



**Hewlett Packard**  
Enterprise

# HPE Operations Orchestration

Software Version: 10.60  
Windows and Linux Operating Systems

## Studio Wizards Guide

Document Release Date: May 2016  
Software Release Date: May 2016

## Legal Notices

### Warranty

The only warranties for Hewlett Packard Enterprise 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. HPE 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 HPE 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

© Copyright 2016 Hewlett Packard Enterprise Development LP

### 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

To download the most recent edition of a document, go to <https://softwaresupport.hp.com>.

## About this PDF Version of Online Help

This document is a PDF version of the online help. This PDF file is provided so you can easily print multiple topics from the help information or read the online help in PDF format. Because this content was originally created to be viewed as online help in a web browser, some topics may not be formatted properly. Some interactive topics may not be present in this PDF version. Those topics can be successfully printed from within the online help.

# Contents

Using the HPE OO Wizards .....	6
Using the REST Wizard .....	7
Purpose of the REST Wizard .....	7
Audience .....	7
Supported Versions .....	7
Getting Started with the REST Wizard .....	8
Installing the REST Wizard .....	8
REST Wizard Logs .....	8
Uninstalling the REST Wizard .....	8
REST Wizard System Requirements .....	8
REST Wizard Steps .....	9
Step 1. Welcome Page .....	9
Step 2. Destination .....	9
Step 3. API Definitions .....	10
Step 4. Operations .....	11
Step 5. Configuration .....	12
Step 6. Summary .....	14
Importing the Generated Flows into Studio .....	14
Generated Flows .....	14
Inputs .....	16
Outputs .....	16
Responses .....	16
Descriptions .....	17
REST Operation .....	18
Troubleshooting .....	18
Troubleshooting Overview .....	18
Unable to locate resource .....	19
Using the Third Party Content Pack Wizard .....	20
Purpose of the Third Party Content Pack Wizard .....	20
Supported Versions .....	20
Getting Started with the Third Party Content Pack Wizard .....	20
Installing the Third Party Content Pack Wizard .....	20
Third Party Content Pack Wizard Logs .....	20
Uninstalling the Third Party Content Pack Wizard .....	20
Third Party Content Pack Wizard Requirements .....	20
Third Party Content Pack Wizard Steps .....	20
Step 1. Welcome Page .....	21
Step 2. Libraries Selection Page .....	21
Third Party Wizard Command Line Tool .....	24
Troubleshooting .....	25
Using the Web Services Wizard .....	26
Overview of the Web Services Wizard .....	26
System Requirements .....	26

Installing the Web Services Wizard .....	26
Web Services Wizard Code Dependencies .....	26
Configuring Logging Settings .....	27
WS Wizard Enhancements from 9.x .....	27
Downloading HPE OO Releases and Documents on HPE Live Network .....	27
Wizard Processing Details .....	27
How the Web Services Wizard Uses SoapUI .....	27
Processing Templates .....	28
Locating Inputs and Creating the inputMap .....	28
Locating Outputs and Creating Operation Outputs .....	31
Populating InvokeMethod2 Default Values for All Operations .....	32
The Invoke Method 2 Operation .....	32
Overview of the Invoke Method 2 Operation .....	32
Building a SOAP Request .....	33
Complete Set of Inputs .....	33
Using the Web Services Wizard to Create Web Service Flows .....	36
Using the Web Services Wizard to Create HPE OO Flows from Selected WSDL Operations .....	36
After Running the Web Services Wizard .....	41
Troubleshooting .....	42
General Troubleshooting Principles .....	42
Troubleshooting Steps .....	42
Creating an OO Web Service Tool with Proxy .....	44
Using the Shell Wizard .....	45
Purpose of the Shell Wizard .....	45
Audience .....	45
Supported Versions .....	45
Getting Started with the Shell Wizard .....	45
Installing the Shell Wizard .....	45
Configuring Logging Settings .....	45
Uninstalling the Shell Wizard .....	45
Shell Wizard Requirements .....	46
Shell Wizard Enhancements from 9.x .....	46
Shell Wizard Steps .....	46
Step 1. Welcome Page .....	46
Step 2. Selecting the Destination .....	46
Step 3. Specifying Flow Information .....	47
Step 4. Entering Connection Settings .....	48
Step 5. Specify Telnet Prompts .....	49
Step 6. Recording Commands .....	50
Step 7. Finish .....	51
Importing the Generated Flows into Studio .....	51
Generated Flows .....	52
Inputs .....	52
Outputs .....	52
Responses .....	53
Descriptions .....	53
Shell Operation .....	54

Operation Inputs .....	54
Operation Results .....	55
Troubleshooting .....	55
Troubleshooting Overview .....	55
Could not connect to the host .....	55
Error Messages .....	55
Using the PowerShell Wizard .....	56
Purpose of the PowerShell Wizard .....	56
Supported Versions .....	56
Getting Started with the PowerShell Wizard .....	56
Downloading the PowerShell Wizard .....	56
Starting the PowerShell Wizard .....	57
Configuring Logging Settings .....	57
Uninstalling the PowerShell Wizard .....	57
PowerShell Wizard Requirements .....	57
PowerShell Wizard Enhancements from 9.x .....	58
PowerShell Wizard Steps .....	58
Step 1. Selecting the Repository .....	58
Step 2. Configuring the PowerShell Connection .....	60
Step 3. Selecting the Modules .....	61
Step 4. Selecting Operations (Cmdlets) .....	62
Using the PowerShell Wizard .....	62
PowerShell Wizard Operations and Flows .....	62
PowerShell Script Operation .....	65
Connection Inputs .....	66
Additional Modules and Snapins .....	68
PowerShell Script and cmdlet Inputs .....	68
Formatting the Result .....	69
Running a PowerShell Script on a Localhost .....	69
Running PowerShell Scripts from a File .....	71
Loading PowerShell Functions from Files .....	72
Running a PowerShell Script on a Remote Host .....	73
Formatting the Result .....	74
Running Multiple PowerShell Cmdlets Scripts in the Same PowerShell Session .....	76
Assigning the Result of One Cmdlet as a Parameter to Another Cmdlet .....	79
Solution 1: Create a New PowerShell Script Step .....	83
Solution 2: Run a PowerShell Script in the Generated Flow Context .....	86
Solution 3: Use Generated Flows Only and Minimize the User Effort .....	88
PowerShell Remoting .....	88
Overview .....	88
Enabling Remoting Using GPO (Group Policy Objects) .....	88
Group Policy Configuration for a Group of Servers .....	90
Enabling Remoting for Non-Administrative Users .....	93
Authentication Types .....	94
Troubleshooting .....	97
Could not connect to the host .....	97
The wizard fails to load modules on a x64 localhost. ....	97
The user has exceeded the maximum allowed number of remote shells .....	97

# Using the HPE OO Wizards

HPE OO Studio comes with five wizards:

- The REST wizard enables administrators to create HP Operations Orchestration (HPE OO) flows based on URL to Resource Listing or local API Definition files.
- The Third Party Content Pack wizard enables authors to generate a new third-party content pack with any required third party jars, which are not delivered with the out-of-the-box content packs.
- The Web Services wizard creates HPE OO flows based on the API described in the Web Service Definition Language (WSDL) of the web service that you identify.
- The Shell wizard enables administrators to create HPE OO flows that are integrated with Shell.
- The PowerShell wizard enables authors to generate HPE OO flows from the selected PowerShell cmdlets found in a list of modules/snapins.

# Using the REST Wizard

## Purpose of the REST Wizard

This integration enables administrators to create HPE Operations Orchestration (HPE OO) flows based on URL to Resource Listing or local API Definition files.

The REST Wizard supports the Swagger, RAML and WADL formats for RESTful web service.

Swagger is a specification and a complete framework implementation for describing, producing, consuming, and visualizing RESTful web services. Swagger is JSON-based. The main goal of Swagger is to enable client and documentation systems to update at the same pace as the server. The documentation of methods, parameters, and models are tightly integrated into the server code, allowing APIs to always stay in sync.

The Web Application Description Language (WADL) is a machine-readable XML description of HTTP-based web applications (typically REST web services). WADL models the resources provided by a service and the relationships between them. WADL is intended to simplify the reuse of web services that are based on the existing HTTP architecture of the web. It is platform- and language-independent and aims to promote reuse of applications beyond the basic use in a web browser. It was designed to provide an alternative to the WSDL.

The RESTful API Modeling Language (RAML) is a YAML-based language for describing RESTful APIs. It encourages reuse, enables discovery and pattern-sharing, and aims for merit-based emergence of best practices.

For more information about creating HPE OO flows, see the *HPE OO Studio Authoring Guide*.

This document explains how this integration has been implemented, and how the integration's operations and flows communicate between HPE OO and RESTful web services.

## Audience

This guide is intended for system administrators who establish and maintain the implementation of the integration between RESTful web services and HPE OO. This guide assumes that you have administrative access to both systems.

## Supported Versions

Content Pack	Operations Orchestration Version	REST Wizard Version
Base Content Pack	10.x	10.20 and later

## Getting Started with the REST Wizard

### Installing the REST Wizard

The wizard is installed if Studio is selected from the Operations Orchestration installer. The REST Wizard is located under **<OOInstallPath>\studio\tools**.

### REST Wizard Logs

The logs are located in the **<OOInstallPath>\studio\tools\logs\rest-wizard.log** file.

### Uninstalling the REST Wizard

The wizard is uninstalled when Studio is uninstalled.

### REST Wizard System Requirements

The following are the minimum software requirements for systems running REST Wizard for HPE Operations Orchestration:

- The environment must have Java SE Runtime Environment 8 (also known as JRE) installed (for running the wizards).



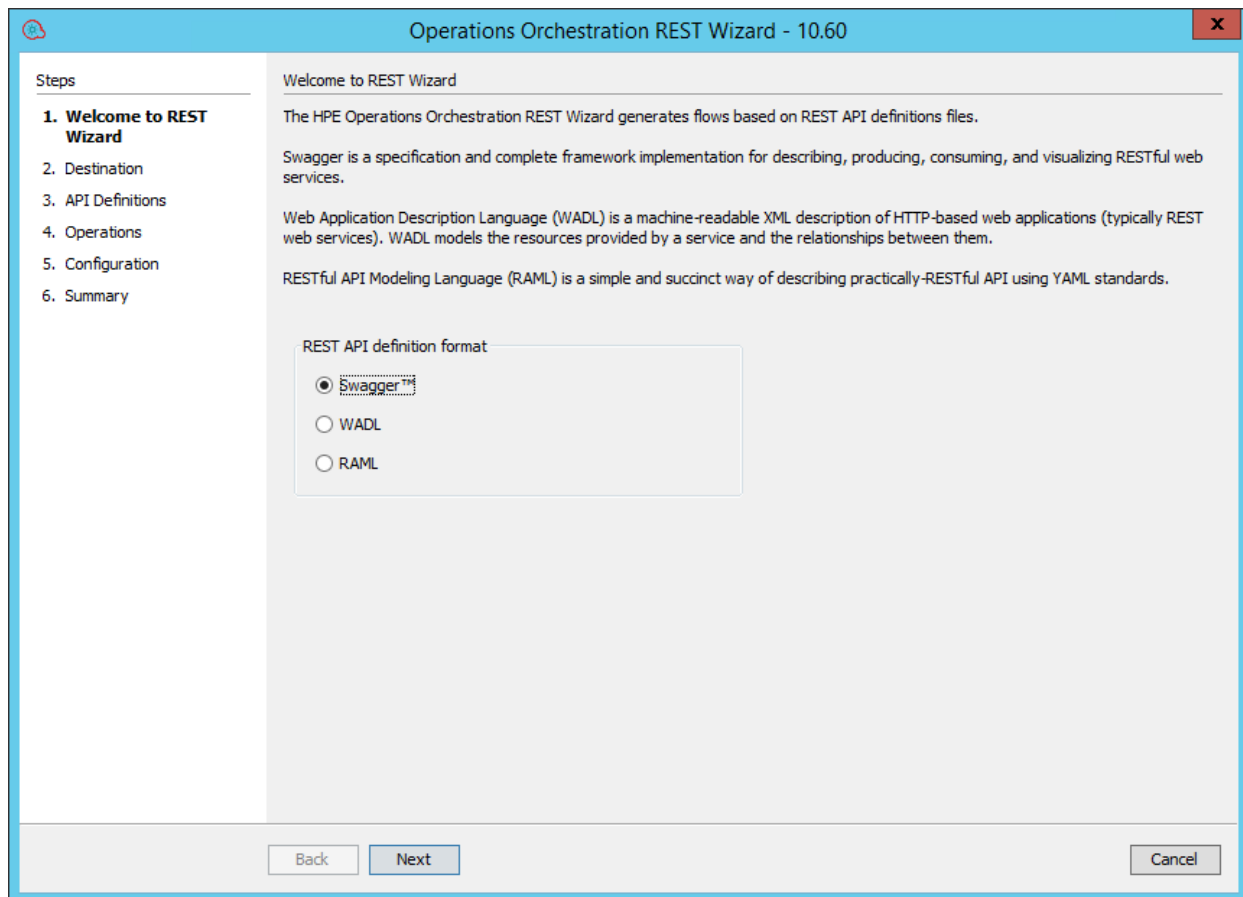
## REST Wizard Steps

To start the REST wizard, run the **rest-wizard.bat** file. This section describes the steps that you have to perform.

### Step 1. Welcome Page

When you start the REST Wizard, the **Welcome** page opens.

Select **Swagger**, **WADL**, or **RAML**.



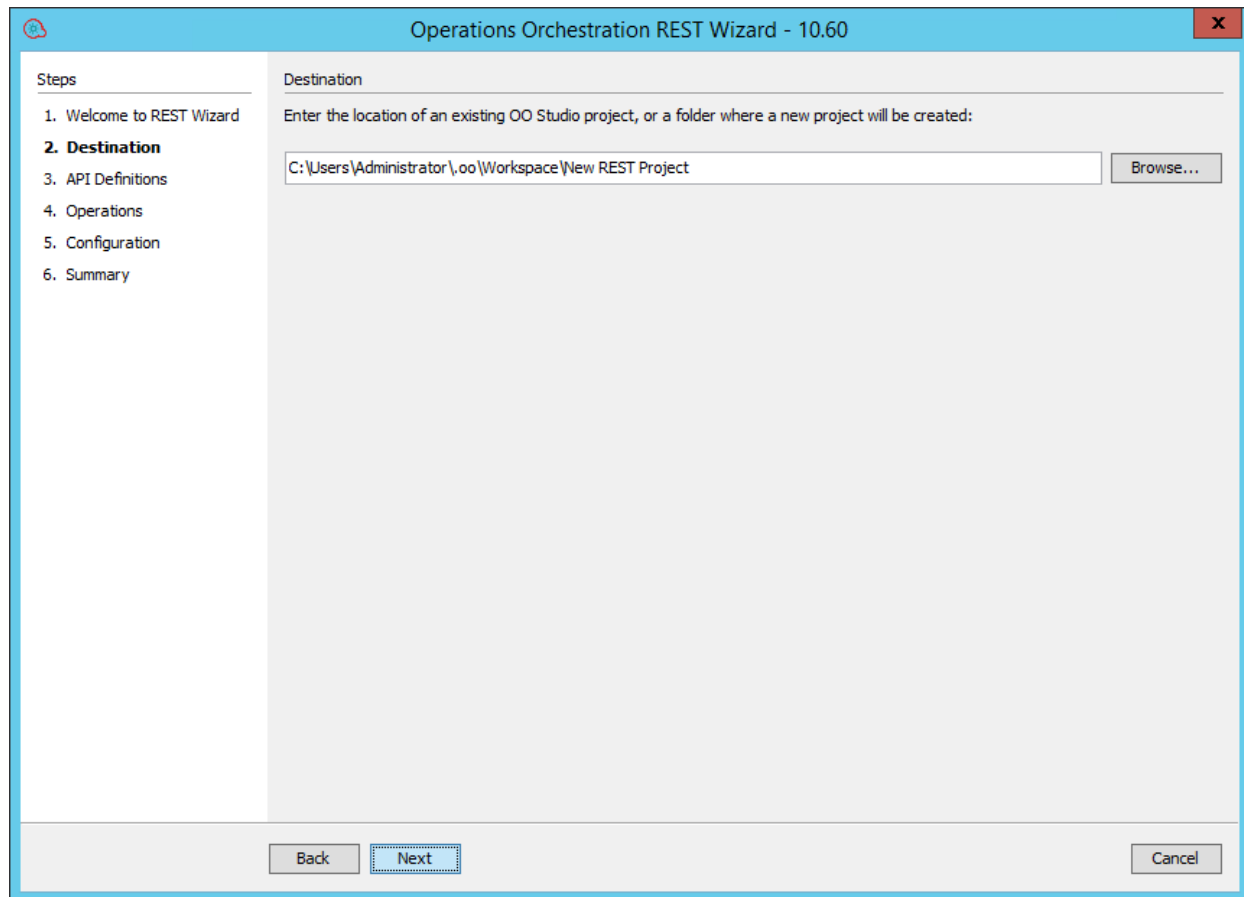
Click **Next** to open the **Destination** page.

### Step 2. Destination

In this step, you select the location of an existing Studio project or a folder where the new project will be created.

Enter or select a location of a Studio project for the flows you want to create and then click **Next**.

If this is not the first time you have run the wizard, the location is populated with the path to the last project created.



## Step 3. API Definitions

The **API Definitions** page will vary, depending on whether you selected Swagger or WADL format.

Enter the base URL to the Swagger or WADL resource listing or click **Browse** to select one or more files that contain REST API Definitions in a Swagger or WADL format. When the base URL is entered, the **Next** button is enabled.

If the base URL is protected by authentication, specify the **Username**, **Password**, and **Authentication** type. The currently supported authentication types are **Basic**, **Digest**, and **Anonymous**.

If the HTTP request to the base URL is done through a proxy, specify the proxy settings. The currently supported proxy authentication types are **Basic**, **Digest**, and **Anonymous**.

(Swagger only): If the **Resolve relative references according to the RFC3986** check box is selected, the relative paths discovered in the Swagger file found at the provided URL will be automatically resolved, and the HTTP requests will be made at the correct resource listening paths. If this check box is not selected, it is possible that for relative paths discovered, the resource listening paths will be incorrect and an exception containing the message "Invalid URL to a Swagger file" will be thrown.

Operations Orchestration REST Wizard - 10.60

Steps

- Welcome to REST Wizard
- Destination
- API Definitions**
- Operations
- Configuration
- Summary

API Definitions

URL to the Swagger Resource Listing (e.g. <https://corporateWebSite.com/api/api-docs>), or select local files:

Authentication

Username:

Password:

Authentication type:  ▼

Trust all roots

Trust keystore

Keystore password

Proxy

Host:

Port:

Username:

Password:

Authentication type:  ▼

## Step 4. Operations

On the **Operations** page, all the resources discovered in the previous page are displayed: the resource path, the HTTP method available for the resource and a brief description of the resource.

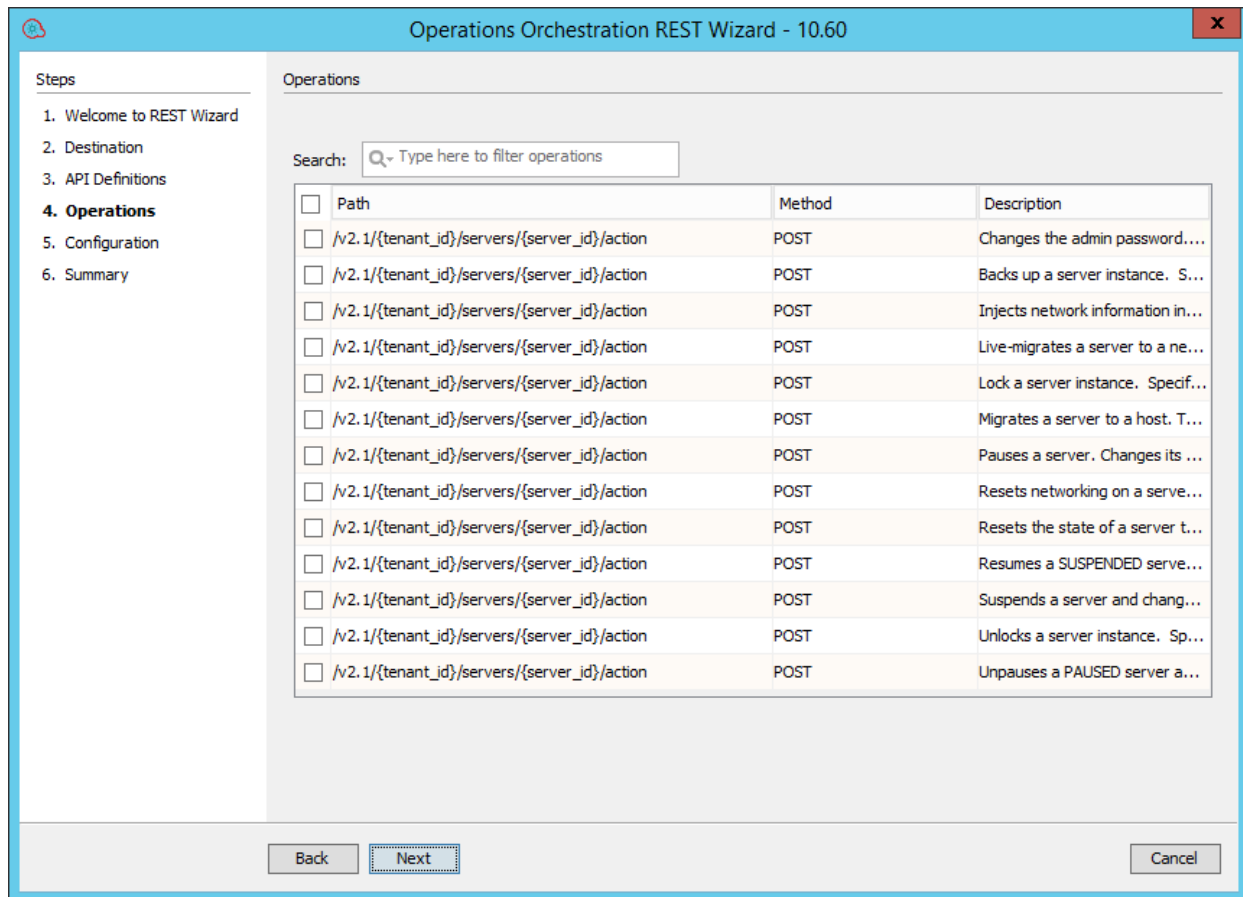
For the selected resources, the OO flows will be generated and will contain an operation that performs an HTTP request based on the HTTP method available for each resource.

Enter text in the top **Search** field, to filter the resources table by this text.

Select one or more resources.

**Note:** To select multiple operations, hold down the CTRL key.

Click **Next**.



