pyVirtualize Documentation

Release alpha

Rocky Ramchandani

Sep 08, 2018

Contents:

CHAPTER 1

Indices and tables

- genindex
- modindex
- search

1.1 pyVirtualize

pyVirtualize is build over pyVmomi, hence it has ability to perform all the operations what vSphere client is able to. Not only vSphere, but pyVirtualize provides an interface for VMware Horizon View session management.

pyVirtualize provides easy interfaces to:

- Connect to VMWare's ESX, ESXi, Virtual Center, Virtual Server hosts and View Connnection server.
- Query hosts, datacenters, resource pools, virtual machines and perform various operations over them.
- VMs operations: power, file, process, snapshot, admin, utilities
- Horizon View: desktop, farms, rdsh, -pool management, querying connection and more.

An of course, you can use it to access all the API through python.

class `pyVirtualize.`**`vSphere`**(*address*, *username='root'*, *password='ca$hc0w'*, *port=443*, *sslContext=None*)

Bases: `pyVirtualize.utils.klasses.PersistableClass`

Just like vSphere client, this class helps to access the vCenter Server and ultimately manages ESXi servers.

Parameters

- **`address`** – (str) Server address of the ESXi / vCenter host.
- **`username`** – (str) Username to login with. [default = root]
- **`password`** – (str) Password to login with. [default = ca$hc0w]
- **`port`** – (int) Port on which object should connect to host. [default = 443]

Example

>> from pyVirtualize.pyvSphere import vSphere

>> vsphere = vSphere(address='10.112.67.60', username='administrator@vsphere.local', password='Ca$hc0w1')

>> vsphere.login()

>>

>> vsphere.Datacenters

{'datacenter1': 'vim.Datacenter:datacenter-2', 'datacenter2': 'vim.Datacenter:datacenter-7'}

>>

>> vsphere.VirtualMachines

{'W7x64': 'VirtualMachine:vm-295', 'W7x32': 'vim.VirtualMachine:vm-183', … }

`Clusters`

List of Clusters objects.

Returns dictionary object with key, value pair or Cluster name and Cluster object.

`ComputeResources`

`Datacenters`

List of Datacenter objects.

Returns dictionary object with key, value pair or Datacenter name and Datacenter object.

`Datastores`

List of Datastore objects.

Returns dictionary object with key, value pair or Datastore name and Datastore object.

`Folders`

`Hosts`

List of Host objects.

Returns dictionary object with key, value pair or Host name and Host object.

`Networks`

`ResourcePools`

`VirtualMachines`

List of VirtualMachines objects.

Returns dictionary object with key, value pair or VirtualMachine name and VirtualMachine object.

`about`

Returns *vim.AboutInfo*, vmomi info object.

```
(vim.AboutInfo) {
    dynamicType = <unset>,
    dynamicProperty = (vmodl.DynamicProperty) [],
    name = 'VMware vCenter Server',
    fullName = 'VMware vCenter Server 6.0.0 build-3634794',
    vendor = 'VMware, Inc.',
    version = '6.0.0',
    build = '3634794',
    localeVersion = 'INTL',
    localeBuild = '000',
    osType = 'linux-x64',
    productLineId = 'vpx',
    apiType = 'VirtualCenter',
    apiVersion = '6.0',
    instanceUuid = '3c6c809d-eb71-4a4b-bd9d-23ca07928f51',
    licenseProductName = 'VMware VirtualCenter Server',
    licenseProductVersion = '6.0'
}
```

`apiType`

Returns (str) API Type of the specified server. VirtualCenter/ESXi

`create_new_datacenter`(*name*)

`create_new_folder`(*name*)

`load`(*cls_file*)

`login`()

Determine the most preferred API version supported by the specified VCenter server, then connect to the specified server using that API version, and logs into it.

`logout`()

Logs out from the specified VCenter server.

`rescan_virtual_machines`()

Refreshes the *VirtualMachines* list associated to server.

`service_instance`

VMOMI ServiceInstance.

