabledState -eq $vmState.running} | shutdown-vm -Reason "Server Upgrade"`

Gets all the VMs in a running state and requests that they shut down and log the reason as “server Upgrade”.

New-VMConnectSession

Parameters

- vm (Accepted from the Pipe)
- server $Env:computerName

This function emulates the Connect function in the GUI, by invoking %programFiles%\hyper-v\VMConnect.exe.

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.

Example 1: `New-VMConnectSession $tenby`
Creates a session the VM pointed to by the $tenby

Example 2: `New-VMConnectSession "tenby" -server James-2008`
Creates a session to the same VM, but this time by pointing to the server and using its name

Get-VMSettingData

Parameters

- vm (Accepted from the Pipe)
- Server Default : "."

Gets a WMI object of the class MsVM_VirtualSystemSettingData which represents the Virtual machine settings
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the function is passed MsVM_VirtualSystemSettingData object (for example one which represent a VM snapshot) it returns that object.
The accompanying format.ps1xml file formats MsVM_VirtualSystemSettingData objects to show the Description of the VMits element name, state, boot device order, creation date and notes

New-VM

Parameters

- [String]Name (Manadatory)
- Server Default : "."

Creates a new virtual machine and returns a MsVM_ComputerSystem WMI object which represents it.

Example 1: `$tenby = new-VM "Tenby"`
Creates an New-VM on the local machine using default settings, and setting the name to "Tenby", storing its MsVM_ComputerSystem object in the variable $tenby

Example 2: `$tenby = new-VM –name "Tenby" -server "James-2008"`
As above, but this time specifying the name of the server explicitly

Set-VM

Parameters

- VM (Accepted from the Pipe)
- [string]Name
- [Int[]]BootOrder
- Notes
- [int]AutoRecovery
- [int]AutoShutDown
- [int]AutoStartup
- Server Default : "."

This function sets properties of a VM excluding the virtual hardware connected to it.
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the name parameter is passed the VM is renamed. If the notes parameter is specified the notes property of the VM is updated.
The BootOrder parameter contains an array of integers specifying which the order in which different boot devices should be tried: the possible values are: "Floppy"=0 ; "CD"=1 ; "IDE"=2 ; "NET"=3. For ease of use these they are stored in a HashTable, $Bootmedia.
The AutoRecovery parameter contains an integer specifying what action should be taken if the VM worker process fails. The possible values are: "None"=0 ; "Restart"=1 ; "RevertToSnapShot"=2. For ease of use these they are stored in a HashTable, $Recoveryaction
The AutoShutDown parameter contains an Integer specifying what action Hyper-v should take for the VM when the host server is shut down, the default is SaveState. The possible values are "TurnOff"=0 ; "SaveState"=1 ; "ShutDown"=2. For ease of use these they are stored in a HashTable, $ShutDownAction
The AutoStartup parameter contains an integer specifying what action Hyper-V should take for the VM when the server is booted, the default is the restart the machine only if it was saved at shutdown. The possible values are: "None"=0 ; "RestartOnly"=1 ; "AlwaysStartup"=2} For ease of use these they are stored in a HashTable, $StartupAction

Example 1: `set-vm -vm core -bootOrder @(3,2,0,1)`
Sets the boot order for the VM named “Core” to Network, IDE, Floppy, CD
Example 2: `Set-vm "CORE%" -bootorder $bootmedia["CD"], $bootmedia["IDE"], $bootmedia["net"], $bootmedia["Floppy"] -autoStart $StartupAction["AlwaysStartup"]`
Sets any VM whose name begins with CORE to a given boot order (this time using the hash table for readability) and to always start when the Server is booted
Example 3: `Set-vm $vm -AutoShutdown 2 -autoStart 0`
Sets the VM pointed to by $VM to shutdown when the server is shut down , and never to autostart (this time using the values instead of the hash tables).

Remove-VM

Parameters

- vm (Accepted from the Pipe)
- server Default : "."
- [Switch]wait

This function deletes a virtual machine, but does not delete its Virtual Hard disk file(s). No warning is given.
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
Internally a job is created to remove the VM. If the wait switch is specified the function will display a progress indicator until the job completes.
Example 1: `Remove-VM "Tenby" -server James-2008`

Removes the VM named Tenby on the Server named “James-2008”

Example 2: `Get-vm | remove-vm`

Gets all VMs on the local server and removes them.

Set-VMMemory

Parameters

- vm (Accepted from the Pipe)
- [int64]memory
- server Default : "."

This function sets the memory for the given VMs to the stated amount in Bytes. Note that in the Server 2008 release of Hyper-V the amount of memory assigned to a virtual machine cannot be changed while it is running or in a saved state

The VM parameter contains either:

a MsVM_ComputerSystem WMI object which represents a Virtual Machine or

a string which contains a VM Element name on the specified server

an array which contains these.

If no VM is passed as a parameter, the function looks for input to be piped to it.

If an array is passed the function calls itself recursively for each item.

For the memory parameter, PowerShell understands terms KB,MB, GB, so the size can be written as 512MB, however passing 512 as a memory parameter will try to set the VM to 512BYTES.

