VMWARE VREALIZE ORCHESTRATOR
Using the Plugin for FlashArray
February 2017
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXECUTIVE SUMMARY</td>
<td>4</td>
</tr>
<tr>
<td>GOALS AND OBJECTIVES</td>
<td>4</td>
</tr>
<tr>
<td>AUDIENCE</td>
<td>4</td>
</tr>
<tr>
<td>REQUIREMENTS</td>
<td>5</td>
</tr>
<tr>
<td>INSTALLATION</td>
<td>5</td>
</tr>
<tr>
<td>INSTALLING THE FLASHARRAY PLUGIN WITH VRO 6.X</td>
<td>6</td>
</tr>
<tr>
<td>INSTALLING THE FLASHARRAY PLUGIN WITH VRO 7.X</td>
<td>8</td>
</tr>
<tr>
<td>UNINSTALLATION</td>
<td>10</td>
</tr>
<tr>
<td>CONFIGURATION</td>
<td>11</td>
</tr>
<tr>
<td>ADDING A FLASHARRAY CERTIFICATE TO VREALIZE ORCHESTRATOR</td>
<td>11</td>
</tr>
<tr>
<td>ADDING A FLASHARRAY TO VREALIZE ORCHESTRATOR</td>
<td>12</td>
</tr>
<tr>
<td>ADDING A VCENTER TO VREALIZE ORCHESTRATOR</td>
<td>14</td>
</tr>
<tr>
<td>CUSTOM OBJECTS AND FLASHARRAY INVENTORY</td>
<td>15</td>
</tr>
<tr>
<td>CUSTOM OBJECTS</td>
<td>16</td>
</tr>
<tr>
<td>SCRIPTING OBJECTS</td>
<td>20</td>
</tr>
<tr>
<td>RUNNING WORKFLOWS FROM OBJECTS</td>
<td>20</td>
</tr>
<tr>
<td>BUILT-IN ACTIONS</td>
<td>22</td>
</tr>
<tr>
<td>ACTION DESCRIPTIONS</td>
<td>22</td>
</tr>
<tr>
<td>CONNECTION ACTIONS</td>
<td>23</td>
</tr>
<tr>
<td>HOST ACTIONS</td>
<td>24</td>
</tr>
<tr>
<td>HOST GROUP ACTIONS</td>
<td>24</td>
</tr>
<tr>
<td>FLASHARRAY MANAGEMENT ACTIONS</td>
<td>25</td>
</tr>
<tr>
<td>HOST GROUP ACTIONS</td>
<td>25</td>
</tr>
<tr>
<td>RUN REST API ACTIONS</td>
<td>26</td>
</tr>
<tr>
<td>SNAPSHOT ACTIONS</td>
<td>26</td>
</tr>
<tr>
<td>VOLUME ACTIONS</td>
<td>27</td>
</tr>
<tr>
<td>VMWARE ACTIONS</td>
<td>28</td>
</tr>
<tr>
<td>BUILT-IN WORKFLOWS</td>
<td>29</td>
</tr>
<tr>
<td>WORKFLOW DESCRIPTIONS</td>
<td>29</td>
</tr>
<tr>
<td>FLASHARRAY CONNECTION MANAGEMENT</td>
<td>29</td>
</tr>
</tbody>
</table>
EXECUTIVE SUMMARY

Pure Storage® provides a RESTful API service on the FlashArray controllers to provide simple and standard API control of the FlashArray. In order to even further simplify the use of the RESTful API service for customers who may not be familiar with leveraging REST, Pure Storage offers a variety of plugins, modules, and SDKs in an assortment of platforms with which the user might be more familiar (e.g. PowerShell, Python).

VMware® offers a GUI-based orchestration engine called vRealize Orchestrator™ (vRO) – tool that provides a mechanism to build workflows that can connect to and control your entire infrastructure. Through default plugins, like REST, SOAP, SSH, PowerShell, and many others, vRO delivers a variety of methods to connect to the different cogs in your environment.

Pure Storage offers a FlashArray plugin for vRO that tightly integrates FlashArray provisioning and management with minimal effort. Features like inventory synchronization, built-in workflows, and actions provide most functionality right out of the box. The official VMware support listing can be found here:


GOALS AND OBJECTIVES

The purpose of this document is to overview the features and functions of the Pure Storage FlashArray Plugin for vRealize Orchestrator™. In detail:

- Installation and Configuration
- Inventory
- Actions
- Workflows
- Custom Orchestration

AUDIENCE

This document is intended for storage, VMware®, automation, and other types of administrators who wish to use the vRealize Orchestrator platform to manage the Pure Storage FlashArray. Familiarity with the VMware vSphere® virtualization platform, vRealize Orchestrator and the FlashArray is recommended but not required before reading this document.
REQUIREMENTS

The Pure Storage FlashArray plugin for vRealize Orchestrator has the following requirements:

FlashArray Models
- FlashArray//M
- FlashArray 400 series

FlashArray Purity Operating Environment Versions
- 4.x

vRealize Orchestrator Versions
- 6.0.x Single-Node
- 7.x Single-Node
- 7.x Multi-Node

VMware® vCenter™
- 5.5.x
- 6.0.x
- 6.5.x

The VMware-integrated workflows require vCenter 6.x as many use SDK calls that are only supported in the 6.x version of the vCenter SDK. In order to use those workflows certain edits will need to be made to the workflows when used with vCenter 5.5. Making custom workflows with vCenter 5.5 is fully supported though by the plugin.

INSTALLATION

The Pure Storage FlashArray Plugin for vRealize Orchestrator can be downloaded from the VMware Solution Exchange:
https://solutionexchange.vmware.com/store/products/pure-storage-flasharray-plugin-for-vmware-vrealize-orchestrator

The plugin is of type .vmoapp:

<table>
<thead>
<tr>
<th>Name</th>
<th>Date modified</th>
<th>Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>o11nplugin-purestorage-vro-plugin.vmoapp</td>
<td>12/1/2016 2:57 PM</td>
<td>VMAPP File</td>
<td>5,185 KB</td>
</tr>
</tbody>
</table>

The plugin is supported with the 6.x release (6.0.1 and later) of vRO and 7.x. The process for installing plugins is slightly different depending on whether you are using 6 or 7.
INSTALLING THE FLASHARRAY PLUGIN WITH VRO 6.X

- Login to your vRO appliance at the following address:
  https://<vRO FQDN or IP>:8283/vco-config/

Click on the **Plugins** menu button and then the search icon to find the plugin file. Click **Upload and install** to start the installation.

Accept the EULA.
Once accepted, the plugin will be installed.

You must now restart the core vRO service in order to activate the plugin.

Restarting the core vRO service is a disruptive operation that will kick users out of the vRO client and will terminate running workflows. Be 100% sure that there are no interfering operations before restarting the service.
vRO 6.0.1 and 6.0.2 do not support, by default, the proper security protocol that the FlashArray requires for REST (or any method) communication. The FlashArray requires TLS 1.1 or later. Additional information on how to configure the proper TLS support in vRO can be found in the appendix here Appendix III: Enabling TLS 1.1 and 1.2. vRO 6.0.3 and later are fine by default. https://www.vmware.com/support/orchestrator/doc/vrealize-orchestrator-603-release-notes.html

INSTALLING THE FLASHARRAY PLUGIN WITH VRO 7.X

- Login to your vRO appliance at the following address
  https://<vRO FQDN or IP>:8283/vco-controlcenter/#

Click on the Plugins menu button and then the Browse... button to find the plugin file. Click Upload and install to start the installation.
Accept the EULA.

Once accepted, the plugin will be installed.
You must now restart the core vRO service in order to activate the plugin.

Restarting the core vRO service is a disruptive operation that will kick users out of the vRO client and will terminate running workflows. Be 100% sure that there are no interfering operations before restarting the service.

UNINSTALLATION

There is no GUI-based uninstallation of plugins for vRealize Orchestrator – uninstallation requires root access to the bash shell (or Microsoft® Windows host). For uninstallation instructions, please refer to the following VMware Knowledge Base article:

https://kb.vmware.com/kb/2064575
CONFIGURATION

When the plugin is installed, a variety of workflows are immediately provided for use. The following categories are available:

- **FlashArray Connection Management** – Adding/removing FlashArray connections into vRO
- **FlashArray Host Management** – Adding/removing/changing FlashArray hosts and host groups
- **FlashArray Management** – Configuring FlashArray-level settings
- **FlashArray Protection Group Management** – Adding/removing/changing FlashArray local snapshot or remote replication policies with Protection Groups
- **FlashArray REST APIs** – Allows for custom and direct API interaction with the FlashArray
- **FlashArray Snapshot Management** – Adding/removing/changing FlashArray snapshots
- **FlashArray Volume Management** – Adding/removing/changing FlashArray volumes
- **VMware ESXi® and FlashArray** – Built-in workflows for managing storage in a VMFS environment

In the section **Built-in Workflows**, the majority of these workflows will be explained in more detail. This section will focus on configuration pre-requisites to use the plugin fully. This includes three steps:

- Adding a certificate for the FlashArray
- Authenticating a FlashArray
- Authenticating a vCenter

**ADDING A FLASHARRAY CERTIFICATE TO VREALIZE ORCHESTRATOR**

The first step is adding the FlashArray certificate to vRO. Run the workflow **Import FlashArray certificate from URL** in the FlashArray Connection Management folder.
Fill out the FQDN or IP of the FlashArray. If the certificate is self-signed, change the accept certificate with warnings option to Yes. Click Submit.

**Note that you may use the default vRO workflows to add a FlashArray certificate if you so choose – you do not need to use the FlashArray-specific certificate workflow. This workflow was provided mainly for convenience.**

You can view the certificate in the **Inventory** tab under vRO Configuration → Keystores → CA Keystore.

**ADDING A FLASHARRAY TO VREALIZE ORCHESTRATOR**

To add a FlashArray to vRO, navigate to the FlashArray Connection Management folder and start the **Add FlashArray Connection** workflow.
Fill out the connectivity information for the FlashArray you would like to add.

The available API versions will auto-populate in the API Version entry drop-down after the FlashArray URL is entered and will default to the newest. The recommendation is to keep with the newest API version unless you have a reason to leverage an older API version of the FlashArray REST service. Change the autopopulate to Yes to retrieve the REST versions. If nothing populates, ensure the certificate was added and that the FQDN/IP is correct. The FlashArray REST service requires port 443 TCP access to function.