`store`(*cls_file*)

class `pyVirtualize.`**`Process`**

Bases: `object`

Utility to launch new processes

static `start_process`(*cmd*)

class `pyVirtualize.`**`WinUtils`**

Bases: `object`

Utilities to work with Windows OS

static `del_map_drive`(*net_share*)

Delete the given network share :param net_share: :return:

static `map_drive`(*user*, *password*, *net_share*)

Map given network share :param user: :param password: :param net_share: :return:

static `read_reg`(*reg_key*, *reg_val*)

Fetch registry given the reg key+value combo :param reg_key: :param reg_val: :return:

CHAPTER 2

VirtualMachine class

class `pyVirtualize.pyvSphere.vm.`**`VirtualMachine`**(*vmomi_obj*, *service_instance*, ***kwargs*)
Bases: `object`

`operations`
Set of operations supported over Virtual machine.

`set_credentials`(*username*, *password*, *credentials_type*, *default=False*)
Adds the credentials for the Guest operations on this virtual machine. :param username: (str) Guest username. ex: domain_nameuser_name :param password: (str) Guest password. :param credentials_type: (any immutable) A dictionary key which helps to different credentials for the system. :param default: (bool) If specified then for all the guest operation these credentials will be used by default.

Unless with the operation one specifies the *credential_type* with it.

Examples

>> vm.set_credentials(username="myDomaindomainUser", password="secret", credentials_type="user", default=True)

>> vm.set_credentials(username="myDomaindomainAdmin", password="secret", credentials_type="admin")

class `pyVirtualize.pyvSphere.vm._base.`**`Details`**(*vobj*)

`config`

Returns *vim.vm.Summary.ConfigSummary*, the config details of the current machine.

(vim.vm.Summary.ConfigSummary) {
dynamicType = <unset>,
dynamicProperty = (vmodl.DynamicProperty) [],
name = 'Jenkins',

```
        template = false,
        vmPathName = ‘[datastore1] Jenkins/Jenkins.vmx’,
        memorySizeMB = 4096,
        cpuReservation = 0,
        memoryReservation = 0,
        numCpu = 4,
        numEthernetCards = 1,
        numVirtualDisks = 1,
        uuid = ‘420c6ef6-eef0-03ff-20f2-5d2479b2afdc’,
        instanceUuid = ‘500ce065-5f5b-17fa-fa8f-d7033e548ecb’,
        guestId = ‘windows7_64Guest’,
        guestFullName = ‘Microsoft Windows 7 (64-bit)’,
        annotation = ”,
        product = <unset>,
        installBootRequired = false,
        ftInfo = <unset>,
        managedBy = <unset>
}
```

guest

Returns *vim.vm.Summary.GuestSummary*, the guest details of the current machine.

```
(vim.vm.Summary.GuestSummary) {
        dynamicType = <unset>,
        dynamicProperty = (vmodl.DynamicProperty) [],
        guestId = ‘windows7_64Guest’,
        guestFullName = ‘Microsoft Windows 7 (64-bit)’,
        toolsStatus = ‘toolsOk’,
        toolsVersionStatus = ‘guestToolsCurrent’,
        toolsVersionStatus2 = ‘guestToolsCurrent’,
        toolsRunningStatus = ‘guestToolsRunning’,
        hostName = ‘W7x64’,
        ipAddress = ‘10.112.19.116’
}
```