## Step 5. Configuration

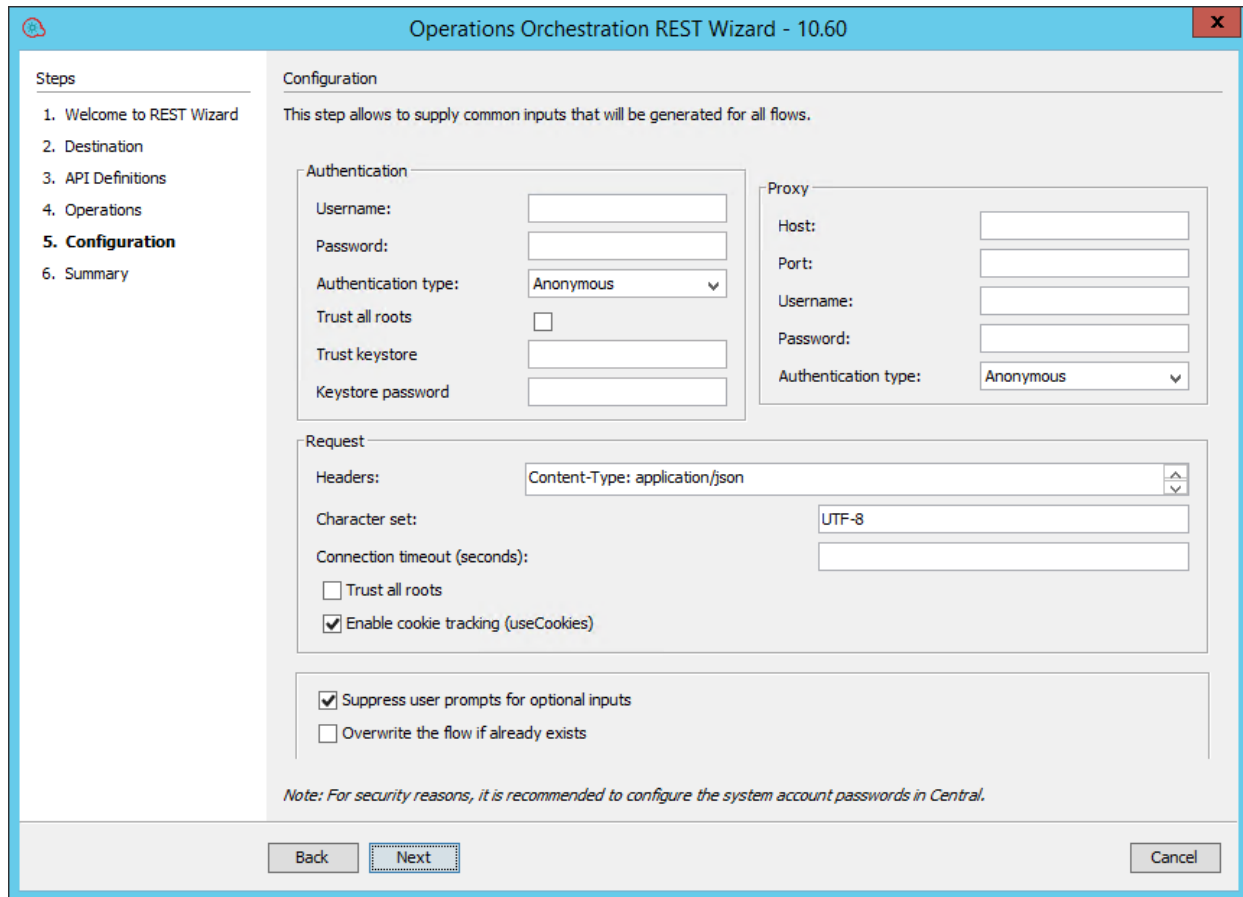
All the values provided in this step will be used as inputs for the generated flows and operations.

- In the **Authentication** section, if the **Username** and **Password** are provided, a new system account will be generated under the **New REST Project\Configuration\System Accounts** folder and the user name and password inputs of generated flows will reference this system account.
- For the **Proxy user name** and **Proxy password** pair, a new system account will be generated with the **Proxy** suffix, following the same rule as for the **Username** and **Password** from the **Authentication** section.
- The currently supported authentication types are: **Basic**, **Digest**, and **Anonymous**.
  - If the selected authentication type in both the **Authentication** section and the **Proxy** section is **Anonymous**, the **authType** input from the generated flows will be initialized with an empty constant.
  - If the selected authentication type in the **Authentication** section is **Basic** and the authentication type from the **Proxy** section is **Digest**, the **authType** input from the generated flows will be initialized with a **basic,digest** constant.
- In the **Headers** text box, you can add multiple headers in the generated flows, in the following format: headerName1=headerValue1\nheaderName2=headerValue2. Use the new-line character to separate the headers.
- If the **Trust all roots** or **Enable cookie tracking** check boxes are selected, the **trustAllRoots**, and

**useCookies** input from the generated HTTP operation will be set to **true** and the **x509HostnameVerifier** input will be set to **allow\_all**.

Otherwise, these inputs will be set to **false** and **x509HostnameVerifier** will be set to **strict**.

- If the **Suppress user prompts for optional inputs** check box is selected, the non-required operation parameters discovered from the API Definitions document will be generated as flow inputs set to use constants with empty values. Otherwise, these flow inputs will be marked as prompts.
- Select the **Overwrite the flow if already exists** check box if the new generated flow should overwrite an existing flow from the same location specified in "[Step 2. Destination](#)" on page 9.
- Some Swagger or WADL files do not have the base URL set correctly. In this case, after the project is imported into Studio, check the **<ProjectName>/Configuration/Sytem Properties** folder, to see if the generated **System Properties SP\_<ProjectName>\_<ResourceName>\_Url** value is initialized correctly.

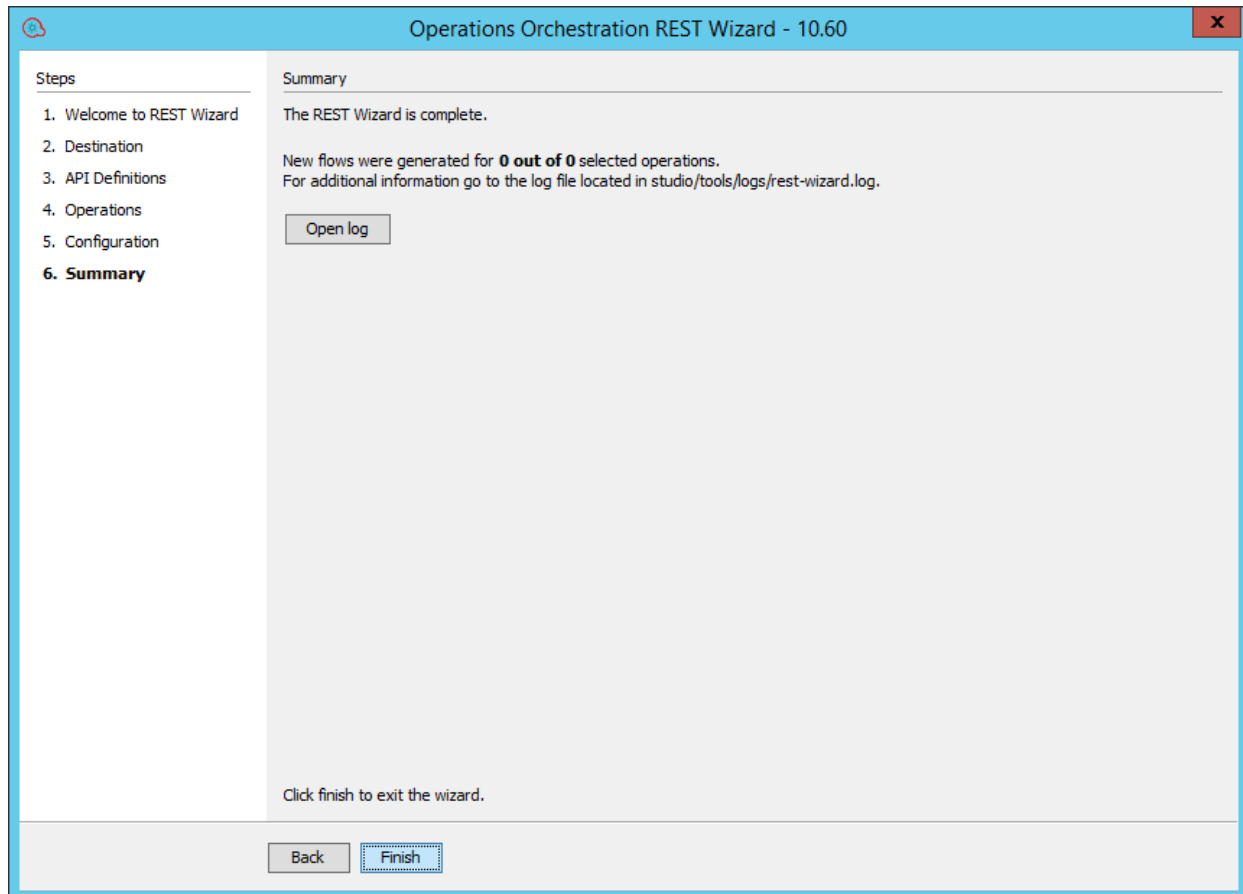


Click **Next** for the **Summary** page.

## Step 6. Summary

This step contains information about the successfully generated flows number from the selected number of resources.

For more information about the flow generation, click **Open log** to view the log files.



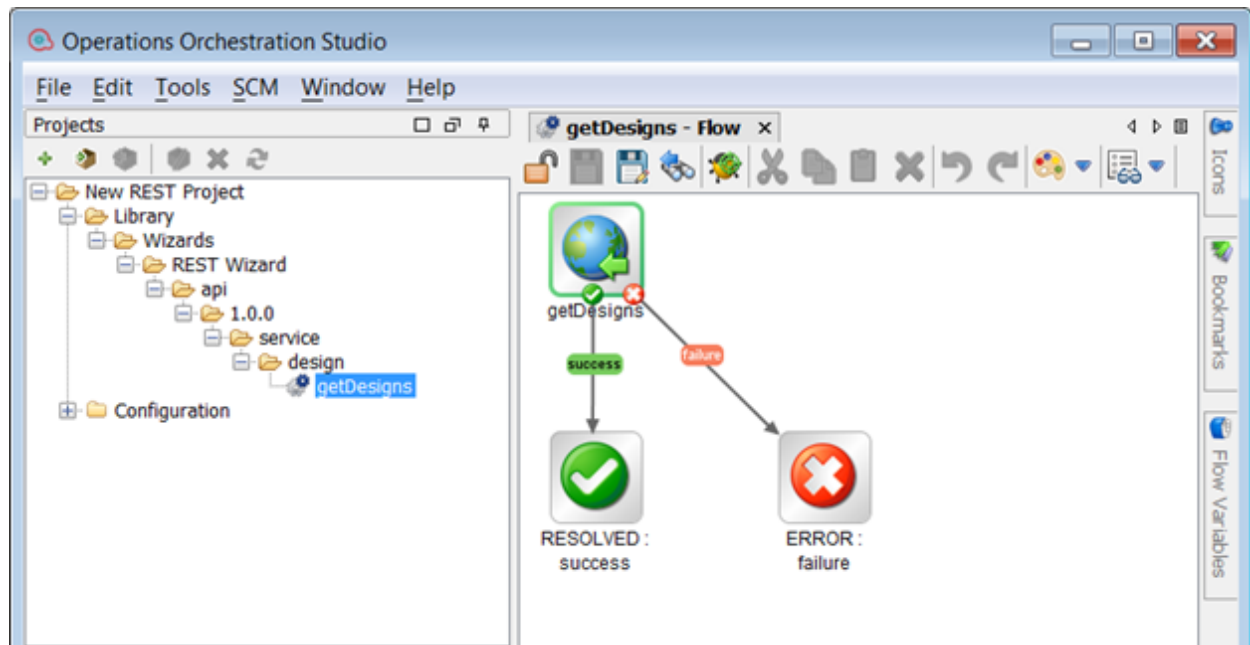
## Importing the Generated Flows into Studio

Before importing the generated flows into Studio, import the project generated by the wizard in Studio. See the "Managing Projects" section in the *HPE OO Studio Authoring Guide* to see how to import a project.

**Note:** In order to correctly run flows generated the REST Wizard, clients have to update the **Service\_Url** from the System Properties with appropriate request address.

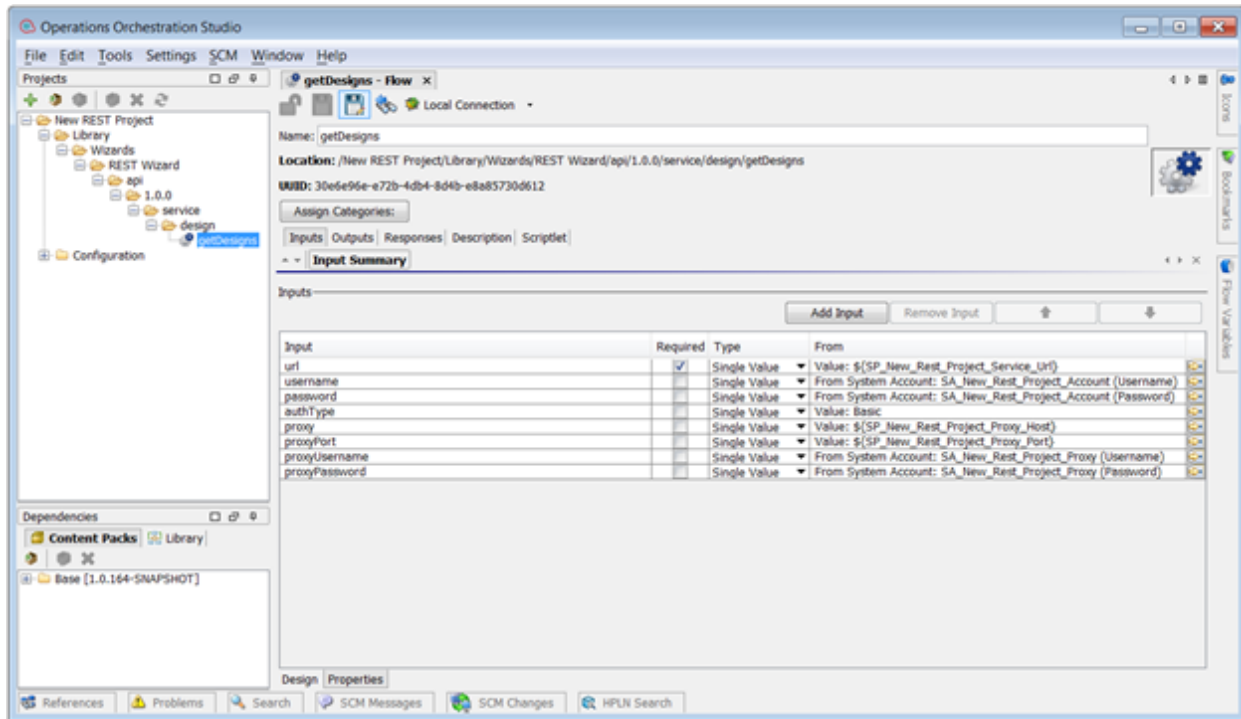
## Generated Flows

The REST Wizard generates a flow with the name specified in the Destination step of the wizard. If a project with the same name already exists, the new flow is added to it.



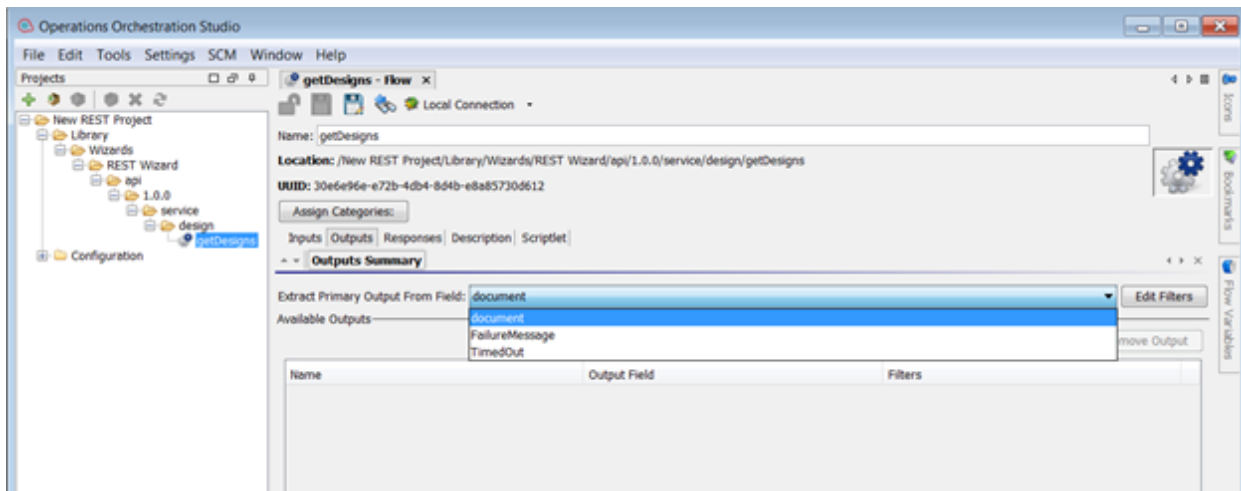
## Inputs

Each flow has the following inputs, which are common to the REST operation:



## Outputs

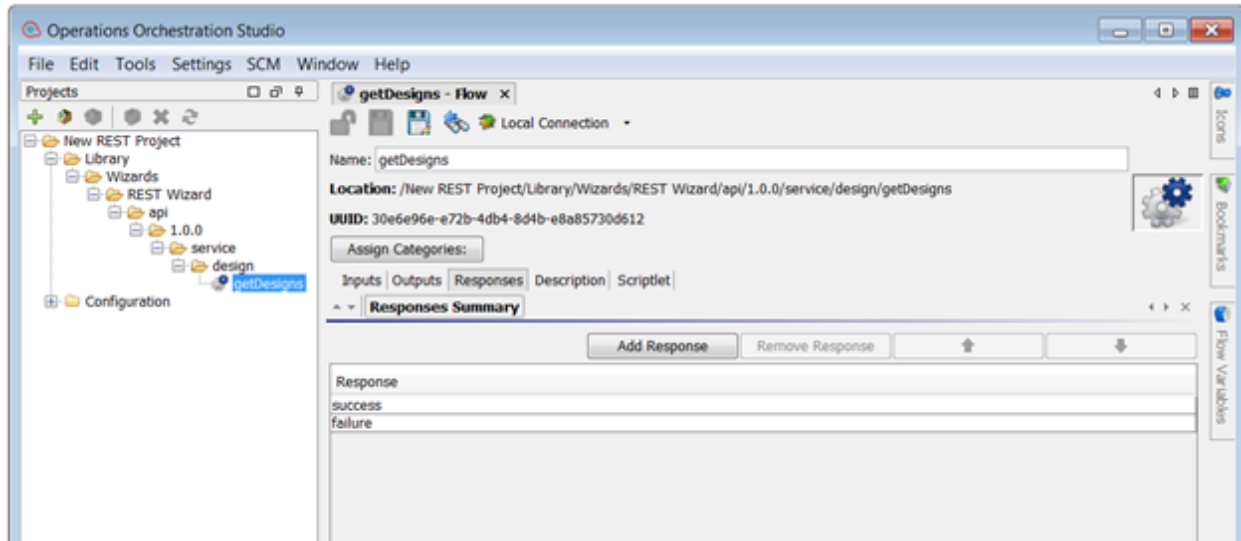
On the **Outputs** tab, the flow's primary result is **document**. All HPE OO flows have the **FailureMessage** and **TimedOut** outputs.



## Responses

The success and failure responses of the flows are the same as the REST operations.

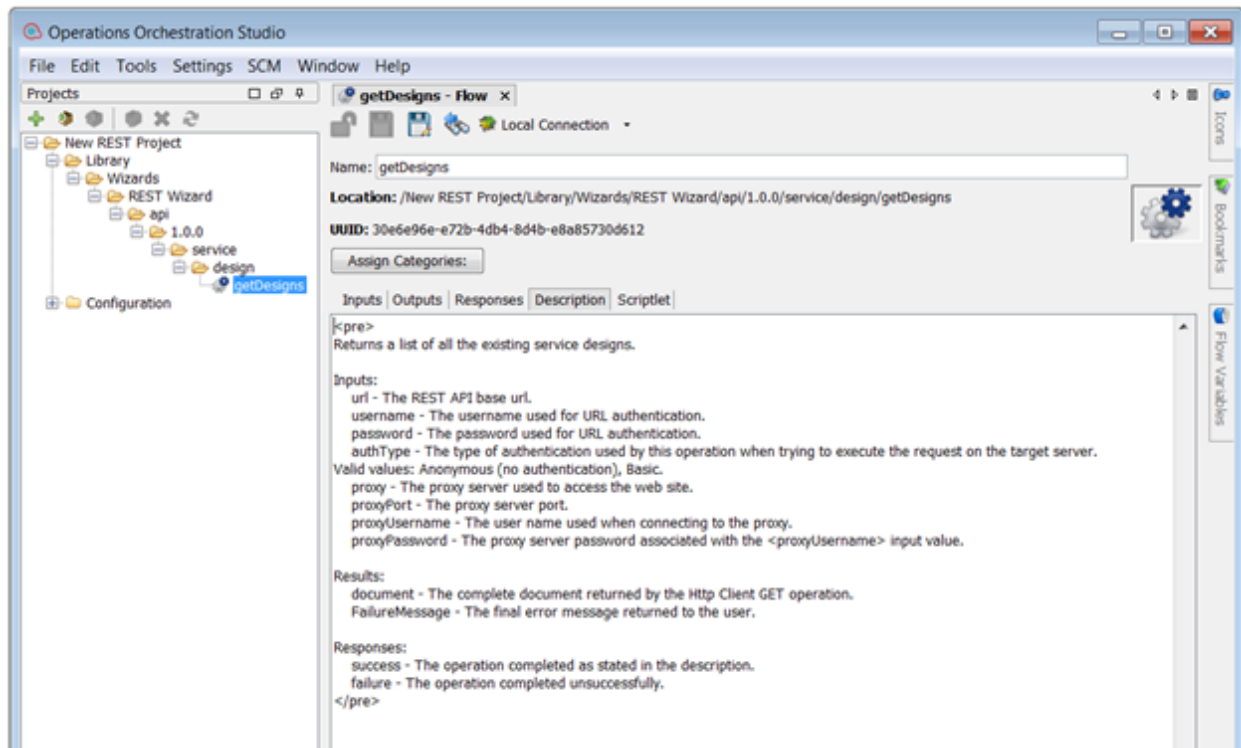




## Descriptions

The description of each generated flow contains the following items:

- The description of the flow
- The description of the flow inputs
- The description of the results
- The description of the responses



## REST Operation

This operation is used to execute a HTTP request on a target host, either local or remote.