Submit the information and look for the FlashArray to be successfully added – indicated by a green check mark on the workflow run instance.
ADDING A VCENTER TO VREALIZE ORCHESTRATOR

Adding a vCenter is an optional task, but if you plan on managing VMware objects (datastores, VMs, etc.) and/or using the VMware-integrated workflows in the FlashArray plugin, it is necessary to have at least one vCenter registered. A full description of the process can be found in VMware documentation.
CUSTOM OBJECTS AND FLASHARRAY INVENTORY

The FlashArray plugin for vRO introduces a few new object types into vRealize Orchestrator. These objects types have been created to allow end users a simple way to manipulate and manage FlashArray-specific objects in your actions, scripts, and workflows.

This inventory screen can be accessed from any view in vRO. If an item that has been created does not appear in the inventory, please refresh the vRO interface to have the object populated.
Destroyed, but not eradicated, objects are not displayed in the vRO inventory. Objects in this state are automatically eradicated from the FlashArray in 24 hours.

CUSTOM OBJECTS

The plugin offers a variety of new object types in to the vRO inventory:

- PS:FlashArrayConnection
- PS:FlashArray
- PS:Volume
- PS:Snapshot
- PS:Host
- PS:HostGroup
- PS:ProtectionGroup

These objects represent their respective objects on the FlashArray and can be passed in and out of workflows and actions and managed in external applications like vRealize Automation. When passed into scripting objects in vRO they are represented by respective “scripting objects” which provide additional methods to manage and alter the object configurations. More information on managing scripting objects can be found in the section Using Scripting Objects and Methods.

See the section, Building a New Action for more details on using objects.

Read on for a detailed description of these object types:

- **PS:FLASHARRAYCONNECTION**: This is an object that defines a specific connection to a FlashArray and is the highest level object. This object has the following properties:
  - **Username** – The username of the user who created the connection; if this user is deleted from the FlashArray the connection will be broken
  - **FlashArray Connection** – The friendly name specified upon creation of the connection
  - **Display Name** – A combination of the display name and the URI used to connect to the FlashArray
- **Credentials** – A more detailed description of the user name who authenticated the session and an internal session ID for the connection
- **Base URI** – The URL used for the REST connection, this will specify the REST version at the end of the URL string

<table>
<thead>
<tr>
<th>User Name</th>
<th>pureuser</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlashArray Connection</td>
<td>m20-1</td>
</tr>
<tr>
<td>Display Name</td>
<td>m20-1 [<a href="https://flasharray-m20-1.flashcloud.local/api/1.7">https://flasharray-m20-1.flashcloud.local/api/1.7</a>]</td>
</tr>
<tr>
<td>Credentials</td>
<td>Credentials [id=97c2a17-bc46-4515-bee2-bf39b615f351, username=pureuser]</td>
</tr>
<tr>
<td>Base URI</td>
<td><a href="https://flasharray-m20-1.flashcloud.local/api/1.7">https://flasharray-m20-1.flashcloud.local/api/1.7</a></td>
</tr>
</tbody>
</table>

- **PS:FLASHARRAY** – An object that represents the FlashArray specified in the FlashArray connection. This information is not tied to a single session. Properties include:
  - **Name** – The name of the FlashArray
  - **ID** – The unique identifier of the FlashArray
  - **Display Name** – This is the friendly name of the FlashArray
  - **Version** – Purity Operating Environment version
  - **Revision** – Specific Purity version build information

<table>
<thead>
<tr>
<th>Name</th>
<th>solutions-lab-m20-c09-25</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>4fb2b26d-4b69-4ddf-8540-86659a6a6846</td>
</tr>
<tr>
<td>Display Name</td>
<td>solutions-lab-m20-c09-25 [4.8.1]</td>
</tr>
<tr>
<td>Version</td>
<td>4.8.1</td>
</tr>
<tr>
<td>Revision</td>
<td>201669270537+0302232</td>
</tr>
</tbody>
</table>

- **PS:VOLUME** – This is a new object type that represents a FlashArray volume. Properties are:
  - **Name** – The volume name
  - **Size** – The provisioned size in bytes of the volume
  - **Display Name** – The volume name (currently will always be the same as the name)
  - **Source** – If the volume was copied from another volume this will have the source volume name. Will be null if it is an original volume
  - **Created** – Time in UTC format when the volume was created
  - **Serial** – The serial number of the volume
PS:SNAPSHOT – This is a new object type that represents a FlashArray snapshot. Properties are:

- **Name** – The snapshot name: if it is a one-off snapshot it will be in the form of `<volume name>.<snapshot name>` (e.g., “myVolume.mySnapshot”); if it is a snapshot from a protection group, it will be in the form of `<protection group name>.<snapshot name>.<volume name>` (e.g., “myPGroup.mySnapshot.myVolume”)
- **Size** – The size in bytes of the data being preserved by the snapshot
- **Display Name** – The snapshot name: if it is a one off snapshot it will be in the form of `<volume name>.<snapshot name>` (e.g., “myVolume.mySnapshot”); if it is a snapshot from a protection group, it will be in the form of `<protection group name>.<snapshot name>.<volume name>` (e.g., “myPGroup.mySnapshot.myVolume”)
- **Source** – The name of the volume the snapshot was created from
- **Created** – Time in UTC format when the snapshot was created
- **Serial** – The serial number of the snapshot

<table>
<thead>
<tr>
<th>Name</th>
<th>SIOCVMFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>8796093022209</td>
</tr>
<tr>
<td>Display Name</td>
<td>SIOCVMFS</td>
</tr>
<tr>
<td>Source</td>
<td></td>
</tr>
<tr>
<td>Created</td>
<td>2016-11-07T00:11:06Z</td>
</tr>
<tr>
<td>Serial</td>
<td>4BFA20D4B694DF7000134E9</td>
</tr>
</tbody>
</table>

PS:HOST – This is a new object type that represents a FlashArray host. Properties are:

- **Name** – The name of the host
- **Display Name** – The host name (currently will always be the same as the name)
- **Host Group** – The name of the host group if the host is in a host group

<table>
<thead>
<tr>
<th>Name</th>
<th>slice60.test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>1099511627776</td>
</tr>
<tr>
<td>Display Name</td>
<td>slice60.test</td>
</tr>
<tr>
<td>Source</td>
<td>slice60</td>
</tr>
<tr>
<td>Created</td>
<td>2016-12-06T01:05:04Z</td>
</tr>
<tr>
<td>Serial</td>
<td>4BFA20D4B694DF7000137D7</td>
</tr>
</tbody>
</table>
- **IQN** – The IQNs assigned to this host; this is an array of strings
- **WWN** – the WWNs assigned to this host; this is an array of strings

<table>
<thead>
<tr>
<th>Name</th>
<th>ESXi-01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Name</td>
<td>ESXi-01</td>
</tr>
<tr>
<td>Host Group</td>
<td>vCenter006A</td>
</tr>
<tr>
<td>IQN</td>
<td>[iqn.1926-01.com.vmware.57384190-54d00-1ed1-aed7-0025b5123426-3468753d]</td>
</tr>
<tr>
<td>WWN</td>
<td>[20000025B511114F, 20000025B54444F]</td>
</tr>
</tbody>
</table>

• **PS:HOSTGROUP** – This is a new object type that represents a FlashArray host group. Properties are:
  - **Name** – The name of the host group
  - **Display Name** – The host group name (currently will always be the same as the name)
  - **Hosts** – A list of the hosts in that host group; this is an array of strings

<table>
<thead>
<tr>
<th>Name</th>
<th>vCenter60U2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Name</td>
<td>vCenter60U2</td>
</tr>
<tr>
<td>Hosts</td>
<td>[ESXi-06, ESXi-07, ESXi-08]</td>
</tr>
</tbody>
</table>

• **PS:PROTECTIONGROUP** – This is a new object type that represents a FlashArray protection group. Properties are:
  - **Display Name** – The name of the protection group: if this is local to the FlashArray it will just be the protection group name; if it is remote to the FlashArray (i.e., replicating to it) it will be in the form of “<FlashArray target name>:<protection group name>” (e.g., “FlashArrayNYC:MyPGroup”)
  - **Source** – The name of the FlashArray the protection group resides on
  - **Volumes** – The volumes currently in the protection group: if the protection group is remote to the FlashArray the volume name will be appended to the source FlashArray name after a colon (e.g., “FlashArrayNYC:MyVolume”). This is an array of strings
  - **Hosts** – The hosts currently in the protection group: if the protection group is remote to the FlashArray the hosts name will be appended to the source FlashArray name after a colon (e.g., “FlashArrayNYC:MyHost”). This is an array of strings.
– **Host Groups** – The host groups currently in the protection group: if the protection group is remote to the FlashArray the host group names will be appended to the source FlashArray name after a colon (e.g., “FlashArrayNYC:MyHostGroup”). This is an array of strings

– **Target** – The FlashArrays that the protection group is replicating to and whether that connection is enabled

<table>
<thead>
<tr>
<th>Display Name</th>
<th>smtest1test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>solutions-lab-m20-c09-25</td>
</tr>
<tr>
<td>Volumes</td>
<td>[srm datastore]</td>
</tr>
<tr>
<td>Hosts</td>
<td></td>
</tr>
<tr>
<td>Targets</td>
<td>[Target[name=solutions-lab-420-c11-31, allowed=true]]</td>
</tr>
<tr>
<td>HostGroups</td>
<td></td>
</tr>
</tbody>
</table>

**SCRIPTING OBJECTS**

The FlashArray plugin also offers scripting objects that allow for easier management of FlashArray provisioning tasks above and beyond the object types mentioned before. Scripting objects offer a variety of methods that provide additional operations that can be executed against that object type. Objects mentioned in the previous section are respective scripting objects when passed into a scriptable task in vRO (JavaScript). Furthermore, there are scripting objects that do not have mapped object types.

The main difference between standard object types and scripting objects is that object types can be passed between actions and workflows. Scripting objects can only be leveraged in JavaScript for custom operations. Certain scripting objects can be passed out of between actions if it has a corresponding object type.