runtime

Returns *vim.vm.RuntimeInfo*, the runtime details of the current machine.

```
(vim.vm.RuntimeInfo) {
        dynamicType = <unset>,
        dynamicProperty = (vmodl.DynamicProperty) [],
        device = (vim.vm.DeviceRuntimeInfo) [
                (vim.vm.DeviceRuntimeInfo) {
                        dynamicType = <unset>,
            dynamicProperty = (vmodl.DynamicProperty) [],
            runtimeState = (vim.vm.DeviceRuntimeInfo.VirtualEthernetCardRuntimeState) {
                dynamicType = <unset>,
                dynamicProperty = (vmodl.DynamicProperty) [],
                vmDirectPathGen2Active = false,
                vmDirectPathGen2InactiveReasonVm = (str) [],
                vmDirectPathGen2InactiveReasonOther = (str) [
                    'vmNptIncompatibleNetwork'
                ],
                vmDirectPathGen2InactiveReasonExtended = <unset>,
                reservationStatus = <unset>
            },
            key = 4000
        }
    ],
    host = 'vim.HostSystem:host-14',
    connectionState = 'connected',
    powerState = 'poweredOn',
    faultToleranceState = 'notConfigured',
    dasVmProtection = <unset>,
    toolsInstallerMounted = false,
    suspendTime = <unset>,
    bootTime = 2017-02-17T14:39:35.245193Z,
    suspendInterval = 0L,
    question = <unset>,
    memoryOverhead = <unset>,
    maxCpuUsage = 9196,
    maxMemoryUsage = 4096,
    numMksConnections = 0,
    recordReplayState = 'inactive',
    cleanPowerOff = <unset>,
    needSecondaryReason = <unset>,
    onlineStandby = false,
    minRequiredEVCModeKey = <unset>,
    consolidationNeeded = false,
    offlineFeatureRequirement = (vim.vm.FeatureRequirement) [
        (vim.vm.FeatureRequirement) {
            dynamicType = <unset>,
            dynamicProperty = (vmodl.DynamicProperty) [],
            key = 'cpuid.lm',
            featureName = 'cpuid.lm',
            value = 'Bool:Min:1'
        }
    ],
    featureRequirement = (vim.vm.FeatureRequirement) [
        (vim.vm.FeatureRequirement) {
            dynamicType = <unset>,
            dynamicProperty = (vmodl.DynamicProperty) [],
            key = 'cpuid.SSE3',
            featureName = 'cpuid.SSE3',
            value = 'Bool:Min:1'
        },...
    ],
    featureMask = (vim.host.FeatureMask) [],
    vFlashCacheAllocation = 0L,
    paused = false,
    snapshotInBackground = false,
    quiescedForkParent = <unset>
}
```

storage

Returns *vim.vm.Summary.StorageSummary*, the storage details of the current machine.

```
(vim.vm.Summary.StorageSummary) {
    dynamicType = <unset>,
    dynamicProperty = (vmodl.DynamicProperty) [],
    committed = 38614424818L,
    uncommitted = 239075873L,
    unshared = 34120663040L,
    timestamp = 2017-05-18T09:16:22.357187Z
}
```

2.1 VirtualMachines Operations

class `pyVirtualize.pyvSphere.vm.operation.file.`**`FileOperations`**(*vim*, *timeout=None*, ***kwargs*)

Bases: `pyVirtualize.pyvSphere.vm.operation._base.BaseOperation`

FileOperations provides APIs to manipulate the guest operating system file options.

`create_local`(*path*, *contents=None*)

`create_remote`(*path*, *contents=None*, *credentials=None*)

Parameters

- **`path`** –
- **`contents`** –
- **`credentials`** –

Returns

`delete_local`(*path*)

`delete_remote`(*path*, *credentials=None*)

`download`(*src*, *dest*, *credentials=None*, *overwrite=True*)

Downloads the file/directory into the guest operating system. In case of file it will 'pyVirtualize' requires the file path and 'dest' also requires the path specifying file name. But when 'pyVirtualize' is path representing folder/directory then it will copy the contents of the directory recursively into specified 'dest' path.

NOTE: It creates the directories automatically missing in the path. Also in case of file, then it overwrites in the 'dest'.

Parameters

- **`src`** – (str) Valid source location that can be either file or directory.
- **`dest`** – (str) A destination location for file or directory respectively.
- **`credentials`** – (str) Specify the credentials type string. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.
- **`overwrite`** – (bool) Overwrites the file if present on local when download.

`get_remote_dir_desc`(*path*, *credentials=None*)

`list_dir_in_vm`(*path*, *credentials=None*)

Returns information about files or directories in the guest.

Parameters

- **`path`** – (str) The complete path to the directory or file to query.
- **`credentials`** – (str) Specify the credentials type string. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.

Returns

vim.vm.guest.FileManager.ListFileInfo: A *GuestListFileInfo* object containing information for all the matching files in the filePath and the number of files left to be returned.

`move_local`(*src*, *dest*, *credentials=None*)

`move_remote`(*src*, *dest*, *credentials=None*)

`remote_path_exists`(*path*, *credentials=None*)

Return the presence of the path.