Example 1: `set-VMmemory –vm "Tenby" –memory 1.5gb -server James-2008`

Sets the VM named “Tenby” on the server named “James-2008” to have 1.5GB of memory

Example 2: `Get-vm “Core%” | set-VMmemory -memory 1073741824`

Gets all VMs on the local server whose names begin with core and sets their memory to 1GB

Get-VMMemory

Parameters

- vm (Accepted from the Pipe)
- server Default : "."

The VM parameter contains either:

a MsVM_ComputerSystem WMI object which represents a Virtual Machine or

a string which contains a VM Element name on the specified server

an array which contains these.

If no VM is passed as a parameter, the function looks for input to be piped to it.

If an array is passed the function calls itself recursively for each item.

If nothing is passed via the pipe or as a parameter all VMs are tried.

The function returns one or more MSVM_MemorySetting objects. The accompanying format.ps1xml file formats MSVM_MemorySetting objects to show the VM’s element name and the memory Virtual quantity, Limit and reservation (which in the Server 2008 Release of Hyper-V are all the same value).

Example 1:`Get-VMMemory “core”`

Returns the memory settings for the machine named core on the local server

Example 2: `Get-VMMemory | ft`

Gets the memory settings for all VMs and displays it as a table.

Set-VMCPUCount

Parameters

- vm (Accepted from the Pipe)
- [int]CPUCount
- server Default : "."

This function sets the number of virtual Processors assigned to a VM. Note that in the Server 2008 release of Hyper-V the processors assigned to a virtual machine cannot be changed while it is running or in a saved state
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
Example 1: `Set-VMCPUCount "tenby" 2 -Server "James-2008"`
Sets the VM named "tenby" on the server named "james-2008" to have 2 CPUs
Example 2: `Get-vm "Core%" | Set-VMCPUCount -CPUCount 2`
Gets the VMs with names begin "Core" and sets them to have 2 CPUs

Get-VMCPUCount

Parameters

- vm (Accepted from the Pipe)
- server Default : "."

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If nothing is passed via the pipe or as a parameter all VMs are tried.
The function returns one or more objects. The accompanying format.ps1xml file formats MSVM_ProcessorSettingData objects to show the VM's element name quantity of CPUs, Limit, reservation and Weight for calculating CPU waiting cores per socket and Socket count exposed in the VM
Example `Get-VMCPUCount "core"`
Returns the CPU Count for the VM named Core on the local server.

New-VMRasd

Parameters

- ResType
- ResSubType
- server Default : "."

This function is used for creating the Resource Allocation Setting Data objects used by most of the NEW hardware functions.

Get-VMDiskController

Parameters

- vm (Accepted from the Pipe)
- ControllerID
- server Default : "."
- [Switch]SCSI
- [Switch]IDE

If the SCSI switch is specified (and IDE is not) only SCSI controllers are returned
If the IDE Switch is specified (and SCSI is not) only IDE controllers are returned

If both switches OR neither switch is specified and the controllerID is not specified then all disk controllers are returned.
To return the nth SCSI and nth IDE controller BOTH switches must be specified as well as the controller ID.
The controller is 0 for the first controller of each type, 1 for the second and so on.
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If nothing is passed via the pipe or as a parameter all VMs are tried.
The function returns MSVM_ResourceAllocationSettingData objects for the disk contoller(s). The accompanying format.ps1xml file formats MSVM_ResourceAllocationSettingData objects to show the VM's element name, the resource's Element name, its subtype (Microsoft Emulated IDE Controller or Microsoft Synthetic SCSI controller) and its connection (empty for disk controllers)
Example 1: `Get-VM -server James-2008| Get-VMDiskController -IDE -SCSI`
Gets all VMs on the server named james-2008 and gets IDE and SCSI Disk controllers for each of them.
Example 2: `Get-VMDiskController -IDE -SCSI -vm "%" -Server "James-2008"`
Effectively the same as example 1
Example 3:`$controller=Get-VMDiskController $Tenby -SCSI -ContollerID 0`
Gets the first SCSI controller on the VM pointed to by $tenby and stores the result in a variable.

Get-VMDriveByController

Parameters

- Controller (Accepted from the Pipe)
- LUN Default = "%"

This function returns the drives attached to a given disk controller.
The Controller parameter is a single MSVM_ResourceAllocationSettingData WMI object which represents an IDE or SCSI controller or an array which contains these.
If no controller is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
The LUN parameter identifies the logical unit number of the device. If no LUN is passed all Drives attached to the controller are returned.
The function returns MSVM_ResourceAllocationSettingData objects for the drives(s). The accompanying format.ps1xml file formats MSVM_ResourceAllocationSettingData objects to show the VM's element name, the resource's Element name, its subtype (Microsoft Synthetic Disk Drive or Microsoft Synthetic DVD Drive) and its connection. (empty for drives)
Example 1: `Get-VMDiskController "Tenby" -server "James-2008" -IDE -ContollerID 0 | Get-VMDriveByController`
Gets IDE disk controller 0 for the VM named "Tenby" on the server named "james-2008" and then gets the Drives attached to it.
Example 2: `$drive=Get-VMDriveByController $controller -lun 0`
Gets the first disk attached to the Controller specified by $controller and stores the result in $drive

Get-VMDiskByDrive

Parameters

- Drive (Accepted from the Pipe)

This function returns the disk mounted in a given drive

The Drive parameter is a single MSVM_ResourceAllocationSettingData WMI object which represents a hard-disk drive or DVD drive or an array which contains these.
If no drive is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
The function returns MSVM_ResourceAllocationSettingData objects for the disk(s). The accompanying format.ps1xml file formats MSVM_ResourceAllocationSettingData objects to show the VM's element name, the resource's Element name, its subtype (Microsoft Virtual Hard Disk or Microsoft Virtual CD/DVD Disk) and its connection (the passthrough disk path, VHD file path or ISO file path).