For a detailed description of scripting objects, please refer to the section **Using Scripting Objects and Methods**.

**RUNNING WORKFLOWS FROM OBJECTS**

In the inventory view of vRealize Orchestrator, it is possible to right-click on an object and then run a workflow from that specific object.

For instance, if you click on a FlashArray connection, choose **Run Workflow**.
Then choose a workflow to run against that FlashArray connection.

Whichever input matches that input type will be auto-populated with the object you originally clicked on.

Note that the right-click menu is not an exhaustive list of workflows that take that object type in. It is also not dynamic – creating new workflows that take in that object type will not auto-populate. Requests to add menu options can be sent to Pure Storage, by contacting support@purestorage.com.
Once the wizard is completed, a dialog box will appear until the workflow completes.
You can close the box to allow the workflow to run in the background and you can then
monitor its process in the normal fashion.

**BUILT-IN ACTIONS**

The FlashArray plugin for vRealize Orchestrator includes a variety of actions to make
custom workflow creation much easier. These actions are used by the built-in workflows
and can also be used by your own workflows.

The plugin actions can be found under the `com.purestorage.flasharray.xxx` namespace.

**ACTION DESCRIPTIONS**

This section will detail the actions built into the FlashArray plugin for vRealize
Orchestrator. There are nine sections:

- Connection actions
- Host actions
- Host group actions
- Array management actions
- Protection group actions
- Custom REST actions
- Snapshot actions
- Volume actions
- VMware actions

**CONNECTION ACTIONS**

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFlashArrayConnection</td>
<td>Adds a new FlashArray connection to vRO</td>
<td>Name [string]</td>
<td>Connection object [PS:FlashArrayConnection]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FQDN/IP [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Username [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Password [SecureString]</td>
<td></td>
</tr>
<tr>
<td>getAllSupportedApiVersions</td>
<td>Identifies all supported REST versions supported by a FlashArray</td>
<td>Connection object [PS:FlashArrayConnection]</td>
<td>REST versions [Array of string]</td>
</tr>
<tr>
<td>getFlashArrayConnection</td>
<td>Returns a connection</td>
<td>Name [string]</td>
<td>Connection object [PS:FlashArrayConnection]</td>
</tr>
<tr>
<td>getFlashArrayConnections</td>
<td>Returns all connections</td>
<td>None</td>
<td>Connection objects [Array of PS:FlashArrayConnection]</td>
</tr>
<tr>
<td>removeFlashArrayConnection</td>
<td>Removes a FlashArray connections</td>
<td>Connection object [PS:FlashArrayConnection]</td>
<td>Success result [boolean]</td>
</tr>
<tr>
<td>updateFlashArrayConnection</td>
<td>Updates a FlashArray connection</td>
<td>Connection object [PS:FlashArrayConnection]</td>
<td>Connection object [PS:FlashArrayConnection]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Name [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FQDN/IP [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Username [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Password [SecureString]</td>
<td></td>
</tr>
<tr>
<td>validateFlashArrayConnection</td>
<td>Validates connection information</td>
<td>Name [string]</td>
<td>Success result [boolean]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FQDN/IP [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>REST version [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Username [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Password [SecureString]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Discover REST versions [boolean]</td>
<td></td>
</tr>
</tbody>
</table>
### HOST ACTIONS

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>createFlashArrayHost</td>
<td>Creates a new FlashArray host</td>
<td>Name [string] WWNs [array of string] IQNs [array of string] Connection [PS:FlashArrayConnection]</td>
<td>Host [PS:Host]</td>
</tr>
<tr>
<td>getAllFlashArrayHosts</td>
<td>Returns all hosts on a FlashArray</td>
<td>Connection object [PS:FlashArrayConnection]</td>
<td>Hosts [Array of PS:Host]</td>
</tr>
<tr>
<td>getHostsOfAllConnections</td>
<td>Returns all hosts on all FlashArrays</td>
<td>None</td>
<td>Hosts [Array of PS:Host]</td>
</tr>
<tr>
<td>removeFlashArrayHost</td>
<td>Deletes a host from the FlashArray</td>
<td>Host [PS:Host] Name of host [string]</td>
<td></td>
</tr>
<tr>
<td>updateFlashArrayHostName</td>
<td>Changes a host name</td>
<td>New name [string] Host [PS:Host]</td>
<td>Host [PS:Host]</td>
</tr>
</tbody>
</table>

### HOST GROUP ACTIONS

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>createFlashArrayHostGroup</td>
<td>Creates an empty host group</td>
<td>Host group name [string] FlashArray [PS:FlashArrayConnection]</td>
<td>Host group [PS:HostGroup]</td>
</tr>
<tr>
<td>getAllFlashArrayHostGroups</td>
<td>Returns all host groups on a FlashArray</td>
<td>Connection object [PS:FlashArrayConnection]</td>
<td>Host groups [Array of PS:HostGroup]</td>
</tr>
<tr>
<td>getHostGroupsOfAllConnections</td>
<td>Returns all host groups on all FlashArrays</td>
<td>None</td>
<td>Host groups [Array of PS:HostGroup]</td>
</tr>
</tbody>
</table>
### HOST GROUP ACTIONS (CONT.)

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>getSpecificHostGroup</td>
<td>Returns a specific host group</td>
<td>Host group name [string]</td>
<td>Host group [PS:HostGroup]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td></td>
</tr>
<tr>
<td>removeFlashArrayHostGroup</td>
<td>Deletes an empty host group from the FlashArray</td>
<td>Host group [PS:HostGroup]</td>
<td>Name of host group [string]</td>
</tr>
<tr>
<td>removeFlashArrayHostGroupWithHosts</td>
<td>Deletes a host group from the FlashArray even if it has hosts in it</td>
<td>Host group [PS:HostGroup]</td>
<td>Name of host group [string]</td>
</tr>
<tr>
<td>updateFlashArrayHostGroupName</td>
<td>Changes a host group name</td>
<td>New name [string]</td>
<td>Host group [PS:HostGroup]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Host group [PS:HostGroup]</td>
<td></td>
</tr>
</tbody>
</table>

### FLASHARRAY MANAGEMENT ACTIONS

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllFlashArrays</td>
<td>Return all FlashArrays</td>
<td>None</td>
<td>FlashArrays [array of PS:FlashArray]</td>
</tr>
<tr>
<td>renameFlashArray</td>
<td>Renames a FlashArray</td>
<td>FlashArray [PS:FlashArray]</td>
<td>Name of host group [string]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>New name [string]</td>
<td></td>
</tr>
</tbody>
</table>

### HOST GROUP ACTIONS

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>createFlashArrayProtectionGroup</td>
<td>Creates an empty protection group</td>
<td>Protection group name [string]</td>
<td>Protection group [PS:ProtectionGroup]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td></td>
</tr>
<tr>
<td>eradicateFlashArrayProtectionGroup</td>
<td>Permanently removes a destroyed protection group</td>
<td>Protection group [PS:ProtectionGroup]</td>
<td>Name of protection group [string]</td>
</tr>
<tr>
<td>getProtectionGroupsForDatastore</td>
<td>Returns all protection groups of a given datastore</td>
<td>Datastore [VC:Datastore]</td>
<td>Protection groups [Array of PS: ProtectionGroup]</td>
</tr>
<tr>
<td>getFlashArrayProtectionGroups</td>
<td>Returns all protection groups of a given FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Protection groups [Array of PS: ProtectionGroup]</td>
</tr>
</tbody>
</table>
### HOST GROUP ACTIONS (CONT.)

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>getProtectionGroupsOfAll Connections</td>
<td>Returns all protection groups on all FlashArrays</td>
<td>None</td>
<td>Protection groups [Array of PS: ProtectionGroup]</td>
</tr>
<tr>
<td>removeFlashArrayProtection Group</td>
<td>Deletes a protection group from the FlashArray</td>
<td>Protection group name [string]</td>
<td>Protection group name [string]</td>
</tr>
<tr>
<td>updateFlashArrayProtection GroupName</td>
<td>Changes a protection group name</td>
<td>New name [string] Protection group name [string]</td>
<td>Protection group name [string]</td>
</tr>
</tbody>
</table>

### RUN REST API ACTIONS

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>runAPI</td>
<td>Runs a REST API call to a FlashArray that is not built into the plugin</td>
<td>FlashArray [PS:FlashArray] Resource URI [string] HTTP method [string] JSON body [string]</td>
<td>JSON result [string]</td>
</tr>
</tbody>
</table>

For more information on the REST API action, refer to the section, **Running Direct REST API Calls to a FlashArray.**

### SNAPSHOT ACTIONS

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllSnapshots</td>
<td>Retrieves all snapshots from a specific FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Snapshots [Array of PS:Snapshot]</td>
</tr>
<tr>
<td>getAllSnapshotsOfAll Connections</td>
<td>Retrieves all snapshots from all FlashArrays</td>
<td>None</td>
<td>Snapshots [Array of PS:Snapshot]</td>
</tr>
</tbody>
</table>
### SNAPSHOT ACTIONS (CONT.)

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>destroySnapshot</td>
<td>Destroys a snapshot. The snapshot is not eradicated.</td>
<td>Snapshot [PS:Snapshot]</td>
<td>Deleted snapshot name [string]</td>
</tr>
<tr>
<td>eradicateSnapshot</td>
<td>Permanently removes a destroyed snapshot.</td>
<td>Snapshot [PS:Snapshot]</td>
<td>Eradicated snapshot name [string]</td>
</tr>
</tbody>
</table>

### VOLUME ACTIONS

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>createVolumeByCopySource</td>
<td>Creates a new volume by copying it from a snapshot or a volume</td>
<td>Name [string] Source [string] FlashArray [PS:FlashArrayConnection]</td>
<td>Volume [PS:Volume]</td>
</tr>
<tr>
<td>disconnectDestroyVolume</td>
<td>Disconnects volume if it connected to host or host groups. After disconnecting it destroys volume. Optionally can eradicate the volume.</td>
<td>Volume [PS:Volume] Eradicate option [boolean]</td>
<td>Volume name [string]</td>
</tr>
<tr>
<td>disconnectVolume</td>
<td>Disconnects volume if it connected to host or host groups.</td>
<td>Volume [PS:Volume]</td>
<td>Removed objects [properties]</td>
</tr>
<tr>
<td>eradicateVolume</td>
<td>Permanently remove a volume from its name</td>
<td>Name [string] FlashArray [PS:FlashArrayConnection]</td>
<td>Volume name [string]</td>
</tr>
</tbody>
</table>
### VOLUME ACTIONS (CONT.)