Parameters

- **`path`** – (str) The complete path to the directory or file to query.
- **`credentials`** – (str) Specify the credentials type string. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.

Returns (bool) 'True' if path exists or else 'False' if it doesn't.

`upgrade_vm_tools`()

`upload`(*src*, *dest*, *credentials=None*, *overwrite=True*)

Uploads the file/directory into the guest operating system. In case of file it will 'pyVirtualize' requires the file path and 'dest' also requires the path specifying file name. But when 'pyVirtualize' is path representing folder/directory then it will copy the contents of the directory recursively into specified 'dest' path.

NOTE: It creates the directories automatically missing in the path. Also in case of file, then it overwrites in the 'dest'.

Parameters

- **`src`** – (str) Valid source location that can be either file or directory.
- **`dest`** – (str) A destination location for file or directory respectively.
- **`credentials`** – (str) Specify the credentials type string. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.
- **`overwrite`** – (bool) Overwrites the file if present on guest while upload.

class `pyVirtualize.pyvSphere.vm.operation.power.`**`PowerOperations`**(*vim*, *timeout=None*, ***kwargs*)

Bases: `pyVirtualize.pyvSphere.vm.operation._base.BaseOperation`

PowerOperations provides APIs to manipulate the guest operating system power options.

`power_off`(*sync=True*)

Performs the hard power off on this virtual machine.

Parameters **`sync`** – (bool) When 'sync' is set to True, command will wait until VM is completely powered Off. Else it wait just trigger the power Off and returns back.

`power_on`(*sync=True*, *wait_for_guest_ready=True*)

Powers on this virtual machine.

Parameters

- **`sync`** – (bool) When 'sync' is set to True, command will wait until VM is completely powered On. Else it will just trigger the power On and returns back.
- **`wait_for_guest_ready`** – (bool) When 'wait_for_guest_ready' is to True, command will wait until guest operations is ready i.e. logged-in successfully. Else it will return when it power's On completely. 'wait_for_guest_ready' works when 'sync' is enabled/True.

`reset`(*sync=True*)

Resets power on this virtual machine. If the current state is poweredOn, then this method first performs powerOff(hard). Once the power state is poweredOff, then this method performs powerOn(option). Although this method functions as a powerOff followed by a powerOn, the two operations are atomic with respect to other clients, meaning that other power operations cannot be performed until the reset method completes.

Parameters **`sync`** – (bool) When 'sync' is set to True, command will wait until VM is completely powered Off and powered On back.

Returns

`restart`(*sync=True*)

Sends the restart command to guest operating system.

Parameters **`sync`** – (bool) When 'sync' is set to True, command will wait until VM is completely powered Off and powered On back. Else it will just trigger the restart and returns back;; but it will wait till shutdown of the system atleast.

`shutdown`(*sync=True*)

Issues a command to the guest operating system asking it to perform a clean shutdown of all services.

Parameters **sync** – (bool) When 'sync' is set to True, command will wait until VM is completely powered Off. Else it will just trigger the shutdown and returns back.

upgrade_vm_tools()

class pyVirtualize.pyvSphere.vm.operation.process.**ProcessOperations**(*vim*, *timeout=None*, ***kwargs*)

Bases: pyVirtualize.pyvSphere.vm.operation._base.BaseOperation

ProcessOperations provides APIs to manipulate the guest operating system processes.

execute(*program*, *arguments=''*, *cwd=''*, *env_vars=None*, *start_minimized=False*, *wait_for_guest_ready=True*, *wait_for_program_to_exit=True*, *credentials=None*, *interactive=True*)

Starts a program in the guest operating system. If program is not executed then it will raise an Exception ”

