HP Operations Orchestration

For the Windows operating system

Software Version: Content Pack 8

PowerShell Wizard Guide

Document Release Date: April 2012

Software Release Date: April 2012

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe™ is a trademark of Adobe Systems Incorporated.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

This product includes an interface of the 'zlib' general purpose compression library, which is Copyright © 1995-2002 Jean-loup Gailly and Mark Adler.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

Chapter 1

Introduction

This chapter includes:

Purpose

This integration enables users to generate OO flows from the selected PowerShell cmdlets found in a list of modules/snapins.

Its main advantages are:

- **Automation.** Avoid having to repeat the same time-consuming process of creating flows which execute PowerShell cmdlets. Perform the following steps as an alternative to using the PowerShell Wizard (multiply by the number of cmdlets):
  a. Create an empty flow
  b. Drag and drop the PowerShell Script operation
  c. Search for the cmdlet description
  d. Set the required input values
  e. Add the cmdlet parameters as flow and step inputs (some cmdlets may have numerous parameters)
  f. Set the description of the flow. The step inherits its description from the PowerShell Script operation, but this is not available for the flow.
- **Authoring ease.** The description of each flow contains the default description of the corresponding cmdlet which it executes. Therefore, the user is not forced to open the cmdlet description in a browser and switch between OO and the Internet.
- **Module and cmdlet discovery.** The wizard discovers the available modules and cmdlets from a target host.

Supported Versions

Operations Orchestration Version	PowerShell Wizard Version
OO Content Pack 8	01.00.01

Chapter 2

Getting Started with the PowerShell Wizard Integration

This chapter includes:

PowerShell Wizard Installer Overview

The PowerShell Wizard Installer installs the PowerShell Wizard (pswizard.exe), which allows you to create OO flows based on the PowerShell cmdlets discovered in modules and snapins from a host that you specify when you run the wizard.

Downloading OO Releases and Documents on HP Live Network

HP Live Network provides an Operations Orchestration Community page where you can find and download supported releases of OO and associated documents.

To download OO releases and documents, visit the following site:

https://hpln.hp.com/

Note: This site requires that you register for an HP Passport and sign-in.

To register for an HP Passport ID:

1. Go to: http://h20229.www2.hp.com/passport-registration.html

   Or

   Click the **New users - please register** link on the HP Passport login page.

2. On the HP Live Network page, click **Operations Orchestration Community**. The Operations Orchestration Community page contains links to announcements, discussions, downloads, documentation, help, and support.
3. On the left-hand side, click **Operations Orchestration Content Packs**.
4. In the **Operations Orchestration Content Packs** box, click **Content**. The HP Passport and sign-in page appears.
5. Enter your user ID and Password to access to continue.
6. Click **HP Operations Orchestration 9.00**.
7. Search for the required HP Operations Orchestration Content Pack.

Installing the PowerShell Wizard

Follow the PowerShell Wizard instructions. After the PowerShell Wizard is installed, a new folder is created in the OO Home folder in **Studio/tools/**. This folder contains the following:

- **pswizard.exe**. The HP OO PowerShell Wizard application executable.
- **pswizard/lib/ folder.** Contains the pswizard.exe jar dependencies.
- **pswizard/dotnet/ folder.** Contains the pswizard.exe dll dependencies.
- **conf/ folder**. Contains logging configuration and user interface message files.
- **OO Home folder/pswizard/uninst.** Contains the uninstall icon and other files.

The PowerShell Wizard is available from the **Start Menu** folder if you navigate to **Hewlett-Packard/Operations Orchestration/Wizards** where you can find icons for running the application or uninstalling it.

Note: On Windows 2008 and Windows 2008 R2 servers, you must have administrative privileges to install the PowerShell Wizard.

To apply administrative privileges:

1. On the Start menu, select **All Programs > Hewlett-Packard > Operations Orchestration > Wizards**.
2. Right-click on **PowerShell Wizard for HP Operations Orchestration**, and then select **Run as administrator**.

You now have administrative (elevated) privileges and can successfully run the PowerShell Wizard installer.

PowerShell Wizard Dependencies

When you run the PowerShell Wizard, it starts a new javaw process and searches for library dependencies in the order defined in the classpath:

pswizard\lib*;

lib*;

..\..\Studio\tools\lib*;

..\..\Studio\tools\thirdparty*;

..\..\Studio\tools\conf (The Wizard SDK searches this conf folder for certain files.)

Besides these dependencies, the wizard uses JNI to invoke C# code. The assembly dependencies loaded at runtime can be found in the **pswizard\dotnet** folder.

Configure Logging Settings

Once the installation succeeds, you will find the new **files psw.properties** and **psw.log4j.properties** in the OO home directory, in the **Studio/tools/conf/** folder. These files allow you to configure basic logging settings for the PowerShell Wizard. When you run the PowerShell Wizard, logging information is written to the **PowerShellWizard.log** file which can be found in the OO Home folder in **Studio/tools/**.

Uninstall the PowerShell Wizard

Before uninstalling the PowerShell Wizard, make sure you back up your installation and repository. For information on backing up HP Central and Studio, see the OO Administrator's Guide.

Note: Uninstalling the PowerShell Wizard deletes all of the resources, files, and folders created when the PowerShell Wizard was installed. However, uninstalling does not delete the **Studio/tools/** subfolder in the OO home directory if a log file was created there or if the PowerShell Wizard created a new repository in this folder. This happens when you run the wizard without providing an absolute repository path.

To uninstall the PowerShell Wizard:

1. Make sure that the PowerShell Wizard is closed.
2. Open Control Panel and click **Add/Remove Programs**.
3. Scroll down to and highlight **HP Operations Orchestration PowerShell Wizard <version_number>** and click **Remove**:
4. When you are prompted to confirm whether you want to remove the PowerShell Wizard and its components, click **Yes**:
   - The Uninstall Status box appears, in which progress of the removal is tracked in a progress bar. When the PowerShell Wizard is completely removed, a message box informs you that the uninstall process is complete
   - If you have any relevant folders open, the message box may tell you that some components could not be removed. This can be ignored
5. Click **OK.**

PowerShell Wizard Requirements

RAS Host

- PowerShell
- HP Operations Orchestration Studio 9.00 (x32 or x64).
- Content repository which contains the PowerShell Script operation (UUID f0b2afd2-5733-47e4-80ba-7f2387cc66d5).

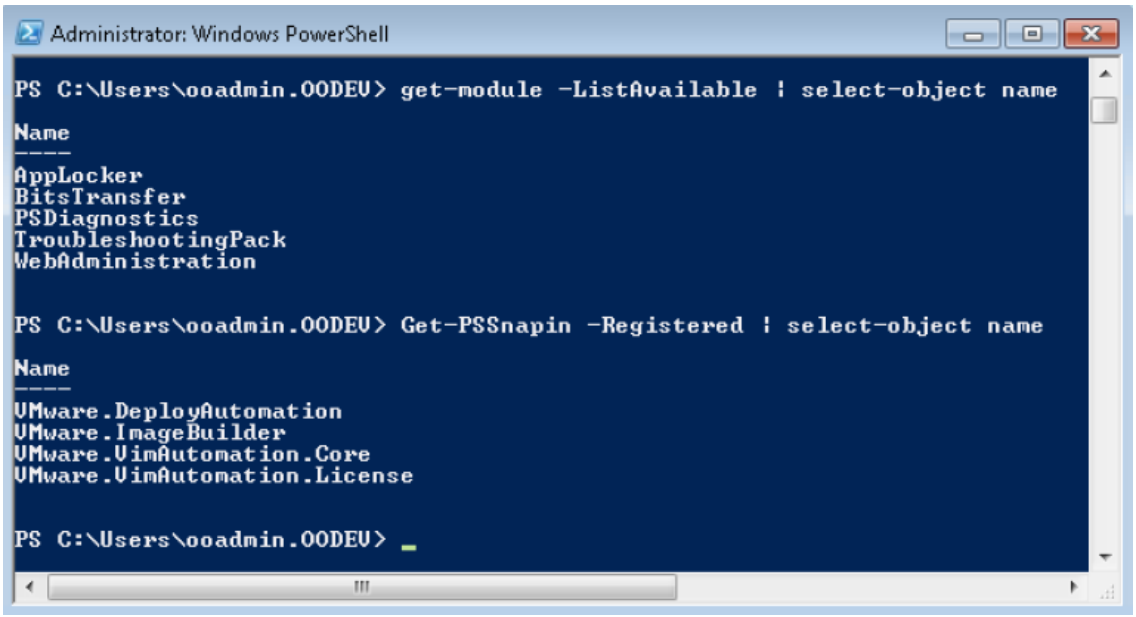
Note:

- The wizard requires an existing environment for products providing PowerShell cmdlets, but it does not require any additional installations on the RAS host.
- If you provide the Central's **rcrepo** as the content repository when running the PowerShell Wizard under a domain account, you may receive an error message stating the repository is locked. This is due to insufficient administrative privileges.

Target Host

- PowerShell with remoting enabled.
- For the products providing PowerShell cmdlets, the target host must have the modules and snapins available. Run one of the following cmdlets to list the required modules and snapins. Otherwise, it means that the host does not have the cmdlets provided for that product.

  Following is an example of how to list the modules and snapins in the PowerShell console.



Chapter 3

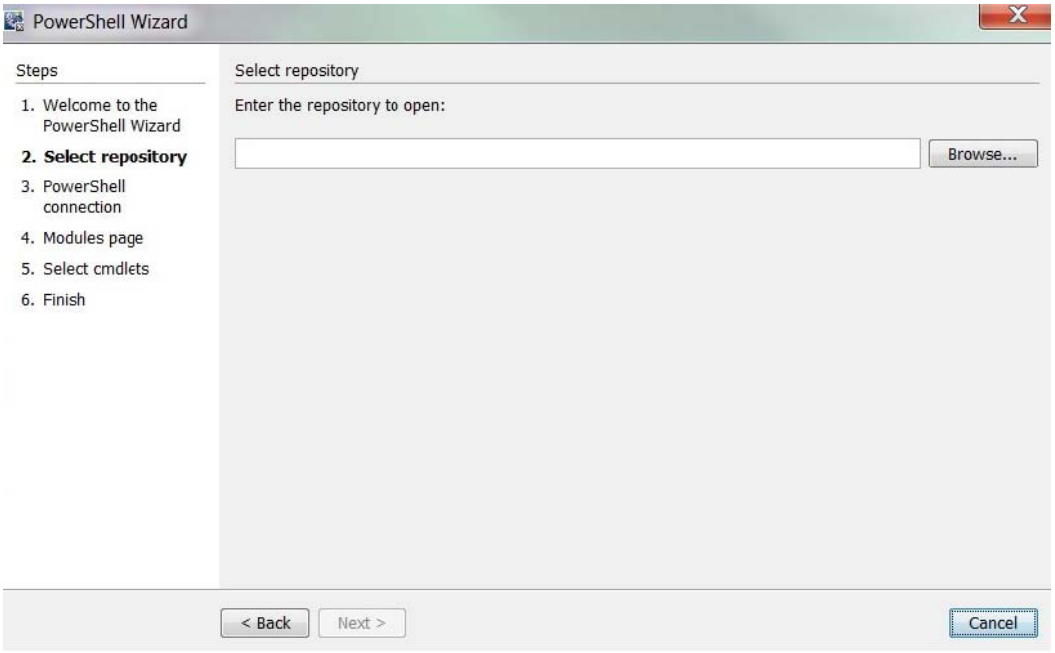
Using the PowerShell Wizard

The following steps describes how to use the PowerShell Wizard.

Step 1 — View the Welcome page

The Welcome page contains a short summary of the wizard.

Step 2 — Select the Required Repository

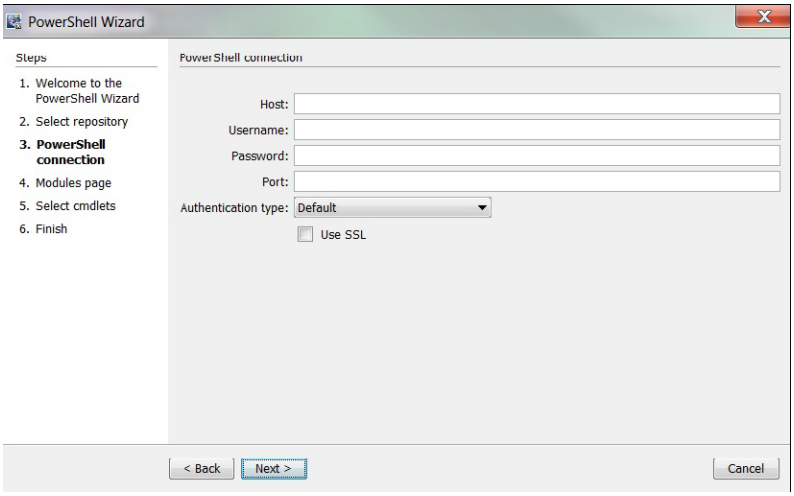


In the **Enter the repository to open** box, type the repository path or click **Browse** to locate the required repository.