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllVolumes</td>
<td>Retrieve all volumes from a FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Volumes [array of PS:Volume]</td>
</tr>
<tr>
<td>getAllVolumesOfAll</td>
<td>Retrieve all volumes from all FlashArrays</td>
<td>None</td>
<td>Volumes [array of PS:Volume]</td>
</tr>
<tr>
<td>Connections</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getAllVolumesPending</td>
<td>Retrieve all volume names of all volume pending eradication from a FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Volumes [array of string]</td>
</tr>
<tr>
<td>Eradication</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getVolume</td>
<td>Retrieve a specific volume from a FlashArray</td>
<td>Name [string], FlashArray [PS:FlashArrayConnection]</td>
<td>Volume [PS:Volume]</td>
</tr>
<tr>
<td>increaseVolumeSize</td>
<td>Increases the size of a FlashArray volume</td>
<td>Volume [PS:Volume], Size [string]</td>
<td>Volume [PS:Volume]</td>
</tr>
<tr>
<td>overwriteVolume</td>
<td>Copies one volume to another</td>
<td>Source name [string], Target volume [PS:Volume]</td>
<td>Volume [PS:Volume]</td>
</tr>
<tr>
<td>renameVolume</td>
<td>Renames a volume</td>
<td>Name [string], Volume [PS:Volume]</td>
<td>Volume [PS:Volume]</td>
</tr>
</tbody>
</table>

### VMWARE ACTIONS

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Return value</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAvailableDisksWithCapacity</td>
<td>Returns a list of NAA numbers of devices that are available for VMFS formatting on an ESXi host.</td>
<td>ESXi [VC:HostSystem]</td>
<td>NAA numbers [array of string]</td>
</tr>
<tr>
<td>getFASnapshotsOfVolume</td>
<td>Returns all snapshots of a VMFS datastore</td>
<td>Datastore [VC:Datastore]</td>
<td>Snapshots [array of PS:Snapshot]</td>
</tr>
<tr>
<td>getHostGroupNameFromvCenterCluster</td>
<td>Returns a valid name of a FlashArray host group by removing any special characters in a cluster name</td>
<td>Cluster [VC:ClusterComputeResource]</td>
<td>Host group name [string]</td>
</tr>
</tbody>
</table>
BUILT-IN WORKFLOWS

The FlashArray plugin offers built-in workflows for users to execute subsequent to installation of the plugin and registering of at least one FlashArray. These workflows are separated into seven categories:

- **FlashArray Connection Management** – this includes all of the workflows to add/change/remove a FlashArray connection inside of vRO.
- **FlashArray Host Management** – these workflows control creating/changing/removing FlashArray hosts and host groups
  - Host Group Management
  - Host Management
- **FlashArray Management** – this includes workflows for altering array-level settings
- **FlashArray Protection Group Management** – these workflows control creating/changing/removing FlashArray protection groups
- **FlashArray REST APIs** – workflows for running custom REST API calls to the FlashArray
- **FlashArray Snapshot Management** – these workflows control creating/changing/removing FlashArray snapshots
- **FlashArray Volume Management** – these workflows control creating/changing/removing FlashArray volumes
- **VMware ESXi and FlashArray** – these workflows are targeted at integrating VMware provision tasks with FlashArray ones to provide end-to-end provisioning

All workflows are in the Pure Storage folder in vRO and are separated into corresponding folders.

WORKFLOW DESCRIPTIONS

**FLASHARRAY CONNECTION MANAGEMENT**

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add FlashArray Connection</td>
<td>This workflow adds a FlashArray connection to vRO.</td>
<td>Friendly name [string]</td>
<td>FlashArray connection object [PS:FlashArrayConnection]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IP/FQDN of FlashArray [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Discover REST version [boolean]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>REST API version [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray username [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray password [secure string]</td>
<td></td>
</tr>
<tr>
<td>Get All FlashArray Connections</td>
<td>Returns all FlashArray connection objects in the vRO inventory</td>
<td>None</td>
<td>FlashArray connections [array of PS:FlashArrayConnection]</td>
</tr>
</tbody>
</table>
**FLASHARRAY CONNECTION MANAGEMENT (CONT.)**

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get FlashArray Connection</td>
<td>Returns the connection object for a given FlashArray name</td>
<td>FlashArray name [string]</td>
<td>FlashArray connection [PS:FlashArrayConnection]</td>
</tr>
<tr>
<td>Import FlashArray Certificate from URL</td>
<td>Imports the certificate for a FlashArray into vRO</td>
<td>FlashArray URL [string] Accept certificate with warnings [boolean]</td>
<td>None</td>
</tr>
<tr>
<td>Remove FlashArray Connection</td>
<td>Removes a specific FlashArray connection</td>
<td>FlashArray connection [FlashArrayConnection]</td>
<td>Operation success [boolean]</td>
</tr>
</tbody>
</table>

*All inputs marked with an asterisk (*) are optional. All others are mandatory.*

**FLASHARRAY HOST MANAGEMENT – HOST GROUP MANAGEMENT**

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create FlashArray Host Group</td>
<td>Creates an empty host group</td>
<td>Host group name [string] FlashArray [PS:FlashArrayConnection]</td>
<td>Host group [PS:HostGroup]</td>
</tr>
<tr>
<td>Create FlashArray Host Group With Hosts</td>
<td>Creates a host group and adds input hosts</td>
<td>Host group name [string] List of hosts [array of PS:Host] FlashArray [PS:FlashArrayConnection]</td>
<td>Host group [PS:HostGroup]</td>
</tr>
</tbody>
</table>
## FLASHARRAY HOST MANAGEMENT – HOST GROUP MANAGEMENT (CONT.)

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get All FlashArray Host Groups</td>
<td>Retrieves all host groups from a specific FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Host groups [Array of PS:HostGroup]</td>
</tr>
<tr>
<td>Get Specific FlashArray Host Group</td>
<td>Retrieves a specific host group from a FlashArray</td>
<td>Host group name [string] FlashArray [PS:FlashArrayConnection]</td>
<td>Host group [PS:HostGroup]</td>
</tr>
<tr>
<td>Remove FlashArray Host Group With Hosts</td>
<td>Removes a host group with hosts. The hosts themselves are not deleted</td>
<td>Host group [PS:HostGroup] Removed host group name [string]</td>
<td>Removed host group name [string]</td>
</tr>
<tr>
<td>Rename Host Group</td>
<td>Renames a host group</td>
<td>Host group [PS:HostGroup] New name [string]</td>
<td>Host group [PS:HostGroup]</td>
</tr>
</tbody>
</table>

## FLASHARRAY HOST MANAGEMENT – HOST MANAGEMENT

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create FlashArray Host</td>
<td>Creates an empty host</td>
<td>Host name [string] FlashArray [PS:FlashArrayConnection]</td>
<td>Host [PS:Host]</td>
</tr>
<tr>
<td>Create FlashArray Host Group with Hosts</td>
<td>Creates a host and adds input initiators</td>
<td>Host name [string] List of IQNs [array of string]* List of WWNs [array of string]* FlashArray [PS:FlashArrayConnection]</td>
<td>Host [PS:Host]</td>
</tr>
<tr>
<td>Get All FlashArray Hosts</td>
<td>Retrieves all hosts from a specific FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Hosts [Array of PS:Host]</td>
</tr>
<tr>
<td>Get Specific FlashArray Host</td>
<td>Retrieves a specific host from a FlashArray</td>
<td>Host name [string] FlashArray [PS:FlashArrayConnection]</td>
<td>Host [PS:Host]</td>
</tr>
<tr>
<td>Remove FlashArray Host</td>
<td>Removes a host</td>
<td>Host [PS:Host]</td>
<td>Removed host name [string]</td>
</tr>
<tr>
<td>Rename Host</td>
<td>Renames a host</td>
<td>Host [PS:Host]</td>
<td>Host [PS:Host]</td>
</tr>
</tbody>
</table>
FLASHARRAY MANAGEMENT

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rename FlashArray</td>
<td>Renames the actual name of the FlashArray</td>
<td>New name [string]</td>
<td>Success result [boolean]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray [PS:FlashArray]</td>
<td></td>
</tr>
</tbody>
</table>

FLASHARRAY PROTECTION GROUP MANAGEMENT

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add FlashArray Protection Group</td>
<td>Creates a new empty protection group</td>
<td>Protection group name [string]</td>
<td>Protection group [PS:ProtectionGroup]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td></td>
</tr>
<tr>
<td>Destroy FlashArray Protection Group</td>
<td>Destroy a protection group and puts it into the deleted items folder</td>
<td>Protection group [PS:ProtectionGroup]</td>
<td>Removed protection group name [string]</td>
</tr>
<tr>
<td>Get All FlashArray Protection Groups</td>
<td>Retrieves all protection groups from a specific FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Protection groups [Array of PS:ProtectionGroup]</td>
</tr>
<tr>
<td>Get Specific FlashArray Protection Group</td>
<td>Retrieves a specific protection group from a FlashArray</td>
<td>Protection group name [string]</td>
<td>Protection group [PS:ProtectionGroup]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New name [string]</td>
<td></td>
</tr>
</tbody>
</table>

FLASHARRAY REST APIS

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run REST API</td>
<td>Runs any REST API supported by the FlashArray to a specific FlashArray</td>
<td>FlashArray [PS:FlashArray]</td>
<td>Result [string]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Resource URI [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP method [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>JSON body [string]</td>
<td></td>
</tr>
</tbody>
</table>

More information on the Run REST API workflow in the section Running Direct REST API Calls to a FlashArray.
## FLASHARRAY SNAPSHOTS MANAGEMENT