Parameters

- **program** – (str) The absolute path to the program to start. For Linux guest operating systems, /bin/bash is used to start the program. For Solaris guest operating systems, /bin/bash is used to start the program if it exists. Otherwise /bin/sh is used. If /bin/sh is used, then the process ID returned by StartProgramInGuest will be that of the shell used to start the program, rather than the program itself, due to the differences in how /bin/sh and /bin/bash work. This PID will still be usable for watching the process with ListProcessesInGuest to find its exit code and elapsed time.
- **arguments** – (str) The arguments to the program. In Linux and Solaris guest operating systems, the program will be executed by a guest shell. This allows stdio redirection, but may also require that characters which must be escaped to the shell also be escaped on the command line provided. For Windows guest operating systems, prefixing the command with "cmd /c" can provide stdio redirection.
- **cwd** – (str) The absolute path of the working directory for the program to be run. VMware recommends explicitly setting the working directory for the program to be run. If this value is unset or is an empty string, the behavior depends on the guest operating system. For Linux guest operating systems, if this value is unset or is an empty string, the working directory will be the home directory of the user associated with the guest authentication. For other guest operating systems, if this value is unset, the behavior is unspecified.
- **env_vars** – (str) An array of environment variables, specified in the guest OS notation (eg PATH=c:bin;c:windowssystem32 or LD_LIBRARY_PATH=/usr/lib:/lib), to be set for the program being run. Note that these are not additions to the default environment variables; they define the complete set available to the program. If none are specified the values are guest dependent.
- **start_minimized** – (bool) This argument applies only for 'Windows' guests. Makes any program window start minimized.
- **wait_for_guest_ready** – (bool) This argument specifies, whether to check for Guest operation are ready before starting the program. If 'True', it halts execution until Guest is ready (or logged in, maybe using auto-logon), Else, it directly starts execution.
- **wait_for_program_to_exit** – (bool) This argument will wait until the programs gets executed and returns back some value. It can also be configured with a

'time-out' period, which will take value from parent VM object.

- **`credentials`** – (str) Specify the credentials type string. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.
- **`interactive`** – (bool) This is set to true if the client wants an interactive session in the guest.

Returns

vim.vm.guest.ProcessManager.ProcessInfo: Attributes:

name (str): The process name

pid (long): The process ID

owner (str): The process owner

cmdLine (str): The full command line

startTime (datetime): The start time of the process

endTime (datetime, optional): If the process was started using StartProgramInGuest then the process completion time will be available if queried within 5 minutes after it completes.

exitCode (int, optional): If the process was started using StartProgramInGuest then the process exit code will be available if queried within 5 minutes after it completes.

Raises

vim.fault.GuestOperationsFault: if there is an error processing a guest operation.

vim.fault.InvalidState: if the operation cannot be performed because of the virtual machine's current state.

vim.fault.TaskInProgress: if the virtual machine is busy.

vim.fault.FileFault: if there is a file error in the guest operating system.

vim.fault.GuestOperationsUnavailable: if the VM agent for guest operations is not running.

vim.fault.InvalidPowerState: if the VM is not powered on.

vim.fault.FileNotFound: if the program path does not exist.

vim.fault.CannotAccessFile: if the program path cannot be accessed.

vim.fault.GuestPermissionDenied: if the program path cannot be run because the guest authentication will not allow the operation.

vim.fault.InvalidGuestLogin: if the the guest authentication information was not accepted.

vim.fault.GuestComponentsOutOfDate: if the guest agent is too old to support the operation.

vim.fault.OperationNotSupportedByGuest: if the operation is not supported by the guest OS.

vim.fault.OperationDisabledByGuest: if the operation is not enabled due to guest agent configuration.

pyVirtualize.utils.exceptions.ProgramNotExecuted `get_env_var` (*names=''*, *credentials=None*, *interactive=True*)

Reads an environment variable from the guest OS. If the authentication uses interactiveSession, then the environment being read will be that of the user logged into the desktop. Otherwise it's the environment of the system user.

Parameters

- **`names`** – (str, optional) The names of the variables to be read. If not set, then all the environment variables are returned.
- **`credentials`** – (str) Specify the credentials type string. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.
- **`interactive`** – (bool) This is set to true if the client wants an interactive session in the guest.

Returns A string array containing the value of the variables,

or all environment variables if nothing is specified. The format of each string is "name=value". If any specified environment variable isn't set, then nothing is returned for that variable.