Following are the repository validation conditions:

- Studio or any other programs should not hold any locks on the repository, that is, it should not be opened in Studio or anywhere else.
- The repository the must contain the PowerShell Script operation **UUID f0b2afd2-5733-47e4-80ba-7f2387cc66d5**.
- The selected repository must be compatible with HP Operations Orchestration 9.00.

Step 3 — Define the PowerShell Connection



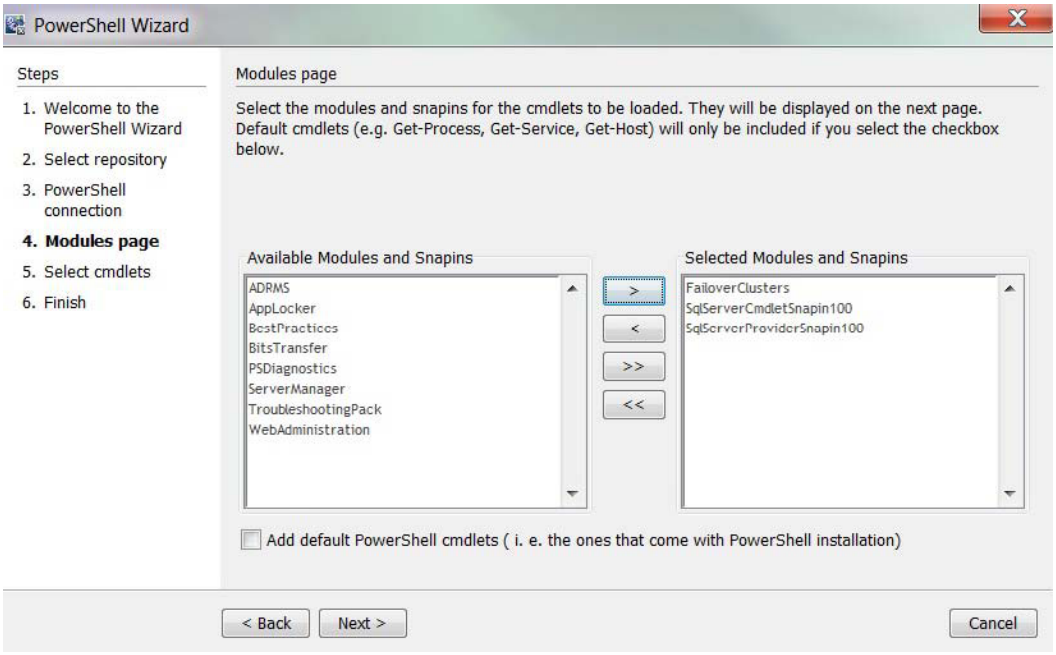
- In the **Host** box, type the name of the host to which you want to connect. If you leave the **Host** field empty, the PowerShell Wizard uses localhost as the default.

  Note: If **Host** is empty, then the authentication type is **NegotiateWithImplicitCredential**. If the host has been defined, the wizard considers the host definition provided by the user.

- In the **Username** box, enter the required user name.
- In the **Password** box, enter the required password.
- In the **Port** box, define the port value in the range of 1-65535.

  Note: If you set the port value to 0, the wizard ignores it and uses the default port values. The default port values are: **5985 (HTTP)** and **5986 (HTTPS)**. The Port and Use SSL inputs are ignored for **localhost**.

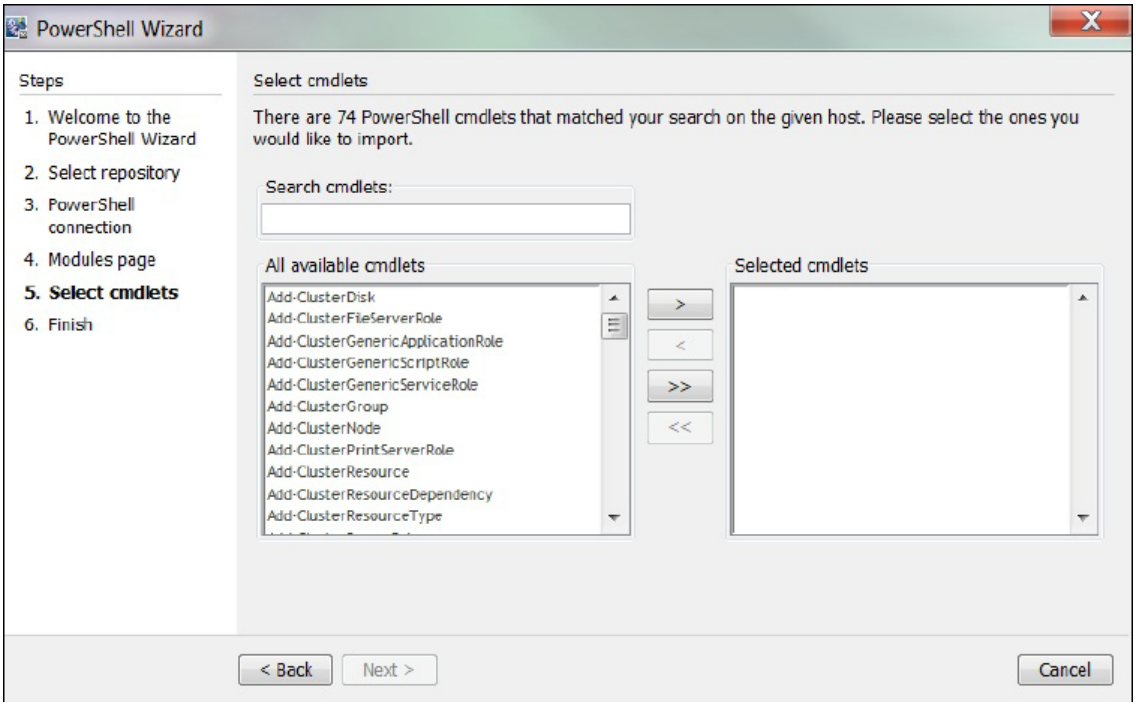
Step 4 — Select the Module



The wizard detects all the available modules/snapins on the target host and displays them in a list as shown above. You can select/unselect any module and the wizard retrieves only those cmdlets contained in the **Selected Modules and Snapins** box.

Cmdlets such as Get-Process and Get-Service are not contained in the list of available modules. These are cmdlets which are available by default in PowerShell. To retrieve the list of default cmdlets, select the **Add default PowerShell cmdlets** box.

Step 5 — Select the Operation



The selected modules are loaded to the PowerShell runspace, and the wizard retrieves the names of the cmdlets from those modules.

You can move the cmdlets from left to right or right to left. Use the search textbox in case the list is very large, and you have difficulties finding the required cmdlet. The wizard searches the list for the cmdlets with names containing the search text. In addition, the wizard updates the list while you are typing.

Chapter 4

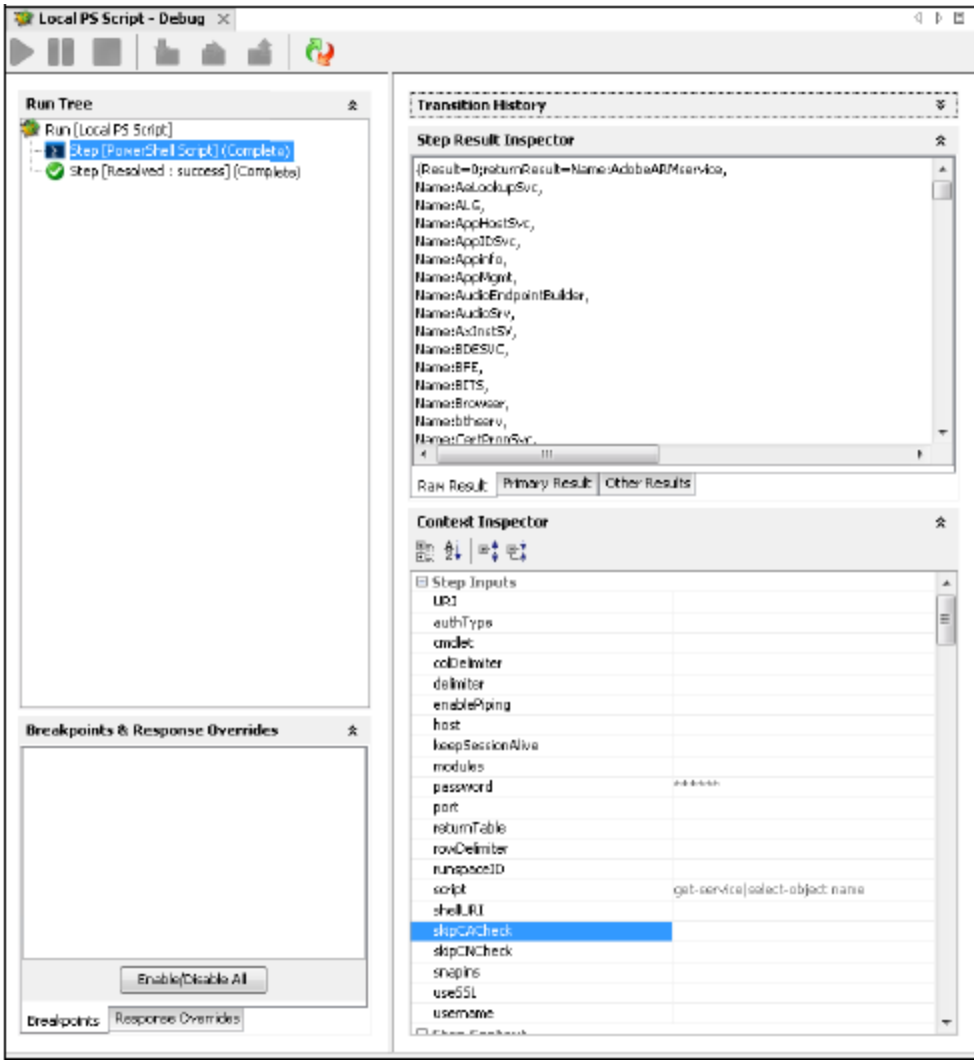
Using the PowerShell Wizard — OO Integration

This chapter includes:

Running a PowerShell Script on a Localhost

The only setting required to execute the PowerShell scripts on the localhost is that the ExecutionPolicy must be RemoteSigned. Use Get-ExecutionPolicy to display the current execution policy and Set-ExecutionPolicy to set the execution policy.

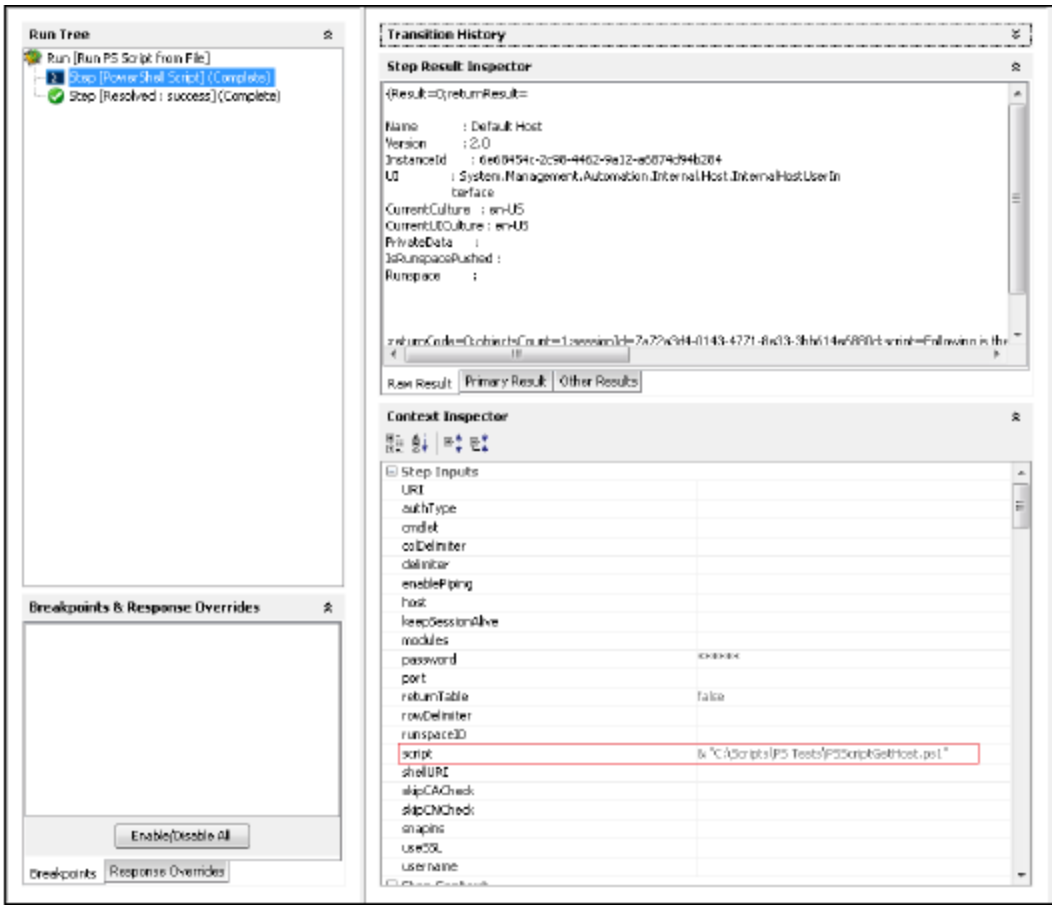
In addition, the required input is the script input.