Example 1: `Get-VMDiskController "Tenby" -server "James-2008" -IDE -ControllerID 0 | Get-VMDriveByController | get-vmdiskByDrive`

Gets IDE disk controller 0 for the VM named "Tenby" on the server named "james-2008" and then gets the Drives attached to it. And then gets the disks mounted in them

Example 2: `get-vmdiskByDrive $drive`

Gets the Disk mounted in the logical drive specified by $Drive.

Get-VMDisk

Parameters

- vm (Accepted from the Pipe)
- [switch]snapshot

This function returns all disks attached one or more VMs, including their controller, and LUN information and the path to the disk file and it's type (ISO or hard disk)
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If nothing is passed via the pipe or as a parameter all VMs are tried.
If the snapshot switch is included disks attached to snapshots are included.
The function returns on or more objects 10 properties

- VMElementname — The display name of the VM
- VMGUID — It's GUID (the MSVM_ComputerSystem's "name" property)
- ControllerName — The Element name property of the disk controller object
- ControllerInstanceID — The WMI Instance ID of the disk controller object
- ControllerID — The logical controller number
- DriveName — The element Name of the drive object
- DriveInstanceID — The WMI InstanceID for the drive
- DriveLun — the Slot occupied by the drive (the address property of the disk object)
- DiskPath — The path to the disk (the connection property of the disk object)
- DiskName — The Element name property of the disk object
- DiskInstnace ID — The WMI Instance ID for the drive.

Example 1: `Get-VMDisk (choose-vm -server "James-2008" -multi) | format-table -autosize -property VMname, DriveName, @{Label="Conected to"; expression={"{0,5}{1}:{2}" -f $_.Controllername.split(" ")[0], $_.ControllerID,$_.DriveLun }} , DiskPath`

Gets the disks (without snapshot disks) for the chosen VMs on Server James-2008, and returns the result as a table with the VMName, Drive name, the connection path, and the DiskPath.

Example 2: `Get-VMDisk | foreach {$_.diskpath}`

Returns a list of disks in use

Example 3: `Get-VMDisk | where {$_.ControllerName -match "^IDE"}`
Returns a list of disks attached to IDE controllers

Add-VMSCSIController

Parameters

- vm (Accepted from the Pipe)
- server Default : "."

This function adds a synthetic SCSI controller to the VM(s) passed to it.
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
Example 1: `Add-VMSCSIController $tenby`
Adds a Scsi controller to the VM pointed to by $Tenby
Example 2 : `Get-vm "Core%" -server james-2008 | Add-VMSCSIController`
Gets all Virtual Machines with names starting "Core" on the server named "James-2008" and adds a SCSI controller to each.

Remove-VMSCSIcontroller

Parameters

- Vm
- [int] controllerID
- server Default : "."

This function removes a synthetic SCSI controller to the VM(s) passed to it. By design it only allows a single controller ID and a Single VM to be passed to it.
The VM parameter contains either: a single MsVM_ComputerSystem WMI object which represents a Virtual Machine ora single a string which contains a VM Element name on the specified server
The controllerID identifies which SCSI controller should be removed, if more than one controller is present this parameter is required.
Example 1: `Remove-VMSCSIController $tenby 0`
Removes SCSI contoller 0 from the vm pointed to by $Tenby

Add-VMDRIVE

Parameters

- vm (Accepted from the Pipe)
- [int] ControllerID Default : 0
- [int] LUN
- Server Default : "."
- [switch]DVD
- [switch]SCSI

This function adds a drive- either a DVD or Hard drive to a controller on the specified VM.
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.

If –SCSI is specified then a SCSI controller is selected otherwise an IDE controller is used. The ControllerID is an integer which identifies the controller, with 0 being the first.
If DVD is specified a DVD drive is added at the position identified by LUN (which is also zero based), otherwise a hard drive is added. Note that SCSI controllers do not support DVD drives.
Example 1: `Add-VMDRIVE "tenby" 1 1 -server "james-2008"`
Adds a hard drive to the second position on the second IDE controller of the VM named Tenby on the server Named james-2008
Example 2: `Add-VMDRIVE $tenby 0 3 -SCSI`
Adds a hard drive at LUN 3 of the first SCSI controller on the Server pointed to by $tenby
Example 3: `Get-vm "Core%" | Add-VMDRIVE -controllerID 0 -lun 1 –DVD`
Gets all VMs with names that begin CORE and adds a DVD drive as the second device on their first IDE controller

Function Add-VMDISK

Parameters

- vm
- [int]ControllerID Default : 0
- [int]LUN Default : 0
- VHDPath
- server Default : "."
- [switch]DVD
- [switch]SCSI