### Operation Inputs

Input	Required	Type	From
url	✓	Single Value	Value: \${url}
username		Single Value	Value:
password		Single Value	Value:
authType		Single Value	Value:
kerberosConfFile		Single Value	Value:
timeout		Single Value	Value:
socketTimeout		Single Value	Value:
useCookies		Single Value	Value: true
followRedirects		Single Value	Value:
proxy		Single Value	Value:
proxyPort		Single Value	Value:
proxyUsername		Single Value	Value:
proxyPassword		Single Value	Value:
encoding		Single Value	Value:
userAgent		Single Value	Value:
characterSet		Single Value	Value: UTF-8
trustAllHosts		Single Value	Value:
keystore		Single Value	Value:
keystorePassword		Single Value	Value:
trustKeystore		Single Value	Value:
trustPassword		Single Value	Value:
headerNamesList		Single Value	Value: Content-Type
headerValuesList		Single Value	Value: application/json
queryNamesList		Single Value	Value:
queryValuesList		Single Value	Value:
delimiter		Single Value	Value:

### Operation Results

Name	From	Assign To	Assignment Action	Filters
document	document	Flow Output Field	OVERWRITE	No Filters
FailureMessage	FailureMessage	Flow Output Field	OVERWRITE	No Filters

## Troubleshooting

### Troubleshooting Overview

This section provides troubleshooting procedures and tools that you can use to solve problems you may encounter while using this integration.

## Unable to locate resource

Possible reasons are:

- Invalid URL to Swagger/WADL/RAML Resource Listing or API Definition local files.
- The connection to a Swagger/WADL/RAML Resource Listing URL must be done through a proxy.
- The URL to the resource is invalid based on the provided JSON/XML document.

# Using the Third Party Content Pack Wizard

With this wizard, you can generate a new third-party content pack with any required third party jars, which are not delivered with the out-of-the-box content packs.

## Purpose of the Third Party Content Pack Wizard

The main advantage of this wizard is automation.

This helps you to avoid repeating the same time-consuming process of creating the **third-party-cp.jar** content pack. You can perform the steps in the wizard as an alternative to using the steps described in **oo10-third-party-cp-release-notes-en.pdf**.

## Supported Versions

Operations Orchestration Version	PowerShell Wizard Version
10.2x and later	10.20 and later

## Getting Started with the Third Party Content Pack Wizard

### Installing the Third Party Content Pack Wizard

The wizard is installed if Studio is selected from the Operations Orchestration installer.

### Third Party Content Pack Wizard Logs

The logs are located in the `<OOInstallPath>\studio\tools\logs\third-party-cp-wizard.log` file.

### Uninstalling the Third Party Content Pack Wizard

The wizard is uninstalled when Studio is uninstalled.

### Third Party Content Pack Wizard Requirements

The following are the minimum software requirements for systems running Third Party Content Pack Wizard for HPE Operations Orchestration:

The environment must have Java SE Runtime Environment 8 (also known as JRE) installed (for running the wizards).

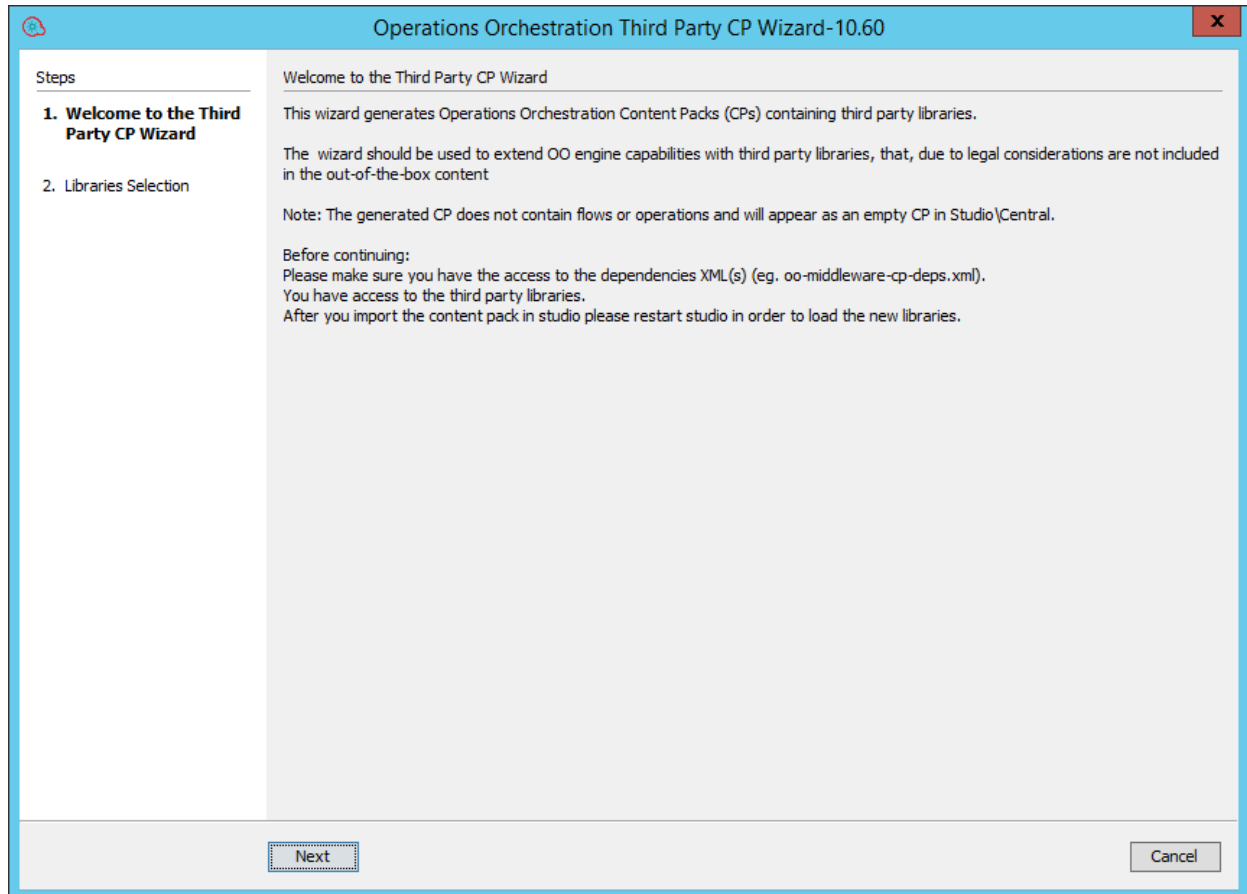
## Third Party Content Pack Wizard Steps

To start the wizard, run the **third-party-cp-wizard.bat** file.

## Step 1. Welcome Page

When you start the Third Party Content Pack wizard, the **Welcome** page is displayed.

Click **Next** to continue.



## Step 2. Libraries Selection Page

In this step, you will import an XML file containing the third party definitions that are needed to generate the third party content pack. This is a standard file that is required for a content pack that has missing third party files, which cannot be delivered with the content pack.

Each entry in the file should have the following format:

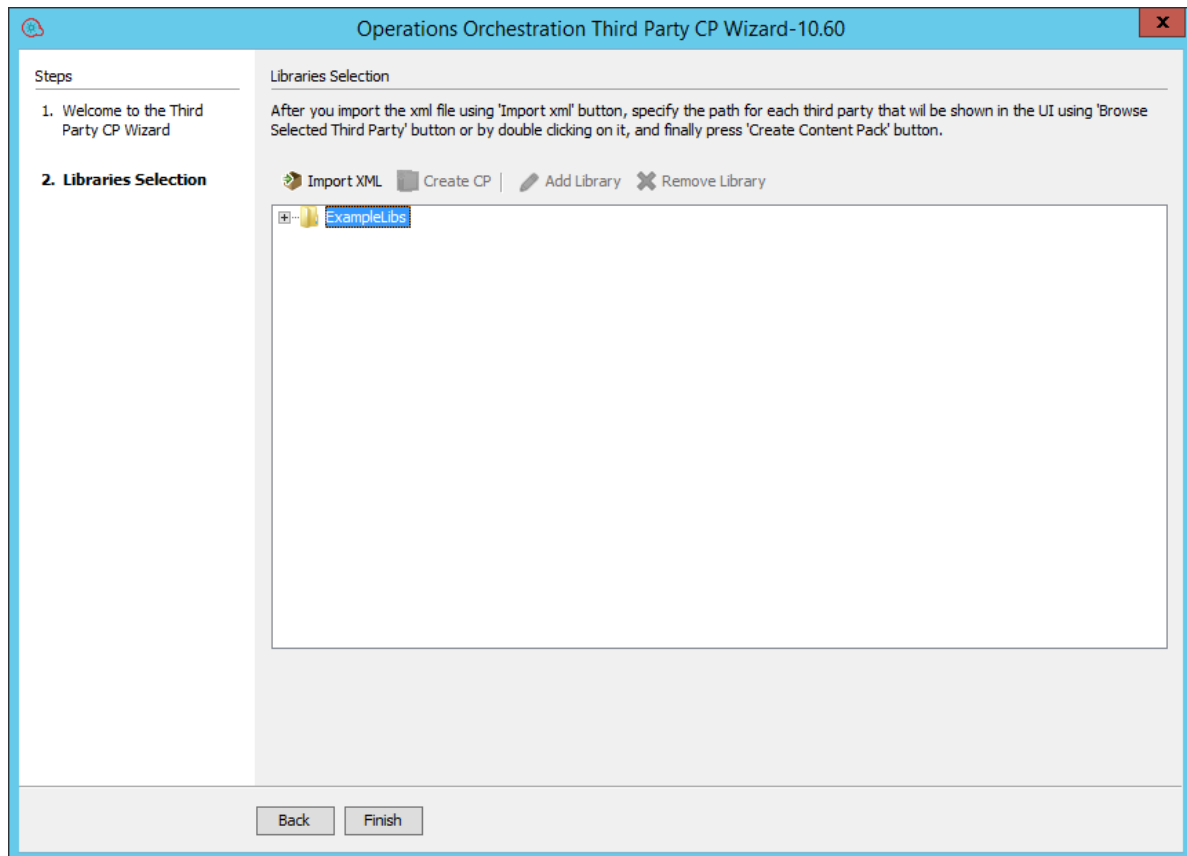
```
<dependencies>
...
  <dependency>
    <groupId>weblogic</groupId>
    <artifactId>weblogic</artifactId>
    <version>10.3</version>
  </dependency>
...
```

```
</dependencies>
```

Based on these entries, Studio and Central will validate a content pack with those missing dependencies.

Currently, there are three XML lists for **oo10-base-cp**, **oo10-middleware-cp**, and **oo10-sap-cp**.

1. Click the **Import XML** button to import one or more third party lists.

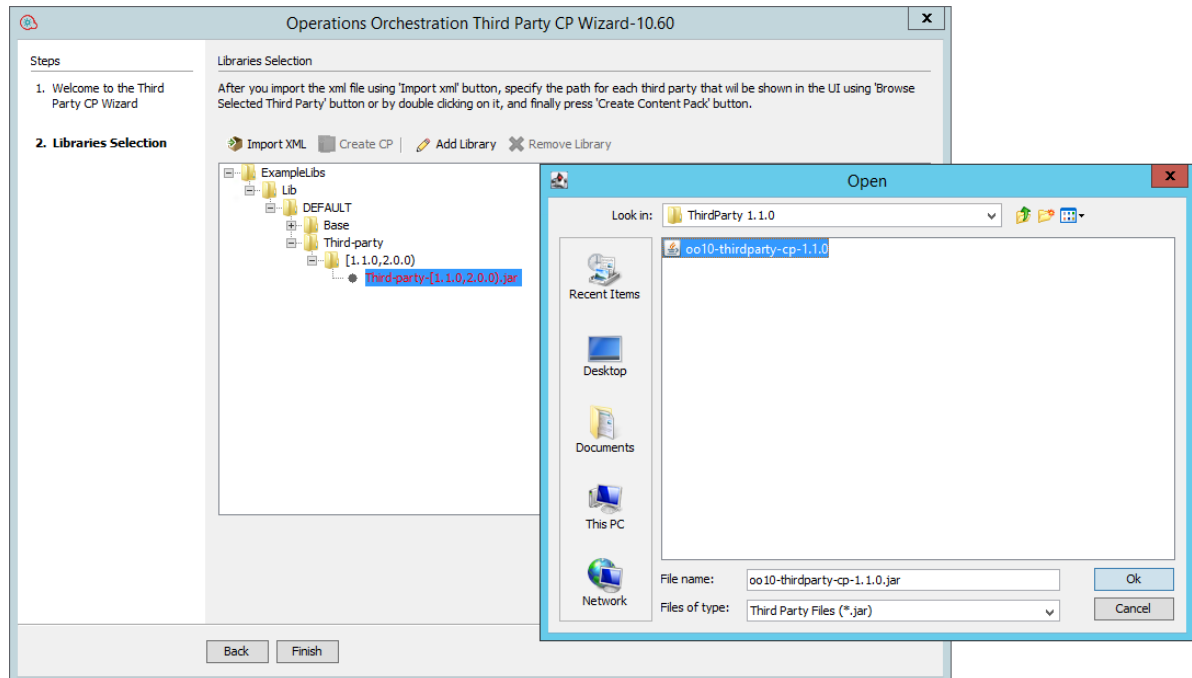


2. Click the **+** button to expand the list.

**Note:** Alternatively, you can use the **Shift + Space** shortcut.

3. After the list is expanded, select an item, click the **Edit** button, and select the path to the selected third party file.

**Note:** The **Edit** button is enabled only when a third party item is selected.

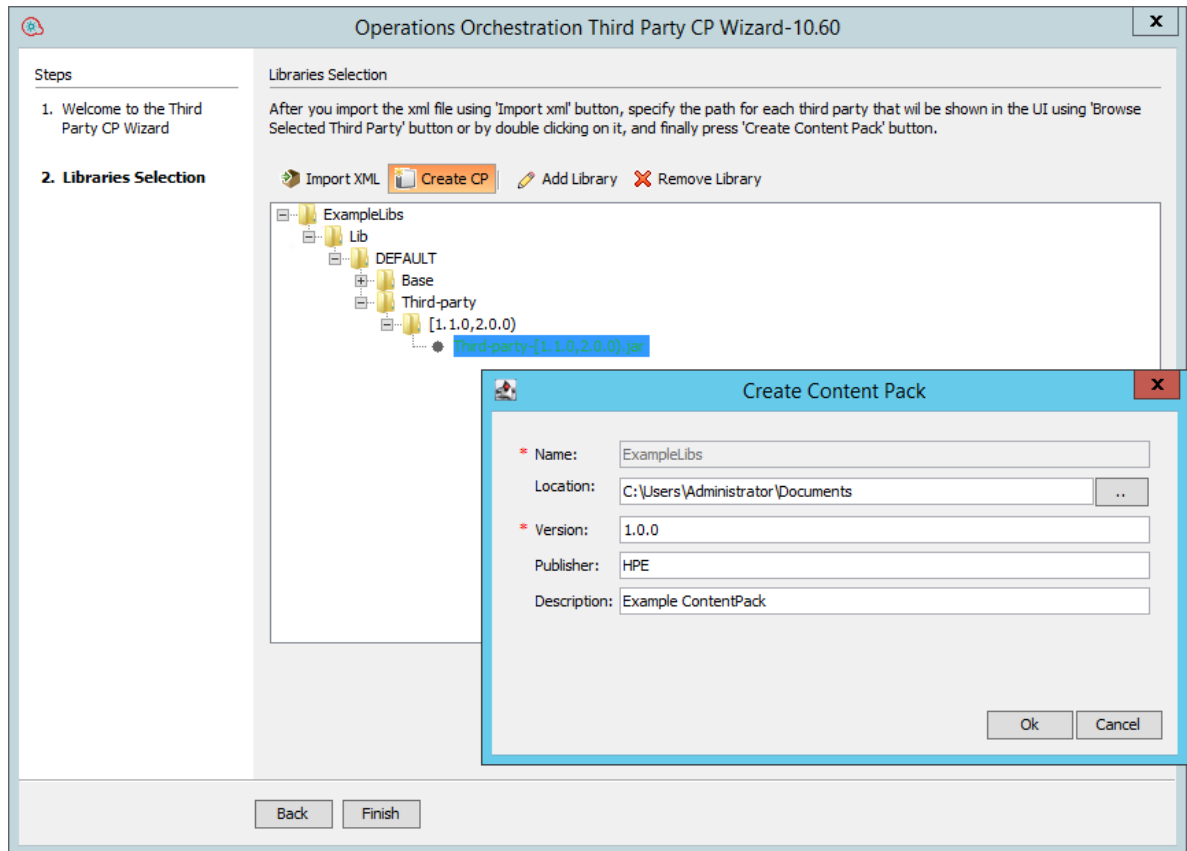


**Note:** You can also double-click any third party item and provide the path.

- By default, when the list is loaded, all items are red. When you give an item the path to a third party file on the disk, the item changes color to green. In most cases, you will need to provide all required third party files to the list. However, there are cases where this is not needed, because you are using only some specific third party files from a content pack. In that case, you will just need to provide the required third party files.

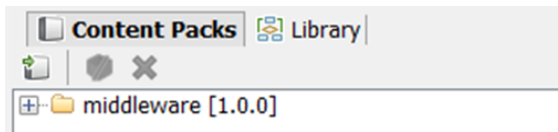
After you have provided a real path for all the third party files that you need, click the **Create CP** button to continue with the generation of the content pack.

**Note:** You need to select the root element of the corresponding list to enable the **Create CP** button.



- In the Create Content Pack dialog box, enter the content pack name, location, and version (the same details as in the corresponding Studio dialog box). It is optional to enter information about the publisher and to add a description of the content pack.

The result is a content pack that can be imported into Studio or Central. For example:



**Note:** In Central, you will not see a deployed content pack.

## Third Party Wizard Command Line Tool

The wizard lets you use a command line tool to generate a content pack from the command line.

To start the wizard in command line mode:

- Navigate to the wizard jar location at `<OOInstallPath>\studio\tools\lib` and start a command line here.
- Run the following java command:

```
java -jar third-party-cp-wizard-jar-with-dependencies.jar -help.
```

After using this command, you will see a list of arguments that can be passed to the wizard in order to generate a content pack.



The most important constraint in command line generation is that all libraries need to respect the format [artifactId]-[version].[type] that is present in the XMLinput list. The default type format is jar.

If an entry in the XML file is in the format shown below, you need to provide a library in the `-third-parties-paths` argument using the `artifactId-version.type` format. In this case, you will need to have the **weblogic-10.3.jar** on disk. This helps the wizard to map between the XML list provided in the `-xmlPath` argument and the items provided in the `-third-parties-paths` argument. In the `-third-parties-paths` argument, you will provide the real paths of the jars needed by the wizard.

```
<dependency>
  <groupId>weblogic</groupId>
  <artifactId>weblogic</artifactId>
  <version>10.3</version>
</dependency>
```

## Troubleshooting

If a content pack cannot be generated, check whether:

- The third party files on the disk are locked or in use
- The path where you want to save the content pack is locked
- The content pack jar file is already generated and locked

By default, if you try to generate a content pack with the same name as an existing one, the original content pack is overwritten by the wizard.

# Using the Web Services Wizard

## Overview of the Web Services Wizard

When you run the Web Services Wizard (`wswizard.exe`), you provide it with the WSDL for a web service. The Web Services Wizard creates OO flows based on the API described in the Web Service Definition Language (WSDL) of the web service that you identify in the wizard. The WSDL string you provide as a pointer can be a file's location and name or a URL.

The Web Services Wizard helps you create OO flows when:

- An HPE OO integration does not exist.
- An HPE OO integration does exist, but the customer has modified the application. For example, a customer using Remedy may have modified a form or added a field. To take advantage of the customer's modifications, the Remedy web service is updated. You can use the Web Services Wizard to create HPE OO flows from the modified web service.
- If a new version of an application with an HPE OO integration is released and the integration content does not support the new version, you can use the Web Services Wizard to create new HPE OO flows.

### Example

You have an application named MyAlert that creates a ticket through a web service and API, and you want to tell MyAlert to create a ticket. The Web Services Wizard extracts the application's APIs from the Web service's WSDL for the actions that can be performed with the application, such as creating or changing a ticket. The WSDL defines the web service's methods, the inputs for each method, and the required format for each input.

When you provide the wizard with the WSDL (in our example, for MyAlert) and run the wizard, it generates flows that can run against the web service. All flows created using the Web Services Wizard have a single step that is built from the Invoke Method 2 operation in the **Library/Operations/Wizards/Web Services Wizard** folder from the Base Content Pack. The flows are created in the project location folder specified by the user. Running the flows requires a Remote Action Service (RAS) that has access to the web service. For information on creating and configuring RAS references, see "Configuring Group Aliases" in the *Studio Authoring Guide*.

## System Requirements

The minimum software requirements for systems running Web Services Wizard for HPE Operations Orchestration:

- The environment must have Java SE Runtime Environment 8 (also known as JRE) installed.

## Installing the Web Services Wizard

The Web Services wizard is automatically installed if Studio is selected in the Operations Orchestration installer.

## Web Services Wizard Code Dependencies

The wizard does not have any dependencies. All third parties are encapsulated into the executable files.

## Configuring Logging Settings

The configure logging settings are no longer supported in the 10.x wizard.

## WS Wizard Enhancements from 9.x

- The wizard now appears in the taskbar and can be closed, minimized, or brought to the front.
- The UI is correctly divided. The scroll bar is now not necessary.
- Operation selection has a search functionality which allows you to quickly find an operation by typing letters of the operation name.
- The wizard includes functionality to override an existing flow (or flows).
- The wizard supports the use of credentials when retrieving the WSDL.
- The flows created by the wizard support HTTP headers and outputs.

## Downloading HPE OO Releases and Documents on HPE Live Network

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

To download HPE OO releases and documents, go to <https://hpln.hp.com/>

This site requires that you register for an HPE Passport and sign-in.

To register for an HPE 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.

On the HPE Live Network page, click **Operations Orchestration Community**.

The Operations Orchestration Community page contains links to announcements, discussions, downloads, documentation, help, and support.

1. On the left-hand side, click **Operations Orchestration Content Packs**.
2. In the **Operations Orchestration Content Packs** box, click **Content**. The HP Passport sign-in page appears.
3. Enter your HP Password User ID and Password and click **Sign-in**.
4. Click **HP Operations Orchestration 10.x**.
5. Search for **Base Content Pack**.

## Wizard Processing Details

### How the Web Services Wizard Uses SoapUI

SoapUI is an open-source web service testing tool. It provides web service inspection, invoking, development, and simulation. The Web Services Wizard uses SoapUI to parse the WSDL and create a template SOAP request.

This template is an XML file with placeholder tokens that are replaced with real data in order to make a request to the server. If you run SoapUI manually and create a project referencing a WSDL, you will see it create these request templates in the tree as nodes named **Request 1** for every operation in the WSDL. This is the template that the Web Services Wizard receives from SoapUI, and uses to populate the **xmlTemplate** input.