If one script requires elevated rights, enter a username and a password.

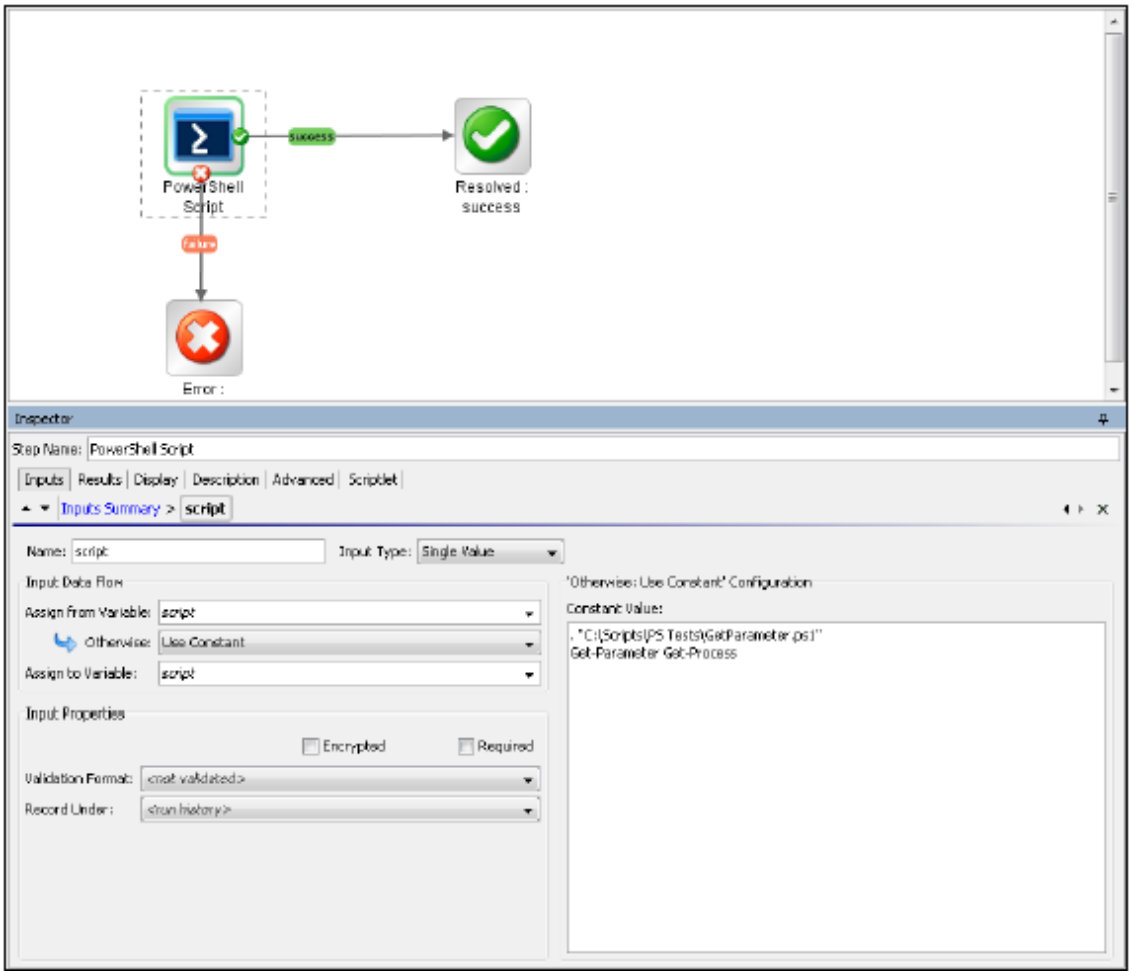
Running PowerShell scripts from a File

To run scripts from a file, just provide the path to that file. This should work in most of the cases, when the path to the file contains white spaces, the operation fails. To fix this provide the path to that file as shown in the picture below.



Loading PowerShell Functions from Files

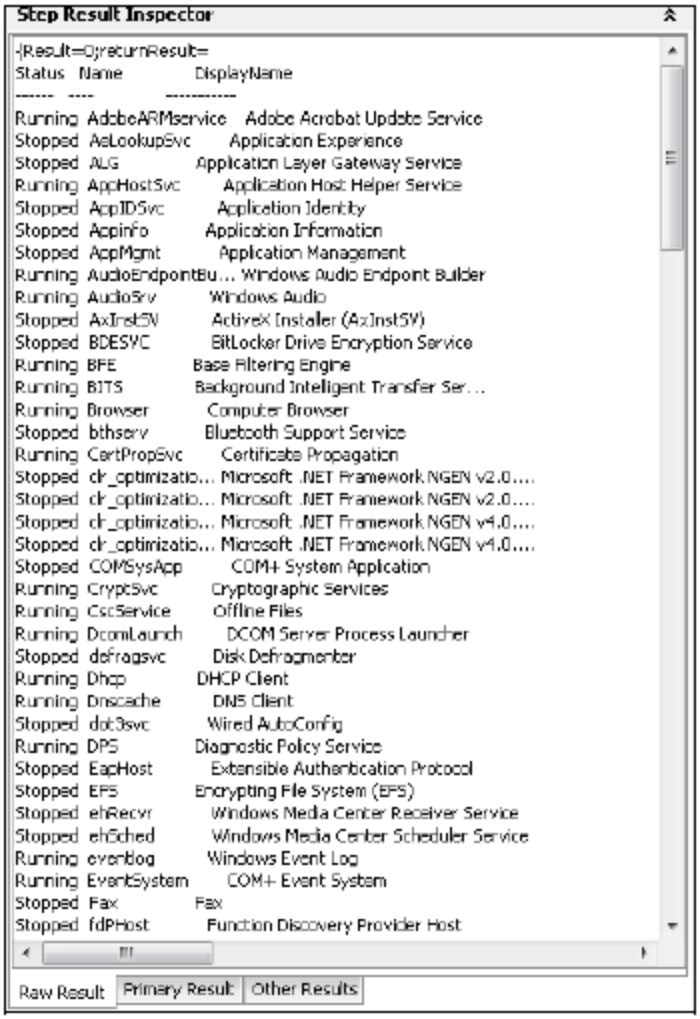
In certain cases, a PowerShell script depends on functions from other files on the disk. The script from the picture below might help to load this file and enable all functions and cmdlets from it. Get-Parameter cmdlet is defined in the file named Get-Parameter.ps1.



Formatting the Results of the PowerShell Script Operations

The result can be formatted as a table or the same way it is displayed in the PowerShell console. The format is determined by the returnTable input. If returnTable is set to false, the operation returns the result as in the PowerShell console.

The example below shows the result if `returnTable=false`.



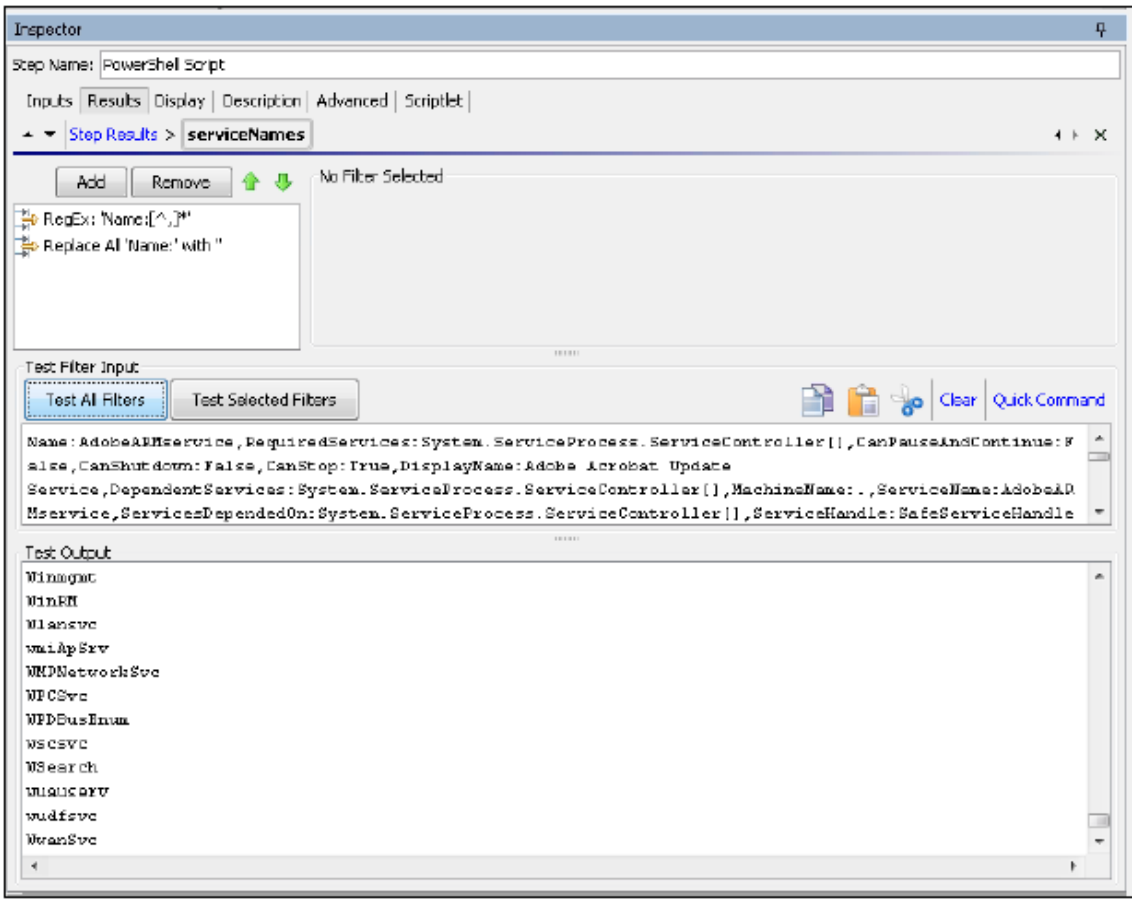
The result is humanly readable, but the problem is that it is very difficult to parse, and does not contain properties which could not be displayed on the screen.

The result can be displayed as a table.

PowerShell session considerations results are displayed in a table. Each PowerShell object (in this case each service), is displayed by default on a line. Each line contains different properties of the service (default delimiter is “,”) and the key-value pairs are delimited by “:”. All these delimiters can be changed. Refer to the Input descriptions for more information.

For example the Get-Service flow returns the following result:

The result contains a full list of properties without any additional PowerShell script, for example, Get-Service|fl, Get-Service|Select-Object Status). Therefore, the status of the services appears and can be parsed by using different filters on the result.



Run Multiple PowerShell cmdlets Scripts in the Same PowerShell Session

This section explains how to run multiple PowerShell script steps in the same PowerShell session. When the PowerShell script has to execute a single script on a remote server, keeping the sessions alive is not necessary. The PowerShell script connects to the remote host, creates a new PowerShell Runspace, for example, a new PowerShell session, runs the full script on the target, and closes the runspace and the connection.

If you want to use the PowerShell Wizard and run the generated flows in a different sequence, you need to consider how much effort is required and whether additional PowerShell scripts need to be created in order to general flows. However, there are situations when you want to keep the PowerShell session alive.

For example, you can use the PowerShell Wizard, connect to a host which has PowerCLI installed on it and follow the wizard steps to generate OO flows for the PowerShell cmdlets to execute VMWare tasks. Suppose that after the wizard finishes, you want to execute one simple cmdlet like Get VM.