This function mounts a disk (usually a VHD file, but also an ISO file or the path to a passthrough disk) into a logical drive
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If –SCSI is specified then a SCSI controller is selected otherwise an IDE controller is used. The ControllerID is an integer which identifies the controller, with 0 being the first.
If DVD is specified a DVD disk is inserted into the DVD drive at the position identified by LUN (which is also zero based), otherwise a hard disk is mounted. Note that SCSI controllers do not support DVD drives.
Example 1: `Add-VMDisk $tenby 0 1 "C:\update.iso" –DVD`
Mounts the Disk image "C:\Update.iso" into the DVD drive at position 1 on IDE controller 0 of the machine pointed to by $Tenby
Example 2: `Add-VMDisk $tenby 0 0 ((get-VHDdefaultPath) +"\tenby.vhd")`
Mounts the Disk image "Tenby.vhd" from the default folder into the hard drive at position 0 on IDE controller 0 of the machine pointed to by $Tenby

Add-VMNewHardDisk

Parameters

- Vm
- [int]ControllerID Default : 0
- [int]LUN Default : 0
- VHDPath
- Size Default: 127GB
- ParentDisk
- Server Default : "."
- [switch]fixed
- [switch]SCSI

This function combines New-VHD, Add-VMDrive and Add VM disk in a single function, it calls NEW-VHD with the ParentDisk, Size, Fixed and VHDPath and Server parameters. If no VHDPath is passed the VHD is created in the default folder using a name which matches the VM. It then calls Add-VMDrive with the Server, VM, SCSI, Controller and LUN parameters to create the drive and Add-VMDisk with VM, ContollerID, Lun, VHDPath, Server and SCSI parameters. Note that the function assumes that the controller exists.
If called with only a VM the function will create a dynamic VHD, named to match the VM, sized at 127GB (the default in the graphical management tools) , attached to IDE controller 0, LUN 0
Example: `Add-VMNewHardDisk –vm $vm -controllerID 0 -lun 3 -vhdpath "(get-VHDdefaultPath)\foo31.vhd" -size 20gb -scsi`
Creates a new 20GB dynamic disk named Foo31.vhd in the default folder, and adds it to position 3 on SCSI controller 0 of the machine pointed to by $VM.

Set-VMDisk

Parameters

- Vm
- [int]controllerID
- [int]LUN
- path
- server Default : "."
- [switch]scsi

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If –SCSI is specified then a SCSI controller is selected otherwise an IDE controller is used. The ControllerID is an integer which identifies the controller, with 0 being the first. The function changes the disk image mounted in the drive at the position identified by LUN (which is also zero based) to whatever is pointed to by Path; this can be a physical drive, an ISO file or a VHD file (hence the name is path, not VHDPath, or image path)

Example 1: `Set-VMDisk Tenby 0 1 (Get-WmiObject -Query "Select * From win32_cdromdrive Where ID  Default : 'D:' " ).deviceID`
Example 2: `Set-VMDiskCore 0 0 "\\?\Volume{d1f72a03-d43a-11dc-8bf1-806e6f6e6963}\Virtual Hard Disks\Core.vhd"`

Remove-VMdrive

Parameters

- vm (Accepted from the Pipe)
- [int]controllerID
- [int]LUN
- server Default : "."
- [switch]scsi
- [switch]Diskonly

Example 1: `Remove-VMdrive "Tenby" 0 1 -SCSI -DiskOnly -Server "James-2008"`
Example 2: `Remove-VMdrive $Core 1 1 -IDE`

Add-VMFloppyDisk

Parameters

- vm (Accepted from the Pipe)
- VFDPath

- server Default : "."

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
The VFDPath contains the path to a virtual floppy disk file, which is added to the specified virtual machine
Example: `add-VMFloppyDisk $core "C:\Users\Public\Documents\Microsoft Hyper-V\Blank Floppy Disk\blank.VFD"`

Remove-VMFloppyDisk

Parameters

- vm (Accepted from the Pipe)
- server Default : "."

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
This function removes any floppy disk attached to the VM provided. If no Floppy is present no error occurs

Get-VMFloppyDisk

Parameters

- vm (Accepted from the Pipe)
- server Default : "."

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If nothing is passed via the pipe or as a parameter all VMs are tried.
The function returns MSVM_ResourceAllocationSettingData objects for the Floppy disk(s). The accompanying format.ps1xml file formats MSVM_ResourceAllocationSettingData objects to show the VM's element name, the resource's Element name, its subtype (Microsoft virtual Floppy disk) and its connection (the path to the VFD file)
Example: `Get-VMFloppyDisk (get-vm -server james-2008) | foreach {$_.connection}`
Produces a list of all the VFD files in the floppy drives of the VMs on the server James-2008

Get-VMBackupScript

Parameters

- vm (Accepted from the Pipe) Default : "%"
- BackupDevice Default : "e:\"
- Server Default : "."
- [switch]dataFiles

This function produces a script to work with the volume shadow copy service to make copies of the files used by Hyper-V . This method of backup is not supported by Microsoft.
Example 1: `Get-VmBackupScript -Server HyperCore -BackupDevice R: | winrs -r:hyperCore cmd.exe`
Example 2: `Get-VmBackupScript -datafiles | CMD`

Get-VMSwitch

Parameters

- Name Default :"%"
- server Default : "."

