



Learning vRealize Orchestrator in action

VMUG LAB

Lab

Learning vRealize Orchestrator in action

Code examples

If you don't feel like typing the code you can download it from the webserver running on 192.168.10.201. It's called code.txt. It is assumed you have some experience with writing code and using a scripting language. If you want to brush up on JavaScript you can use <https://www.w3schools.com/js/>. Many of the web based examples don't apply but you can learn Syntax from this site.

Logging into orchestrator

- Open a web browser and navigate to <http://192.168.10.203> and select Start Orchestrator client.
- This will download the jnlp for orchestrator
- Once downloaded start the jnlp
- This will prompt you to login
 - Username : administrator@vsphere.local
 - Password: VMware1!

Workflows

Workflows are the procedural visual language used to create automation in Orchestrator. The key idea is orchestrator gives you a workspace pane that allows you to drag and drop elements into an order or decision tree providing visual programming. This method provides maximum reusability of elements. If you create your elements in a modular fashion you should be able to reuse them as you add new automation elements. The programming language for orchestrator is JavaScript. It helps to have some experience with programming languages, but it is not required. When people first start using orchestrator they are tempted to create a single JavaScript code that does everything essentially reducing the workflow to a single element, doing this reduces the reusability of your code. This document will teach you how to create your first workflow by learning to create an action.

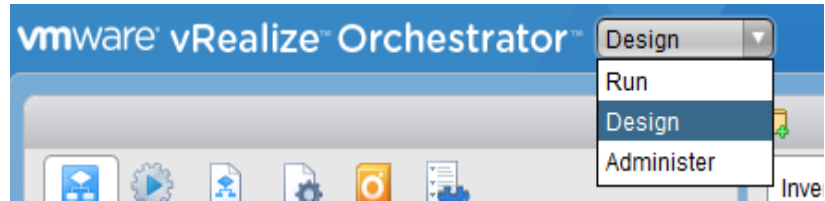
Action to list all virtual machine names

Actions are snippets of JavaScript code that take input and provide output. They can be used in a modular function inside workflows or to populate a drop down in vRealize Automation. Workflows can run other workflows as part of their workflow. It is best to develop actions as workflows then convert into actions

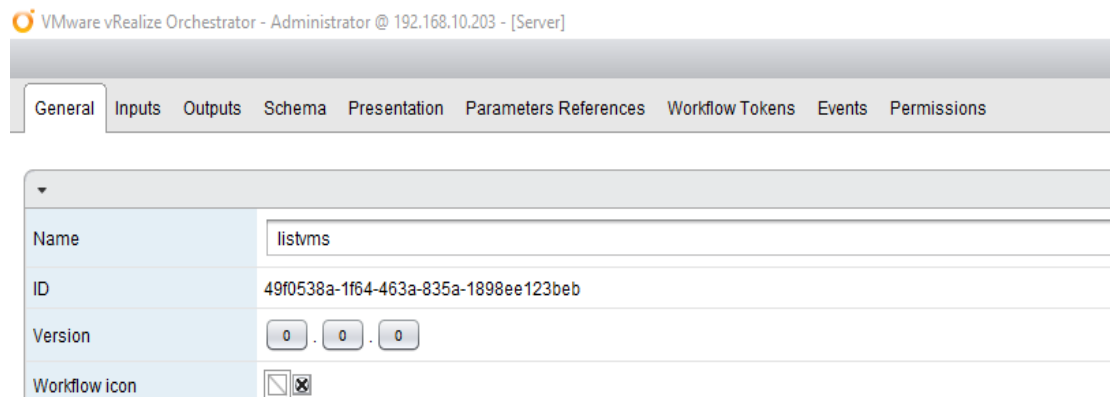


once you have tested them. We will create a basic action to list all virtual machines available to this orchestrator instance.

1. Navigate to the top of the vRealize Orchestrator pane and select the drop down that says **run switch it to design**

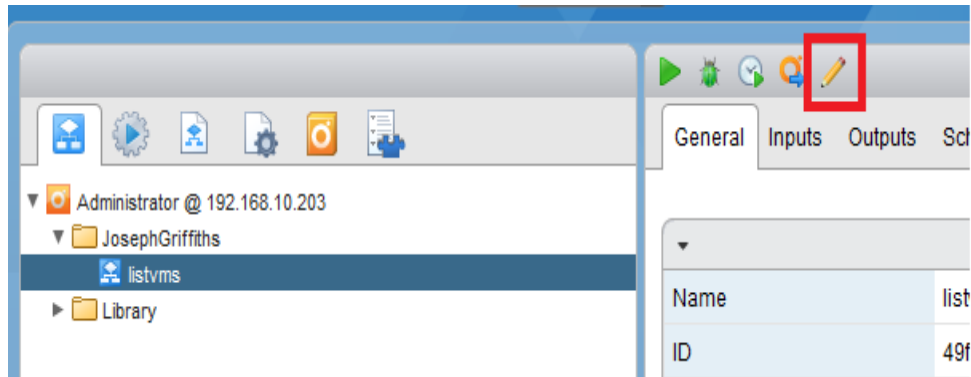


2. This will change your left pane view to include additional icons if you mouse over these icons you will see: Workflows, Actions, Resources, Configurations, Packages, Inventory
3. Select **Workflows**
4. **Right click** on Administrator @ 192.168.10.203 and **select Add folder**. Name the folder after yourself
5. Your new folder should be automatically selected right click on the folder and select **New workflow**. Name the workflow listvms.
6. This will open the **edit workflow pane**

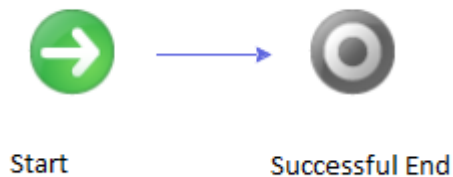


7. If for some reason it does not open the edit workflow pane then you can edit it by selecting the workflow and clicking the edit button shown below:





- 7.1. From here you can modify all aspects of the workflow. You should see the tabs as above. Workflows are identified by their ID which is unique per Orchestrator instance, this allows multiple workflows to have the same name without conflicts. Each workflow has several additional meta data elements as shown on the general screen.
- 7.2. There are three different variable models inside workflows: attributes; input and output. Each of these variables can be of any type (for example numbers, strings, objects or array of strings) these variable types persist through out the workflow here is the key differences:
 - **Attributes** – Exist when a workflow is started and can have an initial value assigned they can be read/write or read only (examples would be configuration elements or environment variables)
 - **Inputs** – expected to be manually set or passed into the workflow at start time by another process (example would be variables passed from another workflow, user input, or vRealize Automation)
 - **Output** – These are variables that are passed out of the workflow when completed to another process (example would be to another workflow or to display as output)
- 7.3. On the General tab at the bottom you can define attributes
- 7.4. On the Inputs / Output tab you can define input and output variables
- 7.5. Schema is your visual pane for creation of workflows
- 7.6. Presentation shows how to prompt for user input -> this is not used in vRealize Automation it is expected that you prompt for user inputs using XaaS models then pass that as inputs into the workflow
- 7.7. The other tabs will not be used in this example
8. **Click on Schema** to go to the visual editor you should be presented with a blank plane for editing



9. Navigate to the **Generic** pane on the left and **click and hold on Scriptable task** and **drag in-between start and successful end**. (a scriptable task is a section of JavaScript code that is used to do something) When you first start with vRealize orchestrator it is tempting to have every workflow be a single scriptable task this removes the power of orchestrator. Its modular nature is one of its greatest strengths. After we create our action we will create a more complex workflow.



10. Now click on the scriptable task and you will notice that the bottom pane of your screen changes with the following elements available local to the task:
- **Info** – Generic information tab notice the name here which allows you to name this task – lets change it to getallvms
 - **In** – This tab allows us to create local input variables and tie the local variables to variables and attributes from the workflow
 - **Out** – This tab allows us to define variables that will be passed out of this task to the workflow, for a variable to be passed out of the workflow it must be tied to a workflow attribute or variable (everything you define in the In or Out tab is local to the task)
 - **Exception** – Allows us to define how to export errors into variables for use outside the task
 - **Visual Binding** – Allow us to see the input parameters and how they are tied to local variables and outputs
 - **Scripting** – This is the JavaScript code to execute
11. Before you type your scripting code a few basic JavaScript / Orchestrator rules:
- **Case sensitive** – Orchestrator JavaScript is case sensitive this is critical and the cause of much pain be very careful with this!
 - **Ending a line** – Lines should always end in semi-colon; (to execute line)
 - **Comments** – Highly recommended
 - **//** Denotes the rest of the line is a comment

- `/*` Denotes a multi-line comment ended with `*/`
- **System.Log("Text")** – This is the command to log information to screen for debugging it's your best friend
- **Indentation** – Highly recommended for readability ignored by JavaScript
- **Variables** – Defined using term `var` – don't specifically have a type – but can have a type defined if the term is also a local in or out
- **Arrays** – groups of variables or objects defined by using the term `new Array()`; and use the word `push` to put on the array;
- **Conditionals** - Your best friend in programming (conditionals can be created on the visual editor as well)
 - If statement with `else` and `ifelse`

```

if (somevariable < 18) {
    dosomething;
} else if (somevariable < 20) {
    dosomething;
} else {
    dosomething;
}

```

11.1. Loops – Your second-best friend in programming
For Loop

```

For (i=0; i<5; i++) {
    dosomething_five_times;
}

```

While loop

```

While (i < 10) {
    i++;
    dosomething;
}

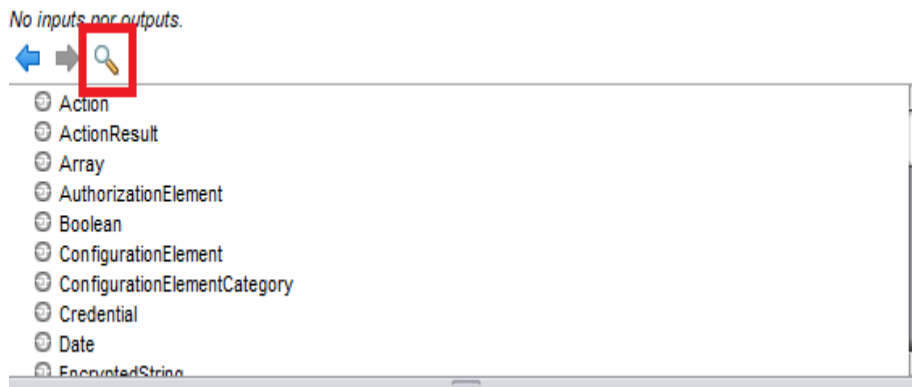
```

- 12.** Let's build our first workflow **click on getallvms** we are going to undock this editor window to allow us to work full screen to do this click on the undock button on the task window as shown below:

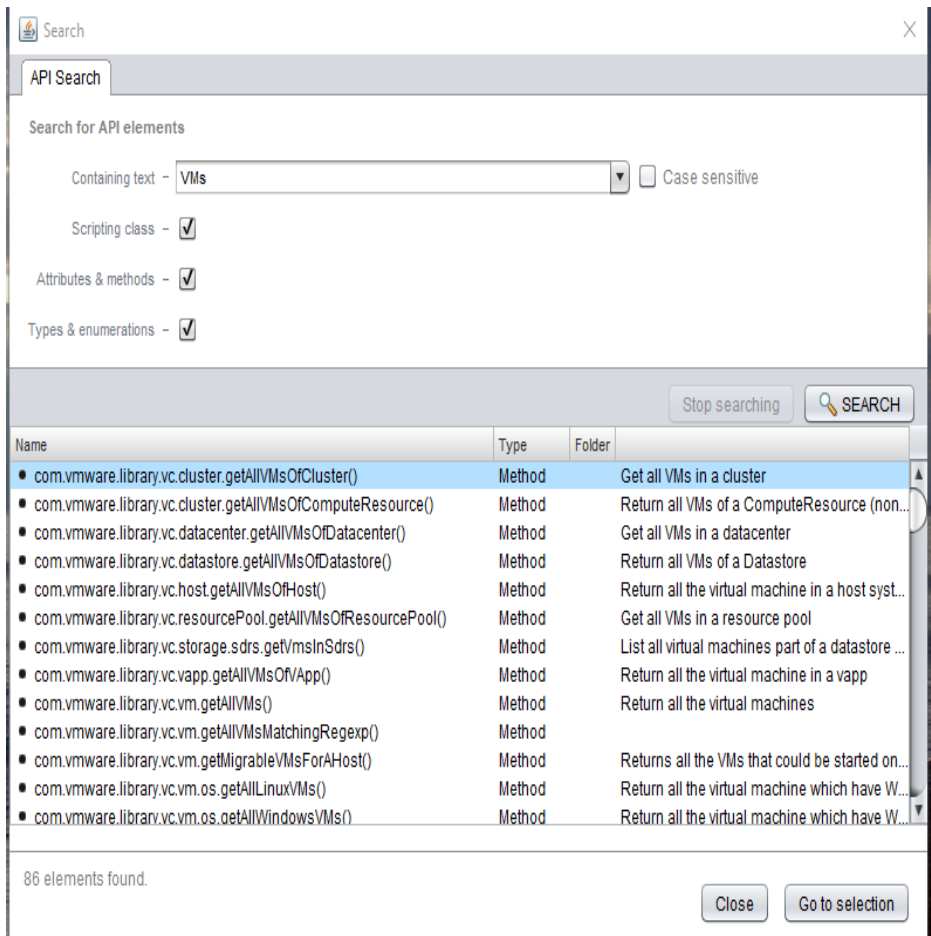




13. This will create a new window of just getallvms **maximize this window**. Click the Scripting tab (once you are done editing you can just close the window)
14. Our Orchestrator instance is already connected to vCenter allowing us to use the API's, so we are going to browse the plugin for a API call to return all virtual machines from vCenter
 - 14.1. On the left side there is a **magnifying glass** that allows you to search the available API's click it (this is the primary method you use to locate data from vCenter)



- 14.2. This will open the API search box type VMs and **click search**



- 14.3.** As you can see many methods exist to get information you can expand the last section to provide additional details on what the call does – for example the top call gets all virtual machines in a cluster
- 14.4.** We are going to use the **com.vmware.library.vc.vm.getAllVMs()** method to return all virtual machines in virtual center (vc) and use the virtual machine (vm) – click go to select then exit the screen
- 14.5.** As you progress with using the API you will begin to understand how the developers organize virtual center and its related elements, in addition the API often exposes elements not available via the GUI allowing you to learn more about the inner workings of the product
- 14.6.** Now that you are back at the scripting window you will notice that `getAllVMs` is selected on the left (because you click go to selection) this provides critical information about the method for example what it returns (Array of objects defined as `VC:VirtualMachines`)

getAllVirtualMachines() : VC:VirtualMachine[]

Method : getAllVirtualMachines

Description

Return all the virtual machines

Signature

VC:VirtualMachine[] getAllVirtualMachines()

Parameters

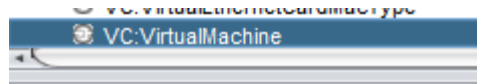
none

Return Type : Array of VC:VirtualMachine

Return all the virtual machines

14.7. What is a VC:VirtualMachine ? well let's click it to find out:





Type : VC:VirtualMachine

Scripting Object : VcVirtualMachine

Description

none

Properties

vimType
overallCpuUsage
memory
vimId
instanceId
hostMemoryUsage
guestMemoryUsage
vmToolsVersionStatus
biosId
productVendor
isTemplate
vmToolsStatus
name
displayName
unsharedStorage
id
cpu
connectionState
committedStorage
hostName
type
sdkId
guestHeartbeatStatus
overallStatus
totalStorage
guestOS
productFullVersion
configStatus
ipAddress
annotation
productName
state
mem
memoryOverhead
alarmActionsEnabled
vmVersion

- 14.8.** This output provides all the fields that make up the object VC:VirtualMachine for example hostName (object name of host which is currently running the virtual machine) or name (name of virtual machine).



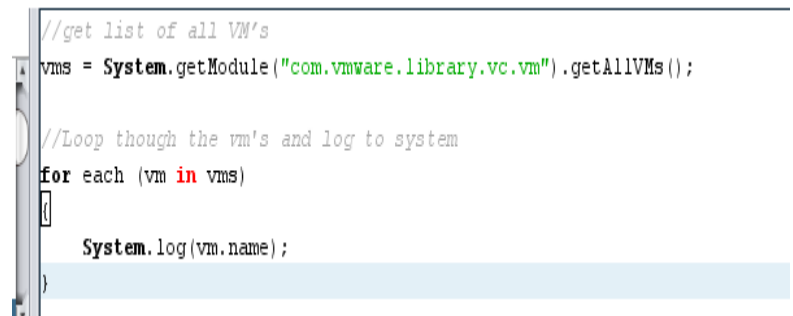
- 14.9.** We need to use the API method to get all virtual machines and assign to a variable type the following code:

```
//get list of all VM's  
vms = System.getModule("com.vmware.library.vc.vm").getAllVMs();
```

- 14.10.** This piece of code has a comment (//) and the command to getAllVM's and assign to the variable vms (which is now an array of Object VC:VirtualMachine)
- 14.11.** Now let's create a loop to log out all the names of virtual machines:

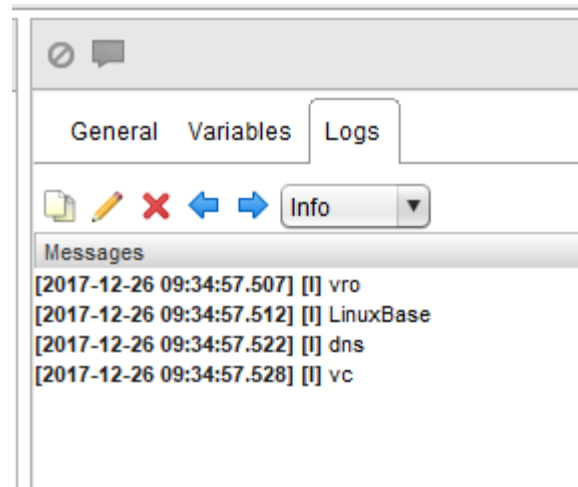
```
//Loop though the vm's and log to system  
for each (vm in vms)  
{  
    System.log(vm.name);  
}
```

- 14.12.** Now exit the full screen editor and return to your workflow editor window
- 14.13.** You should now see the code you wrote in the bottom pane let's try running it by clicking the run button



```
//get list of all VM's  
vms = System.getModule("com.vmware.library.vc.vm").getAllVMs();  
  
//Loop though the vm's and log to system  
for each (vm in vms)  
{  
    System.log(vm.name);  
}
```

- 14.14.** Notice as it's running it highlights in green which step is being run (green means running or successful while red means failed)
- 14.15.** Once completed on the right pane click on logs to see the System.log information