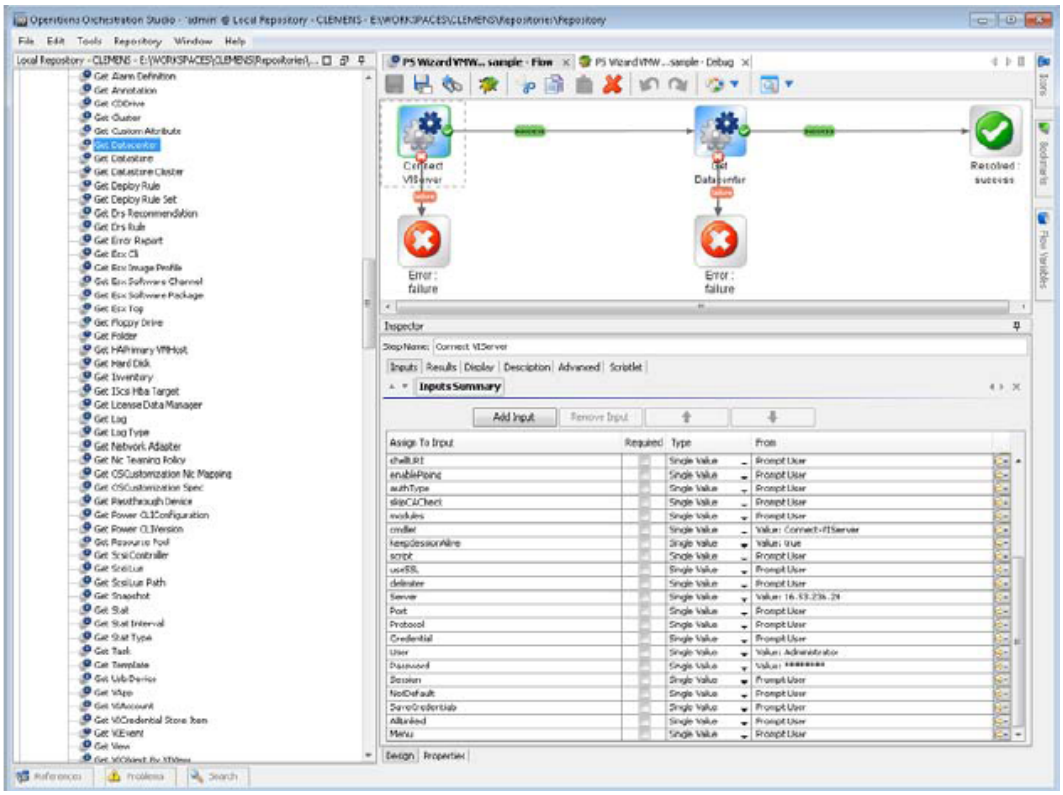
To run cmdlets and keep the sessions alive:.

1. Run the OOTB Get Datacenter flow generated with the PowerShell Wizard.

   Note: This cmdlet does not have any required inputs, however the flow fails to run as you need to run the Connect VIServer cmdlet. The PowerShell Wizard generates this flow to solve this problem.

2. Create a flow sequence as shown below.
   - Do not modify the generated flow.
   - The flow tries to execute Connect VIServer before Get VM.

     In the flow below, the parameters specific to the cmdlet were added as flow inputs. The names appear in capital letters.



You run the flow and provide inputs for the server, user name and password (Connect VIServer). No inputs are required for the Get Datacenter. The Connect VIServer succeeds, however, the Get Datacenter fails.

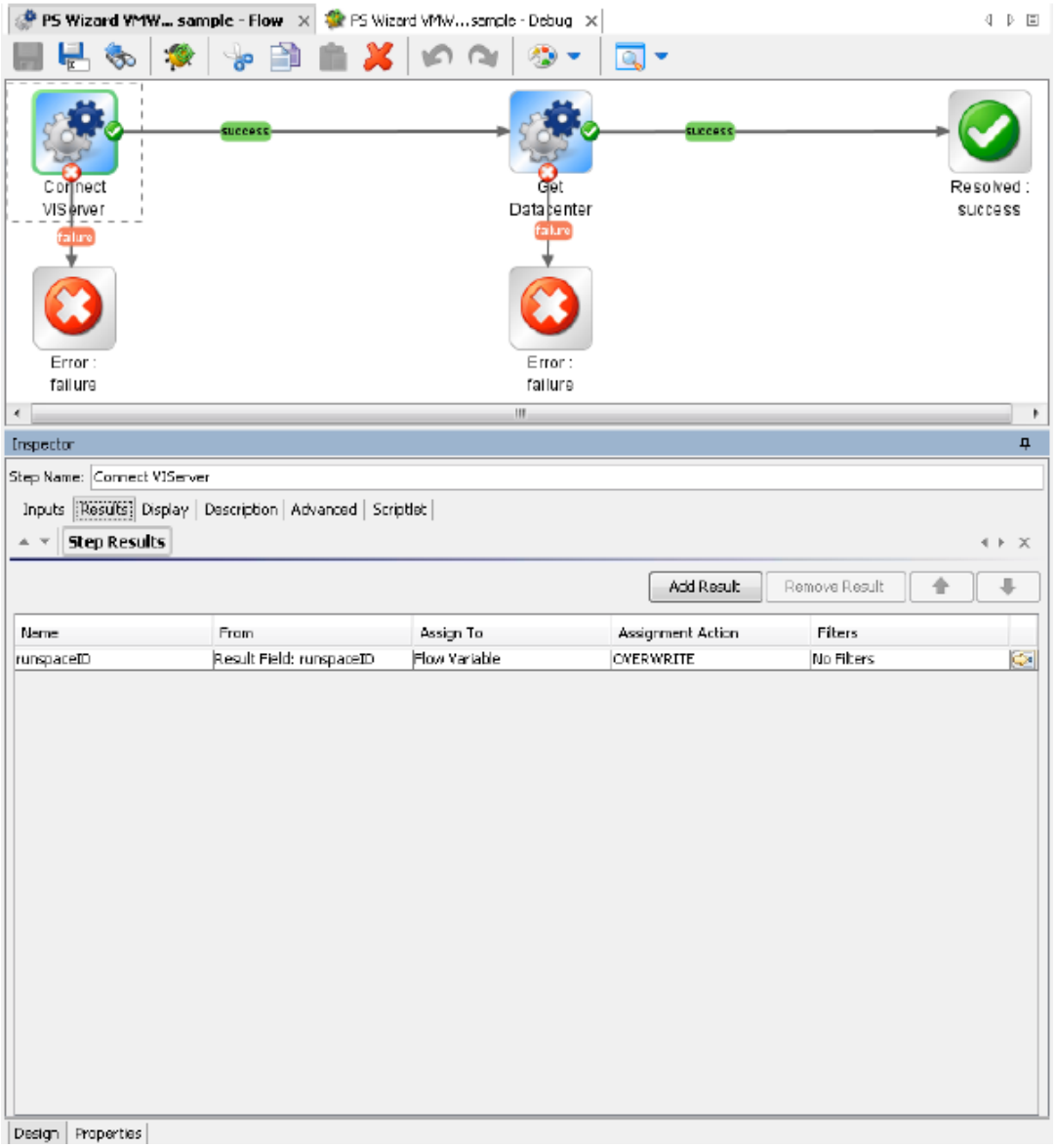
The Get Datacenter fails for the following reasons:

- Connect VIServer passes successfully and the connection to the VMWare server was established.
- Connect VIServer created a new PowerShell runspace, for example, PowerShell session and executed the cmdlet which established a valid connection to the server, however the runspace is closed after the flow runs and the connection is lost.
- Get Datacenter flow creates another PowerShell runspace which is different from the one created by Connect VIServer flow. Therefore, Get-Datacenter cmdlet fails.

3. The solution to the previous step is to keep the session alive during the execution of the two cmdlets.

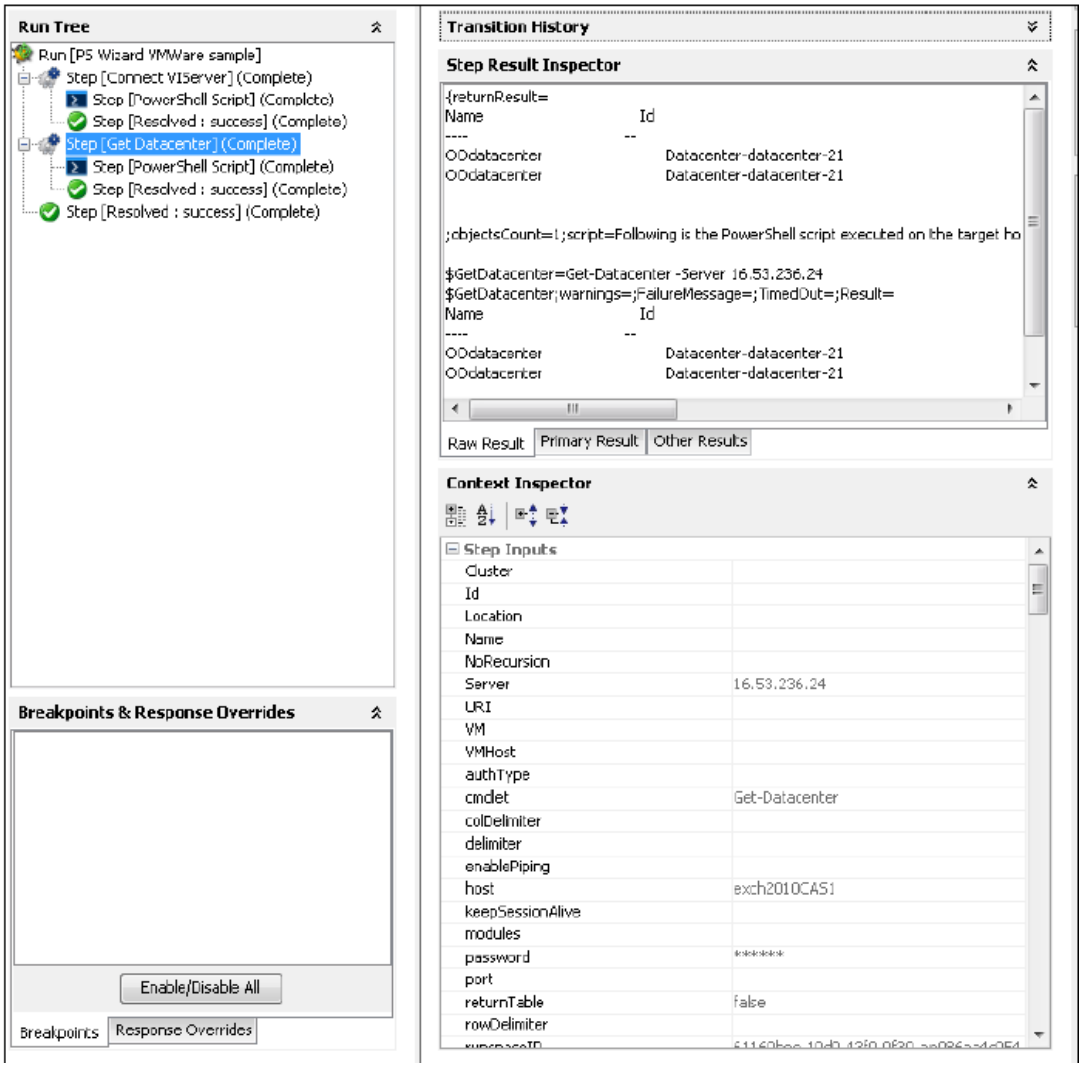
   To do this:

   `keepSessionAlive=true` for the first flow which is Connect VIServer; the runspaceID must be added to the results of the Connect VIServer flow:



4. **runspaceID** of **the Get Datacenter** flow must get its value from the result of the **Connect VIServer**. This happens automatically because runspaceID input assigns its value from the flow variable.

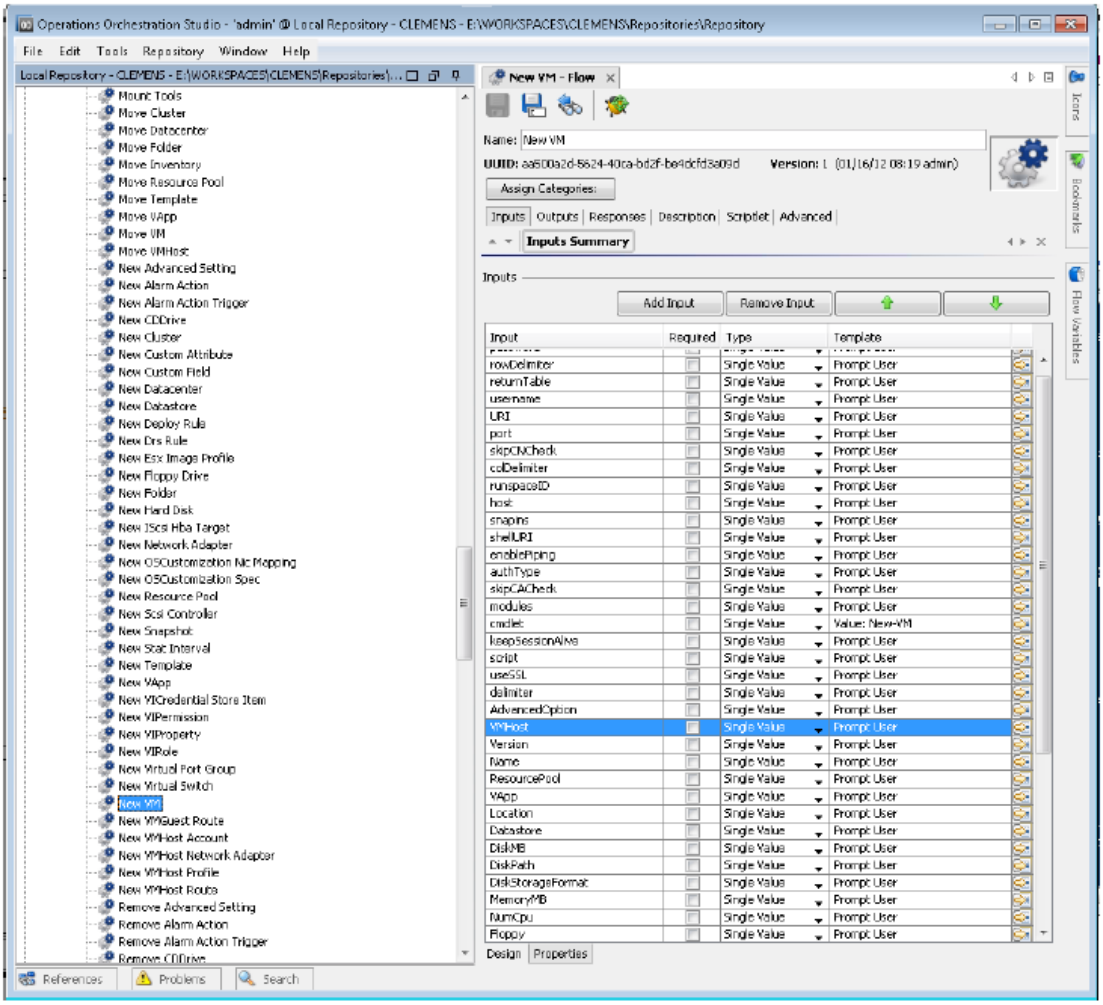
   At this point the flow completes successfully.