Similarly, the Web Services Wizard (in OO versions 9.00 and later) retrieves a SOAP response template with tokens that indicate how the response will look. This is a little more difficult to reproduce in the SoapUI GUI, as it requires creating a Mock Response and then using the Open Editor function to look at the XML.

For OO 9.x, support was added to specify a web proxy via the properties **http.proxyHost** and **http.proxyPort** in the **ws.properties** file in the OO Home folder under **/Studio/tools/conf/**. You only need to enter the configuration information once (the first time you run the wizard against a WSDL outside the firewall). It is then read from the **ws.properties** file and pre-populated in the wizard GUI. You can change it in the file or in the wizard GUI and the values are saved for the next time you run the wizard.

For OO 10.00 and later, specifying a web proxy in the **ws.properties** file is no longer supported. You can change it only from the wizard GUI.

After retrieving the templates for the request and the response, the WSDL is discarded. No further information is obtained from the WSDL, and all subsequent operations in both the Web Services Wizard and the Invoke Method 2 operation are based entirely on the templates returned from SoapUI.

## Processing Templates

The template processing logic parses through a SOAP template (either a request template or a response template) looking for tokens. It is called in different ways for different purposes—for processing the request template and for processing the response template:

- Locating input tokens in the request template to create the input map (in the wizard)
- Locating output tokens in the response template (in the wizard)
- Replacing input tokens with actual values to build the SOAP request (in the **Invoke Method 2** operation)

In all cases, the logic skips any leading XML elements until it finds an element whose namespace prefix is either **soap** or **soapenv** and whose element name is not **envelope**, and then begins with the content of that element; this effectively ends up arriving at the topmost element under the outermost Body element.

## Locating Inputs and Creating the inputMap

In the wizard, the request template is processed, and for each token that is found, a pipe-delimited value is returned indicating its path in the template, but with the outermost SOAP envelope information removed. For example, if the template looks like this:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Test>
      <Name>?</Name>
      <Address>?</Address>
    </Test>
  </soapenv:Body>
</soapenv:Envelope>
```

it returns the values **Test|Name** and **Test|Address**. Note that the whole path is needed, as an element (such as **Name**) may appear in more than one place in a template, and there needs to be a unique path to each.

If, during this input processing, the wizard encounters a comment that indicates that it is at the beginning of an array ("x or more repetitions" or "m to n repetitions"), the value zero (0) is inserted at that point. For example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Test>
      <!--1 or more repetitions-->
      <Name>?</Name>
      <!--1 or more repetitions-->
      <Address>?</Address>
    </Test>
  </soapenv:Body>
</soapenv:Envelope>
```

This returns the values **Test|0|Name** and **Test|0|Address**. As arrays may be nested, there may be templates whose values contain more than one zero (0).

The next task is to define a meaningful set of input names to be created. This is done using an input map. An input map permits a user-friendly name to be associated with each value. For example, **Address** is mapped to **Test|Address** and **Name** is mapped to **Test|Name**. The **inputMap** input that is generated in the operation is a list of these mappings between pipe-delimited paths and user-friendly names. In the first example above, the **inputMap** contains:

```
Test|Name=Name
Test|Address=Address
```

The creation of the **inputMap** is a little complex. Use the following tips when determining a name for each path name:

- Use the last part of the path (for example, **Name** or **Address**) if it is unique within the template.
- Avoid using a friendly name that is already one of the input names to the **Invoke Method 2** operation, such as **xmlTemplate**.
- If there are duplicate names, add a prefix for additional levels (with a period separator) onto the user-friendly name to make the name unique. For example, if the template yielded **One|Name** and **Two|Name**, the following input map would be created:

```
One|Name=One.Name
Two|Name=Two.Name
```

This is because both would otherwise map to the same value of **Name**.

- Single zeros in the pipe-delimited path (indicating the beginning of an array) are replaced with wildcards (\*). The position of the wildcard in the user-friendly name is moved to the end of the next element. In the above example, for **Test|0|Name** and **Test|0|Address**, the following input map would be created:

```
Test|*|Name=^Name*$
```

```
Test|*|Address=^Address*$
```

**Note:** The purpose of moving the wildcard position is to allow more intuitive input names like **Name0** and **Name1**.

The value on the right side of the equal sign for array types is surrounded by the **^** and **\$** symbols as a workaround for resolving the issue of parameters having similar names. These values are used as regex patterns for array types and similarly-named parameters without these symbols corrupting the algorithm.

- The simplification of friendly names (see the first bullet in this list) only applies to the part of an array to the left of the wildcard; all elements to the right will remain. For example, the items Test|0|Extra|Stuff|Name and Test|0|Extra|Stuff|Address results in:

```
Test|*|Extra|Stuff|Name=^Extra*.Stuff.Name$
Test|*|Extra|Stuff|Address=^Extra*.Stuff.Address$
```

This is regardless of the fact that the **Extra** and **Stuff** are otherwise unnecessary.

The wizard then uses the **inputMap** to create step-level and flow-level inputs for each item in the map. Any occurrences of wildcards are replaced with zeros in the input names. If the flow developer wants to provide additional elements (beyond just the 0th), s/he needs to add them both as step level inputs and flow level inputs. Using our previous example:

```
Test|*|Name=^Name*$
Test|*|Address=^Address*$
```

**Name0** and **Address0** are created as inputs to the step and the flow.

The Web Services Wizard accepts JSON-formatted arrays for the array types found in the WSDL. So, instead of entering a new input for each element in the array, you can now enter a JSON-formatted array as the input value instead of creating additional inputs.

When you run the Web Services Wizard, you must check the Use JSON arrays for WSDL array type option on the Select operation(s) screen. This will add the input field "usesJSON" with a value of "true" to the created **Invoke Method 2** step. Then for the inputs, use a JSON format array for the "0" element and the **Invoke Method 2** operation to create the required elements to send in the request.

For example, for an array structure defined by the following in the **xmlTemplate**:

```
<ns:AffectedCI type="Array">
  <!--Zero or more repetitions:-->
  <ns:AffectedCI type="String" mandatory="" readonly=""></ns:AffectedCI>
</ns:AffectedCI>
```

- The **inputMap** entry for this array must use the following wildcard format:

```
CreateChangeTask00Request|model|instance|middle|AffectedCI|*|AffectedCI=^AffectedCI*$
```

- The associated Web Services Wizard created AffectedCI0 input field JSON array formatted value should be similar to:

```
["CValue1","CValue2","CValue3"]
```

## Locating Outputs and Creating Operation Outputs

Locating outputs in the XML template uses the same logic as finding inputs, but instead of returning a pipe-delimited path, the process returns an XML XPath expression. This is nearly the same thing except with a slash as a delimiter rather than a pipe. There are, however a few differences:

- **/text()** is appended to the XPath in order to correctly extract the text of the simple elements. For example, the following template corresponds to the outputs **/Test/Name/text()** and **/Test/Address/text()**:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Test>
      <Name?</Name>
      <Address?</Address>
    </Test>
  </soapenv:Body>
</soapenv:Envelope>
```

- Nothing is appended to the XPath of array elements. This causes the entire portion of the XML document to be returned in a single output, and it is the flow developer's responsibility to use other operations (like XML or JSON ones) to extract the relevant items. This difference is due to the fact that arrays can become arbitrarily nested, and returning such structured data in a simple variable is not an easy task. For example, the following template yields just the single output **/Test**:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <Test>
      <!--1 or more repetitions-->
      <Name?</Name>
      <!--1 or more repetitions-->
      <Address?</Address>
    </Test>
  </soapenv:Body>
</soapenv:Envelope>
```

If JSON arrays are being used, an additional output named **jsonStripped** is populated with the SOAP response in a JSON-formatted string.

The wizard then creates step outputs for each output that was located in the template, assigning an XPath filter to each one (whose value was determined above). At this point, the wizard has completed its main lifting. The remainder of the process resumes when the flow is run, calling the **Invoke Method 2** operation.

## Populating InvokeMethod2 Default Values for All Operations

The Web Services Wizard allows setting **InvokeMethod2** inputs so that each operation created from the WSDL has the inputs set by default. For example, the timeout input can be the same for all web service operations and setting the value once in the wizard will, in turn, set the timeout input value for all operation(s) selected on the selection page. Setting the default values in the Web Services Wizard is optional.

The Web Services Wizard does not validate the default inputs entered. This validation takes place during the flow run. The Web Services Wizard allows you to specify default values only for the authentication type selected. For example, if the HTTP authentication type is selected, the wizard allows you to enter the default inputs for HTTP authentication only and skips the WS-Security page when you click the **Next** button.

## The Invoke Method 2 Operation

### Overview of the Invoke Method 2 Operation

The **Invoke Method 2** operation is called when the flow is run. Its basic tasks are to:

- Build a SOAP request based on the **xmlTemplate**, the **inputMap**, and the inputs supplied to the operation (see the next section).
- Perform security functions as indicated by input values, such as signing the outbound request, encrypting it, and setting up SSL for HTTPS.
- Perform an XSLT transformation on the SOAP reply to populate the **documentStripped** and/or **jsonStripped** output, which strips the namespace prefixes from all of the output fields. For example, the reply:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
    <xyz:Test>
      <xmlns:Name>Real Name</xmlns:Name>
      <abc:Address>An address</abc:Address>
    </xyz:Test>
  </soapenv:Body>
</soapenv:Envelope>
```

would become:

```
<Envelope>
  <Body>
    <Test>
      <Name>Real Name</Name>
      <Address>An address</Address>
    </Test>
  </Body>
</Envelope>
```



or in a JSON formatted string:

```
{"Body":{"Test":{"Name":"Real Name1","Address":"An address1"}}}
```

This conversion is necessary as the operation outputs use XPath filters or JSON to extract values, and XPath expressions do not work well with XML that contains namespaces.

## Building a SOAP Request

Building a SOAP request includes the following steps:

- Input resolution

This step uses the **inputMap** together with the operation inputs, to determine the values to be substituted.

For example, if the **inputMap** contains **Test|Name=Name** and there is an input named **Name** with the value **George Washington**, this step combines them to determine that the element in the request corresponding to **Test|Name=Name** should have the value **George Washington**. This step also handles wildcards in array references. For example, an **inputMap** containing **Test|\*|Name=Name\*** and inputs **Name0** and **Name1** should have values corresponding to the elements in the SOAP request corresponding to **Test|0|Name** and **Test|1|Name**.

- Completing values

This step parses through the SOAP template looking for tokens. When it finds one, it attempts to find a value resolved from the previous step, and substitutes it if found. If no input is found with the specified name, the token is removed.

If the processing encounters the beginning of an array (indicated by the special comments in the template (“x or more repetitions” or “m to n repetitions”), the resolved inputs for that array are sorted numerically (so that 10 appears after 9 rather than between 1 and 2), and then substituted into the SOAP request.

**Note:** Any missing gaps in the input names are ignored. For example, if the inputs are **Name0** and **Name2** (and **Name1** is missing), then only two values are substituted in the template (the values for **Name0** and **Name2**); no empty entries are created for missing values.

## Complete Set of Inputs

Input	Description
contentType	Sets the HTTP Content-Type header to the given value. Defaults to <b>text/xml</b> .
ICONCLUDE_WSW_VERSION	Must be the constant <b>2</b> .
header_*	Any input that begins with <b>header_</b> is processed by the <b>HTTP Client Post Raw</b> operation, which then creates an HTTP header out of it. For example, if the input named <b>header_Accept-Encoding</b> contains the value <b>gzip</b> , the request will be altered to add the HTTP header <b>Accept-Encoding: gzip</b> .
headers	The list containing the headers to use for the request separated by a new line (CRLF). The header name -

Input	Description
	value pair will be separated by ":".
inputMap	Described in Locating Inputs and Creating the inputMap.
password	The password sent to the web service.
proxy	The name of the proxy host that is used to make the web service request across a firewall (optional).
proxyUsername	The proxy user name, if necessary, used when making web service requests across a firewall (optional).
proxyPassword	The proxy password used when making web service requests across a firewall (optional).
proxyPort	The port on the proxy host used to make the web service request across a firewall (optional).
returnXMLRequest	If this input is set to <b>true</b> , a new output named <b>rawXMLRequest</b> is returned by the operation, which contains the text of the SOAP request that was sent. This is useful for troubleshooting purposes.
timeout	The timeout in ms for the HTTP connection. Note that there may be other timeouts that affect the connection, such as the timeout between Central and the RAS.
trimComments	Removes all comments from the outbound SOAP request (hidden input).
trimNullOptionalTypes	By default (true), for every element in <b>xmlTemplate</b> that is marked as <b>Optional</b> and for which no token has been substituted with a value, the element is removed from the outbound SOAP request (hidden input).
trimNullComplexTypes	By default (true), for every element in <b>xmlTemplate</b> that has sub-elements (including arrays) and for which no token has been substituted with a value, the entire element (and all of its embedded elements) is removed from the outbound SOAP request (hidden input).
trustAllRoots	When set to true, when HTTPS connections are made, it ignores the signing authority of the certificate (permitting self-signed certificates) and ignores discrepancies between the host name on the certificate and the actual server name that is

Input	Description
	hosting the web service.
url	The URL of the web service, extracted from the WSDL. This generally has variable references to the host and port so that this value does not need to be changed to send a request to a host or a port different from the one hosting the WSDL.
useCookies	Determines whether the HTTP client will use cookies (store them during the connection and send them back for subsequent HTTP requests to the same server).
usesJSON	Use JSON arrays for all inputs of array type.
username	The user name sent to the web service.
xmlTemplate	Described in How the Web Services Wizard Uses SoapUI.
WSSecurityEncryptRequest	A Boolean value (default false) indicating whether or not to encrypt the SOAP request.
WSSecurityKeystore	When encrypting or digitally signing the SOAP request, this indicates the keystore containing the certificate.
WSSecurityKeystorePassword	When encrypting or digitally signing the SOAP request, this indicates the password to the keystore.
WSSecurityKeystoreType	When encrypting or digitally signing the SOAP request, this indicates the keystore type.
WSSecuritySignRequest	A Boolean value (default false) indicating whether or not to digitally sign the SOAP request with an X509 signature.
WSSecurityTimestampRequest	A Boolean value (default false) indicating whether or not to securely timestamp the SOAP request.
wswAuthenticationType	Can be assigned one of the following values: <b>http</b> , <b>ws-security text</b> , <b>ws-security digest</b> , and <b>none</b> . <b>http</b> is used for normal HTTP authentication, where the user and password are sent as HTTP headers. The two <b>ws-security*</b> options use SOAP WS-Security protocols.
*	All other headers are passed intact to the <b>HTTP Client Post Raw</b> operation, which can interpret them.

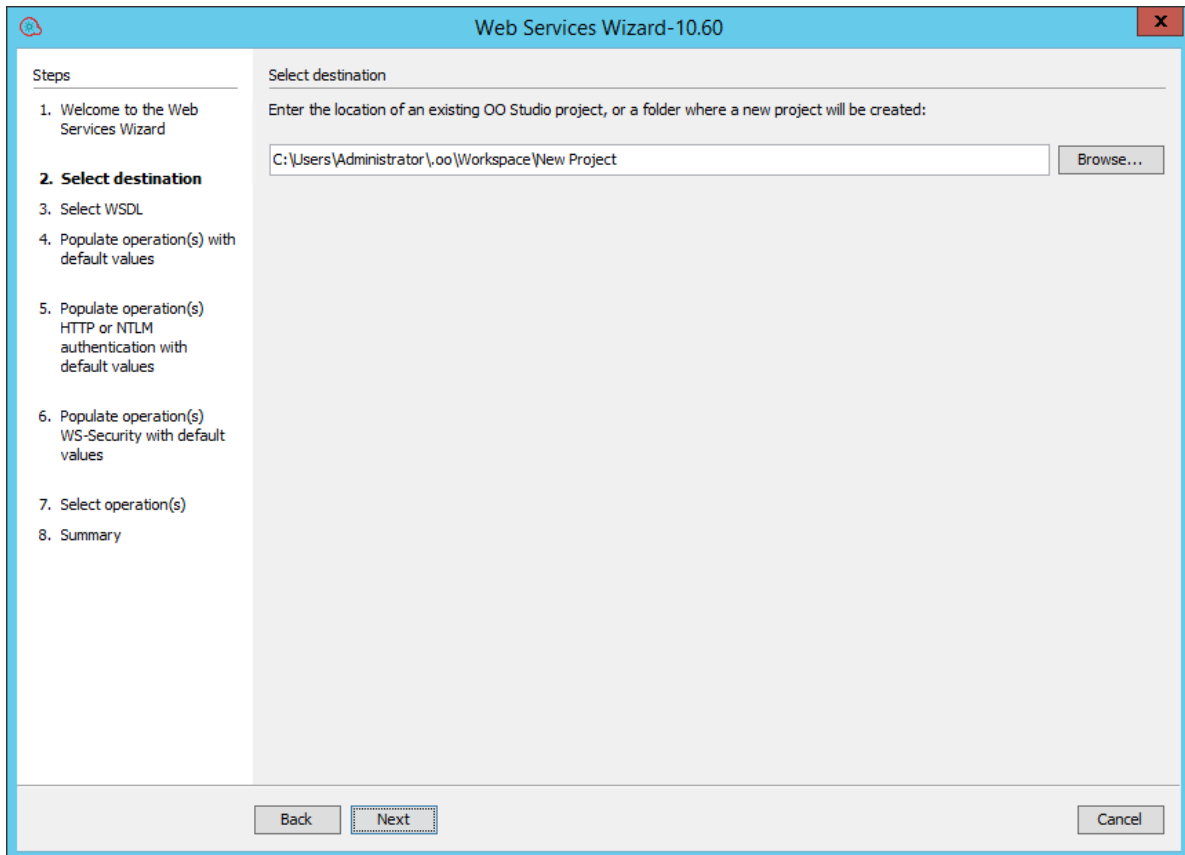
## Using the Web Services Wizard to Create Web Service Flows

### Using the Web Services Wizard to Create HPE OO Flows from Selected WSDL Operations

The Web Services Wizard creates HPE OO flows based on the operations available in the WSDL that you specify when you run the wizard. This tool is available by launching the wizard executable file. The Web Services Wizard is a simple and intuitive tool that leads the user through the tasks and simplifies the process of flow creation.

To use the Web Services Wizard to create an HPE OO flow from a WSDL

1. Start the Web Services Wizard.  
The Welcome page opens.
2. Click **Next** to continue.  
The **Select Destination** page opens.

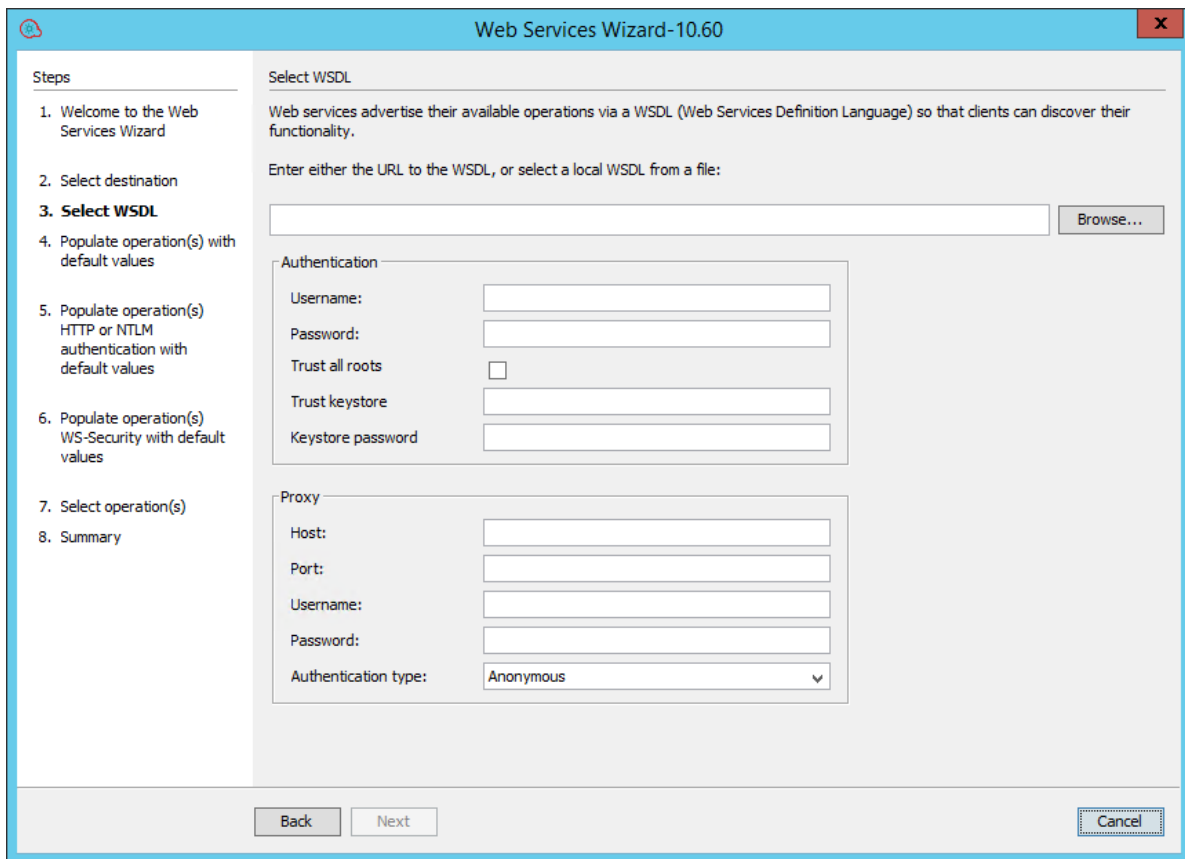


In the **Enter the location** field, enter the required project path or click **Browse** to locate the project location, and then click **Next**.

The wizard generates a 10.x studio project but not a content pack or a repository. The project has a default location: **C:\Users\[username]\.oo\Workspace\New Project**.

If this is not the first time you have run the wizard, the location is populated with the path to the last project created.

3. Enter the URL to the WSDL, or select a local WSDL from a file system.



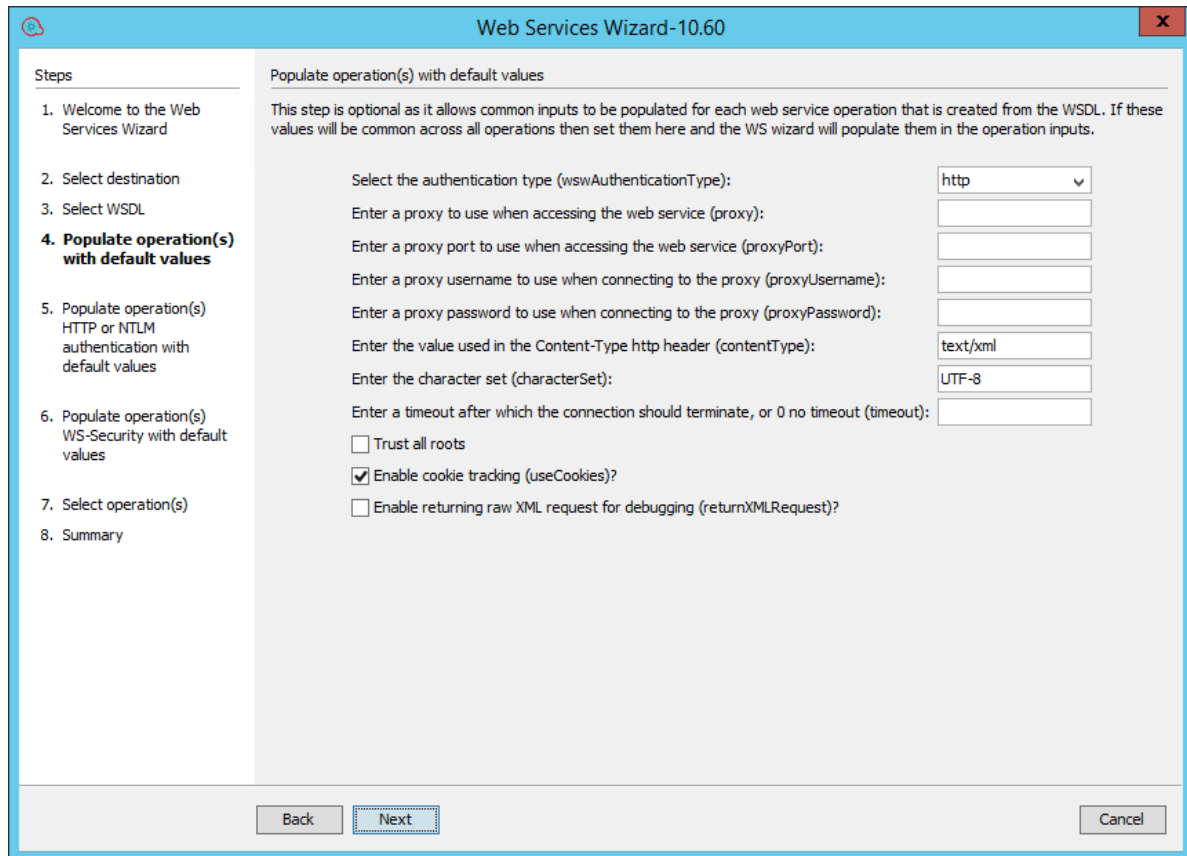
If proxy information is required to access the WSDL URL, enter it here. If loading the WSDL succeeds, the **Populate operation(s) with default values** page opens. In this page, you can set default values for the flows that the Web Services Wizard generates.

4. (Optional ) Enter values for any common inputs (these are the default inputs of Invoke Method 2 operation, so every flow created by the wizard will contain them). If the inputs are common for all flows created, they can be entered on this page. The default values are populated on the page.

**Note:** If you set the values here, each operation will be assigned with the preset values . To change the value, you need to modify each flow in Studio or rerun the Web Services Wizard and select the **Overwrite the flow if already exists** check box.

5. Click **Next** to continue to either the **Populate operation(s) HTTP authentication with default values** page or the **Populate operation(s) WS-Security with default values** page or the **Select operation(s)** page, depending on the authentication type selected.

For example, if you select an authentication type of **ws-security text**, the next page will be the optional step of populating the WS-Security default input values.



6. (Optional) Enter values for the common HTTP authentication inputs. If the inputs are common for all flows created, they can be entered on this page.

**Web Services Wizard-10.60**

**Steps**

- Welcome to the Web Services Wizard
- Select destination
- Select WSDL
- Populate operation(s) with default values
- Populate operation(s) HTTP or NTLM authentication with default values**
- Populate operation(s) WS-Security with default values
- Select operation(s)
- Summary

**Populate operation(s) HTTP or NTLM authentication with default values**

This step is optional as it allows HTTP or NTLM authentication inputs to be populated for each web service operation(s) that is created from the WSDL. If these values will be common across all operation(s) then set them here and the Web Services Wizard will populate them in the operation(s) inputs.

Enter username to use for authenticating with the web service (username):

Enter password to use for authenticating with the web service (password):

Enter the path to the keystore to use for SSL Client Certificates (keystore):

Enter the password for the keystore (keystorePassword):

*Note: For security reasons, it is recommended to configure the system account passwords in Central.*

Back Next Cancel

**Note:** If you set the values here, each operation will be assigned with the preset values. To change the value, you need to modify each flow in Studio or rerun the Web Services Wizard and select the **Overwrite the flow if already exists** check box.

- Click **Next** to continue.
- (Optional) Enter values for the common WS-Security inputs. If the inputs are common for all flows created, they can be entered on this page.

**Web Services Wizard-10.60**

**Steps**

- Welcome to the Web Services Wizard
- Select destination
- Select WSDL
- Populate operation(s) with default values
- Populate operation(s) HTTP or NTLM authentication with default values
- Populate operation(s) WS-Security with default values**
- Select operation(s)
- Summary

**Populate operation(s) WS-Security with default values**

This step is optional as it allows WS-Security inputs to be populated for each web service operation(s) that is created from the WSDL. If these values will be common across all operation(s) then set them here and the WS wizard will populate them in the operation(s) inputs.

Enter username to use for authenticating with the web service (username):

Enter password to use for authenticating with the web service (password):

Enter the keystore to use for encrypting and signing requests (WSSecurityKeystore):

Enter the type of WS-Security keystore (WSSecurityKeystoreType):

Enter the password for WS-Security Keystore (WSSecurityKeystorePassword):

Should the SOAP request be signed using WS-Security (WSSecuritySignRequest)?

Should the entire SOAP request be signed using WS-Security (WSSecurityEncryptRequest)?

Should the Timestamp attribute be added to the WS-Security header (WSSecurityTimestampRequest)?

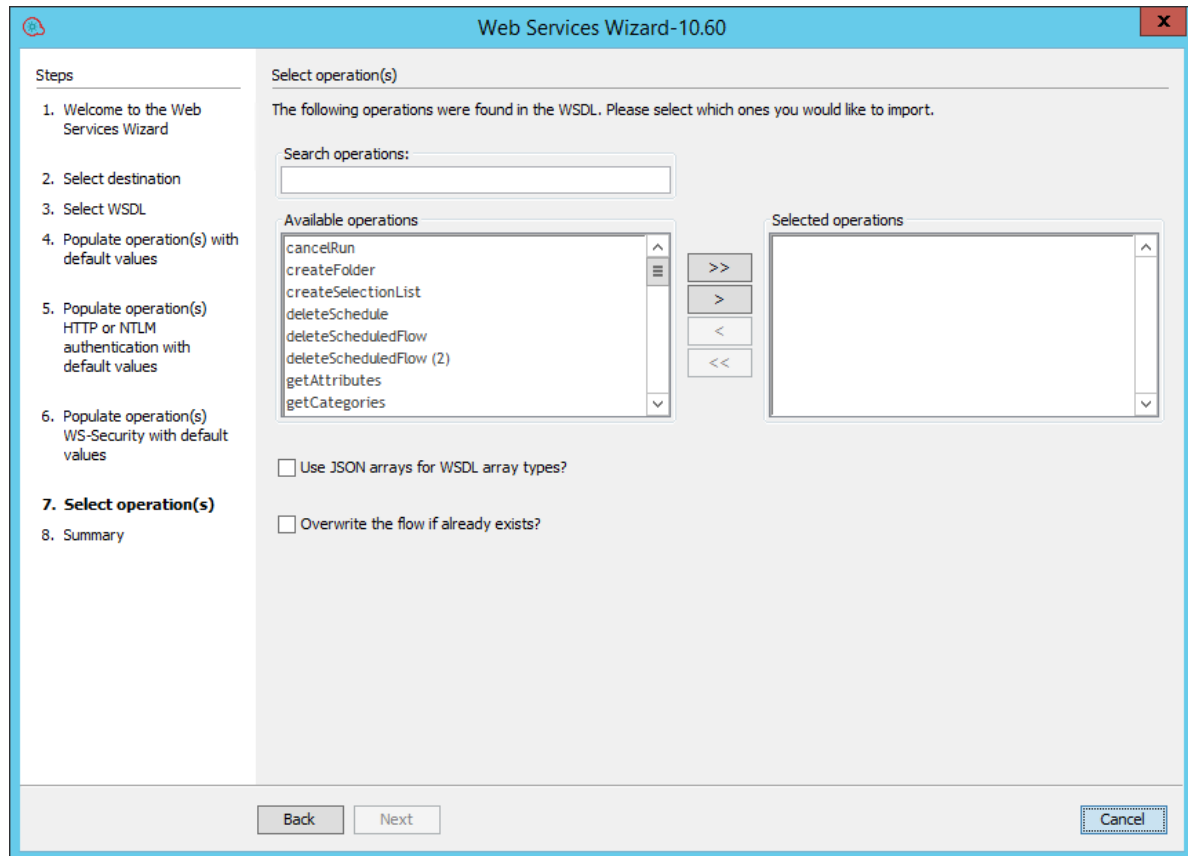
*Note: For security reasons, it is recommended to configure the system account passwords in Central.*

Back Next Cancel

**Note:** If you set the values here, each operation will be assigned with the preset values. To change the value, you need to modify each flow in Studio or rerun the Web Services Wizard and select the **Overwrite the flow if already exists** check box.

- Click **Next** to continue.
- Select the operation(s) for which you are interested in creating flows. The available operations are displayed in the list. If you want to use JSON-formatted arrays for all array type inputs in all the generated flows, check the **Use JSON arrays for WSDL array types** box. If you do not check the box, you can still use JSON formatted arrays, but you have to manually set the input usesJSON in Invoke Method 2 to **True** for all the flows that you want to accept JSON data.





11. Click **Next** to continue.

You can move the operations from one column to the other. Use the search text box if the list is long, and you cannot find the required cmdlet. The wizard searches the list for the operations with names containing the search text. In addition, the wizard updates the list as you type.

After the flows are successfully created and saved in the repository, the Web Services Wizard finishes.

## After Running the Web Services Wizard

If the Web Services Wizard ran successfully, you will have a new set of flows ready for use. However, may have to make some adjustments before the operations can be used, due to the following issues:

- The source WSDL may have problems or may have changed.
- There may be undocumented headers.

To diagnose and correct these situations, read this section, along with the Troubleshooting section.

**Note:** In addition to inputs, the parsing obtains results that can be captured as operation outputs (which are expressed as results in steps). Any arrays in the XML are extracted as a single XML result from which the flow author can extract narrow subsets.

- In the flows that the Web Services Wizard generates, the flow inputs that correspond to web services inputs are optional. Sometimes some of the inputs that the web service definitions indicate as required are not actually required, and mirroring these settings in the flow would force the flow user to enter unused values when running it. So, the Web Services Wizard sets all inputs as optional. When the web service does indicate that a field is optional, it precedes the field with the comment "`<!--`

Optional:>" or "<!zero or more repetitions->". For information on which inputs should be required, see the documentation for the relevant web service.

- If the web service whose WSDL you are accessing resides on the other side of a firewall from your Studio machine, you must specify an HTTP proxy to be used to reach the web service.

## Troubleshooting

### General Troubleshooting Principles

- If you experience difficulties running the Web Services Wizard, first check any changes you make on one input before trying them on all inputs.
- If you experience difficulties running the Web Services Wizard against a WSDL with a URL that starts with HTTPS, try opening the WSDL in a browser and saving it to the local file system. Make sure to copy all dependencies (such as XSD files) because it is difficult to access them through the wizard. These files can be found under `<xs:schema><xs:import>` tags. Then, run the Web Services Wizard against the WSDL file instead.
- If an unexpected error message is returned after running the OO flows that the Web Services Wizard created, try adding and setting the **trimNullOptionalTypes** and/or **trimNullComplexTypes** to **false** in the **Invoke Method 2** operation of your flow. This results in the outbound SOAP request looking more like the request sent by SoapUI when inputs have null values.
- If the Web Services Wizard fails to import the Exchange WSDL due to a `java.lang.OutOfMemoryError: Java heap space` error, this could be because the attached XSD is invalid or because the WSDL is invalid.

### Troubleshooting Steps

If the Web Services Wizard fails to load the operations for selection and returns with a null pointer exception:

- Try removing any white space around the comments section of the WSDL. This is a known issue with the SoapUI utility that the Web Services Wizard uses.
- The Web Services Wizard passes on the SoapUI's "Null-pointer Exception" message followed by the message "Failed to load WSDL" if you attempt to load an invalid WSDL. This is a known issue with the SoapUI utility that the Web Services Wizard uses.
- Validate that the XML request is as you expected. This can be done by setting the **returnXMLRequest** input value to **true** in the **Invoke Method 2** operation in your newly created flow. This will add an output result of the actual XML request that was sent.
- Try the request in SoapUI to verify that the Web service is working correctly. Install SoapUI (<http://www.soapui.org/>), create a project from the WSDL and a request object for the operation in question. Then, replace its content with the XML request from the output above.
- Some WSDLs have been written in a way that causes the Web Services Wizard to fail to recognize some array types. When one of these OO flows runs, it may return the following exception:

```
<faultcode><soapenv:Server.userException</faultcode><faultstring>org.xml.sax.SAXException: Found character data inside an array element while
deserializing</faultstring><
```

The original WSDL file, which was correctly processed by the Web Services Wizard, used the **ArrayOf\_xsd\_String** implementation:

```
<wsdl:message name="createSelectionListRequest">
...
<wsdl:part name="values" type="impl:ArrayOf_xsd_String"/>
</wsdl:message>
```

The modified WSDL file, which is correctly processed by the Web Services Wizard, redefines the type **ArrayOf\_xsd\_String** to **WSListValues** (this is a specific case for the **createSelectionList** operation from the example). Using the **WSListValues** type definition, you can also define your own array of string types (for example, **ArrayOfStrings**) in place of **ArrayOf\_xsd\_String**.

```
<wsdl:types>
...
<complexType name="WSListValues">
<sequence>
<!--Zero or more repetitions!-->
<element maxOccurs="unbounded" minOccurs="0" name="value" type="xsd:string" />
</sequence>
</complexType>
...
</wsdl:types>
<wsdl:message name="createSelectionListRequest">
...
<wsdl:part name="values" type="tns1:WSListValues"/>
...
</wsdl:message>
```

- In some cases, the **Invoke Method 2** operation fails to run when used in a flow in Studio. This occurs because the SOAP envelope is incorrect. A single flow UUID is passed within **<flowUuids></flowUuids>** instead of an array. This occurs for all soap requests that have an element **wsdl:arrayType** that does not have **nillable="true"**.

In the SOAP UI, the generated request contains an empty **flowUuids** array by default. You should edit it manually and insert any relevant UUIDs of interest. For example, the **xmlTemplate** input of the **Invoke Method 2** step in the generated Get Flow Details can be modified manually as shown below:

```
<soapenv:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

```

xmlns:wsc="http://wscentral.service.services.dharma.iconclude.com"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  <soapenv:Header/>
  <soapenv:Body>
    <wsc:getFlowDetails
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  <flowUuids xsi:type="xsd:string"?></flowUuids>
  </wsc:getFlowDetails>
</soapenv:Body>
</soapenv:Envelope>

```

## Creating an OO Web Service Tool with Proxy

If you cannot launch the OO Web Services wizard to create operations from a WSDLfile (local or remote), create an XMLfile named **soapui-settings.xml** in the following format and save it in your home directory:

```

<?xml version="1.0" encoding="UTF-8" ?>
<con:soapui-settings xmlns:con="http://eviware.com/soapui/config">
<con:setting id="WsdSettings@cache-wsdl">true</con:setting>
<con:setting id="WsdSettings@pretty-print-response-xml">true</con:setting>
<con:setting id="HttpSettings@include_request_in_time_taken">true</con:setting>
<con:setting id="HttpSettings@include_response_in_time_taken">true</con:setting>
<con:setting id="WsdSettings@name-with-binding">true</con:setting>
<con:setting id="HttpSettings@max_connections_per_host">500</con:setting>
<con:setting id="HttpSettings@max_total_connections">2000</con:setting>
<con:setting id="ProxySettings@host">111.222.111.322</con:setting>
<con:setting id="ProxySettings@port">3128</con:setting>
<con:setting id="ProxySettings@enableProxy">true</con:setting>
</con:soapui-settings>

```

If this fails due to proxy issues, this file will ask for your credentials for the proxy (hard-coded in the file). Modify the proxy host and port in the file, place it in your home directory, and enter the user name and password, when prompted. This procedure will allow the WSDL file to be loaded correctly.

# Using the Shell Wizard

## Purpose of the Shell Wizard

This integration enables administrators to create HPE Operations Orchestration (OO) flows that are integrated with Shell.

The Shell Wizard helps to create a set of predefined commands that represent a workflow or procedure. The wizard includes a recorder technology that creates steps in a flow from the commands that are used in the wizard's Shell window.

To learn how to create HPE OO flows, see *Studio Authoring Guide* in the documentation set for the current OO release.

This document explains how this integration has been implemented, and how the integration's operations and flows communicate between HPE OO and Shell.

## Audience

This guide is intended for system administrators who establish and maintain the implementation of the integration between Shell and HPE OO. This guide assumes that you have administrative access to both systems.

## Supported Versions

Content Pack	Operations Orchestration Version	Shell Wizard Version
Base Content Pack	10.x	10.x

## Getting Started with the Shell Wizard

### Installing the Shell Wizard

The wizard is installed if Studio is selected from the HPE Operation Orchestration installer. The Shell Wizard is located under `<OOInstallPath>\studio\tools`.

### Configuring Logging Settings

The configure logging settings are no longer supported in the 10.x wizard.

### Uninstalling the Shell Wizard

The wizard is uninstalled when Studio is uninstalled.

## Shell Wizard Requirements

The following are the minimum software requirements for systems running Shell Wizard for HPE Operations Orchestration:

- The environment must have Java SE Runtime Environment 8 (also known as JRE) installed (for running the wizards).

Target Host

- The machine must have SSH or Telnet protocol enabled.

## Shell Wizard Enhancements from 9.x

- The wizard now has the version (10.x) displayed in the title.
- The wizard now appears in the task bar and can be closed, minimized or brought to the front.

## Shell Wizard Steps

### Step 1. Welcome Page

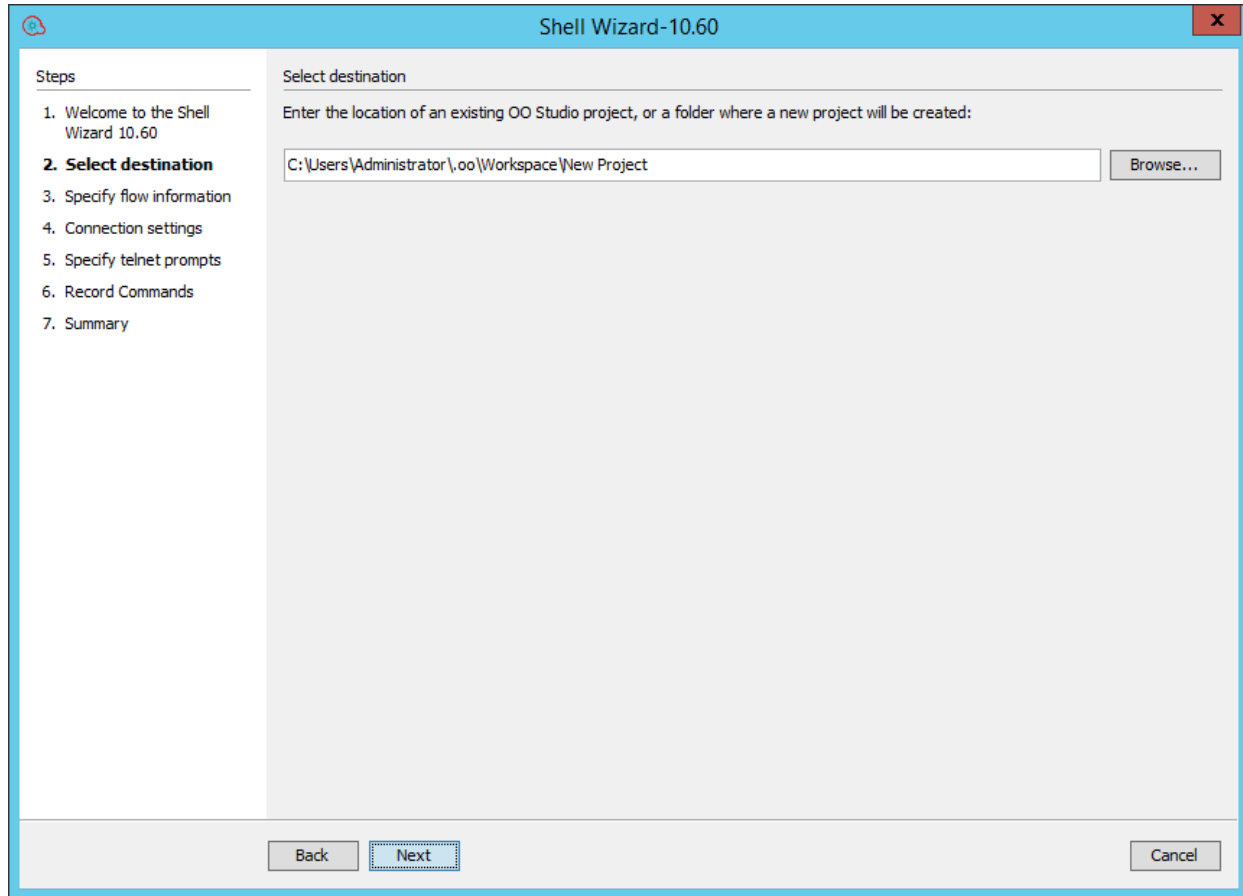
Start the Shell Wizard to open the **Welcome** page. Click **Next** to open the **Select Destination** page.

### Step 2. Selecting the Destination

Here you need to select the location of an existing studio project, or a folder where the new project will be created.

Enter or select a location of a Studio project for the flows you want to create, and then click **Next** to open the **Specify flow information** page.

If this is not the first time you have run the wizard, the location is populated with the path to the last project created.



### Step 3. Specifying Flow Information

Enter the flow name and flow description. If you want to overwrite the flow, check the option with the same name available in this window.

The screenshot shows a window titled "Shell Wizard-10.60" with a close button (X) in the top right corner. On the left, a "Steps" list contains seven items: 1. Welcome to the Shell Wizard 10.60, 2. Select destination, 3. **Specify flow information**, 4. Connection settings, 5. Specify telnet prompts, 6. Record Commands, and 7. Summary. The main area is titled "Specify flow information" and contains the following text: "This wizard will generate a simple flow, which will automate a command that you will enter later on in the wizard. This flow is intended to be used as a subflow to perform the operation." Below this is the instruction: "Enter a name for this new flow, and a brief description of what the flow does." There are two input fields: "Flow name to create\*:" and "Flow description:". A checkbox labeled "Overwrite the flow if already exists?" is located below the input fields. At the bottom of the window, there are three buttons: "Back", "Next", and "Cancel".