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Snapshot</td>
<td>Creates a snapshot of a volume</td>
<td>Name suffix [string]* FlashArray volume [PS:Volume]</td>
<td>Snapshot [PS:Snapshot]</td>
</tr>
<tr>
<td>Create Snapshot of Multiple Volumes</td>
<td>Creates a snapshot of multiple volumes</td>
<td>Name suffix [string]* FlashArray volumes [Array of PS:Volume] FlashArray [PS:FlashArrayConnection]</td>
<td>Snapshots [Array of PS:Snapshot]</td>
</tr>
<tr>
<td>Get All Snapshots</td>
<td>Retrieves all snapshots from a specific FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Snapshots [Array of PS:Snapshot]</td>
</tr>
<tr>
<td>Destroy Snapshot</td>
<td>Destroys a snapshot. The snapshot is not eradicated.</td>
<td>Snapshot [PS:Snapshot] Deleted snapshot name [string]</td>
<td></td>
</tr>
</tbody>
</table>
## FLASHARRAY VOLUME MANAGEMENT

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Volume</td>
<td>Creates a volume</td>
<td>Name [string]</td>
<td>Volume [PS:Volume]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size [number]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size unit (KB, MB, GB, TB, PB) [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td></td>
</tr>
<tr>
<td>Destroy Volume</td>
<td>Destroys a volume. The volume is not eradicated.</td>
<td>Volume [PS:Volume]</td>
<td>Destroyed volume name [string]</td>
</tr>
<tr>
<td>Eradicate Volume</td>
<td>Eradicates a destroyed volume. This causes the volume to be unrecoverable.</td>
<td>Volume name [string]</td>
<td>Eradicated volume name [string]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td></td>
</tr>
<tr>
<td>Get All Volumes</td>
<td>Retrieves all volumes from a specific FlashArray</td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td>Volumes [Array of PS:Volume]</td>
</tr>
<tr>
<td>Get Volume</td>
<td>Retrieves a specific volume from a FlashArray</td>
<td>Volume name [string]</td>
<td>Volume [PS:Volume]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FlashArray [PS:FlashArrayConnection]</td>
<td></td>
</tr>
<tr>
<td>Increase Volume Size</td>
<td>Increases the allocated capacity of a volume</td>
<td>Volume [PS:Volume]</td>
<td>Volume [PS:Volume]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size [number]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size unit (KB, MB, GB, TB, PB) [string]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New name [string]</td>
<td></td>
</tr>
</tbody>
</table>

*Note: The Eradicate Volume command is immediate and permanent; there is no recovery possible after this command.*
## VMWARE ESXI AND FLASHARRAY

<table>
<thead>
<tr>
<th>Workflow Name</th>
<th>Description</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Datastore to FlashArray Protection Group</td>
<td>This takes in a VMFS datastore and then adds the corresponding FlashArray volume to a given protection group</td>
<td>Datastore [VC:Datastore] Protection group [PS:ProtectionGroup]</td>
<td>Volume [PS:Volume] Protection group [PS:ProtectionGroup]</td>
</tr>
<tr>
<td>Create a FlashArray Host Group from a vCenter Cluster</td>
<td>Takes in a VMware cluster and then creates corresponding hosts and then a host group and adds the host to it</td>
<td>vCenter cluster [VC:ClusterCompute Resource] FlashArray [PS:FlashArrayConnection]</td>
<td>Host group [PS:HostGroup]</td>
</tr>
<tr>
<td>Create VMFS Datastore from FlashArray Snapshot</td>
<td>Creates a datastore copy from a snapshot of a VMFS. This will mount it to the specified cluster and resignature the datastore. The datastore will be renamed to the entered name</td>
<td>Snapshot [PS:Snapshot] Datastore name [string] vCenter cluster [VC:ClusterCompute Resource]</td>
<td>Volume [PS:Volume] Resignatured datastore [VC:Datastore]</td>
</tr>
<tr>
<td>Create VMFS Datastore on Existing FlashArray volume</td>
<td>Creates a datastore on a volume that is already provisioned to an ESXi host</td>
<td>ESXi host [VC:HostSystem] NAA of a disk [string] Datastore name [string]</td>
<td>Datastore [VC:Datastore]</td>
</tr>
<tr>
<td>Workflow Name</td>
<td>Description</td>
<td>Inputs</td>
<td>Outputs</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
</tbody>
</table>
| Delete VMFS Datastore                 | This will unmount a datastore and then detach it from all of connected ESXi hosts. It will then destroy it or destroy it and eradicate it depending on what the user chooses.                                          | Destroy option [string]  
Datastore [VC:Datastore] | Destroyed volume name [string]                                                   |
| Expand VMFS Datastore                 | This will increase the FlashArray volume size and then rescan the cluster and perform a VMFS volume grow to use the additional capacity                                                                 | Datastore [VC:Datastore]  
Size [number]  
Size unit (KB, MB, GB, TB, PB) [string] | Volume [PS:Volume]  
Datastore [VC:Datastore] |
| Remove Datastore from FlashArray Protection Group | This takes in a VMFS datastore and then removes the corresponding FlashArray volume from a given protection group                                                                                     | Datastore [VC:Datastore]  
Protection group [PS:ProtectionGroup] |
| Removes a FlashArray Host Group from a vCenter Cluster | Takes in a VMware cluster and then removes the corresponding hosts and host group                                                                                                                          | vCenter cluster [VC:ClusterCompute Resource]  
FlashArray [PS:FlashArrayConnection] | Host group name [string] |
| Restore VMFS Datastore from FlashArray snapshot | Takes in a VMFS datastore and restores it from a snapshot – the process will resignature the VMFS if needed. Default behavior is to unmount/detach/disconnect the datastore before refreshing. This process can be skipped if desired. | Datastore [VC:Datastore]  
Snapshot [PS:Snapshot]  
Skip unmount [boolean] | Volume [PS:Volume]  
Datastore [VC:Datastore]  
Snapshot [PS:Snapshot] |
RUNNING A WORKFLOW

Workflows can be run from the inventory screen (as shown in the above section) or from the workflow listing in the vRO inventory. All workflows require some level of input in order to run, a volume, a datastore, a FlashArray, a capacity, etc. For example, the create VMFS datastore workflow takes in the following information:

- **vCenter Cluster** – A VMware cluster to provision to
- **A FlashArray** – Which FlashArray should host the new volume
- **VMFS Name** – This will be the volume name and the VMFS label
- **Capacity** – A capacity for the volume
- **Unit** – Choose a capacity unit, either KB, MB, GB, TB (default), or PB

Some of these are text fields, some are inventory-related fields. If you have added a vCenter already, your clusters can be selected by drilling-down to the cluster objects.
The workflow will run when submitted and you can monitor the progress in the Schema tab of that workflow instance run.

EDITING A BUILT-IN WORKFLOW

Built-in workflows are not editable by vRO users – these can only be changed and updated by Pure Storage. The main reason behind this is to make support of these workflows much easier for both customers and Pure Storage. That being said you can indirectly edit them. By duplicating the workflows you can create a copy that can be edited as needed.
BUILDING CUSTOM ACTIONS AND WORKFLOWS

A lot of common workflows required by the average user are included by default in the plugin, but in some cases customizations may be required. These customizations may be small, but in other situations, entirely new functionality is required and this may dictate the need for new actions and/or workflows.

In this section, the process of building new actions, customizing existing workflows, and building new workflows will be discussed.

This section is for more advanced users of vRealize Orchestrator who are interested in custom design of workflows. At least an introductory understanding of scripting, in particular Javascript, is recommended. For full understanding of vRO operations refer to VMware documentation.

BUILDING A NEW ACTION

When new functionality is required, the first thought that should be examined is, what is the end result of this workflow? If the end result is one object, or one type of object, an action is probably preferable to a workflow. Furthermore, if this is a function that is likely to be needed by many other workflows, an action is generally a better choice.

This section will focus on building a new action.

The first step is creating a new folder for your action. In this case, I will create one under my own custom folder named `com.purestorage.flasharray.myactions`.

![Image of adding a new action](image)

Right-click that folder and then choose **Add action**...
Assign the new action a name. In this case, I want to obtain all of the timestamps of the snapshots of a certain datastore (or really the FlashArray volume that hosts it), so I will call it **getDatastoreSnapshotTimestamps**.

The first thing you need to decide is what inputs are required of this action – what objects do you want to send it? A volume? A vCenter datastore? A virtual machine? A snapshot? In this case, I want to take in a VMware datastore, so I will add one input, a VC:Datastore and call the input “datastore”. On the scripting tab, click to add a new variable.

Rename the input to something that makes sense and then click on the **Type** and change it to the appropriate type. In this case **VC:Datastore**.
Now you can start adding scripting. My action needs to do the following things:

1. Take in a datastore and figure out what FlashArray volume hosts it
2. Figure out the snapshots for that volume
3. Extract the timestamps of those snapshots

In order to get this information, you can build it yourself using the FlashArray plugin scripting objects and methods (documented in vRO itself or at http://www.vroapi.com). But often, there are already actions (or workflows) that do all or part of what you need, so it is usually easier to re-use those.

So let’s start with figuring out the FlashArray snapshots of a datastore. With built-in functionality of the plugin we have two options:

- Use the action `vmfsVolumeToFAVolume` to translate the datastore to a FlashArray volume and then use the volume method `getSnapshots()` to get the snapshots.
- There is also a built-in action to do both at once called `getFASnapshotsofVolume` which takes in a datastore and returns all of the snapshots of the corresponding FlashArray volume.

Either method is fine, but number two is easier, so let’s do that.

Navigate to this action in the left-hand pane and then you can right-click on it and choose Copy:

```javascript
getFASnapshotsofVolume(VC::Datastore datastore)
```

In the scripting windows, you can then right-click and choose Paste. This will automatically generate the Javascript that is required to call the workflow.
You will also note that it automatically puts in the “datastore” input as the input to the method. Which is exactly what this needs. In other situations, if the automatic choice is not correct, you can always change it.