Assign the Result of One cmdlet as a Parameter to Another cmdlet

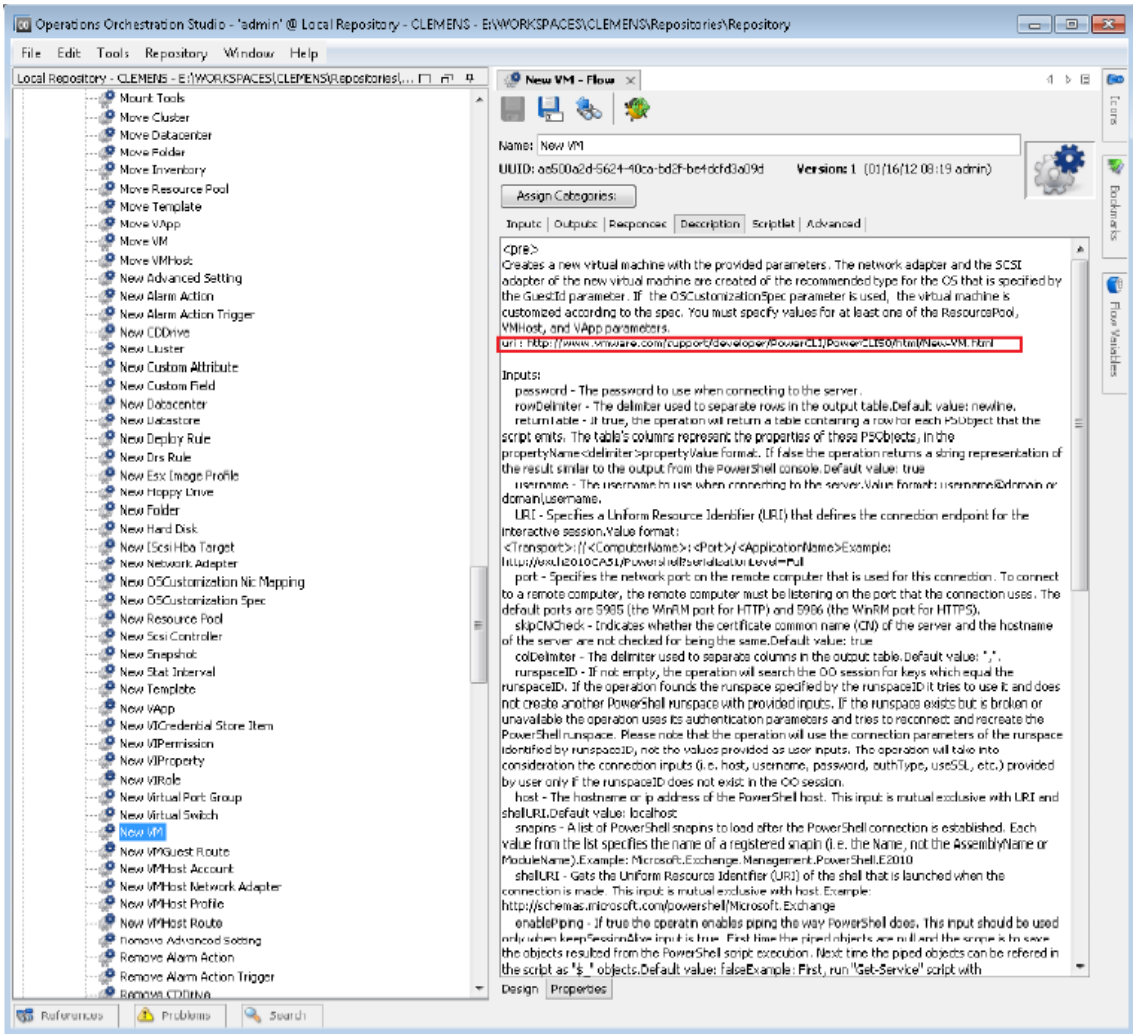
Most of the **get** flows generated through the PowerShell Wizard should work OOTB with minimum effort. But there are some cmdlets, probably the "new" cmdlets, which require as parameters the result of another cmdlet. For example, you can create a new virtual machine using the generated flow New VM.

Even if you follow all the steps described in the section "Run Multiple PowerShell cmdlets Scripts in the Same PowerShell Session" (on page 22), the flow cannot be executed. The PowerShell Wizard generates the flow, but you can run OOTB flows only if their parameters have a built-in type (for example, strings or integers). In case of cmdlets like Get-Help, the parameters can be passed as strings (for example, the name of the cmdlet to search for help information). The New VM flow parameters are below:



New VM has an input named VMHost. What is the type of this parameter?

The description of the generated flows contains information about the PowerShell cmdlet, but for size reasons and other considerations, we could not include the full description of the cmdlet as it is displayed when someone executes Get-Help New-VM –full. The description of the operation contains the original link where you can find detailed information about the cmdlet.



If you open the link in a browser, you can observe the types of the cmdlet parameters. In our case, VMHost is of the same type as the name suggests.

Parameters

NAME	TYPE
VMHost	VMHost
Template	Template
AdvancedOption	AdvancedOption[]
AlternateGuestName	String
CD	SwitchParameter
Confirm	SwitchParameter
Datastore	StorageResource

The problem is that VMHost is an object, not a built-in type which can be resolved as a string. Following the link provided for the VMHost type, you can find which cmdlets return VMHost objects (as seen below). In our case, the type suggests that a cmdlet like Get-VMHost would return this kind of object.

vSphere PowerCLI Reference

VMHost - Object

Property of
VMHostProfileInput, HostVMKernelVirtualNic, EsxCli, VirtualMachine, VMHostFirewallDefaultPolicy, Log, VMHostPatchResult, HostService, VMHostAut
HostVirtualNic, VMHostDiagnosticPartition, VirtualSwitch, VmHostModule, VMHostNetworkInfo, HostNic, IScsiHba, VMHostProfile, VMHostPatch, Hos
NicTeamingPolicy, VMHostProfileIncompliance, PciPassthroughDevice

Parameter to
Start-VMHost, Restart-VMHost, Get-VMHostHba, New-VMHostNetworkAdapter, Get-PassthroughDevice, Get-Datacenter, Get-Cluster, Remove-Data
VMHostNtpServer, New-VMHostProfile, Get-VMHostService, New-VM, Import-VMHostProfile, Add-VmHostNtpServer, Suspend-VMHost, Stop-VMHost
VMHostFirewallDefaultPolicy, Get-VMHostStartPolicy, Get-VMHostProfile, Get-VMHostDiagnosticPartition, Get-VMHostNetworkAdapter, Set-VMHostF
Datastore, Get-VMHostStorage, Get-VMHostNetwork, Get-VMHostFirmware, Get-VirtualSwitch, Test-VMHostProfileCompliance, Get-VirtualPortGrou
VMHostDisk, Get-EsxCli, Get-VMHostProfileRequiredInput, Get-VMHostAuthentication

Returned by
Start-VMHost, Add-VMHost, Set-VMHost, Suspend-VMHost, Stop-VMHost, Restart-VMHost, Get-VMHost, Move-VMHost, Get-HAPrimaryVMHost

Extends
VIContainer

The section "Run Multiple PowerShell cmdlets Scripts in the Same PowerShell Session" (on page 22) described how to execute in the same PowerShell session multiple OO flows generated with

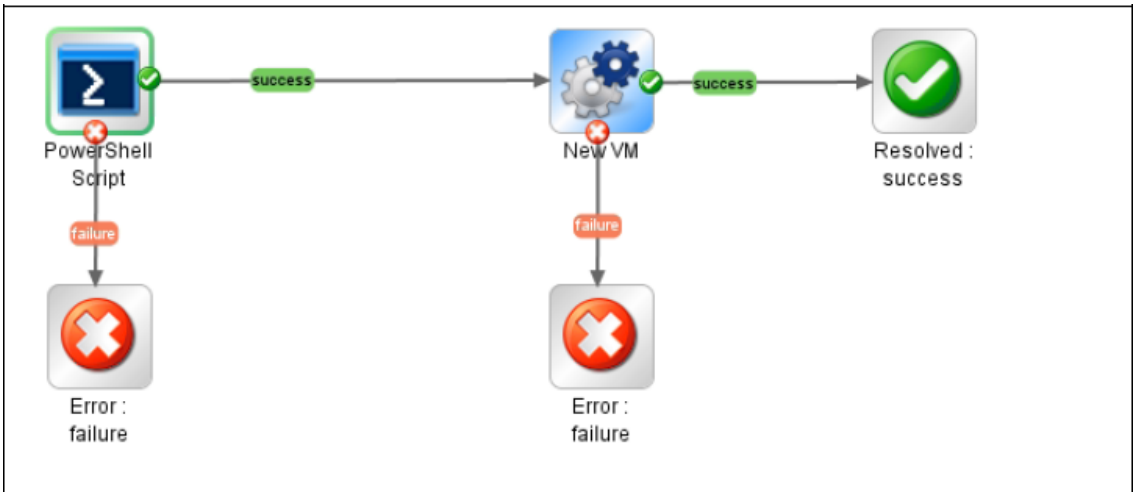
the PowerShell wizard. At this point, the following chain of cmdlets need to be executed to create a new vm:

- **Connect-VIServer.** This must be executed before any VMWare cmdlet.
- **Get-VMHost.** You need the result of this cmdlet as parameter for the next cmdlet.
- **New-VM**. This cmdlet actually creates a new virtual machine.

New VM has other parameters beside VMHost which are not built-in. See the following sections to learn how to solve the VMHost parameter. Use the process described for other parameters as well.

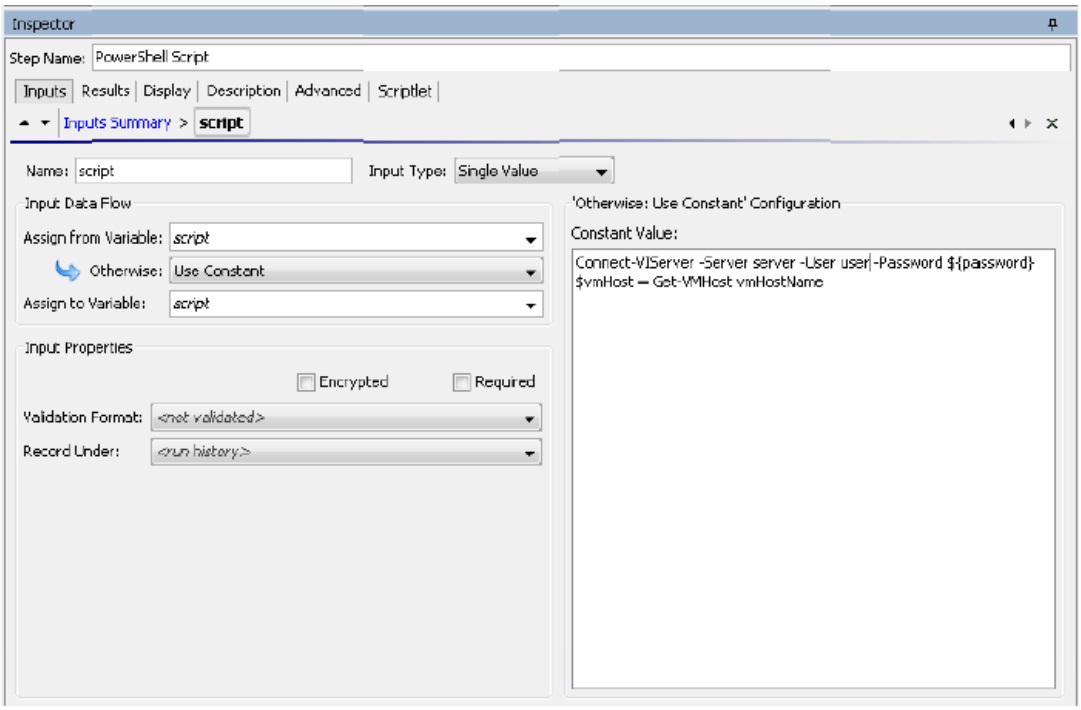
You need to execute 3 cmdlets in the same PowerShell session. You have generated flows for each of the cmdlets and executed them in the same session. The next step is to take the result of the Get-VMHost cmdlet and pass it to the New-VM cmdlet? You can choose any of the following solutions offered