Click **Next** to open the **Connection settings** page.

## Step 4. Entering Connection Settings

Enter the connection settings inputs.

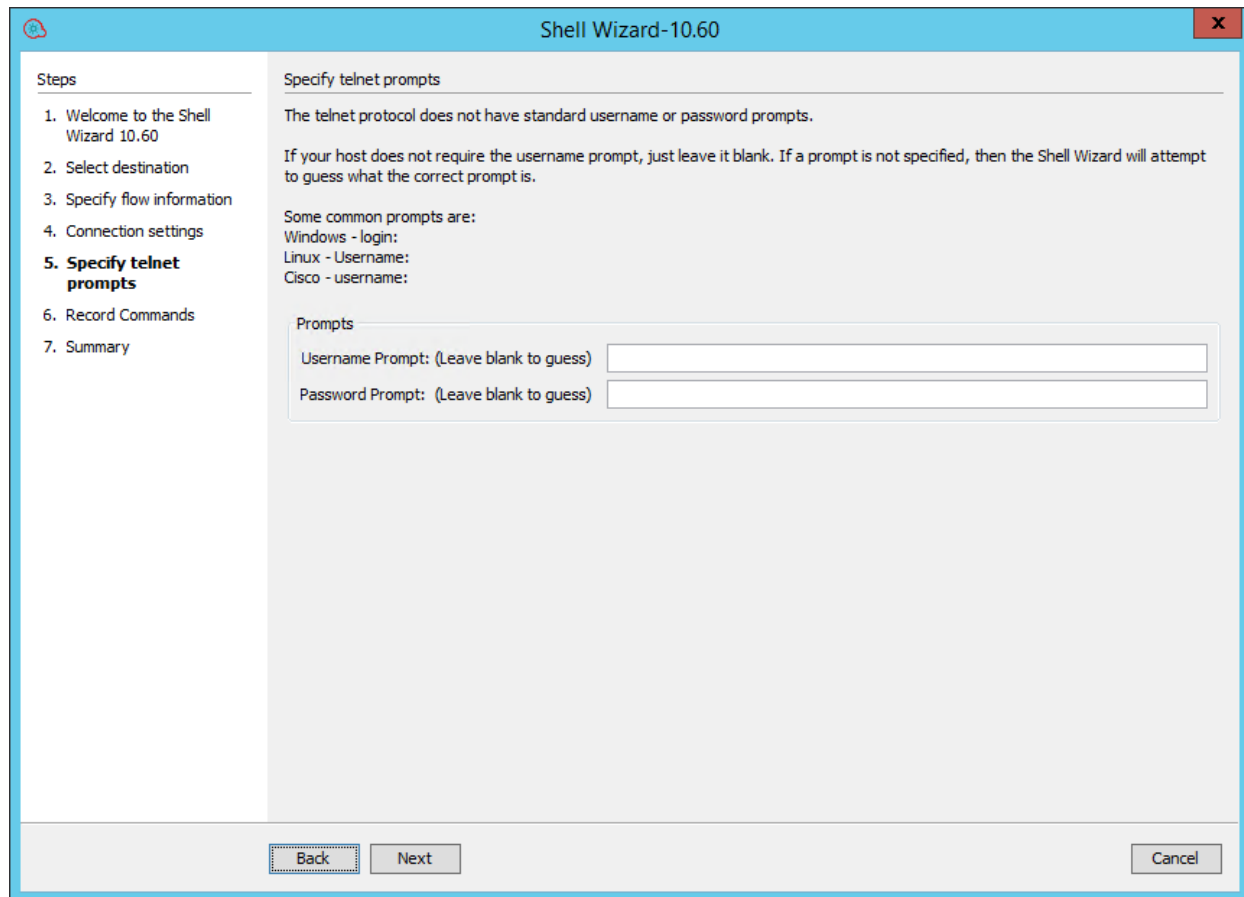


The screenshot shows a window titled "Shell Wizard-10.60" with a blue header bar. On the left, a "Steps" sidebar lists seven steps: 1. Welcome to the Shell Wizard 10.60, 2. Select destination, 3. Specify flow information, 4. **Connection settings** (highlighted), 5. Specify telnet prompts, 6. Record Commands, and 7. Summary. The main area is titled "Connection settings" and contains the instruction "Provide detailed connection information below." Under "Connection Info", there are three input fields: "Host to Connect to\*", "Username\*", and "Password\*". Below this is a "Choose a protocol" section with two radio buttons: "ssh" (selected) and "telnet". At the bottom, there are "Back", "Next", and "Cancel" buttons.

- If the chosen protocol is SSH, the **Record Commands** page opens.
- If the chosen protocol is Telnet, the **Specify telnet prompts** page opens.
- If the chosen protocol is SSH, the Authenticating window opens until the connection is established.

## Step 5. Specify Telnet Prompts

If the Telnet protocol was chosen, the **Specify Telnet prompts** page opens.

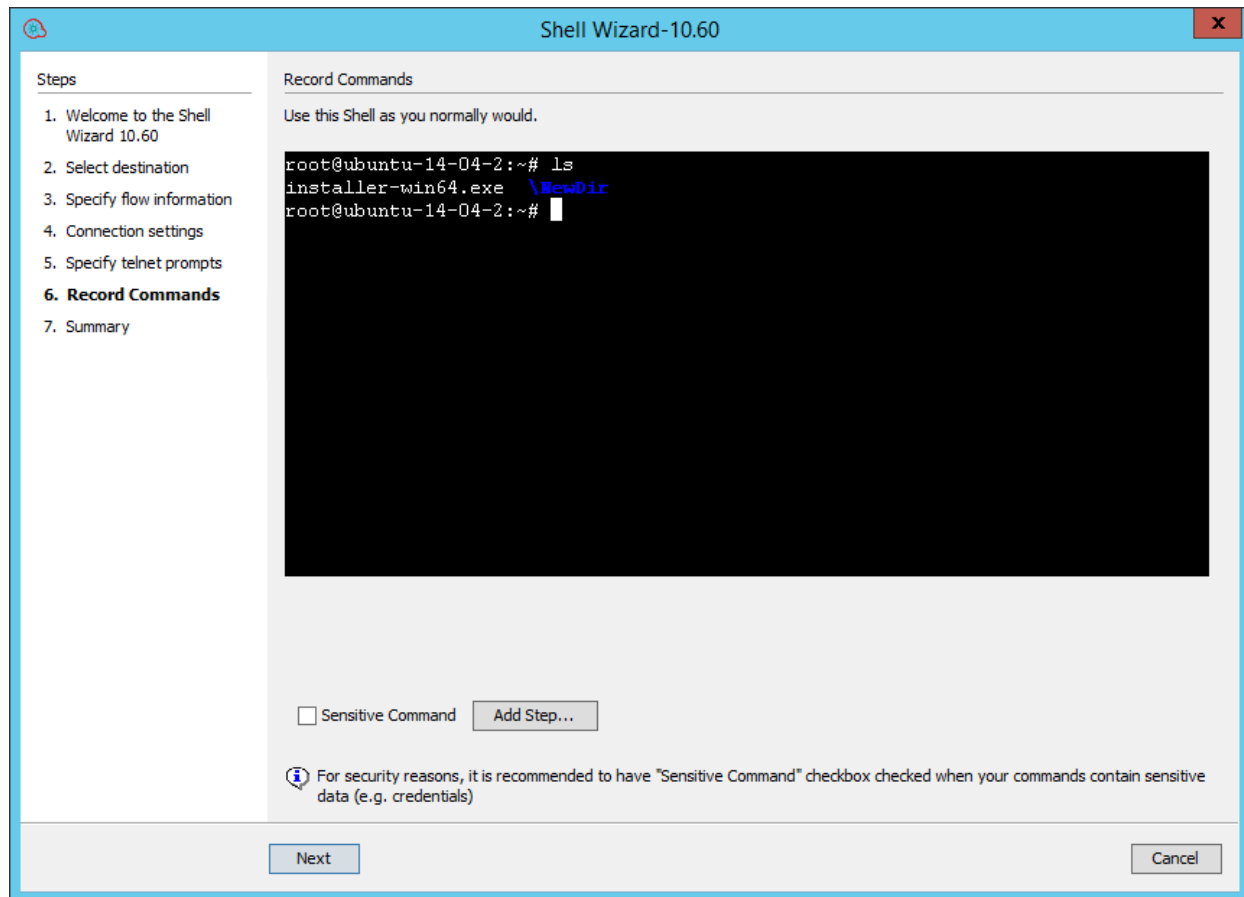


Click **Next** to open the connection. The Authenticating window opens until the connection is established.

## Step 6. Recording Commands

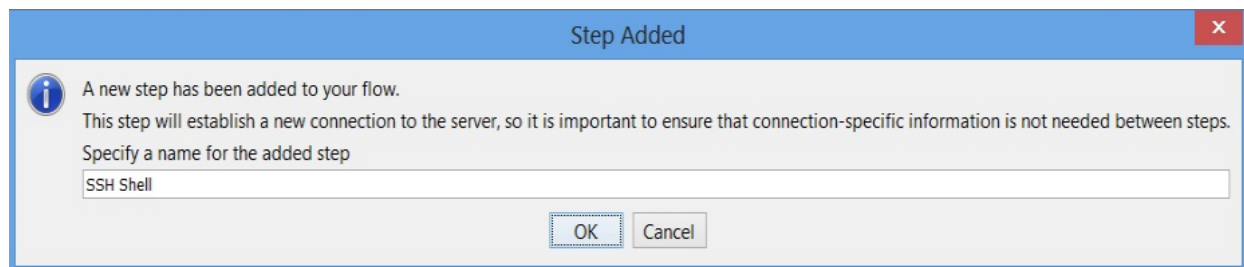
If the SSH protocol was chosen, the **Record Commands** page opens.

The commands written in this screen are recorded and executed as a step in the generated flow.



Click **Next** to go to the **Finish** page.

If the **Add Step** button is clicked, the Step Added window is displayed for a new connection to the server as a new step of the flow.



## Step 7. Finish

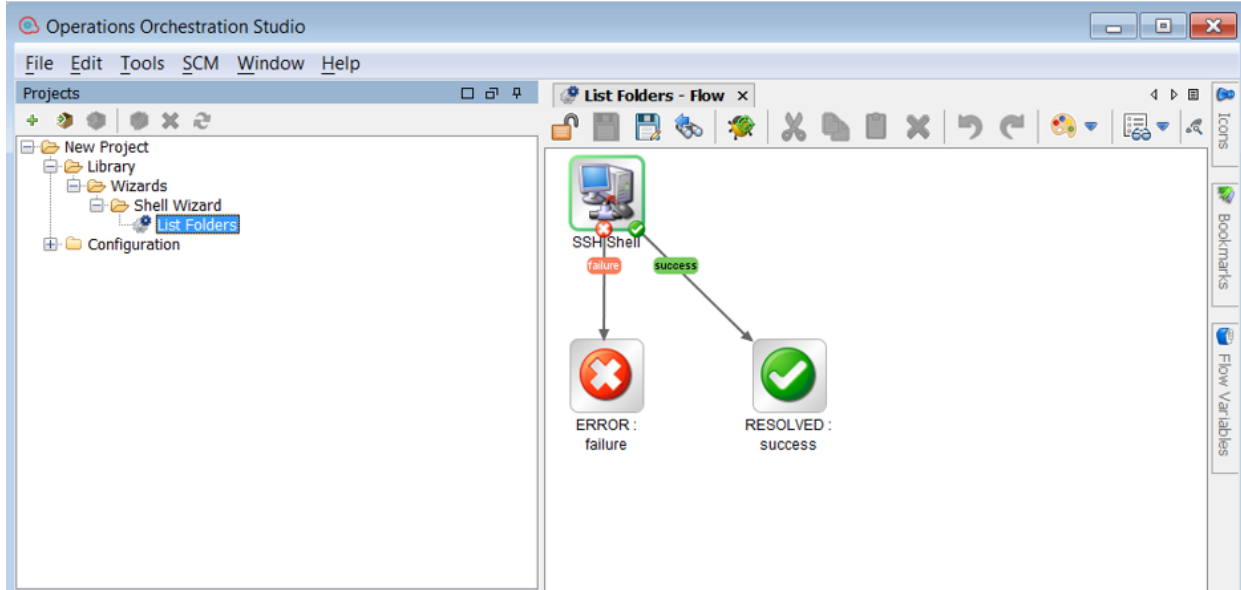
Click **Finish** to exit the wizard.

## Importing the Generated Flows into Studio

Import the project generated by the wizard in HPE OO Studio. See the “Managing Projects” section in the *Studio Authoring Guide* to see how to import a project.

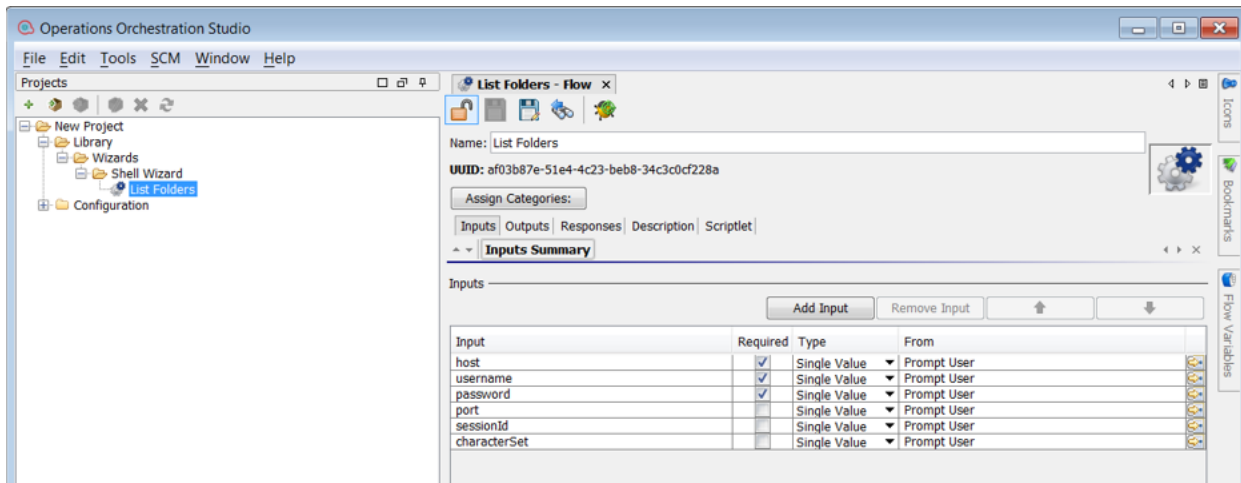
## Generated Flows

The Shell Wizard generates a flow with the name specified in the **Specify step information** step of the wizard. If a project with the same name already exists, the new flow will be added to.



## Inputs

Each flow has the following inputs, which are commons to the Shell operation:

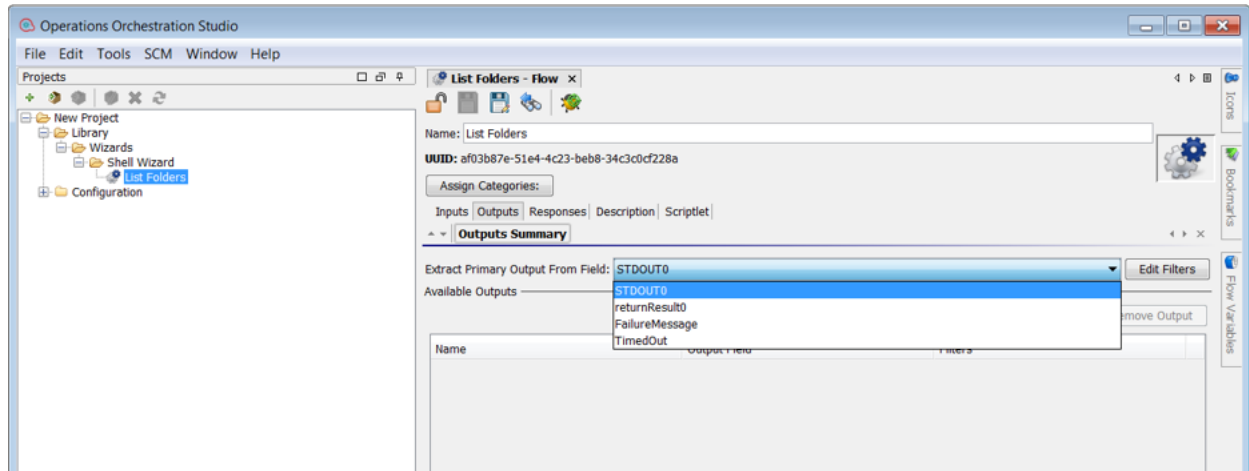


## Outputs

On the **Outputs** tab, the flow primary result is **stdout** in addition to the **returnResult** result.

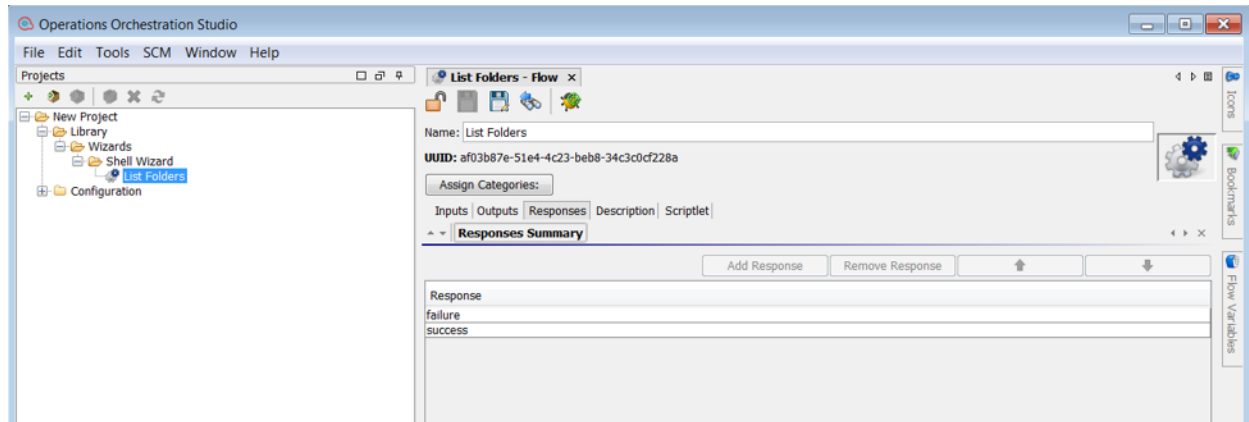
All HPE OO flows have the **FailureMessage** and **TimedOut** outputs.

All generated flows have the **returnResult** output (which is the generic operation's result), and **stdout** result.



## Responses

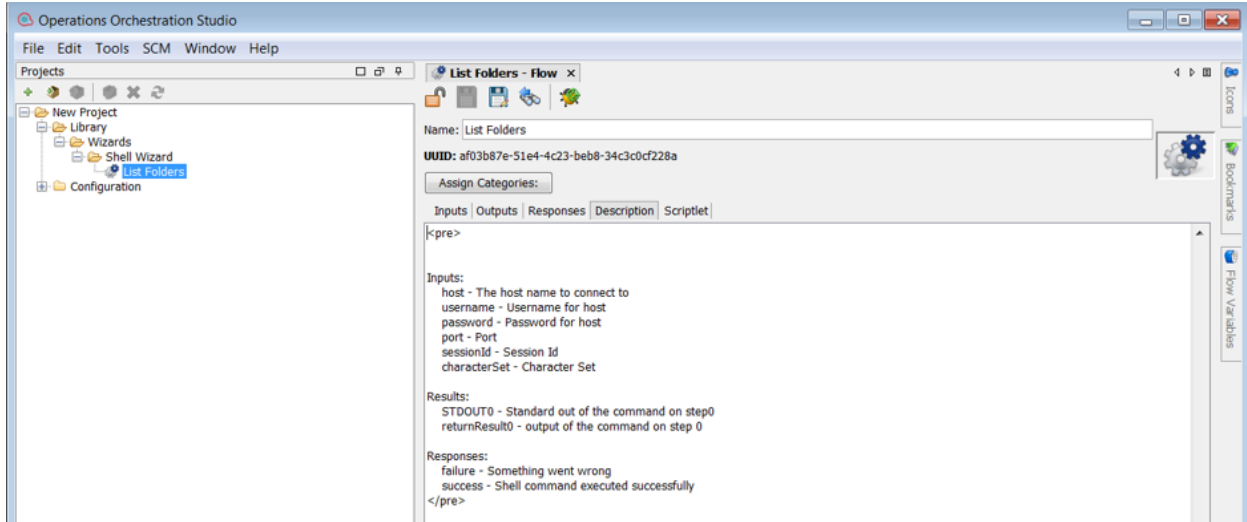
The success and failure responses of the flows are the same as in the Shell operations.



## Descriptions

The description of each generated flow contains the following items:

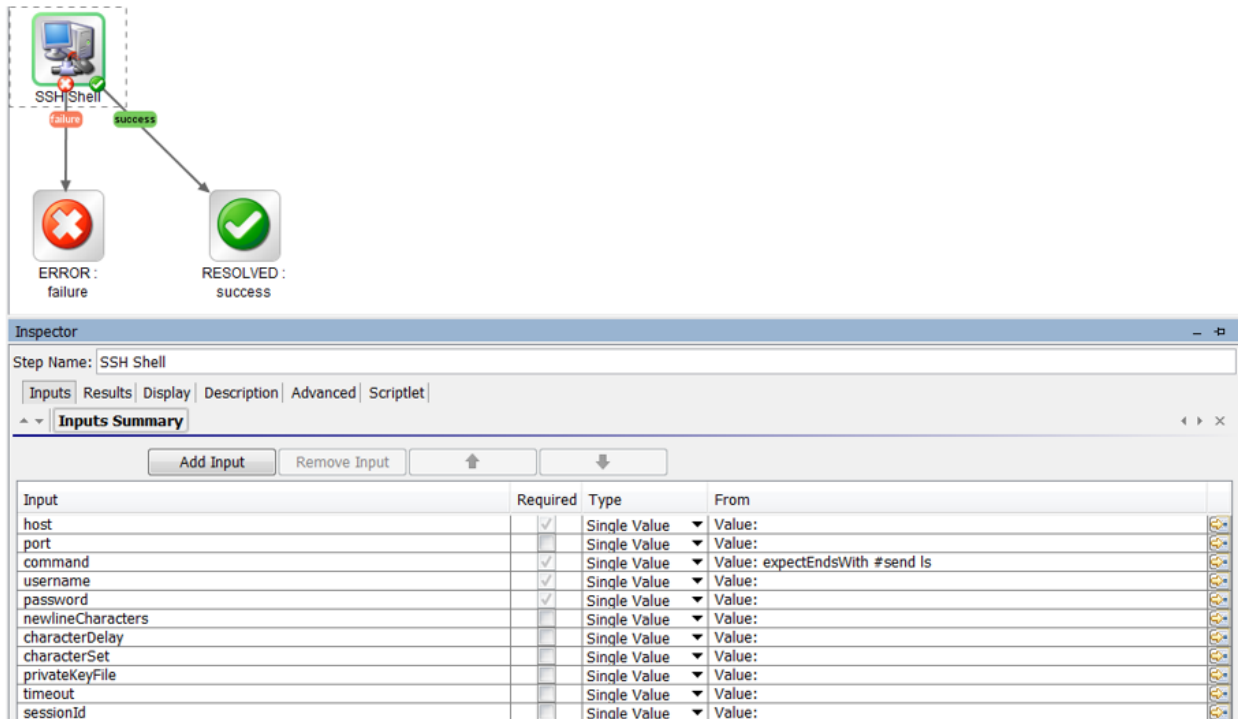
- The description of the flow inputs
- The description of the results
- The description of the responses



## Shell Operation

The operation is used to execute a Shell script on a target host, either local or remote.