This function returns MS_VMVirtualSwitch Objects matching the name Parameter or, if no name is specified, all switches from the given servers. The function will convert * in a name to the % sign used in WMI queries. The accompanying format.ps1xml file formats MS_VMVirtualSwitch objects to show the switches name, learnable addresses and Status

Example:`Get-VMswitch "Internal*"`

Choose-VMSwitch

Parameters

- server Default : "."

This function allows the user to select a Virtual Switch from those on one or more servers. It does this by passing the results of a call to the Get-VMSwitch function into the Choose-List function. Usually the choice will be stored in a variable or piped into another command. If only one switch is available choose will return it, otherwise the user is prompted to choose a switch and the appropriate MSVM_VirtualSwitch object is returned

Example: `Choose-VmSwitch -Server James-2008`

Get-VMNic

Parameters

- vm (Accepted from the Pipe)
- server Default : "."
- [switch]Legacy
- [switch]VMBus

If the Legacy switch is specified (and VMBus is not) only legacy/Emulated Adapters are returned
If the VMBus Switch is specified (and legacy is not) only VMbus/Synthetic Adapters are returned
If both switches OR neither switch is specified then all disk adapters are returned.
The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If nothing is passed via the pipe or as a parameter all VMs are tried.
The function returns MSVM_SyntheticEthernetPortSettingData and MSVM_EmulatedEthernetPortSettingData objects for the NIC(s). The accompanying format.ps1xml file formats these objects to show the VM's element name, the Nics's Element name, its subtype (Microsoft Emulated Ethernet Port or Microsoft Emulated Ethernet Port) it's MAC address, whether the MAC address is static, and the ID used to identifiy Synthetic/VM Bus Nics inside the VM. It also obtains the name of the switch that the NIC is connected to which is not a property of the Object.

Example: `Get-VMNic $core -legacy -vmbus`

Get-VMNicport

Parameters

- nic

The NIC parameter contains either single a MSVM_SyntheticEthernetPortSettingData object or a single MSVM_EmulatedEthernetPortSettingData object which represents a Network interface card If no NIC is passed as a parameter, the function looks for input to be piped to it.
The function returns the MSVM_switchPort object that the NIC is connected to. The accompanying format.ps1xml file formats MSVM_switchPort objects to show their caption, element name (a guid) and description, protocol type and status

Example: `Get-VMNic $core -legacy -vmbus | get-vmNicPort`

Get-VMnicSwitch

Parameters

- Nic

The NIC parameter contains either single a MSVM_SyntheticEthernetPortSettingData object or a single MSVM_EmulatedEthernetPortSettingData object which represents a Network interface card If no NIC is passed as a parameter, the function looks for input to be piped to it.
The function returns an MS_VMVirtualSwitch Object for the switch connected to the given NIC The accompanying format.ps1xml file formats MS_VMVirtualSwitch objects to show the switches name, learnable addresses and Status

Example: `(Get-VMNic $vm -legacy -vmbus | get-vmNicSwitch) | foreach-object {$_.elementName}`

Choose-VMNIC

Parameters

- vm (Accepted from the Pipe)
- server Default : "."

The VM parameter contains either:
a single MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
If no VM is passed as a parameter, the function looks for input to be piped to it.
The function returns an object which represents a single NIC from the VM – if the VM has more than one NIC a list is displayed for the user to select from.

Example: `choose-vmnic $Core`

Add-VMNIC

Parameters

- vm (Accepted from the Pipe)
- Virtualswitch
- Mac
- GUID
- server Default : "."
- [switch]legacy

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If the legacy switch is specified the NIC created is an emulated/legacy one, otherwise it is a Synthetic / VMbus one.
The NIC will usually be connected to a virtual switch, but it is possible to leave it disconnected by omitting the VirtualSwitch Parameter

Default NICs are assigned a MAC address dynamically, however the function can assign a Static MAC address through the MAC parameter. No checking is done on this parameter.
Synthetic / VM Bus devices are given an GUID to identify them. If a machine is recreated from its VHD file and the NIC is given a new GUID, it will appear to the VM as a new NIC and be configured with default, so the function supports a GUID parameter to allow the ID to be set to a known value.
Example 1: `Add-VMNIC $tenby (choose-VMswitch)`
Adds a VMBus NIC to the VM pointed to by $tenby, but does not connect it to a switch
Example 2: `Add-VMNIC $tenby (choose-VMswitch) -legacy`
Adds a legacy NIC to the VM pointed to by $tenby, but does not connect it to a switch
Example 3: `get-vm "core%" -Server James-2008 | add-vmnic -virtualSwitch "Internal Virtual Network" -legacy`
Gets all the VMs with names beginning "Core" on the server James-2008, and adds a legacy NIC to each bound to the switch named "internal virtual network"

Set-VMNICSwitch

Parameters

- vm (Accepted from the Pipe)
- nic
- VirtualSwitch
- Server Default : "."