- 14.16.** You can see the names of the four virtual machines in the lab
- 14.17.** Now it's time to save our work in the bottom right it offers a save and close click that button. It will ask if you want to increase the version go ahead and increase (version can be found on the general tab under version – it can be manually changed)
- 15.** We now have a workflow that produces the list of virtual machine names and logs it to the console which is essentially useless. Next, we are going to log additional information and load the virtual machine names into an array for output.
 - 15.1.** Edit the listvms workflow again and edit the getallvms
 - 15.2.** Navigate to the Schema tab click on getallvms
 - 15.3.** Maximize or edit in lower window by selecting Scripting
 - 15.4.** Create an array to hold our virtual machine names using this code and place it above the for loop:

```
//create an array to hold vm names  
var vmname = new Array();
```

- 15.5.** As shown below:

```

//get list of all VM's
vms = System.getModule("com.vmware.library.vc.vm").getAllVMs();

//create an array to hold vm names
var vmname = new Array();

//Loop though the vm's and log to system
for each (vm in vms)
{
    System.log(vm.name);
}

```

- 15.6. Then let's load the virtual machine names into the array using this command inside the loop:

```

//Add vm names to array
vmname.push(vm.name);

```

- 15.7. Place it as shown below:

```

//get list of all VM's
vms = System.getModule("com.vmware.library.vc.vm").getAllVMs();

//create an array to hold vm names
var vmname = new Array();

//Loop though the vm's and log to system
for each (vm in vms)
{
    System.log(vm.name);
    vmname.push(vm);
}

```

- 15.8. It's a good idea to check the elements of your array to do this we will loop through the array once to confirm it works **using a for loop at the end of the code:**

```

//output vmname contents
for (i=0;i<vmname.length;i++)
{
    System.log("Array vmname at position " + i + " has the following element " + vmname[i].name);
}

```



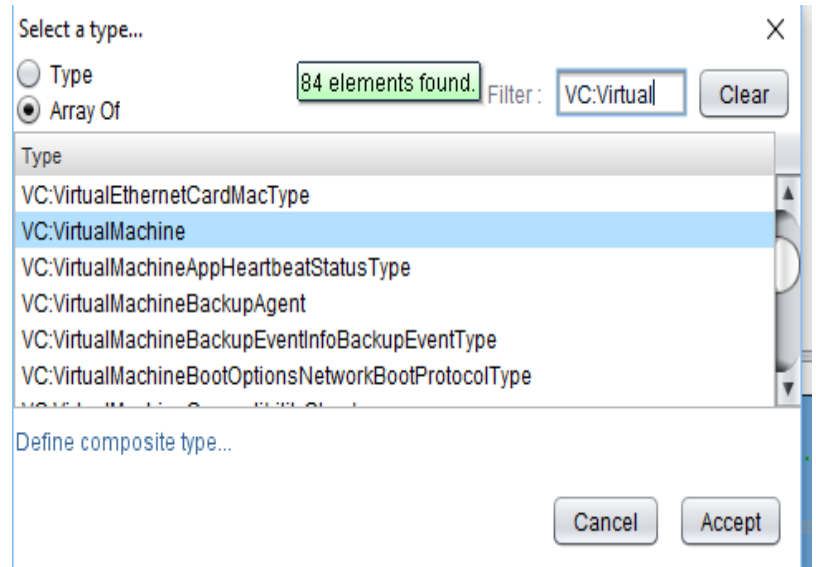
- 15.9. **Run the workflow** to ensure everything works log tab should output the following:

```
Messages
[2017-12-27 08:22:57.143] [I] vro
[2017-12-27 08:22:57.145] [I] LinuxBase
[2017-12-27 08:22:57.146] [I] dns
[2017-12-27 08:22:57.148] [I] vc
[2017-12-27 08:22:57.150] [I] Array vmname at position 0 has the following element vro
[2017-12-27 08:22:57.152] [I] Array vmname at position 1 has the following element LinuxBase
[2017-12-27 08:22:57.153] [I] Array vmname at position 2 has the following element dns
[2017-12-27 08:22:57.155] [I] Array vmname at position 3 has the following element vc
```

- 15.10. **Save and close your action**
16. We are now going to take our current scriptable task and turn it into an action
- 16.1. Navigate to the action tab across the top as shown below



- 16.2. **Right click on the username @ 192.168.10.203** section at the top and **select new module.**
- 16.3. Name the module **org.yourname** (so in my case it's org.josephgriffiths)
- 16.4. Use the slider on the left pane to locate your new module and right click on it
- 16.5. **Select add action**
- 16.6. **Name the action yourfirstname_vmname** (in my case it's joseph_vmname)
- 16.7. This will open the action editor window which has a lot less items than our workflow window
- General tab is very similar to the workflow tab notice that attributes are missing since they are not used in actions
 - Scripting tab is our scripting window
 - Events & Permissions will not be used in this module
- 16.8. **Click on scripting**
- 16.9. An action does some type of work then produces an output notice at the top the **current return type is set to void** we need to change this to be array of string for our vmname script
- 16.10. **Click on void word to open object type window.** We need to change to array of the object type VC:VirtualMachine



16.11. Now we can cut and paste the scriptable task into our action as shown (remember you can download all code from 192.168.10.201:

```
//get list of all VM's
vms = System.getModule("com.vmware.library.vc.vm").getAllVMs();

//create an array to hold vm names
var vmname = new Array();

//Loop though the vm's and log to system
for each (vm in vms)
{
    System.Log(vm.name);
    vmname.push(vm);
}

for (i=0;i<vmname.length;i++)
{
    System.Log("Array vmname at position " + i + " has the following element " + vmname[i].name);
}

```

16.12. Now we need to tell the action what to return as an output using the return key word **add at the very bottom:**

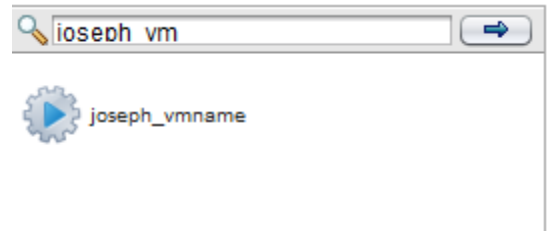
```
return vmname;
```

16.13. Now you can save and close the action

17. Let's use the action in our workflow **return to the workflow window and locate listvms and edit**

17.1. On the left side where you see the generic tab do a search for your action `firstname_vmname`





17.2. Drag and drop this action right after our scriptable task for getallvms



17.3. Let's **remove the getallvms scriptable task** because it's now redundant by selecting it and pressing the red x button



17.4. Try **running your new workflow** and see if the output is the same

17.5. **Save and close** your workflow (notice **it complains about the fact that vmname is not assigned to anything at this time you can ignore this error**)

18. You have now successfully developed a piece of scriptable task into an action. Actions are particularly important because they can be used to feed drop down lists in vRealize Automation. So, if you wanted to create a drop down that contained all virtual machines this action could feed that drop down if we filtered the outcome to name only. Within the scriptable task you could remove or exclude specific virtual machines thus creating a customized drop-down list.