Since this module returns an array (list) of snapshots (PS:Snapshot) and we need to store that object, we need to store it in a variable. We can do this by setting that action call we just pasted to a new variable. I will call it “snapshots”. Add a semi-colon to the end of the statement to complete it. So it will look like this:

```javascript
var snapshots = System.getModule("com.purestorage.flasharray.vmware.vcenter").getFASnapshotsOfVolume(datastore);
```

Now that we have all of the snapshots of that volume stored in a variable we can pull the information out that we need. This is an array of snapshots, and if you look at the FlashArray snapshot object listed section called Custom Objects, you can see the properties of a snapshot object. I want to return the timestamp of its creation (PS:Snapshot.created) and will also return the name (PS:Snapshot.name) to give context. Both properties are strings – so I will join them together. If the name is “mysnapshot” and the timestamp is “2016-12-21T16:13:02Z”, I will put the name first, followed by a colon, then the timestamp, so it would look like:

```text
mysnapshot: 2016-12-21T16:13:02Z
```

I will now write the Javascript to iterate through all of the snapshots and build my custom name/timestamp string for each one. I will then store each value in an array object I created called `snapTimestamps`. Finally I want to return the array of timestamp strings, so I will return that final variable out of the scripting object. If the volume has no snapshots I will just return text indicating there are no snapshots, skipping the iteration.

```javascript
var snapshots = System.getModule("com.purestorage.flasharray.vmware.vcenter").getFASnapshotsOfVolume(datastore);
var snapTimestamps = [];
if (snapshots.length > 0)
{
  for (i=0; i<snapshots.length; i++)
  {
    var snapTimestamp = snapshots[i].name + "":" + snapshots[i].created;
    snapTimestamps.push(snapTimestamp);
  }
  return snapTimestamps;
}
else
{
  return "This datastore has no snapshots";
}
```

The last part is to change the output (return type) to the appropriate type. This is an array of objects, and these objects are all string values, so the return type should be an array of strings.
The action is now complete.

Note that for simplicity of explanation, error/exception handling is not included in the script in this action. It is important, though, to include that in your scripts whenever possible. Error/exception handling explanation is beyond the scope of this document.

UNDERSTANDING WORKFLOWS

Workflows are essentially scripts that you can run. Workflows take in parameters (a name, a virtual machine, etc.) and then return a result (success or failure, or a virtual machine, or many virtual machines, etc.).

Inside of a workflow, just like a script, are multiple functions. Workflows have internal objects often referred to as tasks. These tasks can be many different things, they can be decisions, iteration logic, scripts, actions, or even other workflows.
VARIABLES

In order to provide the tasks with the information they need, the workflow can take the variables it has and pass them into and out of those tasks. There are three types of variables in a workflow:

- **Inputs** – A variable that a user enters into the workflow when they run it. This variable is a constant – it cannot be written to – it can only be read from by tasks.

- **Outputs** – What the workflow returns to the user. The object that was created for instance, information about an object, or a result. An output cannot be read – it can only be written to by tasks.

- **Attributes** – A variable that can be hard coded or empty when the workflow is first run. Can be passed in and out of objects in a workflow and its value can be changed in the process. This is read/write accessible to tasks. An attribute cannot be passed into, or out of, a workflow.

Once a workflow has inputs or outputs, for a task to be able to use them, you must explicitly pass them into (or out from) the task. Details on this process can be found in the next few sections.
SCRIPTABLE TASKS

Workflows, actions, and generic tasks are sometimes not specific enough to manipulate the data in the way that you might want. This is where the “scriptable task” comes in.

Scriptable tasks provide a scripting environment to make custom and detailed handling of objects inside of vRO. Scriptable tasks exclusively use Javascript, which provides direct access to built-in objects and methods from the various plugins.
CREATING A FLASHARRAY WORKFLOW

Occasionally users may want to build their own custom workflows to fit their use case. This can be easily done with the FlashArray plugin for vRealize Orchestrator. Let’s look at the situation where you want to create a FlashArray volume and also connect it to a host group.

First, create a workflow. I will call it “Create Volume and Connect to Host Group”. Very creative.

Once the workflow has been created, you can start building the logic. The first step is to include an action to create a volume. There are two ways to do this:

1. You can create a scriptable task and copy/paste the action into it.
2. Directly import the action into the workflow.

To see an example of the first option, please refer to Appendix I: Using an Action Inside of a Scriptable Task. For this example we will use option two.

1. Add the create volume action. Navigate to the Pure Storage action folder and drag and drop the `createNewVolume` action.
2. Click the Setup button that appears to configure the inputs and outputs of the action.

3. Accept the input defaults. You can rename things if you want, as shown below. When done, click Promote.

4. Now follow the same process for the `connectVolumeToHostGroup` action.

5. The variable setup is very similar. This action takes in a volume name, a host group name and FlashArray. The volume name and FlashArray connection inputs will default to the ones already added. vRO intelligently matches the input and outputs that are already there. If the “type” of the object (a string, a datastore, an array, etc.) is already there as an input or output it will default to that. You can of course separate them, but in this case the inputs for the create volume (the FlashArray
and volume name) and the add to host workflow (FlashArray and volume name) are supposed to be the same, so they can be re-used as the default. The only inputs/outputs that are new are the result of adding to the host group and the host group name. This is denoted by the green plus sign. When done, click promote.

6. The workflow will now completely function without any additional work!
CREATING A FLASHARRAY AND VMWARE WORKFLOW

The Pure Storage FlashArray Plugin for vRealize Orchestrator includes larger workflows that provide VMware-integrated operations out of the box. The following workflows are included automatically:

These workflows run FlashArray operations, but from a VMware (VMFS) context. Instead of inputting a FlashArray volume, you can give it a VMFS datastore, and the workflow will perform the FlashArray operation (like creating a snapshot) on the volume that hosts it. The workflow figures out the translation (which FlashArray hosts the VMFS datastore and which volume is it on). The large majority of the common workflows you might do are already built in. But how do you create your own?

For example, maybe you want to create a simple workflow that renames the VMFS but also renames the FlashArray volume. How would that be done? The plugin includes some helper actions that can achieve this:

Let’s walk through building this workflow.

1. Create a new workflow and add the `vmfsVolumeToFAVolume` action. Click Setup to configure the variables. Change the variable names if you want something more descriptive. Below the output has been changed from `actionResult` (which is the identified FlashArray volume) to `flasharrayVolume`. Also change the output type from an output to a local variable (frequently referred to as an attribute). This will allow the resulting volume to be passed to another action or object inside of the workflow.
2. Now import the `renameVolume` action.

3. In the variable setup for this action, change the `volumeObject` input to the local variable from before (flasharrayVolume), the returned volume object. This will take the returned object of the previous action, `vmfsVolumeToFAVolume`, and send it to the rename action.
4. Now, we need to create an object to rename the VMFS datastore. There is no built-in workflow, so we need to create a new one. Drag in a **Scriptable task** object from the inventory section labeled **Generic**. Drag it anywhere onto the workflow path.

5. Now edit the scriptable object. Rename it to something that makes sense, like “Rename VMFS” on the **Info** tab.

6. Add a new input that is a datastore on the **IN** tab when the edit screen appears.

7. Click the connector button to bind a new variable to this object. We need to “bring in” the datastore from the overall workflow to this new object.
8. Click on the datastore (type VC:Datastore) object. Also choose the `newVolumeName` (type string). This will associate the datastore to be renamed and the new name with the custom scripting object.

![Scriptable task](image)

9. In the **Scripting** tab add the following Javascript to rename the volume:

```javascript
datastore.rename_Task(newVolumeName);
```

![VMware vRealize Orchestrator](image)

10. Now save and run the workflow. It will now rename both objects!
WORKFLOW PRESENTATION LAYER

Once the logic has been completed for a workflow, the presentation layer can be customized. The presentation layer controls how the user interacts with configuring inputs when running a workflow. This allows the designer to control what is entered by the user and also to assist them when choosing options.

There are some basic controls, advanced, dynamic inputs, and also methods to validate inputs before running a workflow.

BASIC INPUT CONTROL

For the majority of inputs, basic controls will suffice. In the presentation layer, you can view all of your workflows inputs.

If you click on a variable you can control how the input for that variable is presented to the user. If you click on the property addition symbol (the triangle with a plus sign), you can add controls.
If you make the input mandatory, the user will not be able to execute the workflow until that input is entered.

When the workflow is run, that input will have a red asterisk next to it and the submit button will be gray until it is populated.

Furthermore, you can force certain selections of inputs by hard-coding some choices. If I want to enforce a LUN ID to be one of a few numbers, I can add them into the “Pre-defined answers” option.
When run, they will get a drop-down of these choices.

This is just a sub-set of what can be done with the basic input presentation controls. Refer to VMware documentation for more information. But what about if you want something that isn’t static? Choices that depend on previous inputs the user has entered? Dynamic inputs is the option for that.

**DYNAMIC INPUTS**

Dynamic inputs allow you to provide choices for an input based on some other variable that the user has entered. Let’s take the example of restoring a datastore from a snapshot. When the user enters a datastore, I want the snapshot restore point to only list snapshots that belong to the FlashArray volume hosting that datastore.

This can be achieved by linking an action to an input. When the appropriate field is filled out, an action will run with that input (in this case a datastore) and return an array of objects that can be selected from.

In the presentation tab, choose the snapshot input and add a property for “Predefined list of elements”.

---

©2017 Pure Storage, Inc. 55
Click on the blue puzzle piece icon to link an action.

A list of possible actions will appear – the return value of the action must be an array of the input you are configuring, in this case a snapshot. vRO will filter the actions that are compatible.

The workflow, getFASnapshotsofVolume, takes in a datastore and returns the respective snapshots. So I will choose that one. When selected, you must tell it in the line entry which input it should act upon. The input must match the input of the action – in this case the datastore input.
When **Apply** is clicked, the process is complete.

When the wizard is run and the datastore is chosen, the correct snapshots will appear as options.

**CUSTOM VALIDATION**

Another option for using actions in the presentation layer of a workflow is via custom validation. Custom validation allows the workflow to check and verify input as soon as the input is entered and before the workflow run is actually submitted. This will greatly increase the rate of successfully run workflows. Examples of custom validation could be:

1. Checking that an input size is in a valid range
2. Checking to see that a name is not in-use
3. Seeing if an object has a required relationship
Let’s look at an example of number two – creating a new volume and checking first that the name is unique.

The important thing to understand is how vRO custom validation decides if the input is invalid or valid. If, when run, the action returns null, the validation succeeds. If the action returns anything else, the input is considered invalid and the validation fails. Ideally, you want to return a string in the case of an error. If you return a string, the string will be interpreted as an error message and will be surfaced in the wizard. You can technically return anything else but it will not be particularly informative.