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
The Nic Parameter contains a WMI object representing the Ethernet port on VM – the type will be either MsVM_EmulatedEthernetPortSettingData or MSVM_SyntheticEthernetPortSettingData
The VirtualSwitch parameter either contains an MSVM_VirtualSwitch WMI object representing a virtual switch or a string containing the name of switch

This function connects the port on the specified NIC a newly created port on the specified switch: note that as NICs tend to use the same names, the function does not support using the NIC name. Both the NIC and the VM must be passed as parameters.
Example: `Set-VMNICSwitch $core (choose-vmNic $core) (choose-VMswitch $core.__server)`
Changes the connection for a NIC on the VM pointed to by $core. If there is more than one nic, the user is prompted to choose the NIC. If there is only one switch specified on the server that is used, if there is more than one the user is prompted to choose the switch

Set-VMNICAddress

Parameters

- vm (Accepted from the Pipe)
- nic
- mac

The VM parameter contains either:
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
The Nic Parameter contains a WMI object representing the Ethernet port on VM – the type will be either MsVM_EmulatedEthernetPortSettingData or MSVM_SyntheticEthernetPortSettingData
The MAC parameter a Mac address

This function sets a Network interface card to use the provided static MAC address – a simple check is done to check that the MAC address is a 12 digit Hex number , but nothing else. As NICs tend to use the same names, the function does not support using the NIC name. Both the NIC and the VM must be passed as parameters.

Example: `Set-VMNICAddress $core (choose-vmNic $core) "00155D010101"`

Changes the connection for a NIC on the VM pointed to by $core. If there is more than one nic, the user is prompted to choose the NIC. If there is only one switch specified on the server that is used, if there is more than one the user is prompted to choose the switch

New-VMSwitchPort

Parameters

- virtualSwitch
- Server Default : "."

The VirtualSwitch parameter either contains an MSVM_VirtualSwitch WMI object representing a virtual switch or a string containing the name of switch.

The function returns a new port on the switch which can then be connected to a network card. This function is not intended to be called directly, only by other functions

Remove-VMNIC

Parameters

- vm
- nic
- Server Default : "."

The VM parameter contains either:

a MsVM_ComputerSystem WMI object which represents a Virtual Machine or

a string which contains a VM Element name on the specified server

an array which contains these.

The Nic Parameter contains a WMI object representing the Ethernet port on VM – the type will be either MsVM_EmulatedEthernetPortSettingData or MSVM_SyntheticEthernetPortSettingData or an array of these of objects. It contains an array the function calls itself recursively using each object in turn

This function deletes the specified NIC(S) from the Virtual machine

Example: `Remove-VMNIC $core (choose-vmNiccore)`

Get-VMByMACaddress

Parameters

- mac
- Server Default : "."

The MAC parameter contains either:

A String holding all or part of a MAC address or

an array which contains these.

If the parameter is empty then the fun

The function returns the VM attached to the NIC(s) matching the mac address(es) passed

Example : `Get-VMbymacAddress "00155D000101"`

New-VMPrivateSwitch

Parameters

- VirtualSwitchName (mandatory)
- Ports Default: 1024
- Server Default : "."

Creates a Private virtual Network Switch with the given name and given number of switch ports. A Private switch is one which is only accessible to the VMs connected to it, and not connected to a physical NIC or to the a virtual NIC in the host OS

Example: `New-VMPrivateSwitch "VM network" -server "HVCore"`
Creates a private switch named “VM Network” on the Server “HVcore”

New-VMInternalSwitch

- VirtualSwitchName (mandatory)
- Ports Default: 1024
- Server Default : "."

Creates an Internal Virtual Network Switch with the given name and given number of switch ports. An Internal switch is one which is accessible to the VMs connected to it and to the host OS via a virtual NIC, but is not connected to physical NIC

Example: `New-VMInternalSwitch "Host and VM network`
Creates an internal switch named “Host and VM Network” on the local server

Choose-VMExternalEthernet

Parameters

- server Default : "."

Returns an MSVM_ExternalEthernetPort WMI object. This function allows the user to select a physical Ethernet port which is not already bound to a switch. If there is only one such port then it is returned, otherwise the user is asked to select one from a list. The purpose of this is to avoid having to enter the NIC name in the New-VMExtenalSwtich

New-VMExternalSwitch

- VirtualSwitchName (mandatory)
- ExternalEthernet
- Ports Default: 1024
- Server Default : "."

The ExtenalEthernet Parameter is either
A MSVM_ExternalEthernetPort WMI Object or
A String which uniquely identifies an Ethernet port – the full name does not need to be given, a trailing wildcard is applied so “Intel” will match “Intel (R) 82566MM Gigabit network Connection”
If the parameter is not provided the function looks to the pipe for a single matching object
The function Creates an External virtual Network Switch with the given name and given number of switch ports, bound to the specified physical Ethernet card An External switch is one which is accessible to the VMs connected to it, to the host OS via a virtual NIC and provides access to and from a physical Network via a physical NIC in the host OS. When a switch is configured there is a temporary loss of connectivity while the protocols bound to the host’s physical NIC are moved to a virtual NIC.

Example 1: `choose-VMExternalEthernet | New-VMExternalSwitch -virtualSwitchNameName "Wired virtual Network"`
Allows the user to select a free connection (if there is more than one) and creates a switch named “Wired Virtual Network”
Example 2: `New-VMExternalSwitch -virtualSwitchNameName "Wired virtual Network" -ext "Broadcom" -Server HVCore`
Creates a switch named “Wired Virtual Network” bound to the BroadCom network card on the server HVCore.

Get-VMSerialPort

Parameters

- vm (Accepted from the Pipe)
- Portno
- Server Default : "."