Workflow to unmount CD-ROM drives

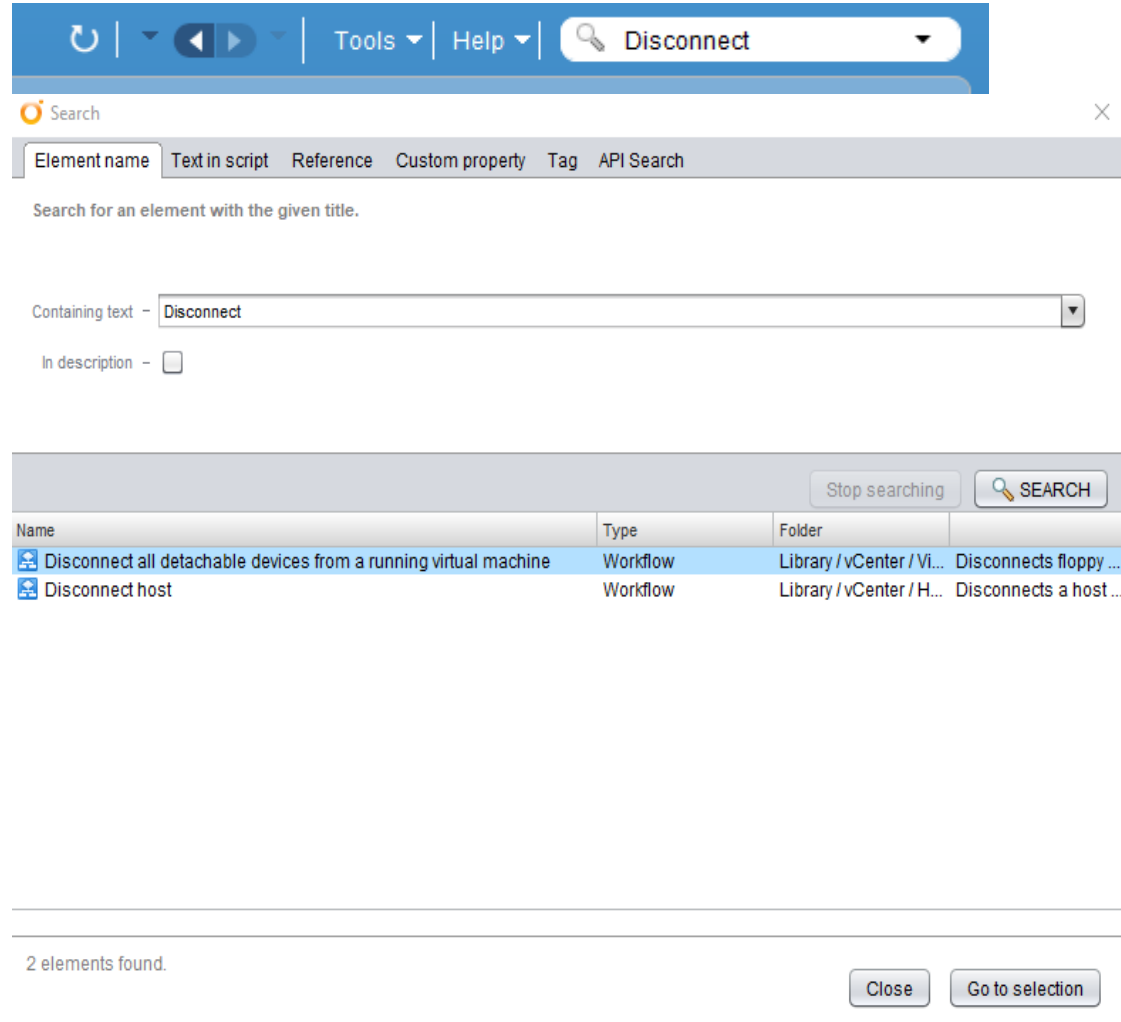
In this workflow we are going to un-mount all currently mounted CD-ROM drives. This could be run as a scheduled task to clean up any mounted CD-ROM's. This example



allows us to use multiple prebuild workflows and actions to create our completed products.

19. Let's start by **navigating to workflows**

20. VMware has created a workflow called "Disconnect all detachable devices from a running virtual machine" **use the search box to locate this workflow**



The screenshot shows the VMware vSphere Search interface. At the top, there is a navigation bar with a search box containing the text "Disconnect". Below the navigation bar, the search results are displayed in a table. The table has columns for Name, Type, Folder, and Description. Two results are shown: "Disconnect all detachable devices from a running virtual machine" and "Disconnect host".

Name	Type	Folder	Description
Disconnect all detachable devices from a running virtual machine	Workflow	Library / vCenter / Vi...	Disconnects floppy ...
Disconnect host	Workflow	Library / vCenter / H...	Disconnects a host ...

2 elements found.

21. **Click the go to selection box** then close the search box

22. You will notice that you are **unable to edit this workflow because it's a built-in workflow**. VMware has provided lots of built in workflows but does not allow you to edit them because they want to have the ability to update them in future releases without breaking your custom changes. **If you want to modify a built-in workflow just right, click on it and select duplicate workflow**. This allows you to create your own workflow to edit and modify which is separate from VMware's workflow.