Raises

vim.fault.GuestOperationsFault: if there is an error processing a guest operation.

vim.fault.InvalidState: if the operation cannot be performed because of the virtual machine's current state.

vim.fault.TaskInProgress: if the virtual machine is busy. accepted by the guest OS.

vim.fault.GuestOperationsUnavailable: if the VM agent for guest operations is not running.

vim.fault.InvalidPowerState: if the VM is not powered on.

vim.fault.GuestPermissionDenied: if there are insufficient permissions in the guest OS.

vim.fault.InvalidGuestLogin: if the the guest authentication information was not accepted.

vim.fault.GuestComponentsOutOfDate: if the guest agent is too old to support the operation.

vim.fault.OperationNotSupportedByGuest: if the operation is not supported by the guest OS.

vim.fault.OperationDisabledByGuest: if the operation is not enabled due to guest agent configuration.

`list_processes` (*pids=[]*, *credentials=None*, *interactive=True*)

List the processes running in the guest operating system, plus those started by StartProgramInGuest that have recently completed.

Parameters

- **`pids`** – (list) If set, only return information about the specified processes. Otherwise, information about all processes are returned. If a specified processes does not exist, nothing will be returned for that process.
- **`credentials`** – (str) Specify the credentials type string. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.

Returns

[vim.vm.guest.ProcessManager.ProcessInfo]: The list running processes is returned in an array of **'GuestProcessInfo'_** structures. Attributes:

name (str): The process name

pid (long): The process ID

owner (str): The process owner

cmdLine (str): The full command line

startTime (datetime): The start time of the process

endTime (datetime, optional): If the process was started using StartProgramInGuest then the process completion time will be available if queried within 5 minutes after it completes.

exitCode (int, optional): If the process was started using StartProgramInGuest then the process exit code will be available if queried within 5 minutes after it completes.

Raises

vim.fault.GuestOperationsFault: if there is an error processing a guest operation.

vim.fault.InvalidState: if the operation cannot be performed because of the virtual machine's current state.

vim.fault.TaskInProgress: if the virtual machine is busy.

vim.fault.GuestOperationsUnavailable: if the VM agent for guest operations is not running.

vim.fault.InvalidPowerState: if the VM is not powered on.

vim.fault.GuestPermissionDenied: if there are insufficient permissions in the guest OS.

vim.fault.InvalidGuestLogin: if the the guest authentication information was not accepted.

vim.fault.GuestComponentsOutOfDate: if the guest agent is too old to support the operation.

vim.fault.OperationNotSupportedByGuest: if the operation is not supported by the guest OS.

vim.fault.OperationDisabledByGuest: if the operation is not enabled due to guest agent configuration.

`upgrade_vm_tools`()

class `pyVirtualize.pyvSphere.vm.operation.snapshot.`**`SnapshotOperations`**(*vim*, *timeout=None*, ***kwargs*)

Bases: `pyVirtualize.pyvSphere.vm.operation._base.BaseOperation`

SnapshotOperations provides APIs to manipulate the snapshot operations on the Virtual Machine.

`create`(*name*, *desc=''*, *memory=True*, *quiesce=False*)

Creates a new snapshot of this virtual machine. As a side effect, this updates the current snapshot. Snapshots are not supported for Fault Tolerance primary and secondary virtual machines. Any % (percent) character used in this name parameter must be escaped, unless it is used to start an escape sequence. Clients may also escape any other characters in this name parameter.

Parameters

- **`name`** – (str) The name for this snapshot. The name need not be unique for this virtual machine.
- **`desc`** – (str, optional) A description for this snapshot. If omitted, a default description may be provided.
- **`memory`** – (bool, optional) If TRUE, a dump of the internal state of the virtual machine (basically a memory dump) is included in the snapshot. Memory snapshots consume time and resources, and thus take longer to create. When set to FALSE, the power state of the snapshot is set to powered off. **'capabilities'_** indicates whether or not this virtual machine supports this operation. Default: 'True'
- **`quiesce`** – (bool, optional) If TRUE and the virtual machine is powered on when the snapshot is taken, VMware Tools is used to quiesce the file system in the virtual machine. This assures that a disk snapshot represents a consistent state of the guest file systems. If the virtual machine is powered off or VMware Tools are not available, the quiesce flag is ignored.

`remove`(*name*)
Removes the specified snapshot from the machine.

Parameters **`name`** – (str) Name of the snapshot.

`remove_all`()
Removes all snapshots from the machine.

`remove_current`()
Removes the current snapshot from the machine.