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If nothing is passed via the pipe or as a parameter all VMs are tried.
PortNo Is a serial port number
The function returns the Serial Port information for the specified VMs (or all VMs if no parameter is specified)
Example : `Get-VMSerial Port –VM Core –port 1`

Set-VMSerialPort

Parameters

- vm (Accepted from the Pipe)
- Portno Default: 1
- Connection (Mandatory)
- Server Default : "."

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
PortNo Is a serial port number
Connection is the name of a named pipe
The function connects the specified VM(s) serial Port to a named pipe.
Example : `set-VMSerialPort –VM Core –port 1 "\\.\pipe\core"`

Export-VM

Parameters

- vm (Accepted from the Pipe)
- path
- Server Default : "."
- [switch]copyState
- [switch]Wait
- [switch]Preserve

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
This function exports one or more VMs to the destination specified in the Path parameter

If the copy state switch is included the export includes the current state of VHD(s) and Memory.
If the Wait Switch is specified execution is paused and a progress indicator shown until the export is complete
If the preserve switch is specified in addition to the wait switch, the supporting files which are normally destroyed during the import process are packed into a zip file named importFiles.Zip.
Example : `Export-VM Core -path  D:\exports\core -copyState`

Import-VM

Parameters

- path (Accepted from the Pipe)
- [switch]ReImportVM
- [switch]ReUseIDs
- [switch]Wait
- [switch]Preserve
- Server Default: "."

Path is either a System.IO.Directory info object representing a directory containing an exported VM or
A String containing the path to such a directory
If no path is passed the filter looks to the Pipe for a value
If the preserve switch is specified the supporting files which are normally destroyed during the import process are packed into a zip file named importFiles.Zip before starting the import.
If the reImportVM switch is specified importFiles.Zip is unpacked before starting the import
If the ReUseIDs switch is specified the imported VM is created with the same GUIDs that were used in the exported VM. This must not be used if export and import are used to clone a VM on the same host.
If the Wait Switch is specified execution is paused and a progress indicator shown until the export is complete
Example 1: `import-VM -path  D:\exports\core`
Imports the VM found in D:\Exports\core
Example 2: `Dir D:\Exports | import-VM -preserve -reuseIDs -wait`
Imports each of the VMs found in D:\Exports one at a time, zipping their files and using the old GUIDs

Get-VMSnapshot

Parameters

- vm (Accepted from the Pipe)
- Name Default :"%"
- Server Default : "."
- [Switch]Newest

VM is either
 a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
 a string which contains a VM Element name on the specified server
 an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
If nothing is passed via the pipe or as a parameter all VMs are tried.
This function returns one or MSVM_VirtualSystemSettingData WMI object(s) representing SnapShots for the specified VM(s). An individual snapshot can be selected by specifying its name, and the newest snapshot can be selected by specifying the -Newest switch.
Example: `Get-Vmsnapshot $Core -newest`
Returns the newest snapshot of the VM pointed to by $Core

Get-VMSnapshotTree

Parameters

- vm
- Server Default : "."

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
The function displays the snapshots of a Virtual machine – returned by Get-VMSnapshot, arranged in tree format so you can see which snapshots are descended from which others.

Example: `Get-VmsnapshotTree $Core`
Gives a tree view of the snapshots of the VM pointed to by $Core

Choose-VMSnapshot

Parameters

- vm (Accepted from the Pipe)
- Server Default : "."

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
If no VM is passed as a parameter, the function looks for input to be piped to it.
This function retrieves the snapshots for the given VM, if there is a single snapshot that snapshot is returned, if there is more than one the user is prompted to select one.

Example: `Choose-Vmsnapshot $Core`
If the VM pointed to by $Core has more than one snapshot prompts the user to select one of them

Filter New-VMSnapshot

Parameters

- vm (Accepted from the Pipe)
- Note
- Server Default : "."
- [Switch]wait

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
The function creates a new snapshot of one more VMs. The Note parameter holds text which can be stored for the Snapshot(s) . If the –wait switch is specified the Filter does not continue until the snapshot process is complete

Example 1: `new-vmsnapshot $Core`
Makes a snapshot of the server pointed to by $core

Example 2: `get-vm "core%" -server "James-2008" | new-VmSnapshot –wait`
Gets all the VMs with names that begin “core” and snapshots them one by one.

Remove-VMSnapshot

Parameters

- snapshot
- [Switch]Tree

The snapshot parameter holds an MSVM_VirtualSystemSettingData WMI Object representing the snapshot to be removed. If no snapshot is passed as a parameter, the function looks for input to be piped to it.
If the –tree switch is specified all snapshots below the specified one are also deleted.
Example: `choose-vmsnapshot $Core | remove-vmsnapshot -tree`

Apply-VMSnapshot

Parameters

- SnapShot
- [Switch]force
- [Switch]Restart
- [Switch]wait

The snapshot parameter holds an MSVM_VirtualSystemSettingData WMI Object representing the snapshot to be apply. If no snapshot is passed as a parameter, the function looks for input to be piped to it.
A snapshot cannot be applied to a running VM so the –force switch specifies that the VM should be stopped.
If the -restart switch is specified the function waits until the snapshot has been applied and then restarts the VM
If the –wait switch h is specified the function waits until the snapshot has been applied, without restarting the VM
Example: `choose-vmsnapshot $Core | Apply-vmsnapshot –force –restart`
Prompts the user to choose a snapshot (if there is more than one) for the server pointed to by $core, applies this snapshot – even if the VM is running, and restarts the VM.