Example of returning a string:

Example of returning an object:

It is apparent that returning a string for a failure is much more helpful to the requestor.

So back to our action. I want to create a new volume, but before I complete the volume creation wizard to start the workflow, I want to make sure that the name is not in use by another volume on that FlashArray.

In this case, I have a simple action to use for checking this. This action takes in a FlashArray connection and a name and looks through all of its volumes to see if that name is in use.
I have a workflow called “Create FlashArray Volume”. This workflow takes in a FlashArray connection, a name, and a size.

The first step is to choose the input in the workflow you want to validate and configure its presentation layer. Add a property to the name input and choose Custom validation.
Then configure the validation by clicking on the puzzle piece icon.

Type in the name of your action in the search box and select it – in this case my action is called `checkVolumeUniqueName`.

Then you need to link the inputs of the workflow to the action used in the validation. This is in the below box.

It lists all of the required inputs of the action. You need to associate inputs of the workflow to each mandatory input of the action – they must be the same type.

In this case, there are two action inputs, a name (a string) and a FlashArray connection. The name input is the input you are validating, so enter `#___current` (note that is two underscores preceding “current”) into the box for “name”. Since you are not hardcoding the name you should change the input type from static to OGNL in the drop down.

The other input `flashArrayConnection` needs to be linked to a different input of the initial workflow to bring in the FlashArray connection. You can either type in the variable name with a preceding `#` or click the pencil icon and select the proper one.
Now that the validation is complete I can run the workflow and if I enter a name that is in-use there immediately is an error:

If I make it unique, the error goes away:
TROUBLESHOOTING

LOGS
You must collect the following logs in case you need to contact the Pure Storage support team:
1. Workflow Execution Logs (Include both Information logs and Debug logs)
2. Server.log

HOW TO COLLECT LOGS
Collect the logs that you need to send to the support team as follows:

WORKFLOW EXECUTION LOGS

INFO LOGS
1. From the Logs tab of the Workflow execution, select Info in the drop-down list.
2. Select All the messages and send it as Info logs.

DEBUG LOGS
1. From the Logs tab of the Workflow execution, select Debug in the drop-down list.
2. Select All the messages and send it as Debug logs.

SERVER.LOG
1. Go to the Orchestrator Control Center of vRO server.
2. From the Monitor and Control section, click on the File System Browser.
3. From the File System Browser, click on the app-server-logs.
4. From app-server-logs, download the server.log file and send it as server.log.
ADVANCED TECHNIQUES

USING AN ACTION INSIDE OF A SCRIPTABLE TASK

To execute an action into a scriptable task you need first to add a scriptable task into the workflow. Generally it is preferred to just add the action directly into your workflow as its own object, to make logic easier to follow, but occasionally actions are used so frequently or the action is simple enough that it’s better to make a direct call inside of a scriptable task.

This can be done by finding the task in the left-hand pane and dragging and dropping it onto the appropriate place in the schema.

Next, edit the task and go to the scripting tab. Navigate to the action you want to use or use the search function to auto-locate it by name. Right-click on the action and choose copy.

Then right-click in the scripting pane and choose paste.
The line to call the action will be added and it will include suggested variable names that need to be passed into the action. In this case a volume to snapshot (PS:Volume object variable called volume) and a snapshot suffix (string variable called suffix). I have two inputs in this workflow, one for the volume which is called volume and another for the suffix which is called snapSuffix:

<table>
<thead>
<tr>
<th>IN</th>
<th>OUT</th>
<th>Exception</th>
<th>Visual Binding</th>
<th>Scripting</th>
</tr>
</thead>
<tbody>
<tr>
<td>volume</td>
<td>volume</td>
<td>[in-parameter]</td>
<td>PS:Volume</td>
<td></td>
</tr>
<tr>
<td>snapSuffix</td>
<td>snapSuffix</td>
<td>[in-parameter]</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>

The input for the volume is right so it will stay named “volume”. The other I will rename to match my variable “snapSuffix”.

The action returns a snapshot object, so if I want to reuse that object, I can set the whole operation to store the result (a snapshot) in a variable I will call snapshot.

```javascript
var snapshot = System.getModule("com.purestorage.flasharray.snapshot").createSnapshotFromSingleVolume(volume, snapSuffix);
```

**USING SCRIPTING OBJECTS AND METHODS**

The FlashArray plugin for vRealize Orchestrator offers custom objects and methods that provide functionality above and beyond what the built-in actions and workflows can do. These are used inside of scripting objects in a workflow or inside of an action through JavaScript. This section will overview how to use these objects and their methods.

**AVAILABLE OBJECTS AND METHODS**

The available list of scripting objects and their methods can be found at the following URL:

http://www.vroapi.com/Plugin/PS/1.0.0

Additionally, documentation of these can be found inside of vRO itself. When in a scripting window, there is an object explorer to help find workflows and actions, but it can also show scripting objects and their methods. The Pure Storage one can be found under PS:
When expanded you can see the available objects, scripting objects, and methods. Objects were already defined earlier – these can be variables that can be passed into and out of a workflow. These objects are based on scripting objects, which is essentially a custom variable that has certain property names and types that are pre-defined. These scripting objects can be instantiated and their properties populated. Each object also has methods that can be run against it to perform a certain operation. The gray items are regular objects and the green ones are scripting objects. Scripting objects may have properties and methods, some just have methods, and some just have properties.

When you expand a scripting object, you will see the properties of it, as well as the methods. The methods describe what inputs they require when invoked, and also what the methods return when successfully run.
USING SCRIPTING OBJECT METHODS

In this section, I will use scripting objects and methods to update a protection group. Specifically, I will be updating a protection group replication schedule to replicate once an hour instead of every four hours. The current protection group looks like this:

The vRO plugin does not have a built-in action to change this, so you need to use the vRO plugin scripting actions and methods to do so. The PS:ProtectionGroup object refers to the ProtectionGroup scripting object. That object has a variety of methods, one is called update. So if I have a protection group object stored in a scripting object called pgroup, I can run the method on it.
The `update` method takes in another type of scripting object called `PSProtectionGroupRequest`. This object has the following properties:
I need to create the `PSProtectionGroupRequest` before I can update the protection group. This can be done by copying and pasting the scripting object into the scripting window:

My workflow has the following inputs:

I am asking the user for a replication interval in either minutes, hours or days – depending on the input I need to enter the proper information. The replication interval is taken in as seconds, so I need to convert what they are entering into seconds. If they choose “hours” I will multiply that number by 3600 to convert to seconds, and so on. This is achieved through a switch statement.
You can click on the property to find more information:

```plaintext
PSProtectionGroupRequest
  - action
  - addreplicationset
  - addhostset
  - addtargetset
  - addwindowset
  - all_lfr
  - allowed
  - days
  - flipgroupset
  - hostset
  - name
  - per_day
  - replicationset
  - replicationgroupset
  - replicationhostset
  - replicationtargetset
  - replicate
  - replicate_all
  - replicate_interval
  - replicate_state
  - replicate_threshold
  - replicate_windowset
  - remove
  - removeall
  - removehostset
  - removetargetset
  - replicate
  - replicate_all
  - replicate_interval
  - replicate_state
  - replicate_threshold
  - replicate_windowset
  - remove
  - removeall
  - removehostset
  - removetargetset
  - replicate
  - replicate_all
  - replicate_interval
  - replicate_state
  - replicate_threshold
  - replicate_windowset
  - remove
  - removeall
  - removehostset
  - removetargetset
  - replicate
  - replicate_all
  - replicate_interval
  - replicate_state
  - replicate_threshold
  - replicate_windowset
  - remove
  - removeall
  - removehostset
  - removetargetset
  - replicate
  - replicate_all
  - replicate_interval
  - replicate_state
  - replicate_threshold
  - replicate_windowset

Attribute: replicate_frequency

Description
Host Update Request replicate_frequency

Return Type: Integer
```

You can also lookup that value in the FlashArray REST API guide for more information as well.

The next part is to add the replication interval as the value in my scripting object (which is now just empty). So I set the scripting object replication_interval property equal to my identified new value:

```plaintext
myPSProtectionGroupRequest.replicate_frequency = replicationInterval;
```

The last part is actually executing the update. I can run the `update` method against my protection group object by passing in my scripting object with the new value:

```plaintext
pgroup.update(myPSProtectionGroupRequest);
```
My full script is below:

```java
var myPSProtectionGroupRequest = new PSProtectionGroupRequest();
switch (replicationUnit) {
    case "days":
        replicationInterval = replicationInterval*86400;
        break;
    case "hours":
        replicationInterval = replicationInterval*3600;
        break;
    case "minutes":
        replicationInterval = replicationInterval*60;
        break;
}
myPSProtectionGroupRequest.replicate frequency = replicationInterval;
agroup.update (myPSProtectionGroupRequest);
```

Now when I run my workflow, I can change it to 2 hours:

It is now changed:

**RUNNING DIRECT REST API CALLS TO A FLASHARRAY**

In some situations, it may be necessary to make direct REST API calls to a FlashArray. This is necessitated when the built-in workflows, actions, or scripting objects and their respective methods do not provide the functionality you need, like if a new feature has been released on the FlashArray subsequent to the latest vRO plugin, that introduces new respective REST API calls.

In order to use newer features, or something custom that happens to not be included, you can leverage the built-in REST functionality inside of the FlashArray vRO plugin to do anything that you need.
There are two options for this:
1. REST API Workflow
2. REST API Action

Both methods leverage the same underlying process – the workflow encapsulates the action, the action encapsulates a scripting object method. Whichever you use is entirely a matter of preference and personal familiarity.

**REST API REQUIRED INPUTS**

Regardless of the method you use (workflow or action) both require a few inputs to run:

- **FlashArray target**: The FlashArray to which the REST API call should be directed
- **Uniform Resource Identifier (URI) Suffix**: The URI defines the type of object, and often the specific object to act upon and is the HTTP URL that is used to make the call
- **HTTP Method**: This is the type of operation being performed; **GET**, **POST**, **PUT**, or **DELETE**
- **JSON Body**: This is additional information often included in **POST**, **PUT**, or **DELETE** REST calls

**FlashArray Target**

Any REST call should be targeted at a specific FlashArray – generally this is dictated in the URI (discussed in the next sub-section) but in vRO, a FlashArray object must be chosen from the inventory.