Solution 1— Creating a New PowerShell Script Step

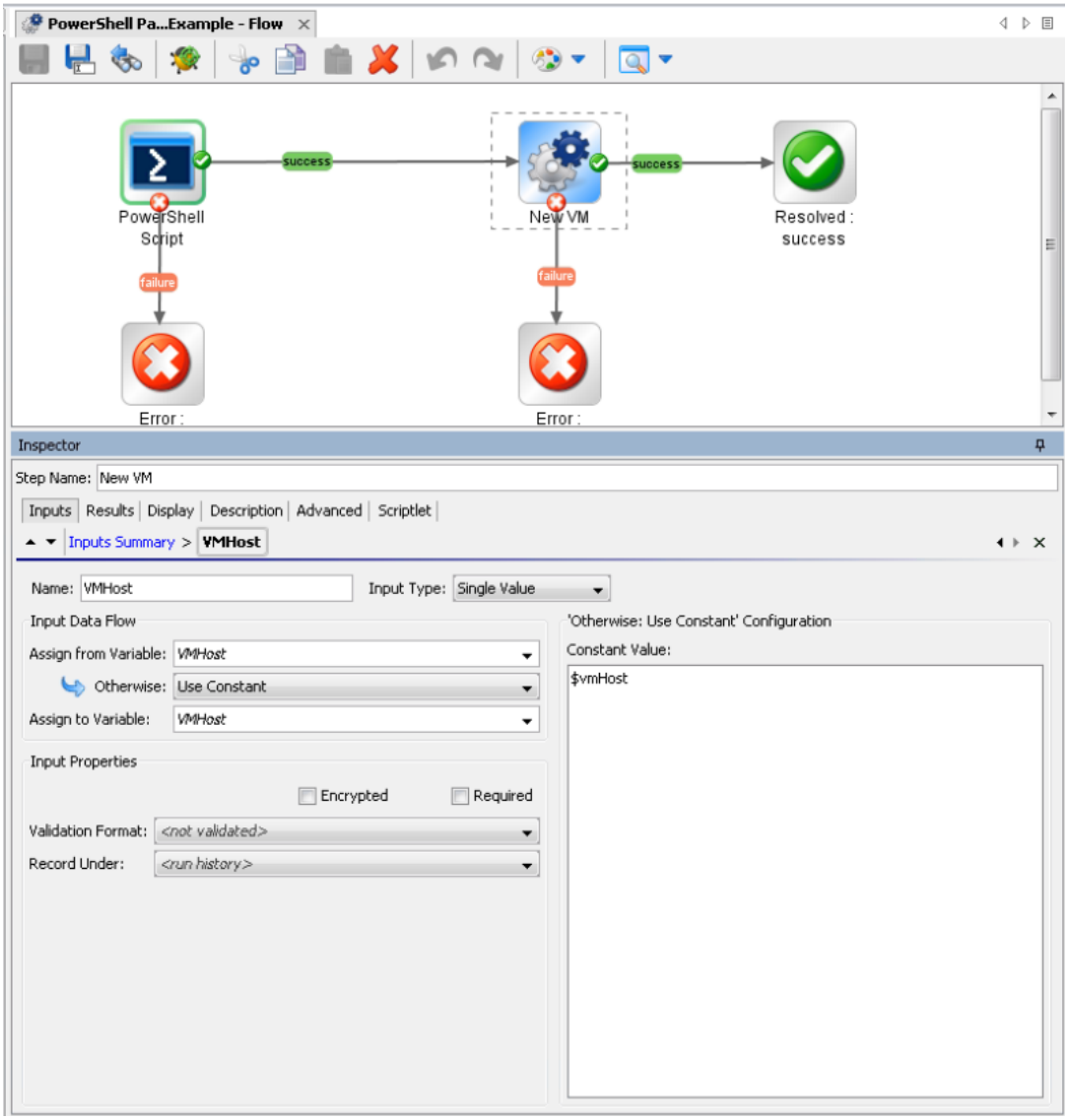


Create a new PowerShell script in addition to the generated flow that you want to run. If you keep the session alive during the execution of the two PowerShell script steps, you can use the PowerShell script variables defined in the first step to pass them in the script of the second step or as parameters for the generated flow. In this case, you are not using the generated flows for Connect-VIServer and Get-VMHost. However, you need to write the script.

1. Execute the following script, then save the result of the Get-VMHost cmdlet in the PowerShell variable named $vmHost.



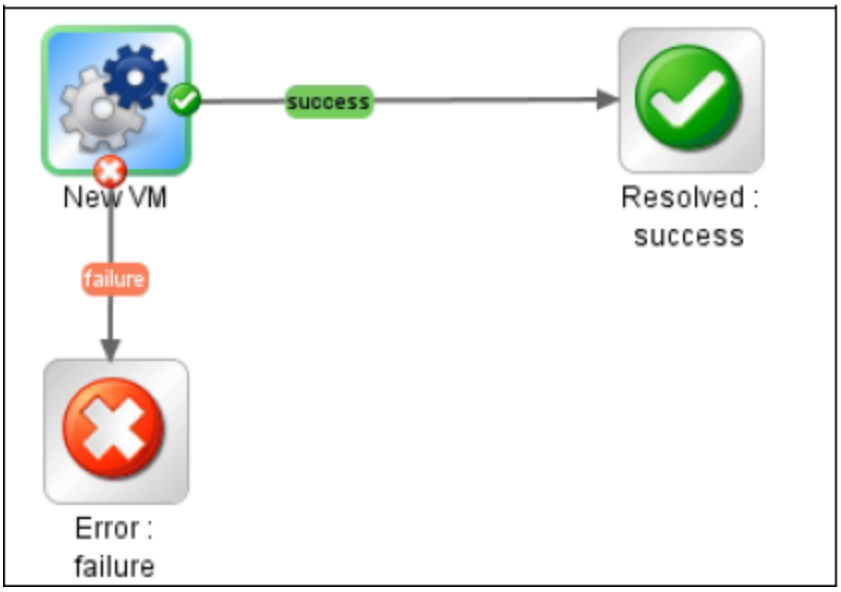
2. Assign the value of the VMHost input from the $vmHost variable.



This enables you to pass PowerShell cmdlets results between OO flows generated with the PowerShell wizard.

Note: The $var refers to PowerShell variables and ${var} refers to OO flow variables.

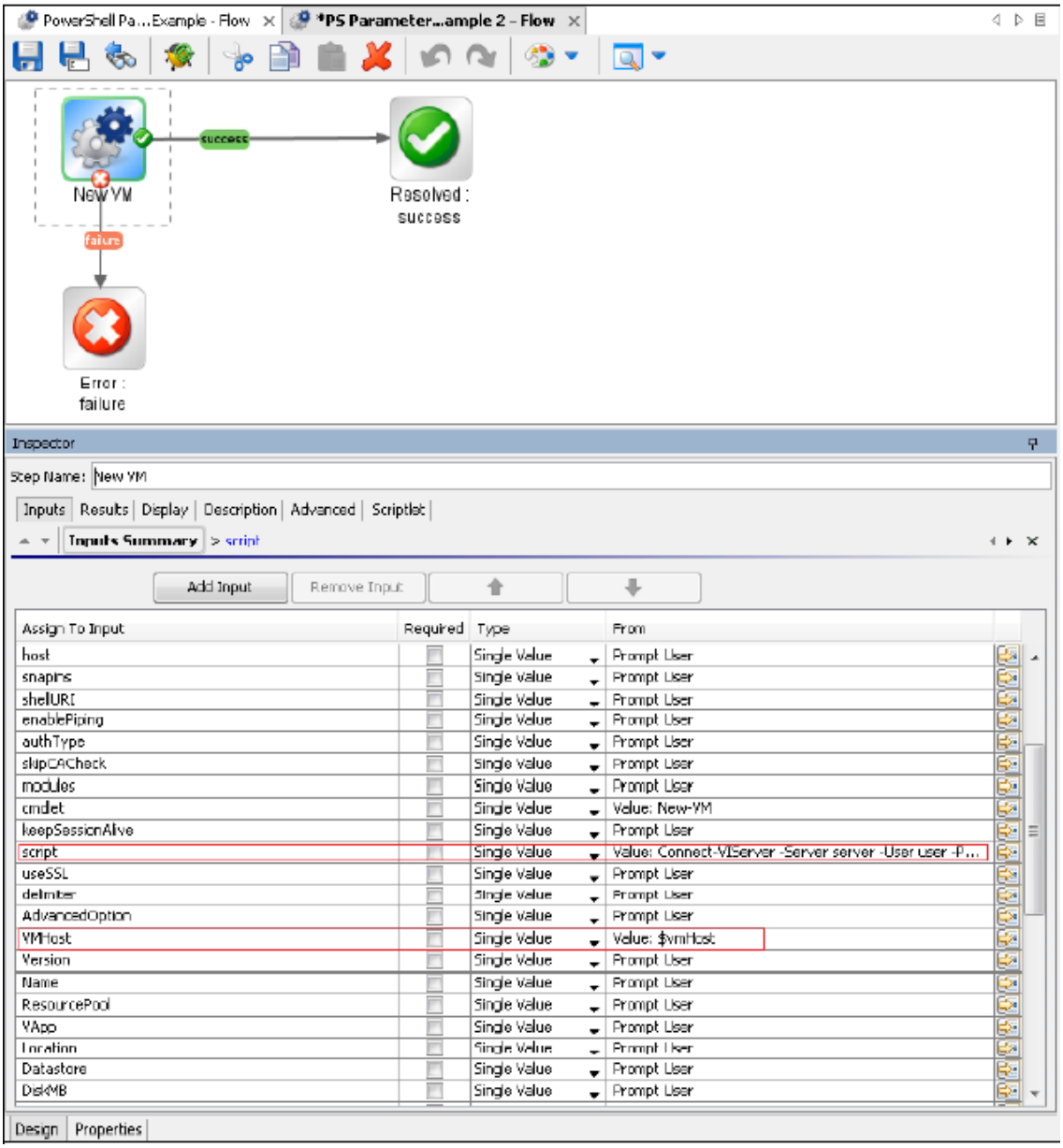
Solution 2 — Run a PowerShell Script in the Generated Flow Context



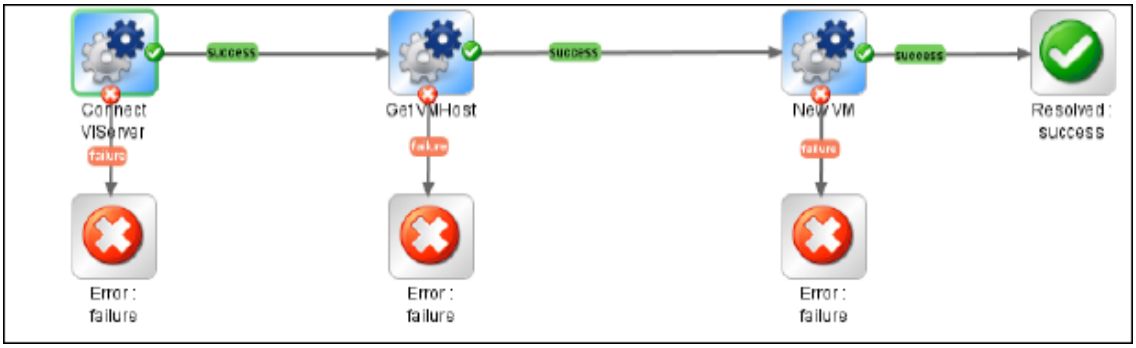
This solution executes the script previously defined in **Solution 1**, in the context of the generated flow. The **PowerShell script** operation has two inputs which build the script that is going to be executed:

- **Script.** A PowerShell script to execute on target host. If the script input is not empty, then the PowerShell script defined by this input is going to be executed before the cmdlet.
- **Cmdlet.** The PowerShell cmdlet name.

The solution is shown in the following flow:



Solution 3 — Using Only Generated Flows to Minimize the Effort



The result of each PowerShell cmdlet executed from a generated flow is saved in a PowerShell variable with the same name as the cmdlet, for example, for Get-VMHost, the variable is $GetVMHost.

Without writing any PowerShell script, you can execute the Get VMHost generated flow and know that the result of this cmdlet is saved in the $GetVMHost variable. Pass the variable to the VMHost input of the New VM flow.

Chapter 5

Powershell Remoting

This chapter includes:

Powershell Remoting Overview

- You can enable PowerShell remoting by running the following cmdlet: Enable-PSRemoting.
- In workgroup environments, you can enable classic mode authentication for network logons.