`revert`(*name*)
Change the execution state of the virtual machine to the state of this snapshot.

Parameters **`name`** – (str) Name of the snapshot.

`revert_to_current`()
Change the execution state of the virtual machine to the state of current snapshot.

`upgrade_vm_tools`()

class `pyVirtualize.pyvSphere.vm.operation.vmutils.`**`VMUtils`**(*vim*, *timeout=None*, ***kwargs*)
Bases: `pyVirtualize.pyvSphere.vm.operation._base.BaseOperation`

`clone`(*template_name*, *vm_name*, *datacenter_name*, *resource_pool_cluster=None*, *host=None*, *is_template=False*, *snapshot=None*, *memory=False*, *datastore_name=None*, *vm_folder=None*, *power_on=False*, *changeSID=False*, *nw={}*, *identification={}*, *timezone=40*, *autologon=False*, *autologonCount=1*, *autologonAdminPwd="*, *fullName=None*, *orgName=None*)
Creates a clone of this virtual machine. If the virtual machine is used as a template, this method corresponds to the deploy command. Any % (percent) character used in this name parameter must be escaped, unless it is used to start an escape sequence. Clients may also escape any other characters in this name parameter. The privilege required on the source virtual machine depends on the source and destination types:

- source is virtual machine, destination is virtual machine - VirtualMachine.Provisioning.Clone
- source is virtual machine, destination is template - VirtualMachine.Provisioning.CreateTemplateFromVM

- source is template, destination is virtual machine - VirtualMachine.Provisioning.DeployTemplate
- **source is template, destination is template - VirtualMachine.Provisioning.CloneTemplate**
  If customization is requested in the CloneSpec, then the VirtualMachine.Provisioning.Customize privilege must also be held on the source virtual machine. The Resource.AssignVMToPool privilege is also required for the resource pool specified in the CloneSpec, if the destination is not a template. The Datastore.AllocateSpace privilege is required on all datastores where the clone is created.

Parameters

- **`template_name`** – (str) Source VirtualMachine/Template name.
- **`vm_name`** – (str) The name of the new virtual machine.
- **`datacenter_name`** – (str) Datacenter name on which source image resides.
- **`resource_pool_cluster`** – (str) Specify a Cluster name from which Resource pool will be picked. The resource pool to which this virtual machine should be attached. For a relocate or clone operation to a virtual machine, if the argument is not supplied, the current resource pool of virtual machine is used. For a clone operation to a template, this argument is ignored. For a clone operation from a template to a virtual machine, this argument is required.
- **`host`** – (str) The target host for the virtual machine. If not specified,
  - if resource pool is not specified, current host is used.
  - if resource pool is specified, and the target pool represents a stand-alone host, the host is used.
  - **if resource pool is specified, and the target pool represents a DRS-enabled cluster,**
    a host selected by DRS is used.
  - **if resource pool is specified and the target pool represents a cluster without DRS enabled,**
    an InvalidArgument exception be thrown.
- **`is_template`** – (bool, optional) Specifies whether or not the new virtual machine should be marked as a template. Default, 'False'.
- **`snapshot`** – (str, optional) Snapshot reference from which to base the clone. If this parameter is set, the clone is based off of the snapshot point. This means that the newly created virtual machine will have the same configuration as the virtual machine at the time the snapshot was taken.If this property is not set then the clone is based off of the virtual machine's current configuration. Setting this is only supported if the host this virtual machine is currently residing on supports cloning from a snapshot point. Such support does not need to exist on the destination host for the clone. Setting this is only supported if the virtual machine supports reporting snapshot configuration information. See snapshotConfigSupported. Such support does not need to exist on the destination host for the clone.
- **`memory`** – (bool, optional) Flag indicating whether to retain a copy of the source virtual machine's memory state in the clone. Retaining the memory state during clone results in a clone in suspended state with all network adapters removed to avoid network conflicts, except those with a VirtualEthernetCard.addressType of "manual". Users of this flag should take special care so that, when adding a network adapter back to the clone, the VM is not resumed on the same VM network as the source VM, or else MAC address conflicts could occur. When cloning between two hosts with different CPUs outside an EVC cluster, users of this flag should be

aware that vCenter does not verify CPU compatibility between the clone's memory state and the target host prior to the clone operation, so the clone may fail to resume until it is migrated to a host with a compatible CPU. This flag is ignored if the snapshot parameter is unset. This flag only applies for a snapshot taken on a running or suspended virtual machine with the 'memory' parameter set to true, because otherwise the snapshot has no memory state. This flag defaults to false.