*Please note that what is required here is not a FlashArray Connection (PS:FlashArrayConnection) object, like many workflows, but instead a FlashArray (PS:FlashArray) object.*

**Uniform Resource Identifier (URI) Suffix**

A standard REST call includes a full URI, which has a FlashArray IP/FQDN, an API version, and then the information for the specific REST call like below:

https://pure-001.example.com/api/1.7/array?space=true

When using the vRO REST feature in the FlashArray plugin, all that is needed is the string after the REST API version. So instead of the full URI above, you just enter:

/array?space=true

Yes, include the preceding forward slash. In this you put object names, types, filters, query parameters, etc.
For more information on building URI’s please refer the FlashArray REST API documentation.

**HTTP Method**

The third input is the HTTP method. While there are many types of HTTP method the following are the ones supported by the FlashArray:

- **GET**: List resources
- **POST**: Create a resource
- **PUT**: Modify a resource
- **DELETE**: Delete a resource.

**JSON Body**

For most POST, PUT, and DELETE operations, it is required to also submit a body with certain parameters populated. The body is formatted using JavaScript Object Notation (JSON). More info on JSON can be found here: [http://www.json.org/](http://www.json.org/).

There are a variety of different data types used in JSON. The following are the most common.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>A parameter that is either true or false</td>
<td></td>
</tr>
<tr>
<td>object</td>
<td>A special parameter with custom variable names and values</td>
<td></td>
</tr>
<tr>
<td>list</td>
<td>A variable with multiple values</td>
<td></td>
</tr>
<tr>
<td>number</td>
<td>A variable that is a number only</td>
<td></td>
</tr>
<tr>
<td>string</td>
<td>A variable that can be a series of characters</td>
<td></td>
</tr>
</tbody>
</table>

Note that it is very important to pay close attention to the placement and proper use of the special characters (brackets, colons, double quotes, etc.) or the JSON will be invalid and REST calls will fail or create unintended results.

An example of a JSON body for a REST call to create a snapshot of two different volumes respectively named myvolume and yourvolume is below:

```json
{
    "snap": true,
    "source": ["myvolume","yourvolume" ]
}
```
**USING THE REST API WORKFLOW**

For this example, we will use the PUT operation that enables an alert recipient on the FlashArray. Details can be found in your REST API guide for the FlashArray under alert management.

The Run REST API workflow is in the FlashArray REST APIs folder. To execute the workflow, right click and click Run workflow.

In the first screen, enter the FlashArray object you want to run the REST API against.

Then click Next.

Enter your resource URI, HTTP Method, and a Body in JSON if applicable. In this case the alert recipient I want to enable is the email address vroadmin@purestorage.com.
The URI suffix would be:
/alert/vroadmin@purestorage.com

The method is **PUT**. The request body would be:

```
{ "enabled": true }
```

Click submit to run the workflow. The workflow completes and the now enabled status of that recipient is returned:

The return result is stored as a string inside of the output named `actionResult`. The string is formatted in the form of JSON. Refer to the section, **Parsing the REST API Response with JavaScript** for details on how to manage that output if it is passed into a JavaScript-based scriptable task.
USING THE REST API ACTION

Using the REST API action by including the action as an object directly in a workflow (the process of doing so is shown in the section Creating a FlashArray Workflow) is no different in concept from the previous section on running the REST API workflow – since the workflow is only the REST API action by itself inside of a workflow. Include the action into a workflow and then follow the steps from the previous section.

If you are calling the action directly from JavaScript like the method described in the section Using an Action Inside of a Scriptable Task, then follow these steps.

In this example, we will use the same situation as described in the previous section – enabling an email address as a FlashArray alert recipient. What the workflow will do is take in a FlashArray and an alert recipient (an email address) and then enable that recipient. The workflow is already built with two inputs:

The workflow has a single scriptable task where the REST API action will be called directly via JavaScript. Both inputs are passed into the scriptable task.

The action takes in four inputs:
- FlashArray
- URI
- HTTP Method
- JSON Body

When the action is called in the JavaScript directly it suggests some variable names (flashArray, httpMethod, body, and uri).

```
System.getModule("com.purestorage.flasharray.restapi").runAPI(flashArray, httpMethod, body, uri)
```

Since I used the variable name “FlashArray” instead of the suggested “flashArray” I will change the suggested to match mine (JavaScript is case sensitive). I will re-use the other variable names.
When the variable matches an input name, it will change to the pink color as seen above.

Now I need to choose my HTTP method, which is PUT. So I will declare the httpMethod variable that was suggested (any variable needs to be declared if it is not directly passed into the JavaScript from an input) and set it equal to PUT.

```
var httpMethod = "PUT";
```

Now I will build my URI. All I need is the following to be sent to the action:

```
/alert/vroadmin@purestorage.com
```

As the alert recipient I want to enable is vroadmin@purestorage.com. I can do this by also declaring the uri variable and building this string through concatenation:

```
var uri = "/alert/" + alertRecipient;
```

Finally, I will need to build the body. The body needs to look like this:

```
{ "enabled": true }
```

There are a couple ways to create this. You could literally set the body equal to that string – this would be the simplest option. Though at scale doing this can be tedious and error prone – making it easy to mess up syntax. I recommend building the data types separately as their own variables first. Then using the `JSON.stringify()` method to build the proper syntax.

Using the `JSON.stringify()` method, you can build the variety of properties in the body request using standard JavaScript variable types (strings, booleans, arrays, objects) and then have them automatically formatted with proper JSON.

In this case I only have one boolean property called enabled that I need to set to true. So first I will declare a new object variable:

```
var enabledVar = new Object();
```

Then add the boolean value of false to a new property called enabled:

```
enabledVar.enabled = true;
```

Then “stringify” it and store it in a newly declared variable (as suggested by the action) called body which will store the information for the body request as a JSON string.

```
var body = JSON.stringify(enabledVar);
```

Lastly, I will call the action using my variables (FlashArray, httpMethod, body, and uri):

```
System.getModule("com.purestorage.flasharray.restapi").runAPI(FlashArray, httpMethod, body, uri);
```
The action returns a string object in the form of JSON, so if you want to use or log the response, declare a new variable and set the action equal to it:

```javascript
var restResponse = System.getModule("com.purestorage.flasharray_restapi").runAPI("FlashArray","httpMethod","body","uri");
```

My whole script looks like:

```javascript
var httpMethod = "PUT";
var uri = "/alert/" + alertRecipient;
var enabledVar = new Object();
enabledVar.enabled = true;
var body = JSON.stringify(enabledVar);
var restResponse = System.getModule("com.purestorage.flasharray_restapi").runAPI("FlashArray","httpMethod","body","uri");
```

If desired you can log out the response, or pass it to another part of your workflow.

The returned JSON string can be easily converted back to JavaScript standard objects for easier reference and indexing. The `JSON.parse()` method can be run on the object and the result stored into a new variable. This will return an object, array, etc. depending on the REST result.

Save it, and then run it:
APPENDIX I: ENABLING TLS 1.1 AND 1.2

The FlashArray Purity Operating Environment requires a more secure form of communication than vRealize 6.0.2 and earlier use by default. While this is fixed in vRO 6.0.3 and later, a change needs to be made to vRO 6.0.2 and earlier to get the plugin communication to the FlashArray to function. The FlashArray does not support TLS versions earlier than 1.1.

**LINUX APPLIANCE**

SSH into vRO and VI into /var/lib/vco/app-server/bin/setenv.sh

Using VI, edit the line starting with `JVM_OPTS=`. Add at the start (after the double quote) the following:

```
-Dhttps.protocols=TLSv1,TLSv1.1,TLSv1.2
```

Then a space before the next value. It should look like this:

![Screenshot of Linux VI editor](image1)

Save it and then reboot your vRO appliance.

**WINDOWS**

If you are running the Windows version of vRO it is a little different. You need to edit two files:

1. Open the file with notepad "C:\Program Files\VMware\Orchestrator\app-server\bin\wrapper.conf"
2. Add the following line:

```
java.additional.21="-Dhttps.protocols=TLSv1,TLSv1.1,TLSv1.2"
```

![Screenshot of Windows notepad](image2)
3. Add the exact same line to the file named "wrapper-auto.conf" in the same directory.
4. Reboot the vRO Windows machine.
ABOUT THE AUTHOR

Cody Hosterman is the Technical Director for VMware Solutions at Pure Storage. His primary responsibility is overseeing, testing, designing, documenting, and demonstrating VMware-based integration with the Pure Storage FlashArray platform. Cody has been with Pure Storage since 2014 and has been working in vendor enterprise storage/VMware integration roles since 2008.

Cody graduated from the Pennsylvania State University with a bachelor’s degree in Information Sciences & Technology in 2008. Special areas of focus include core ESXi storage, vRealize (Orchestrator, Automation and Log Insight), Site Recovery Manager and PowerCLI. Cody has been named a VMware vExpert from 2013 through 2016.

Blog: www.codyhosterman.com
Twitter: www.twitter.com/codyhosterman
YouTube: https://www.youtube.com/codyhosterman

© 2017 Pure Storage, Inc. All rights reserved. Pure Storage and the “P” Logo are trademarks or registered trademarks of Pure Storage, Inc. in the U.S. and other countries. VMware, Orchestrator, vSphere, vCenter, and ESXi are registered trademarks of VMware in the U.S. and other countries. Microsoft Windows is a registered trademark of Microsoft in the U.S. and other countries. Linux is a registered trademark of Linux Foundation in the U.S. and other countries. The Pure Storage product described in this documentation is distributed under a license agreement and may be used only in accordance with the terms of the agreement. The license agreement restricts its use, copying, distribution, decompilation, and reverse engineering. No part of this documentation may be reproduced in any form by any means without prior written authorization from Pure Storage, Inc. and its licensors, if any.

THE DOCUMENTATION IS PROVIDED “AS IS” AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID. PURE STORAGE SHALL NOT BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH THE FURNISHING, PERFORMANCE, OR USE OF THIS DOCUMENTATION. THE INFORMATION CONTAINED IN THIS DOCUMENTATION IS SUBJECT TO CHANGE WITHOUT NOTICE.