  To enable classic mode authentication for network logons:

  a. From **Control Panel** open **Local Security Policy**.

  b. Select **Administrative Tools**.

  c. Select **Local Policies** and then **Security Options**.

  d. Double-click **Network Access: Sharing and Security Model for local accounts** and set it to **Classic**.
- Modify the WSMan trusted hosts setting by adding the IP addresses of all remote clients to the list of trusted hosts. To do this, use one of the following commands:
  - Set-item wsman:localhost\client\trustedhosts -value * (adds all computers as trusted hosts)
  - Set-item wsman:localhost\client\trustedhosts -value Computer (only adds Computer to the trusted hosts)
  - Set-item wsman:localhost\client\trustedhosts -value *.domain.com (adds all computers in the specified domain)

- Set-item wsman:localhost\client\trustedhosts -value 10.10.10.1 (adds the remote computer with the IP address 10.10.10.1 to the trusted hosts list).

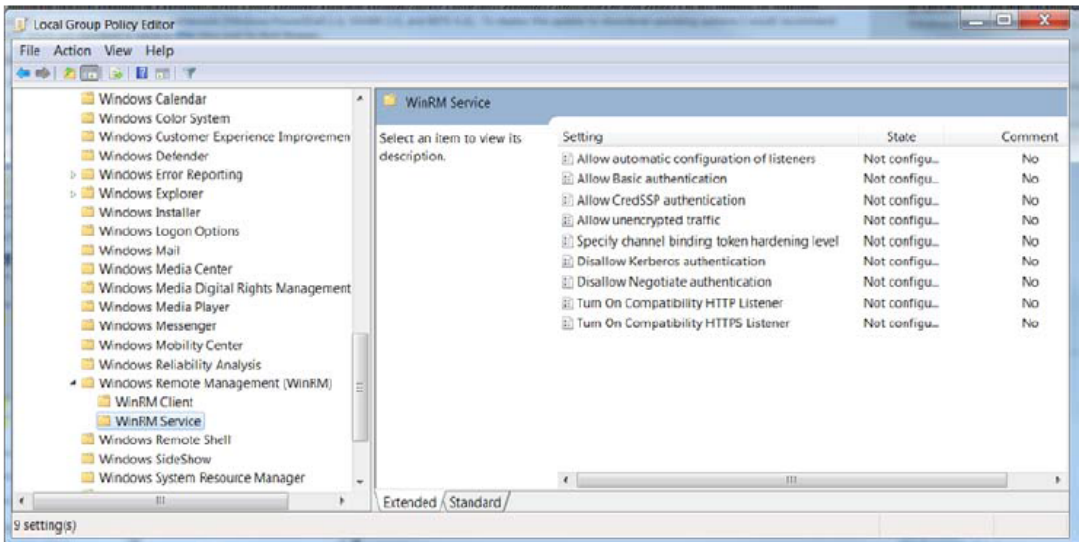
Enable Remoting Using GPO (Group Policy Objects)

While remoting can be enabled manually using Enable-PSRemoting, it is recommended to use GPO management tools whenever possible. Use GPO to apply policies on a single host (for example, the target PowerShell host) or a group of servers.

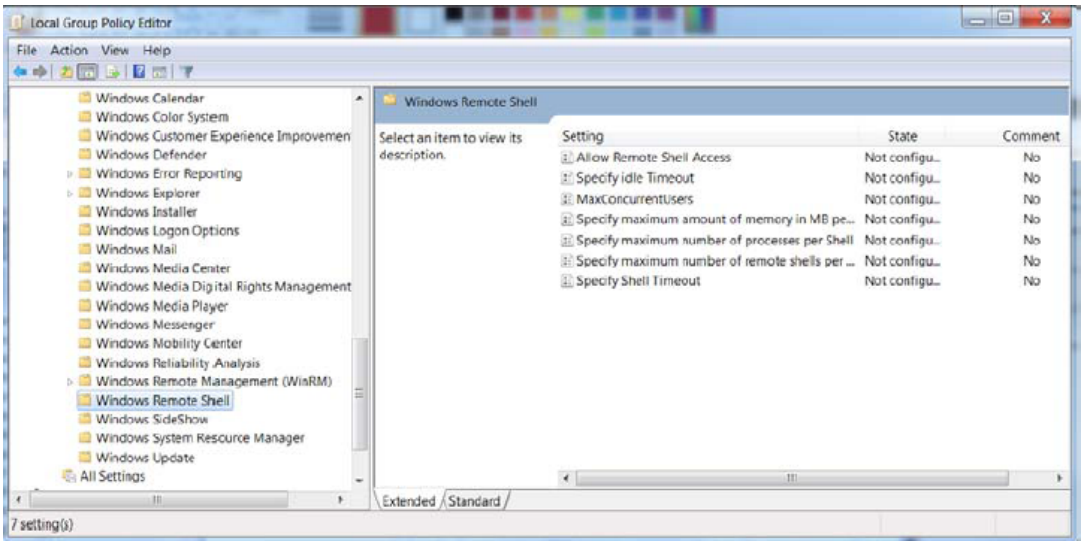
Group Policy Configuration for a Single Host

To enable PowerShell remoting for a single host:

1. Open the Group Policy Management console, for example, gpedit.msc.
2. Go to **Local Computer Policy** > **Computer Configuration** > **Administrative Templates** > **Windows Components** > **Windows Remote Management (WinRM)**.



3. WinRM is the service which PowerShell uses for remote sessions. WinRM can be configured as client or service, depending on the role the host is going to have in a PowerShell connection (that is, request access to execute scripts on other hosts or allow other hosts to execute scripts on the current host). At this point, the user can enable different authentication types, specify the trusted hosts, enable HTTP or HTTPS listeners, and so on.
4. Under **Local Computer Policy** > **Computer Configuration** > **Administrative Templates** > **Windows Components** > **Windows Remote Shell**, there are some other default settings (see below) you might want to change in a production environment.



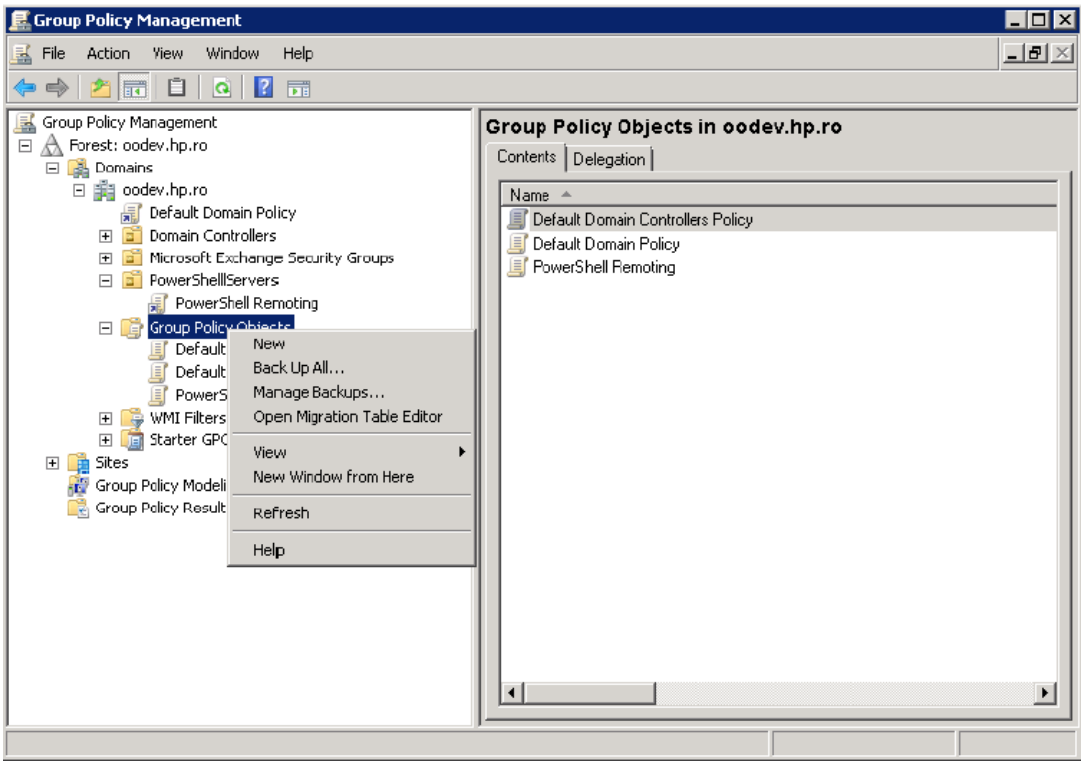
5. You can specify the maximum number of remote shells per user (default is 2) or the maximum amount of memory in MB for shell (default is 150).
6. After configuring the GPO, you might need to restart the computer in order to apply the policies, or try to run the command `gpupdate`.

Group Policy Configuration for a Group of Servers

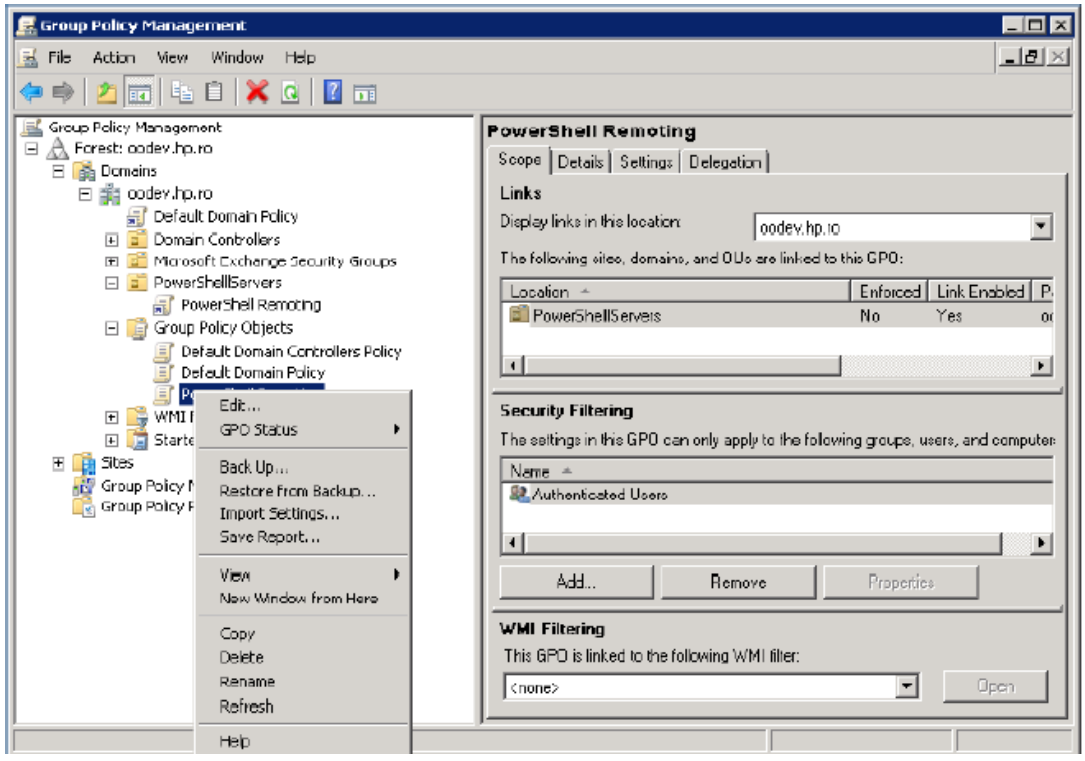
In certain cases, GPO policies must be applied to multiple server hosts and repeating the steps in "Group Policy Configuration for a Single Host" (on page 36) for every server might not be the best solution. Therefore, you can create a new GPO policy, configure it and apply it to a list of servers.

To enable PowerShell remoting for multiple server hosts:

1. Go to the domain controller or a server where **gpmc.msc** is available and open it.
2. Right click **Group Policy Object** and select **New**. Enter a name for the new GPO and then select the policy from which the GPO inherits.