- **`datastore_name`** – (str, optional) Name of the datastore where target image is going to reside. Default, first datastore alphabetically from the list will be selected.
- **`datastore_name`** – (str, optional) The datastore where the virtual machine should be located. If not specified, the current datastore is used.
- **`vm_folder`** – (str, optional) Name of the folder under which image should be listed post deployment. Default, will be shown under root folder.
- **`power_on`** – (bool, optional) Power on the target VirtualMachine after deployment. Default, 'False' i.e. it won't power On.
- **`changeSID`** – (bool, optional) The customization process should modify the machine's security identifier (SID).

  For Vista OS, SID will always be modified.
- **`identification`** – (dict, optional) The Identification data object type provides information needed to join a workgroup or domain. The Identification data object type maps to the Identification key in thesysprep.inf answer file. These values are transferred into thesysprep.inf file that VirtualCenter stores on the target virtual disk. ex: {'domain': DOMAIN_NAME, 'admin': ADMIN_NAME, 'pwd': ADMIN_PWD}
- **`timezone`** – (int, optional) The time zone for the new virtual machine. Numbers correspond to time zones listed in sysprep documentation at in Microsoft Technet. Default, 40 to IST time zone.
- **`autologon`** – (bool, optional) Flag to determine whether or not the machine automatically logs on as Administrator.
- **`autologonCount`** – (int, optional) If the AutoLogon flag is set, then the AutoLogonCount property specifies the number of times the machine should automatically log on as Administrator. Generally it should be 1, but if your setup requires a number of reboots, you may want to increase it.
- **`autologonAdminPwd`** – (str, optional) The new administrator password for the machine. To specify that the password should be set to blank (that is, no password), set the password value to NULL.
- **`nw`** – (dict, optional) IP settings that are specific to a particular virtual network adapter. {"ip": "x.x.x.x", "subnet": "x.x.x.x", "gateway": "x.x.x.x", "dns": [PRIMARY_DNS_IP, SECONDARY_DNS_IP]}
- **`fullName`** – (str, optional) User's full name.
- **`orgName`** – (str, optional) User's organization.

`upgrade_vm_tools`()

class `pyVirtualize.pyvSphere.vm.operation.admin.`**`AdminOperations`**(*vim*, *timeout=None*, ***kwargs*)

Bases: *`pyVirtualize.pyvSphere.vm.operation.file.FileOperations`*, *`pyVirtualize.pyvSphere.vm.operation.process.ProcessOperations`*

execute_process_using_ps_tools(*program*, *arguments=()*, *admin_credentials='admin'*, *user_credentials='user'*)

Starts a program in the guest operating system using Admin credentials. If program is not executed then it will raise an Exception ''

Parameters

- **program** – (str) The absolute path to the program to start. For Linux guest operating systems, /bin/bash is used to start the program. For Solaris guest operating systems, /bin/bash is used to start the program if it exists. Otherwise /bin/sh is used. If /bin/sh is used, then the process ID returned by StartProgramInGuest will be that of the shell used to start the program, rather than the program itself, due to the differences in how /bin/sh and /bin/bash work. This PID will still be usable for watching the process with ListProcessesInGuest to find its exit code and elapsed time.
- **arguments** – (str) The arguments to the program. In Linux and Solaris guest operating systems, the program will be executed by a guest shell. This allows stdio redirection, but may also require that characters which must be escaped to the shell also be escaped on the command line provided. For Windows guest operating systems, prefixing the command with "cmd /c" can provide stdio redirection.
- **admin_credentials** – (str) Specify the credentials type string for Administrator. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.
- **user_credentials** – (str) Specify the credentials type string for User. Which was added using 'set_credentials' function. If string is left empty, it will find the 'default' credentials type and will use them. And if it doesn't find any, then it will raise an Exception.

Python Module Index

Index