Rename-VMSnapShot

Parameters

- VM
- SnapName
- newName
- Server Default : "."

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks at for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
The function finds the Snapshot with the specified name and renames it to the name held in newName.
Example **:**
```
Rename-vmSnapshot –v $core –s (choose-vmsnapshot $core).elementName`
                          –n "defeault"
```
Prompts the user to select one of the snap shots on the VM pointed to by $core and renames it to “default”

Update-VMSnapshot

- VM
- SnapName
- Note
- Server Default : "."

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.

If no VM is passed as a parameter, the function looks at for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
This function is a wrapper for New-Snapshot, remove-Snapshot, and rename-snapshot.
If no snapshot name is specified the function gets the most recent snapshot. It renames that snapshot to “Delete-me”. It creates a new snapshot, using the note if one is specified, and renames the new snapshot to the given name. Finally it removes the old version of the snapshot.
Example `Update-VmSnapshot –VM “Core-1” –snapName “Lab start”`
Replaces the snapshot named “lab start” on the VM named “core-1” with a new snapshot

Get-VMJPEG

Parameters

- vm (Accepted from the Pipe)
- [int]Width Default : 800
- [int]Height Default : 600
- Path
- Server Default : "."

VM is either
a MsVM_ComputerSystem WMI object which represents a Virtual Machine or
a string which contains a VM Element name on the specified server
an array which contains these.
If no VM is passed as a parameter, the function looks at for input to be piped to it.
If an array is passed the function calls itself recursively for each item.
The function saves a JPEG image of a running VM. Width and Height specify the size of the image, but must be less than or equal to the screen size set inside the VM
Path is specifies where the file is saved. If the Path is only a folder the file name matches the VM name, if the path is only a file name the JPG is stored in the current directory.

Example 1 : `Get-VMJPEG “core”`
Creates an 800x600 image named core.jpg in the current directory
Example 2: `Get-vm -Running -server "James-2008" | Get-VMJPEG -w 320 -h 240 -path images`
Gets all VMs running on the server James-2008 and stores a 320x240 thumbnail of them in the folder named images.
Example 3: `While (true) { Get-VMJPEG -vm "core" -w 640 -h 480 -path ((get-date).toLongTimeString().replace(":","-") + ".JPG") ; Sleep -Seconds 10}`
Loops until interrupted , getting a 640x480 image of the VM named “core” and saving it with a path based on the date and time every 10 seconds.

WMI Objects used by the functions

MsVM_AllocationCapabilities	New-VMRasd
MsVM_ComputerSystem	Apply-VMSnapshot, Get-VM, Get-VMDiskController, Get-VMSettingData, New-VM, Set-VM,
MsVM_EmulatedEthernetPortSettingData	Get-VMNic
MsVM_ExternalEthernetPort	Choose-VMExternalEthernet, New-VMExternalSwitch,
MsVM_HeartbeatComponent	Test-VMHeartBeat
MsVM_ImageManagementService	Compact-VHD, Convert-VHD, Expand-VHD, Get-VHDInfo, Get-VMBackupScript, Merge-VHD, Mount-VHD, New-VFD, New-VHD, Test-VHD, UnMount-VHD
MsVM_KvpExchangeComponent	Get-VMKVP
MsVM_KvpExchangeDataItem	Add-VMKVP, Remove-VMKVP,
MsVM_MemorySettingData	Get-VMMemory
MsVM_MountedStorageImage	Mount-VHD
MSVM_Processor	Get-VMCPUCount, Get-VMProcessor ,
MsVM_ResourceAllocationSettingData	Add-VMFloppyDisk, Get-VMDiskByDrive, Get-VMDiskController, Get-VMDriveByController, Get-VMFloppyDisk, Get-VMSerialPort
MsVM_SettingsDefineCapabilities	New-VMRasd
MsVM_ShutdownComponent	Shutdown-VM
MsVM_SwitchLANEndpoint	New-VMInternalSwitch
MsVM_SwitchPort	Get-VMNicport, Set-VMNICSwitch,
MsVM_SyntheticEthernetPortSettingData	Get-VMNic
MsVM_VirtualSwitch	Get-VMnicSwitch, Get-VMSwitch, New-VMExternalSwitch, New-VMInternalSwitch, New-VMPrivateSwitch, New-VMSwitchPort, Set-VMNICSwitch
MsVM_VirtualSystemGlobalSettingData	Get-VMBackupScript, Set-VM,
MsVM_virtualSystemManagementService	Add-VMDisk, Add-VMDrive, Add-VMFloppyDisk, Add-VMKVP, Add-VMNIC, Add-VMSCSIController, Apply-VMSnapshot, Export-VM, Get-VhdDefaultPath, Get-VMJPEG, Import-VM, New-VM, New-VMSnapshot, Remove-VM, Remove-VMdrive, Remove-VMKVP, Remove-VMNIC, Remove-VMSCSIcontroller, Remove-VMSnapshot, Rename-VMsnapshot, Set-VM, Set-VMCPUCount, Set-VMDisk, Set-VMMemory, Set-VMNICAddress, Set-VMNICSwitch, Set-VMSerialPort
MsVM_VirtualSystemSettingData	Get-VMSettingData, Get-VMSnapshot, New-VM