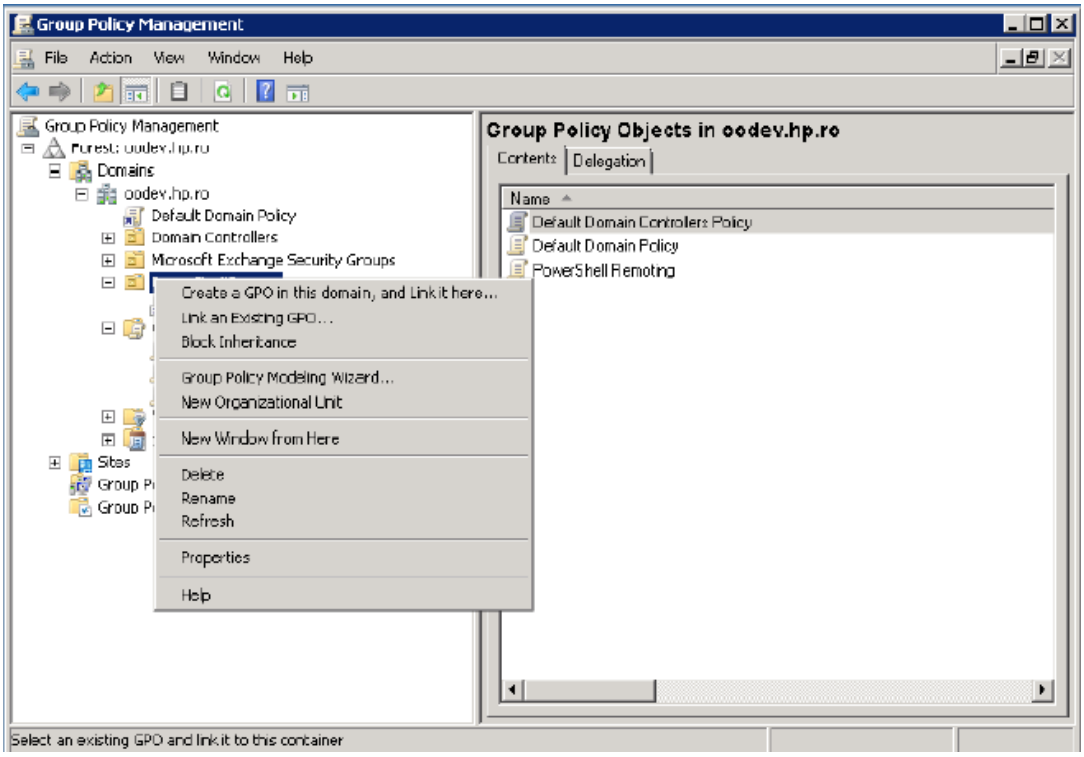


3. Click **Next**.
4. Right-click the new GPO and select **Edit**.



5. Configure the GPO at the domain level (instead of for each host individually).

6. You can now apply the new GPO to a group of servers. The following example shows you how to link it to an existing Organizational Unit (OU) from the Active Directory (AD).



You can also apply the new GPO to other groups. The GPO interface displays the existing OUs from the domain controller AD. To link a GPO to an OU, go to that OU, right click it, and select **Link an Existing GPO**.

The GPO settings are applied to all servers contained in the selected OU. A GPO update and a reboot for the servers might be required before the policies are actually applied.

> **Note:** Local policies overwrite domain policies.

Enable Remoting for Non-Administrative Users

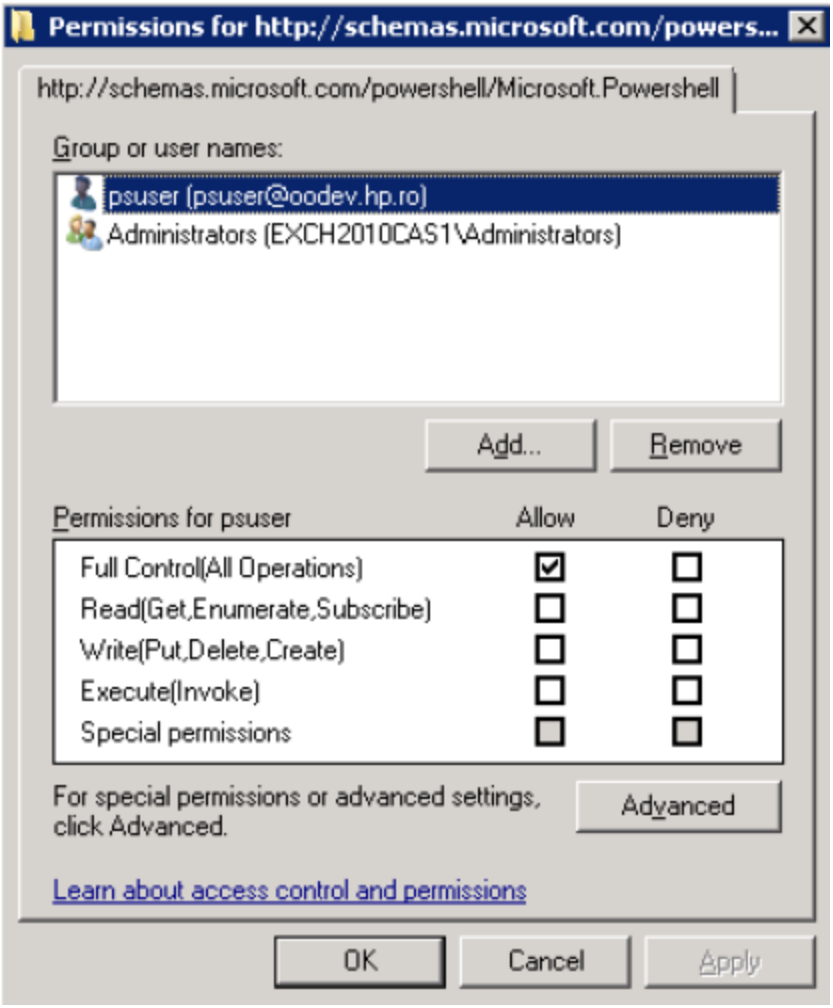
To establish a PSSession or run a command on a remote computer, you must have permission to use the session configurations on the remote computer.

By default, only members of the Administrators group on a computer have permission to use the default session configurations. Therefore, only members of the Administrators group can connect to the computer remotely.

To allow other users to connect to the local computer, give the user Execute permissions to the default session configurations on the local computer.

The following command opens a property sheet that lets you change the security descriptor of the default Microsoft.PowerShell session configuration on the local computer:

```
Set-PSSessionConfiguration Microsoft.Powershell -
ShowSecurityDescriptorUI
```



Authentication Types

Basic

- Client side steps
  - Allow unencrypted communication for the client, by running the following PowerShell command: `set-item wsman:\localhost\client\AllowUnencrypted -value true`.
  - Enable Basic authentication for the client, by running the following PowerShell command: `set-item wsman:\localhost\client\auth\Basic -value true`.
- Server side steps
  - Turn off encryption for the WinRM service, by running the following PowerShell command: `set-item wsman:\localhost\service\AllowUnencrypted -value true`.
- Enable Basic authentication for the service, by running the following PowerShell command: `set-item wsman:\localhost\service\auth\Basic -value true`.

Note:

- The client and server can be in different domains.
- When using Basic authentication, a local user account must be provided for authentication on the remote host.
- Basic can be used when the destination is an IP address.
- Basic can be used when the destination is one of the following: localhost, 127.0.0.1, [::1].
- The cluster name, as well as the hostnames of the cluster nodes can be used for the destination host.

CredSSP

- Client side steps
  - Enable CredSSP authentication for the client, by running the command: `Enable-WSManCredSSP -Role Client -DelegateComputer WSMAN/*`.
  - Allow delegating fresh credentials by performing the following steps.
    - Open **gpedit.msc** and go to **Computer Configuration** > **Administrative Templates** > **System** > **Credentials Delegation**.
    - Enable **Allow Delegating Fresh Credentials** and add the wsman hosts to the server list.
    - Run **gpupdate /force** from command line to force policy update.
- Server side steps
  - Enable CredSSP authentication, by running the following PowerShell command: `Enable-WSManCredSSP -Role Server`.
  - Create a new https listener by using the following command: `winrm create winrm/config/Listener?Address=*+Transport=HTTPS`.
- Domain Controller side steps

  If the NETWORK SERVICE does not have **Validated write to service principal name**, do one of the following:

  - Run the following command: `dsacls "CN=AdminSDHolder,CN=System,DC=domain,DC=com" /G "Sn-1-5-20:WS;Validated write to service principal name"`

    Or

  - Do the following:
    - Open ADUC.
    - Go to **Computers** > **DC object** > **Security**.
    - Select **Network Service**.
    - Give it **Validated write to SPN**.

Default

When Default authentication is used, the following situations can occur:

- Kerberos is the method of authentication used if the client is in the same domain as the destination host, and the value specified for that host is not one of the following: localhost, 127.0.0.1, [::1].
- Negotiate is the method of authentication used if the client is not in the same domain as the destination host, or the value specified for that host is one of the following: localhost, 127.0.0.1, [::1].

Digest

Digest authentication is not supported for remote connections. It cannot be configured for the WinRM server component.

Kerberos

- **Client side steps**. Enable Kerberos authentication for the client, by running the following PowerShell command: `set-item wsman:\localhost\client\auth\Kerberos -value true`.
- **Server side steps**. Enable Kerberos authentication for the service, by running the following PowerShell command: `set-item wsman:\localhost\service\auth\Kerberos -value true`.

Note:

- The client and server must be in the same domain.
- Either a local or a domain user account can be provided for authentication on the server host.
- Kerberos cannot be used when the destination is an IP address.
- Kerberos cannot be used when the destination is one of the following: localhost, 127.0.0.1, [::1].
- The cluster name cannot be used to specify the host. Only the hostnames of the cluster nodes can be used for the destination host.

Negotiate

- **Client side steps.** Enable Negotiate authentication for the client, by running the following PowerShell command: `set-item wsman:\localhost\client\auth\Negotiate -value true`.
- **Server side steps.** Enable Negotiate authentication for the service, by running the following PowerShell command: `set-item wsman:\localhost\service\auth\Negotiate -value true`.

Note:

- The client and server can be in different domains.
- Either a local or a domain user account can be provided for authentication on the server host. Local accounts can only be provided when connecting to the localhost.
- Negotiate can be used when the destination is an IP address.

- Negotiate can be used when the destination is one of the following: localhost, 127.0.0.1, [::1].
- The cluster name, as well as the hostnames of the cluster nodes can be used for the destination host.

NegotiateWithImplicitCredential

Note:

- When using NegotiateWithImplicitCredentials, no credentials should be provided. The current logged-on user account will be used for authentication. This can either be a local or a domain user account.
- NegotiateWithImplicitCredential can only be used when the destination is one of the following: localhost, 127.0.0.1, [::1].

Chapter 6

Powershell Troubleshooting

This section provides troubleshooting procedures that you can use to solve problems you may encounter while using the wizard. It also includes an error message you may receive while using the integration and offers descriptions and possible fixes for the error.

Could Not Connect to the Host

The possible reasons are:

- The user credentials are not correct.
- The user does not have permission to execute PowerShell scripts on the target host. Make sure the user has admin rights or refer to the section "Enable Remoting for Non-Administrative Users" (on page 39).
- Authentication problems (most common).
- The WinRM service is stopped on the target host.
- WinRM default ports (5985 and 5986) were changed. You need to provide the correct port in the connection page of the wizard.

The Wizard Fails to Load modules on a x64 Localhost

Some modules cannot be loaded using the wizard, but they are loaded from the PowerShell console. By default, the wizard runs in a x32 process (depending on the OO jre) which ends up calling x32 PowerShell. The x32 version of PowerShell cannot load some modules (for example, FailoverClusters) and therefore, the wizard fails. In order to fix this, do not leave the host input empty. Instead, you need to provide the **localhost**. This way, the wizard tries to authenticate the localhost like any other remote host.

Note: Remoting rules should be satisfied for localhost in this case. If user is left empty, the wizard connects using the NegotiateWithImplicitCredential. Otherwise, you need to provide user credentials and authentication type as for any other remote host.

The User Has Exceeded the Maximum Allowed Number of Remote Shells

This error would probably occur if the user stresses the wizard with too many **back** and **next** actions without running the wizard from start to end. Refer to " Group Policy Configuration for a Single Host" (on page 36) in order to increase the allowed number of remote shells per user.


PowerShell Connection Error
Could not connect to target PowerShell host.
Error:dotNETWebActions_PowerShell.PowerShellExceptions
.SessionException: Could not connect to host. ---> System
.Management.Automation.Remoting.PSRemotingTransportE
xception: Connecting to remote server failed with the follow
ing error message : The WS-Management service cannot pr
ocess the request. This user is allowed a maximum number o
f 5 concurrent shells, which has been exceeded. Close exist
ing shells or raise the quota for this user. For more informati
on, see the about_Remote_Troubleshooting Help topic.
--- End of inner exception stack trace ---
at PowerShellV2.PSRunspaceV2.NewRemoteSession(String
host, PSCredential credential, AuthenticationMechanism au
thentication)
at PowerShellV2.PSRunspaceV2.CreateRunspace(String h
ost, PSCredential credential, AuthenticationMechanism auth
entication)
at PowerShellV2.PSRunspaceV2.CreateNewRunspace()
at PowerShellV2.PSRunspaceV2..ctor(PSAuthenticationInf
o authInfo)
at PowerShellV2.PowerShellFactoryV2.CreateRemoteRun
space(PSAuthenticationInfo psAuth, Boolean keepSessionAl
ive, Boolean enablePiping)
at PowerShellCommon.PowerShellFactory.CreateRunspac
e(PSAuthenticationInfo authInfo, Boolean keepSessionAlive,
Boolean enablePiping)
at PowerShellCmdlets.Program.CreateRunspace(String ar
gs)
OK
Details >>