## Operation Inputs



## Operation Results

The diagram illustrates the execution flow of an SSH Shell step. It starts with a central 'SSH Shell' icon. From this icon, two arrows branch out: one labeled 'failure' pointing to an 'ERROR: failure' icon (a red square with a white X), and another labeled 'success' pointing to a 'RESOLVED: success' icon (a green square with a white checkmark).

Below the diagram is a screenshot of the 'Inspector' window. The 'Step Name' is 'SSH Shell'. The 'Step Results' tab is active, showing a table with the following data:

Name	From	Assign To	Assignment Action	Filters
STDOUT0	STDOUT	Flow Output Field	OVERWRITE	No Filters
returnResult0	returnResult	Flow Output Field	OVERWRITE	No Filters

## Troubleshooting

### Troubleshooting Overview

This section provides troubleshooting procedures and tools that you can use to solve problems you may encounter while using this integration. It also includes a list of the error messages you may receive while using the integration and offers descriptions and possible fixes for the errors.

### Could not connect to the host

The possible reasons are:

- The user credentials are not correct.
- Telnet protocol is not enabled on the machine.
- The user does not have permission to execute Shell scripts on the target host. Make sure the user has admin rights for executing the file.
- Authentication problems (most common). See "Running a PowerShell Script on a Remote Host".

### Error Messages

This section lists the error messages you may receive while using this integration. Each error message includes possible causes and fixes for the error.

```
java.net.ConnectException: Connection refused: connect
```

This error occurs when SSH or Telnet protocol are not supported on the machine.

# Using the PowerShell Wizard

## Purpose of the PowerShell Wizard

With this integration, users can generate HPE 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):
  - Create an empty flow
  - Drag and drop the PowerShell Script operation
  - Search for the cmdlet description
  - Set the required input values
  - 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
10.x with Base Content Pack	10.x

## Getting Started with the PowerShell Wizard

### Downloading the PowerShell Wizard

The PowerShell Wizard Installer is an executable file that can be downloaded from the HPE Live Network page.

1. From <https://hpln.hp.com>, click **Operations Orchestration Community** and log in. The Operations Orchestration Community page contains links to announcements, discussions, downloads, documentation, help, and support.
2. On the left-hand side, click **Operations Orchestration Content Packs**.
3. In the Operations Orchestration Content Packs box, click **Content**. The HPE Passport and sign-in page appears.



4. Enter your user ID and Password to access to continue.
5. Click HPE Operations Orchestration 10.x, and then select the items that you want to download.

## Starting the PowerShell Wizard

If Studio is selected from the Operation Orchestration installer, the wizard is located under **<installation folder>\studio\tools**.

Double-click the **ps-wizard.bat** file under **<installation folder>\studio\tools**.

## Configuring Logging Settings

The configure logging settings are no longer supported in the 10.x wizard.

## Uninstalling the PowerShell Wizard

The wizard is uninstalled when Studio is uninstalled.

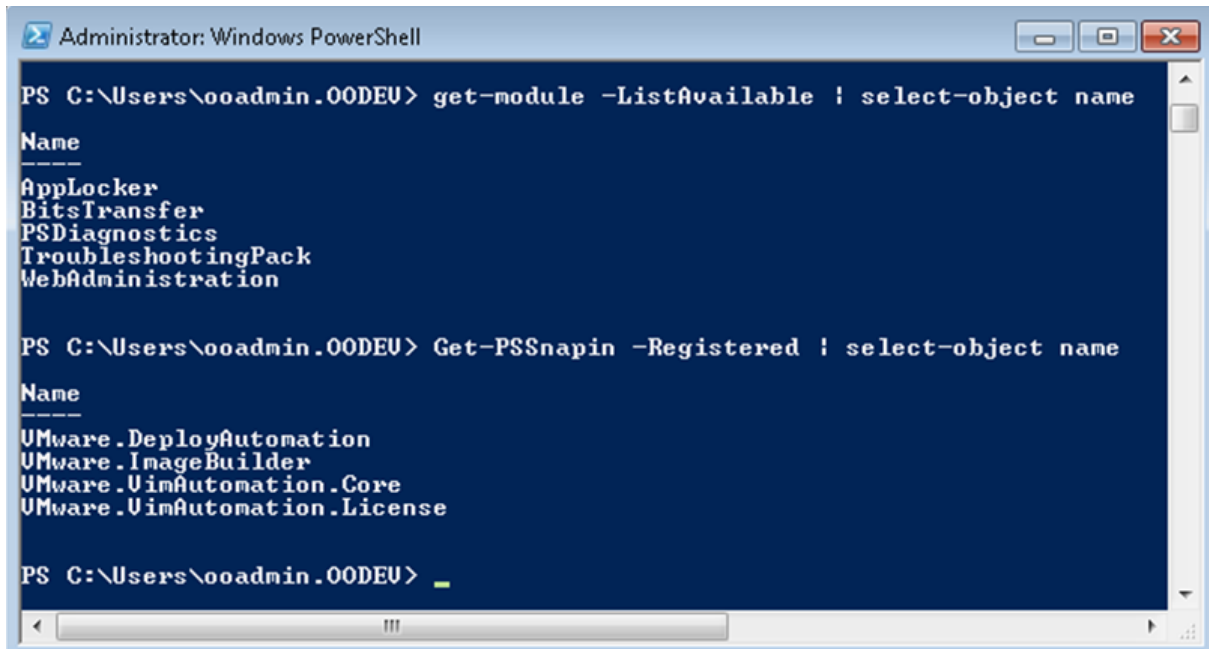
## PowerShell Wizard Requirements

The minimum software requirements for systems running the PowerShell Wizard for HPE Operations Orchestration are:

- Java SE Runtime Environment (also known as JRE) 8
- .NET Framework 2 or a later version

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.



```

Administrator: Windows PowerShell

PS C:\Users\ooadmin.00DEU> get-module -ListAvailable | select-object name
Name
----
AppLocker
BitsTransfer
PSDiagnostics
TroubleshootingPack
WebAdministration

PS C:\Users\ooadmin.00DEU> Get-PSSnapin -Registered | select-object name
Name
----
UMware.DeployAutomation
UMware.ImageBuilder
UMware.UimAutomation.Core
UMware.UimAutomation.License

PS C:\Users\ooadmin.00DEU>

```

Figure 1: How to list the modules and snapins in the PowerShell console

## PowerShell Wizard Enhancements from 9.x

- The wizard now has the version (10.x) displayed in the title.
- The wizard now appears in the task bar and can be closed, minimized or brought to the front.

## PowerShell Wizard Steps

The PowerShell Wizard contains only a few steps. The Welcome page contains a short summary of the wizard. This section describes the steps that you have to perform.

### Step 1. Selecting the Repository

In the **Enter the location** field, type the required project path or click **Browse** to locate the project location.

The wizard generates a 10.x Studio project, but not a content pack or a repository. The project has a default location: **C:\Users\[username]\1.oo\Workspace\New Project**.

If this is not the first time you have run the wizard, the location is populated with the path to the last project created.

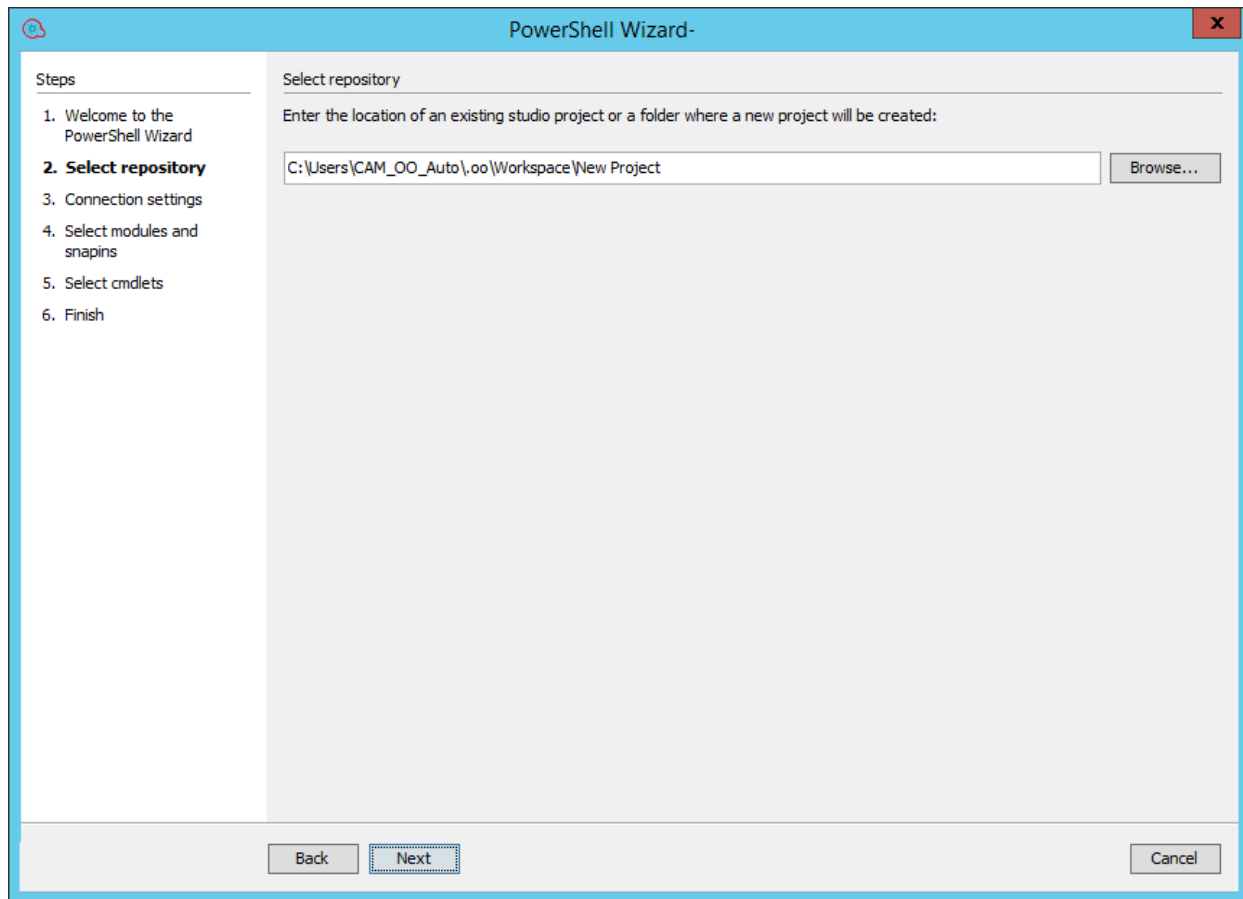


Figure 2: PowerShell Wizard: Select repository page

## Step 2. Configuring the PowerShell Connection

Figure 3: PowerShell Wizard: Connection settings page

**Host** - Type the name of the host that you want to connect to. If you leave the **Host** field empty, the PowerShell Wizard uses localhost as the default.

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

**Username** - Enter the user name to connect to the target host..

**Password** - Enter the password.

**Port** - The port values can be in the range of 1- 65535. 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).

## Step 3. Selecting the Modules

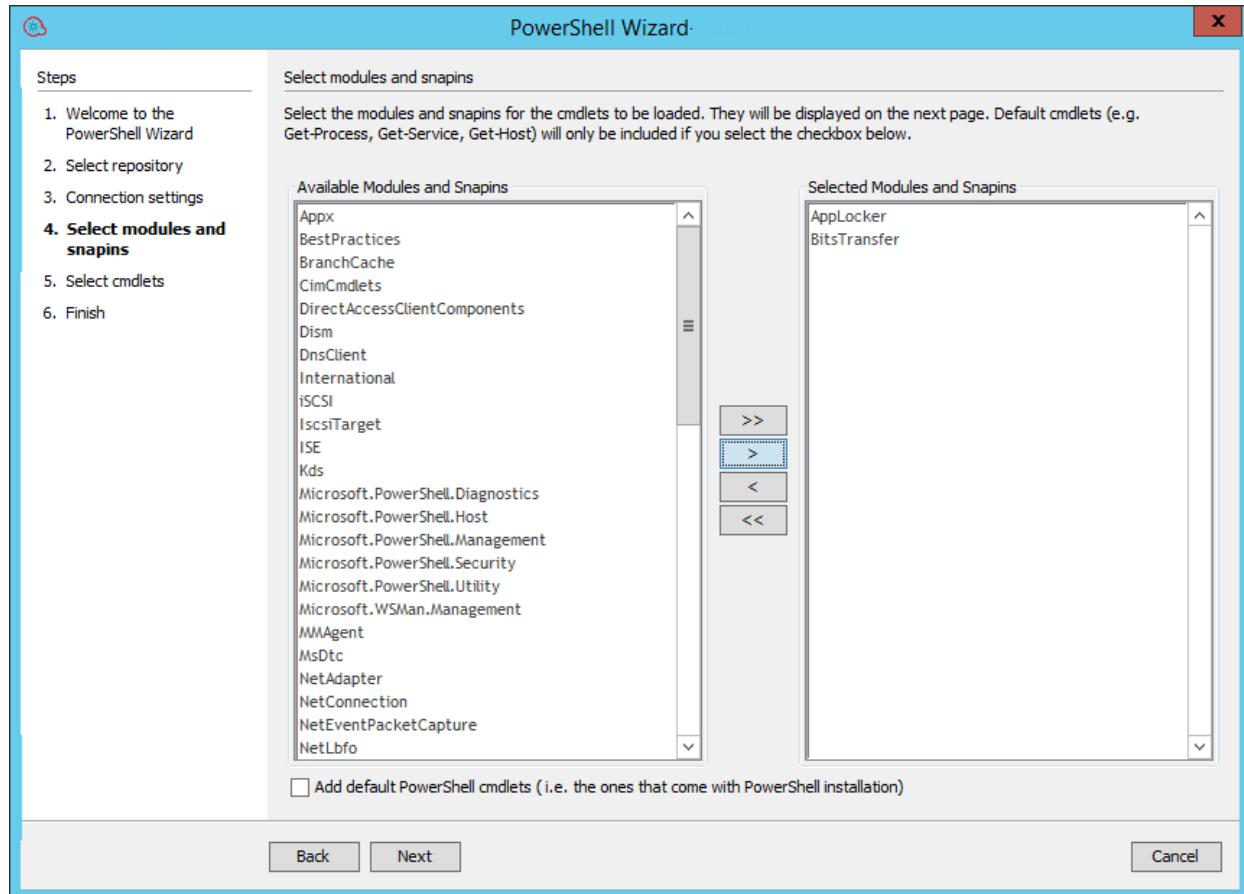


Figure 4: PowerShell Wizard: Select modules and snapins page

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

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** check box .

**Note:** If you select the **Add default PowerShell cmdlets** check box, the lists are disabled.

## Step 4. Selecting Operations (Cmdlets)

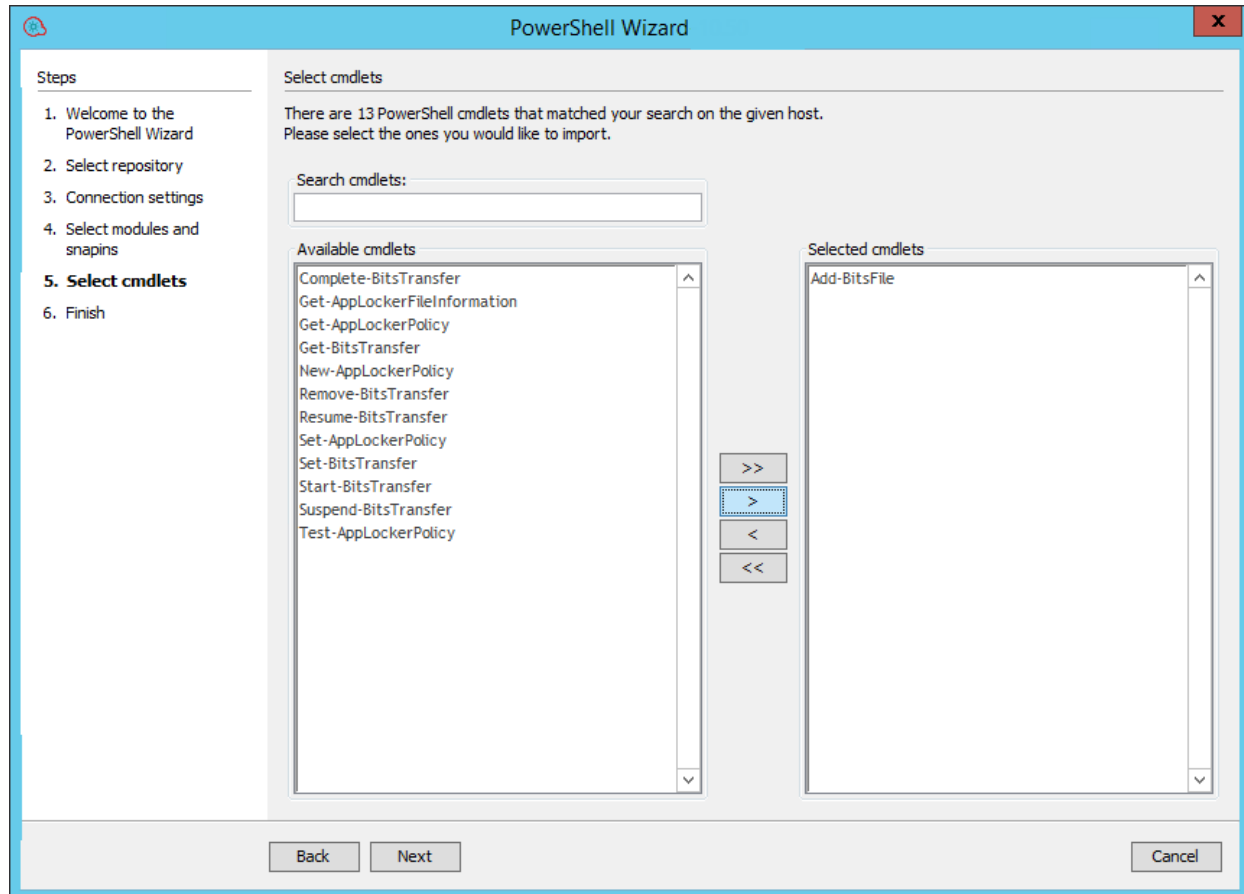


Figure 5: PowerShell Wizard: Select cmdlets page

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 text box if 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.

## Using the PowerShell Wizard

### PowerShell Wizard Operations and Flows

This section describes the operations and flows in the PowerShell wizard.

#### Generated Flows

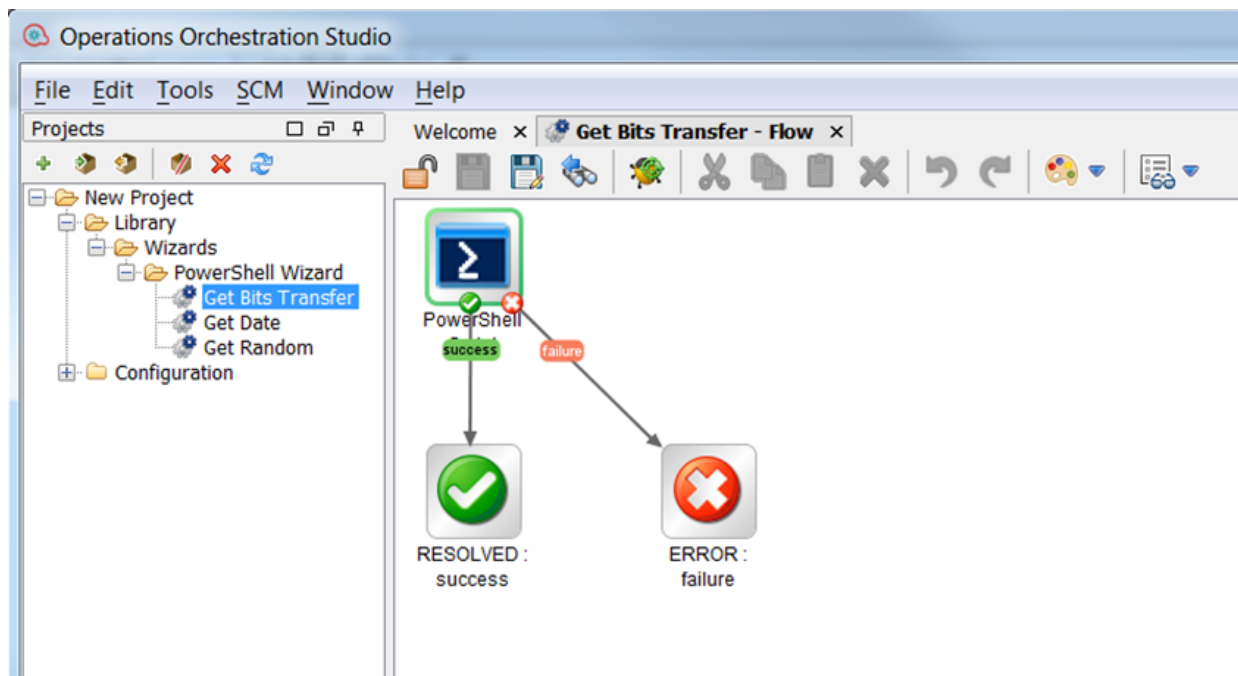


Figure 6: Example of a generated flow

The PowerShell wizard generates one flow for each selected cmdlet unless the project already contains the flow. For example, if the flow was generated in a previous run of the wizard.

The name of the flow is obtained from the name of the cmdlet by applying the following rule:

- Replace "-" with " ".

For example, the name of the flow for the cmdlet **Get-IScsiHbaTarget** is changed to **Get IScsi Hba Target**.