23. Take a minute to examine their built-in workflow. It's a little more complex than previous workflows showing a decision tree. In addition, the scriptable task has more complexity. **Read the scriptable task to see if you can understand the flow.**
- 23.1. Notice how devices is an array of each virtual machine and the scriptable task uses a for loop to work through the devices. This task has a lot of error checking to ensure smooth running. Notice the use of System.log to provide output and help you diagnose errors.
24. Visit the input page to identify what type of input it requires (VC:VirtualMachine)
- 25. Try running the workflow from inside orchestrator against the dns virtual machine**
26. We are now going to **navigate back to our folder** and create a new workflow called unmount_cdroms
- 26.1. Once inside the workflow **click on Schema** to build the workflow
- 26.2. Use the **search pane to locate our action firstname_vmname** (joseph_vmname in my case) and drag on to the pane as shown:



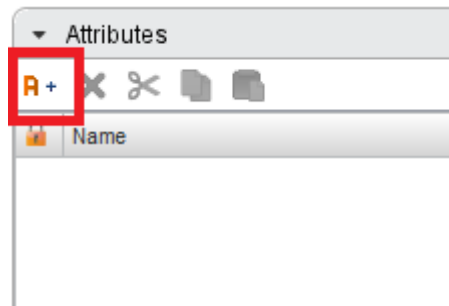
- 26.3. You can **try running** to ensure it works and check logs for correct output

```

Messages
[2017-12-27 10:12:55.632] [I] vro
[2017-12-27 10:12:55.633] [I] LinuxBase
[2017-12-27 10:12:55.635] [I] dns
[2017-12-27 10:12:55.636] [I] vc
[2017-12-27 10:12:55.637] [I] Array vmname at position 0 has the following element vro
[2017-12-27 10:12:55.638] [I] Array vmname at position 1 has the following element LinuxBase
[2017-12-27 10:12:55.648] [I] Array vmname at position 2 has the following element dns
[2017-12-27 10:12:55.649] [I] Array vmname at position 3 has the following element vc

```

- 26.4. We need to **create an attribute to hold our array** as it's returned from the action
- 26.4.1. Click on General**
- 26.4.2. Locate the **R+ button** and click on it



26.4.3. **Click on att0** to change the name of the attribute and name it vmname

26.4.4. **Change the type to array of VC:VirtualMachine**

26.4.5. Put the following in description for usability **“Attribute to hold object of virtual machines”**

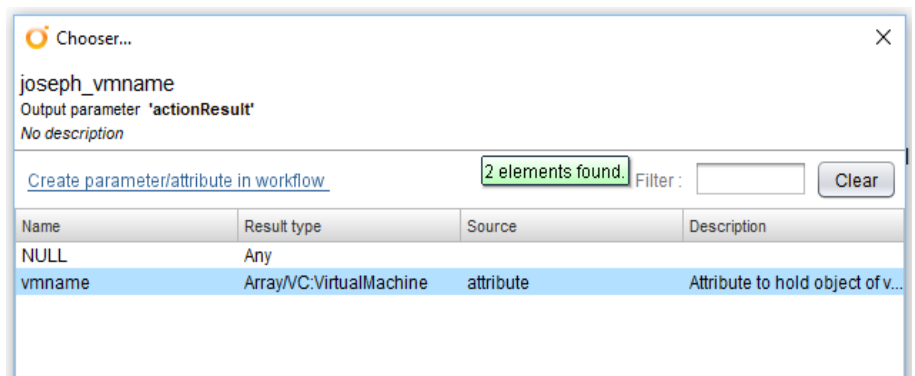
26.5. **Click on Schema again**

26.6. **Left click on your action**

26.6.1. This should open the **action pane below**

26.6.2. **Select OUT tab**

26.6.3. Click on **source parameter “not set”** and select your attribute called **vmname**

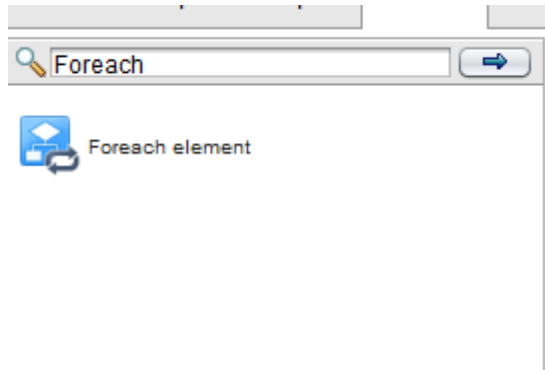


26.6.4. **Select the visual binding tab** and make sure your attribute and output are linked as shown

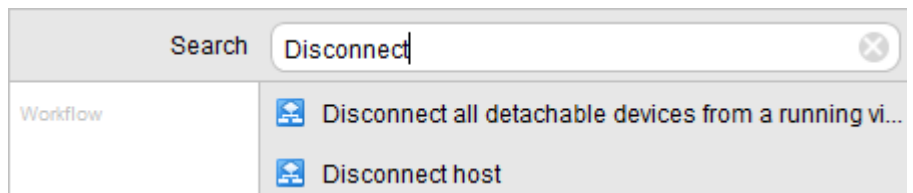


26.7. Now we are going to **add a foreach element** by searching in the pane **“foreach”**



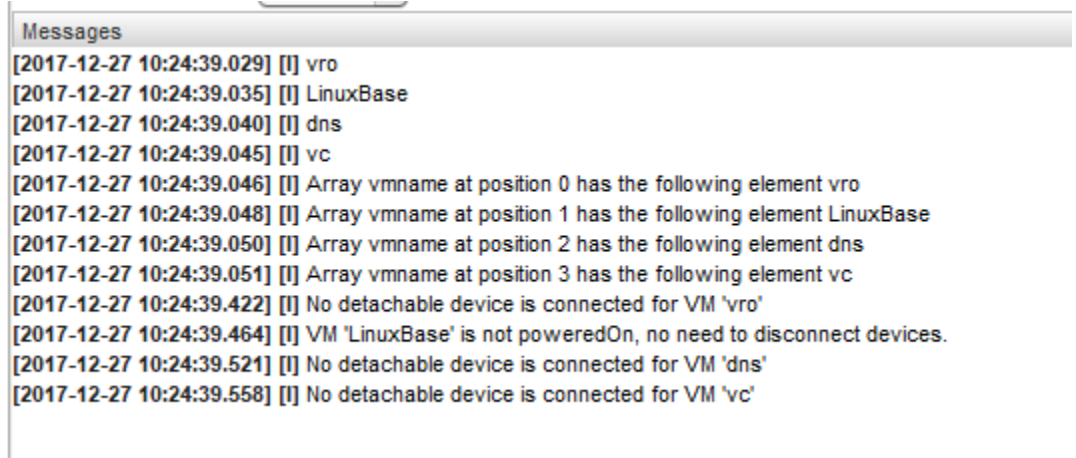


26.8. Drag the foreach element to be after the action when you do this an additional window will pop up asking for a workflow to run on each element do a search for disconnect:



26.9. It will automatically pull in the only available attribute that can be used to feed the loop vmname as shown in the **IN box of the foreach loop**. This can be changed by clicking on the names

27. **Try running the workflow** and see if you get the correct output or something similar:



28. **Save and close the workflow** increasing the version



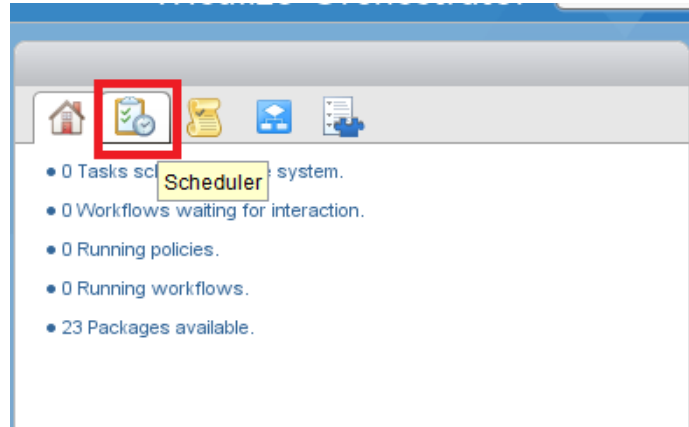
29. You have now completed the job of creating a workflow to disconnect all virtual machines CD-ROM devices. **Now let's schedule it to run every day.**

30. To schedule run of a workflow:

31. At the top of the screen switch from **Design to run**

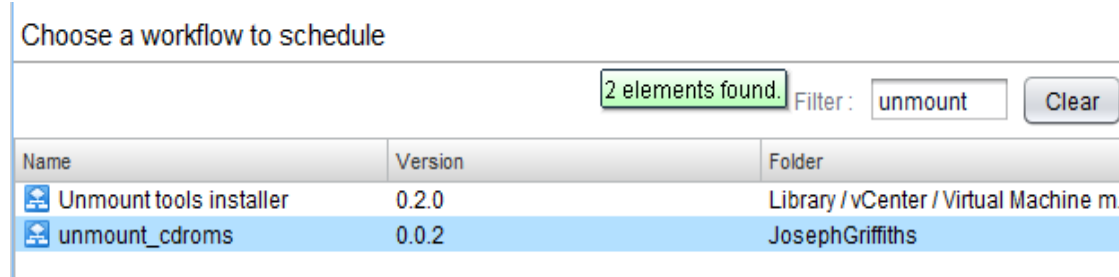
32. This should change the icons at the top

33. Select Scheduler



34. Click the green icon at the top to schedule a task

35. In the search for workflow locate **your unmount workflow** by searching (there may be many different versions of it make sure you select the one in your folder)



36. Name the task **Daily unmount CD-ROMs** (mine would be JosephGriffiths daily unmount cdroms)

37. **Select every day for recurrence** and leave time at 12 am then submit

38. From this same panel you could return and see if it successfully ran each time like this:

Name	State	Start date	End date	Current item name	User Owner
unmount_cdroms	[completed]	12/29/17 18:00	12/29/17 18:00		Administrator@VSPHERE.LOCAL
unmount_cdroms	[completed]	12/28/17 18:00	12/28/17 18:00		Administrator@VSPHERE.LOCAL
unmount_cdroms	[completed]	12/27/17 18:00	12/27/17 18:00		Administrator@VSPHERE.LOCAL

Using the vSphere REST API

In most automation projects you will have to work with another endpoint other than vCenter. The most common API endpoint are REST (Representational State Transfer)



endpoints. You can explore the vCenter REST endpoint by visiting vCenter at: <https://192.168.10.202/> then selecting **Browse vSphere REST APIs** on the bottom right hand side. REST uses http methods (POST, GET, UPDATE, etc..) combined with URL's and XML. The method combined with the URL denotes what to do while the XML provides configuration details. The vCenter REST endpoint does not yet have all actions available more will become available over time. Each REST query is a single web call to the API which then returns results or data. Normally GET represents gathering information while POST or PUT are used to execute a change or action. We will be using Orchestrator to get a list of all virtual machines in JSON.

39. Browse the REST API's via vCenter by visiting 192.168.10.202 and selecting Browse vSphere REST API's

40. Locate and Show the VM category

40.1. This should display the methods available (GET, POST, DELETE) with their respective URL's

VM		Show/Hide	List Operations	Expand Operations
GET	/vcenter/vm	Returns information about at most 1000 visible (subject to permission checks) virtual machines in vCenter matching the VM.FilterSpec.		
GET	/vcenter/vm/{vm}	Returns information about a virtual machine.		
POST	/vcenter/vm	Creates a virtual machine.		
DELETE	/vcenter/vm/{vm}	Deletes a virtual machine.		

40.2. The base URL for all REST calls is: <https://vcenter/rest/> followed by the specific information URL for example if I wanted to get information on virtual machines I would issue a GET to <https://192.168.10.202/rest/vcenter/vm>

40.3. Normally a REST call includes a header that contains authentication information

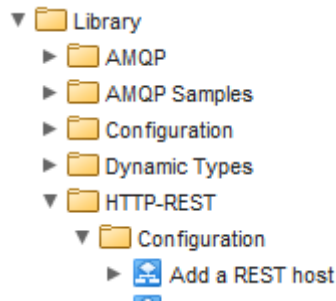
40.4. You might want to use a REST client to explore API's a common client is Postman

41. Let's return to Orchestrator and create a workflow to list virtual machines using REST

41.1. Once in Orchestrator switch to Design mode

41.2. Select Workflows and expand the Library folder (pre-build)

41.3. Expand HTTP-REST, Expand Configuration



41.4. Inside the configuration you will be able to add a new REST endpoint by running the Add a REST host workflow

41.4.1. Right click on the Add a REST host button and select start workflow...

41.4.2. On the Host Properties dialog

- Name : Your full name vc.lab.local (for example Joseph Griffiths vc.lab.local)
- URL: <https://vc.lab.local/rest>
- Accept other defaults and click next

41.4.3. On host authentication type select Basic and click next

41.4.4. On the user credentials page

- Session mode : Shared Session
- Authentication user name: administrator@vsphere.local
- Authentication password: VMware1!
- Click Next

41.4.5. Click Submit and watch it complete

41.5. You can see the REST host information by browsing to the Inventory tab on the left pane

41.5.1. Expand out HTTP-REST to see current REST connections

42. Now you have a defined endpoint you could add specific REST operations with the add REST operations workflow or turn REST operations into workflows with the Generate new workflow from a REST operation. VCenter requires that you create a session ID before using any REST operations. For ease we will create our own workflow that uses the REST host and returns the JSON information.

43. Return to your folder (YourFullName)

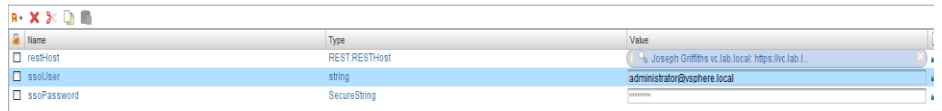
43.1. Right click to create a new workflow which will be named REST_Get_VM_Names

43.2. When your new workflow is created click on general, so we can add some attributes, add the following attributes:

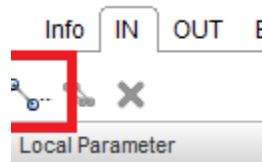
Name	Type	Initial Value
restHost	REST:REStHost	Your rest host (Full name)



		vc.lab.local) click to enter values
ssoUser	String	administrator@vsphere.local
ssoPassword	SecureString	VMware1!



- 43.3. Navigate to Schema
- 43.4. Drag a scriptable task and place between the start and finish
- 43.5. Edit the scriptable task
 - 43.5.1. Select the IN tab and locate the bind to workflow parameter/attribute and click on it



- 43.5.2. This will open the Chooser... menu here you need to select each general attribute you created and click on select

	Name	Result type	Source	De
<input checked="" type="checkbox"/>	restHost	REST:RESTHost	attribute	
<input checked="" type="checkbox"/>	ssoUser	string	attribute	
<input checked="" type="checkbox"/>	ssoPassword	SecureString	attribute	

- 43.5.3. Once they are all displayed in the IN window you are ready to type your scriptable task. Remember all code samples are downloadable from <http://192.168.10.201>

Local Parameter	Source parameter	Type
restHost	restHost [attribute]	REST:RESTHost
ssoPassword	ssoPassword [attribute]	SecureString
ssoUser	ssoUser [attribute]	string

- 43.6. DO a POST to create a session will include the following steps
 - 43.6.1. Create a correctly formed request to POST
 - 43.6.2. Execute the request
 - 43.6.3. Parse the return for a sessionId



43.7. Insert the following code and run it to test

```
//Create a correctly formed request to POST (method, FQDN+REST_URL, requesttype)
var request = restHost.createRequest("POST", restHost.url + "/com/vmware/cis/session", "");
//Execute formed request and return to response
var response = request.executeWithCredentials(ssoUser, ssoPassword);
//Parse return looking for session id
var sessionid = JSON.parse(response.contentAsString).value;

System.log("response code: " + response.statusCode);
System.log("Session id: " + sessionid);
```

43.8. If you correctly wrote the code it should return something like this:

```
Messages
[2018-01-04 14:27:47.377] [I] response code: 200
[2018-01-04 14:27:47.378] [I] Session id: cf3b22c2cfab945a8ff15a98f04597b5
```

43.9. Do a GET to return JSON information on virtual machines

43.9.1. Create a correctly formed request to GET

43.9.2. Execute the request

43.9.3. Log the JSON out

43.10. Insert the following code and run it to test

```
//Create a correctly formed request to GET (method, FQDN+REST_URL, requesttype)
var request = restHost.createRequest("GET", restHost.url + "/vcenter/vm/", "");
//Execute formed request and return to response
var response = request.executeWithCredentials(ssoUser, ssoPassword);

System.log("response code: " + response.statusCode);
System.log("response body: " + response.contentAsString);
```

43.11. If your script is correct it should return something like this:

```
Messages
[2018-01-04 14:30:52.219] [I] response code: 200
[2018-01-04 14:30:52.220] [I] Session id: c1b21da7034578d18a352dfdcd6ff7b
[2018-01-04 14:30:53.354] [I] response code: 200
[2018-01-04 14:30:53.355] [I] response body:
{"value":[{"memory_size_MiB":10240,"vm":"vm-14","name":"vc","power_state":"POWERED_ON","cpu_count":2}
```

43.12. As you can see the response body contains information on my virtual machines in raw text. Notice the internal VM reference is vm-14 not by name. This allows the virtual machine to be renamed without causing any challenges it is known to vCenter as vm-14 and translates into vm name for usability.

43.13. RAW JSON is not the most useful thing in the world so let's parse it for the vm object name only.

43.13.1. This requires a few steps first we need to convert the output text into a JSON object using the following code:



```
//Convert into a JSON object
var jsonObj = JSON.parse(response.contentAsString);
```

43.13.2. Create some variables and loop through all the JSON elements in our object

```
//Create variables for our loop
var key, count = 0;

//Loop through each JSON object in value one at a time
for (key in jsonObj.value)
{
    //Log out the vm element (could do name or anything here)
    System.log("Info : " + jsonObj.value[count].vm);
    //increase the count
    count++;
}
```

43.13.3. Run your code and it should return something like this:

```
Messages
[2018-01-04 15:41:20.312] [I] response code: 200
[2018-01-04 15:41:20.313] [I] Session id: 4a657dfae73abe3b53e74bb4763ee956
[2018-01-04 15:41:21.435] [I] response code: 200
[2018-01-04 15:41:21.436] [I] response body:
{"value":[{"memory_size_MiB":10240,"vm":"vm-14","name":"vc","power_state":"POWER
[2018-01-04 15:41:21.437] [I] Info : vm-14
[2018-01-04 15:41:21.438] [I] Info : vm-15
[2018-01-04 15:41:21.439] [I] Info : vm-19
[2018-01-04 15:41:21.440] [I] Info : vm-20
```

44. You now understand the basics of working with REST API calls.

Last Step

We have provided working sample workflows and actions in the <http://192.168.10.201/VRO> folder please feel free to download for use later. Please download the code samples for later use. In addition you can download the lab manual and code samples for your own use later from the webserver.

Stretch goals

As a challenge with potential prizes take your newly learned skills and try to do the following:

1. Using the REST API code attempt to gather more detailed information from each virtual machine by using the URL <https://vc.lab.local/rest/vcenter/vm/vm-id/>.



Specifically Gather the name of each virtual machine and if hot add is enabled or not.

2. Modify your unmount_cdrom workflow to exclude virtual machines with the tag Exclude_unmount. There is a provided action joseph_return_vm_with_tag to provide you virtual machines with the tag (it requires a tag name passed into it and returns an array of strings containing virtual machine names) Consider using an attribute to hold the returned information from the action.
3. Create something unique and helpful to your environment and share with the group