### Inputs

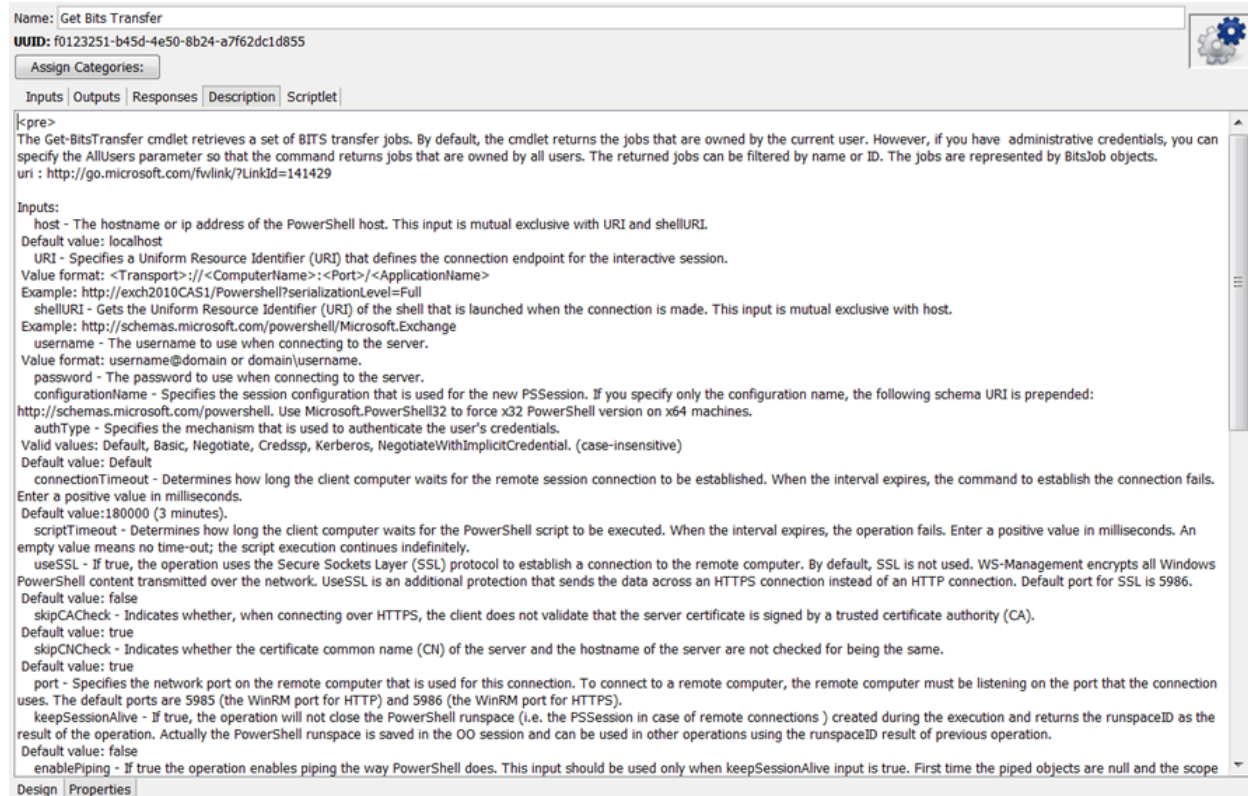
Input	Required	Type	From
host		Single Value	Prompt User
URI		Single Value	Prompt User
shellURI		Single Value	Prompt User
username		Single Value	Prompt User
password		Single Value	Prompt User
configurationName		Single Value	Prompt User
authType		Single Value	Prompt User from List - Selection List
connectionTimeout		Single Value	Prompt User
scriptTimeout		Single Value	Prompt User
useSSL		Single Value	Prompt User from List - Selection List
skipCACheck		Single Value	Value:
skipCNCheck		Single Value	Value:
port		Single Value	Prompt User
keepSessionAlive		Single Value	Prompt User from List - Selection List
enablePiping		Single Value	Prompt User from List - Selection List
runspaceID		Single Value	Prompt User
modules		Single Value	Value: BitsTransfer
snapins		Single Value	Prompt User
script		Single Value	Prompt User
cmdlet		Single Value	Value: Get-BitsTransfer
returnTable		Single Value	Prompt User from List - Selection List
delimiter		Single Value	Prompt User
colDelimiter		Single Value	Prompt User
rowDelimiter		Single Value	Prompt User

Figure 7: Inputs of a generated flow

Each flow has the following inputs:

- Common inputs. All the inputs of the PowerShell Script operation

### Descriptions



Name: Get Bits Transfer  
 GUID: f0123251-b45d-4e50-8b24-a7f62dc1d855  
 Assign Categories:

Inputs | Outputs | Responses | Description | Scriptlet

```
<pre>
The Get-BitsTransfer cmdlet retrieves a set of BITS transfer jobs. By default, the cmdlet returns the jobs that are owned by the current user. However, if you have administrative credentials, you can specify the AllUsers parameter so that the command returns jobs that are owned by all users. The returned jobs can be filtered by name or ID. The jobs are represented by BitsJob objects.
uri : http://go.microsoft.com/fwlink/?LinkId=141429

Inputs:
  host - The hostname or ip address of the PowerShell host. This input is mutual exclusive with URI and shellURI.
  Default value: localhost
  URI - Specifies a Uniform Resource Identifier (URI) that defines the connection endpoint for the interactive session.
  Value format: <Transport>://<ComputerName>:<Port>/<ApplicationName>
  Example: http://exch2010CAS1/PowerShell?serializationLevel=Full
  shellURI - Gets the Uniform Resource Identifier (URI) of the shell that is launched when the connection is made. This input is mutual exclusive with host.
  Example: http://schemas.microsoft.com/powershell/Microsoft.Exchange
  username - The username to use when connecting to the server.
  Value format: username@domain or domain\username.
  password - The password to use when connecting to the server.
  configurationName - Specifies the session configuration that is used for the new PSSession. If you specify only the configuration name, the following schema URI is prepended:
  http://schemas.microsoft.com/powershell. Use Microsoft.PowerShell32 to force x32 PowerShell version on x64 machines.
  authType - Specifies the mechanism that is used to authenticate the user's credentials.
  Valid values: Default, Basic, Negotiate, Credssp, Kerberos, NegotiateWithImplicitCredential. (case-insensitive)
  Default value: Default
  connectionTimeout - Determines how long the client computer waits for the remote session connection to be established. When the interval expires, the command to establish the connection fails.
  Enter a positive value in milliseconds.
  Default value: 180000 (3 minutes).
  scriptTimeout - Determines how long the client computer waits for the PowerShell script to be executed. When the interval expires, the operation fails. Enter a positive value in milliseconds. An empty value means no time-out; the script execution continues indefinitely.
  useSSL - If true, the operation uses the Secure Sockets Layer (SSL) protocol to establish a connection to the remote computer. By default, SSL is not used. WS-Management encrypts all Windows PowerShell content transmitted over the network. UseSSL is an additional protection that sends the data across an HTTPS connection instead of an HTTP connection. Default port for SSL is 5986.
  Default value: false
  skipCACheck - Indicates whether, when connecting over HTTPS, the client does not validate that the server certificate is signed by a trusted certificate authority (CA).
  Default value: true
  skipCNCheck - Indicates whether the certificate common name (CN) of the server and the hostname of the server are not checked for being the same.
  Default value: true
  port - Specifies the network port on the remote computer that is used for this connection. To connect to a remote computer, the remote computer must be listening on the port that the connection uses. The default ports are 5985 (the WinRM port for HTTP) and 5986 (the WinRM port for HTTPS).
  keepSessionAlive - If true, the operation will not close the PowerShell runspace (i.e. the PSSession in case of remote connections ) created during the execution and returns the runspaceID as the result of the operation. Actually the PowerShell runspace is saved in the OO session and can be used in other operations using the runspaceID result of previous operation.
  Default value: false
  enablePiping - If true the operation enables piping the way PowerShell does. This input should be used only when keepSessionAlive input is true. First time the piped objects are null and the scope
  </pre>
```

Design | Properties

Figure 8: Description of a generated flow

The description of each generated flow contains the following items:

- A description of the cmdlet as found in its native documentation
- A link where the user can find a detailed description of the cmdlet written by the provider of the module
- The description of the common inputs copied from the PowerShell Script operation's description.
- The description of the common results

Most of the information included in the description can be obtained from the PowerShell console as shown below:



```

Select Administrator: Windows PowerShell
PS C:\Users\ooadmin.00DEU> get-help get-datastore -full
NAME
    Get-Datastore
SYNOPSIS
    Retrieves the datastores available on a vSphere server.
SYNTAX
    Get-Datastore [-Server <UIServer[]>] [-Id <String[]>] [[-Name] <String[]>] [-Datacenter <Datacenter[]>] [-UMH
    MHost[]] [-VM <VirtualMachine[]>] [-Entity <VIObject[]>] [-Refresh] [<CommonParameters>]
DESCRIPTION
    Retrieves the datastores available on a vSphere server. Returns a set of datastores that correspond to the fi
    riteria defined by the cmdlet parameters. To specify a server different from the default one, use the -Server
    eter.
PARAMETERS
    -Server <UIServer[]>
        Specify the vSphere servers on which you want to run the cmdlet. If no value is given to this parameter,
        mmand runs on the default servers. For more information about default servers, see the description of Con
        IServer.
        Required?                false
        Position?                named
        Default value
        Accept pipeline input?    false
        Accept wildcard characters? true

    -Id <String[]>
        Specify the Ids of the datastores you want to retrieve.
        Required?                false
        Position?                named
        Default value
        Accept pipeline input?    false
        Accept wildcard characters? true

    -Name <String[]>
        Specify the names of the datastores you want to retrieve.
        Required?                false
  
```

Figure 9: Obtaining the description from the PowerShell console

## PowerShell Script Operation

The operation is used to execute a PowerShell script or cmdlet on a target host, either local or remote. If the operation executes a single cmdlet, the parameters of the cmdlet should be passed to the operation inputs.

**Note:** When interacting with a Microsoft Exchange server, use the generic PowerShell operation available in the **Business Application** content pack, under **/Library/Operations/Exchange/<version specific>** folder, instead of loading the Exchange snap-in to this PowerShell Script operation. This is a Microsoft Exchange design.

See <http://www.get-exchange.info/2012/12/30/powershell-scripting-for-exchange-server-some-tips/>

### Inputs

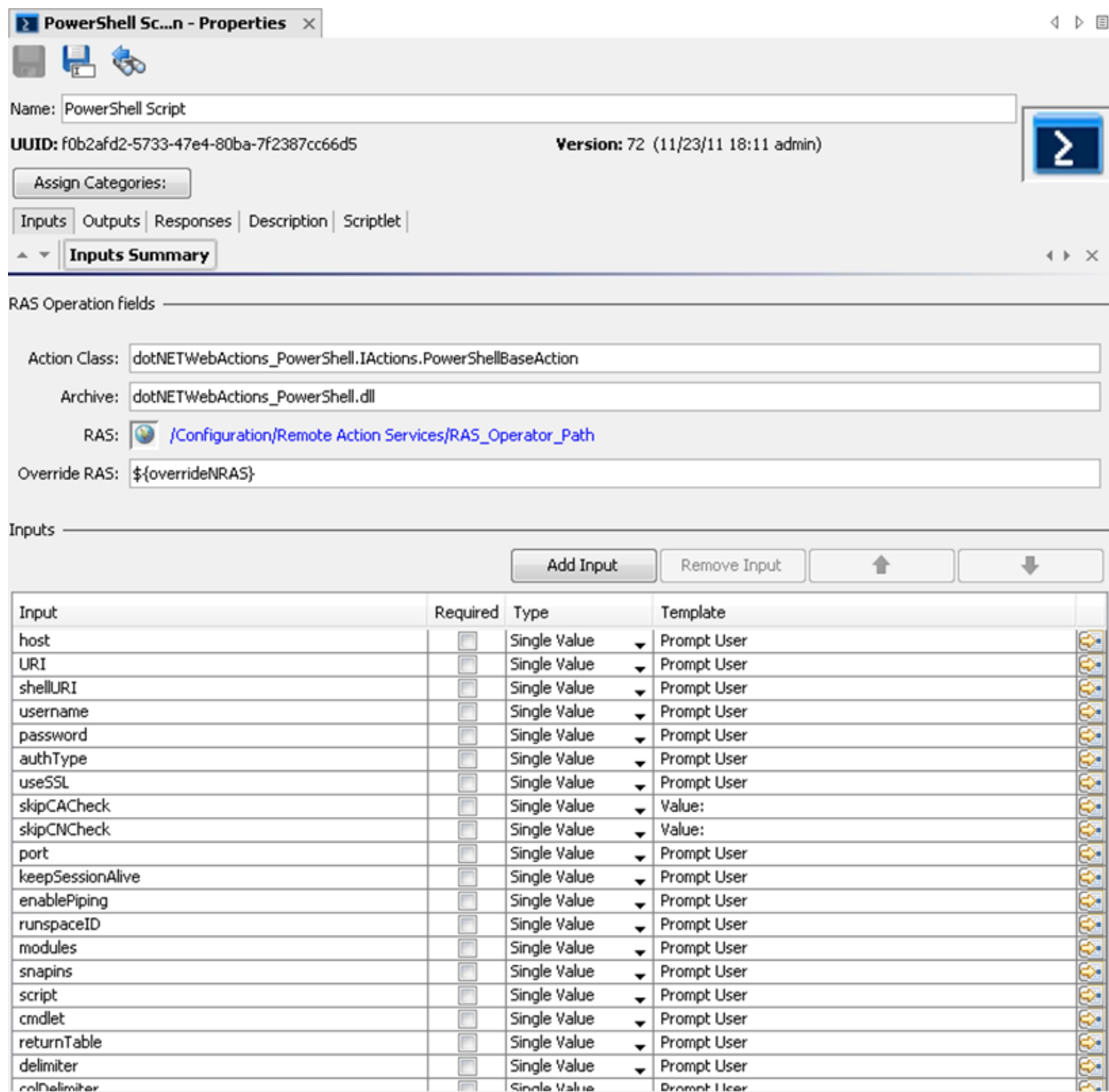


Figure 10: Inputs of the PowerShell Script operation

## Connection Inputs

### host

The hostname or IP address of the PowerShell host. This input is mutual exclusive with URI and shellURI

Default value: localhost

### URI

Specifies a Uniform Resource Identifier (URI) that defines the connection endpoint for the interactive session.

Value format: <Transport>://<ComputerName>:<Port>/<ApplicationName>

Example: **http://exch2010CAS1/Powershell?serializationLevel=Full**

**shellURI**

Gets the Uniform Resource Identifier (URI) of the shell that is launched when the connection is made. This input is mutual exclusive with host.

Example: **http://schemas.microsoft.com/powershell/Microsoft.Exchange**

**Username**

The user name to use when connecting to the server.

Value format: **username@domain or domain\username**

**password**

The password to use when connecting to the server

**authType**

Specifies the mechanism that is used to authenticate the user's credentials. Valid values: **Default, Basic, Credssp, Digest, Kerberos, Negotiate, NegotiateWithImplicitCredential** (case-insensitive).

Default value: Default

**useSSL**

If true, the operation uses the Secure Sockets Layer (SSL) protocol to establish a connection to the remote computer. By default, SSL is not used. WS-Management encrypts all Windows PowerShell content transmitted over the network. UseSSL is an additional protection that sends the data across an HTTPS connection instead of an HTTP connection.

Default port for SSL: 5986.

Default value: false

**skipCACheck**

Indicates when connecting over HTTPS that the client does not validate that the server certificate is signed by a trusted certificate authority (CA).

Default value: true

**skipCNCheck**

Indicates whether the certificate common name (CN) of the server and the hostname of the server are not checked for being the same.

Default value: true

**port**

Specifies the network port on the remote computer that is used for this connection. To connect to a remote computer, the remote computer must be listening on the port that the connection uses. The default ports are 5985 (the WinRM port for HTTP) and 5986 (the WinRM port for HTTPS).

Session and Piping Inputs

**keepSessionAlive**

If true, the operation will not close the PowerShell runspace (i.e. the PSSession in case of remote connections ) created during the execution and returns the runspaceID as the result of the operation. Actually the PowerShell runspace is saved in the OO session and can be used in other operations using the runspaceID result of previous operation.

Default value: false

**enablePiping**

If true, the operation enables piping the way PowerShell does. This input should be used only when keepSessionAlive input is true. First time the piped objects are null and the scope is to save the objects resulted from the PowerShell script execution. Next time the piped objects can be referred in the script as "\$\_" objects.

Default value: false

Example: First, run "Get-Service" script with enablePiping=true and keepSessionAlive=true, then run the script "Select-Object -property name,displayname,status|where-object {\$\_.Name -lt "C"}" with enablePiping=true keepSessionAlive=true. Observe the use of "\$\_".

**runspaceID**

If this is not empty, the operation searches the OO session for keys which equal the runspaceID. If the operation finds the runspace specified by the runspaceID it tries to use it and does not create another PowerShell runspace with provided inputs. If the runspace exists but is broken or unavailable the operation uses its authentication parameters and tries to reconnect and recreate the PowerShell runspace.

**Note:** The operation uses the connection parameters of the runspace identified by runspaceID, not the values provided as user inputs. The operation processes the connection inputs, for example, host, username, password, authType, useSSL that is provided by the user only if the runspaceID does not exist in the OO session.

## Additional Modules and Snapins

**Modules**

A list of PowerShell modules that is loaded after the PowerShell connection is established. Each value from the list specifies the name of the module to import. Enter the name of the module or the name of a file in the module, such as a .psd1, .psm1, .dll, or ps1 file. File paths are optional. Wildcards are not permitted. Specify only the module name whenever possible. When you specify a file name, only the members that are implemented in that file are imported. If the module contains other files, they are not imported, and you might be missing important members of the module. The list of modules should be separated by the comma "," delimiter.

Example: FailoverClusters

**Snapins**

A list of PowerShell snapins loaded after the PowerShell connection is established. Each value from the list specifies the name of a registered snapin, for example, the Name, not the AssemblyName or ModuleName.

Example: Microsoft.Exchange.Management.PowerShell.E2010

## PowerShell Script and cmdlet Inputs

**script**

The script to execute on the PowerShell host. If you want to execute a script from a file just provide the file path.

Example:

```
C:\PowerShellScripts\GetHost.ps1
```

**Cmdlet**

The name of the PowerShell cmdlet to invoke. If the cmdlet has additional parameters, please provide them as inputs to the operation. If the parameter has the same name as one of the operation's inputs just prefix it with "\_". This input is intended to be used together with the PowerShell wizard so please use the script input whenever possible.

## Formatting the Result

**returnTable**

If true, the operation will return a table containing a row for each PSObject that the script emits. The table's columns represent the properties of these PSObjects, in the propertyName<delimiter>propertyValue format. If false the operation returns a string representation of the result similar to the output from the PowerShell console.

Default value: false.

**delimiter**

The delimiter used to separate each property name from the property value in the output table.

Default value: ":".

**colDelimiter**

The delimiter used to separate columns in the output table.

Default value: ",".

**rowDelimiter**

The delimiter used to separate rows in the output table.

Default value: newline.

## 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.

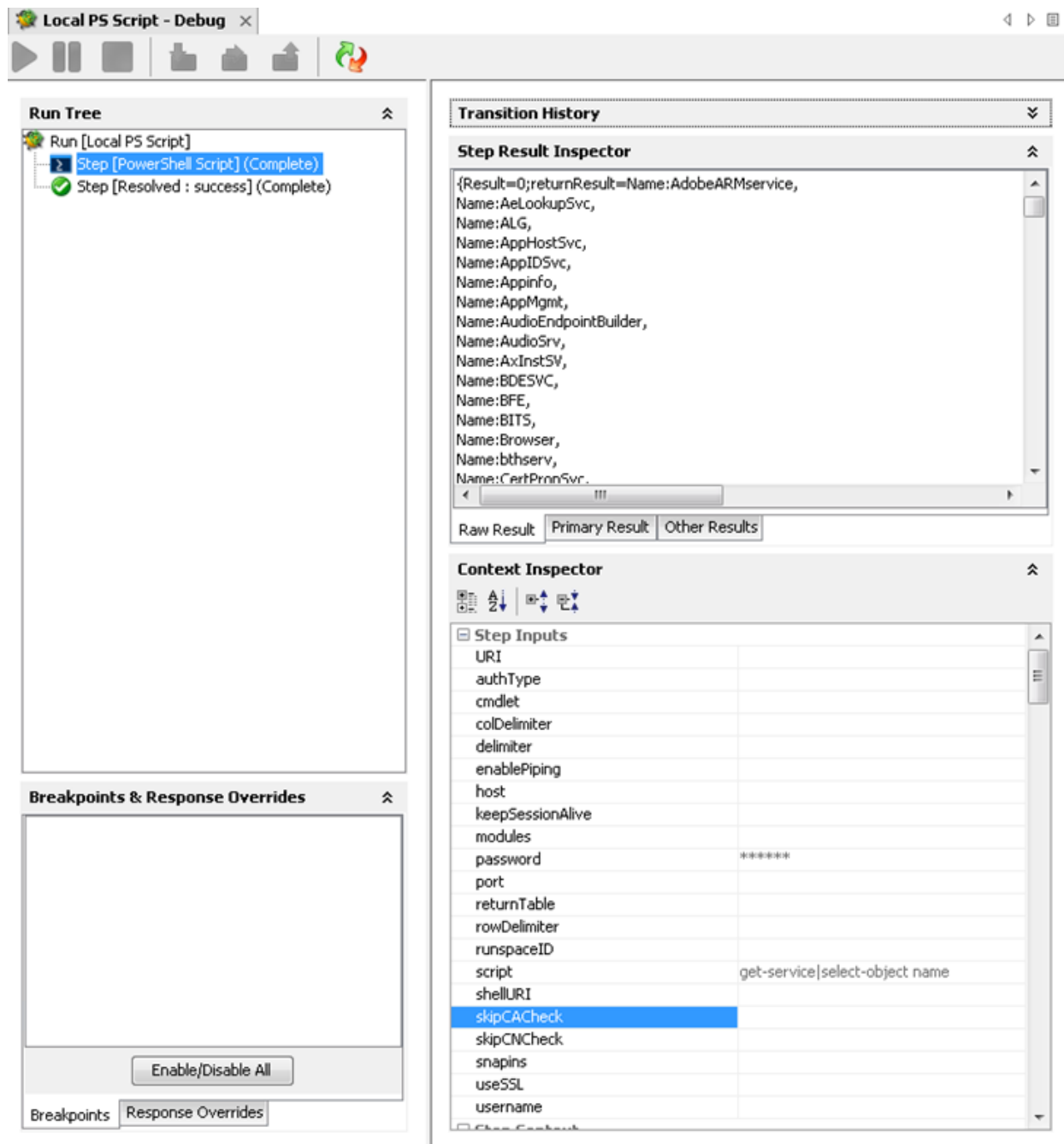


Figure 11: Running the PowerShell script operation on localhost  
 If one script requires elevated rights, enter a user name and a password.

## Running PowerShell Scripts from a File

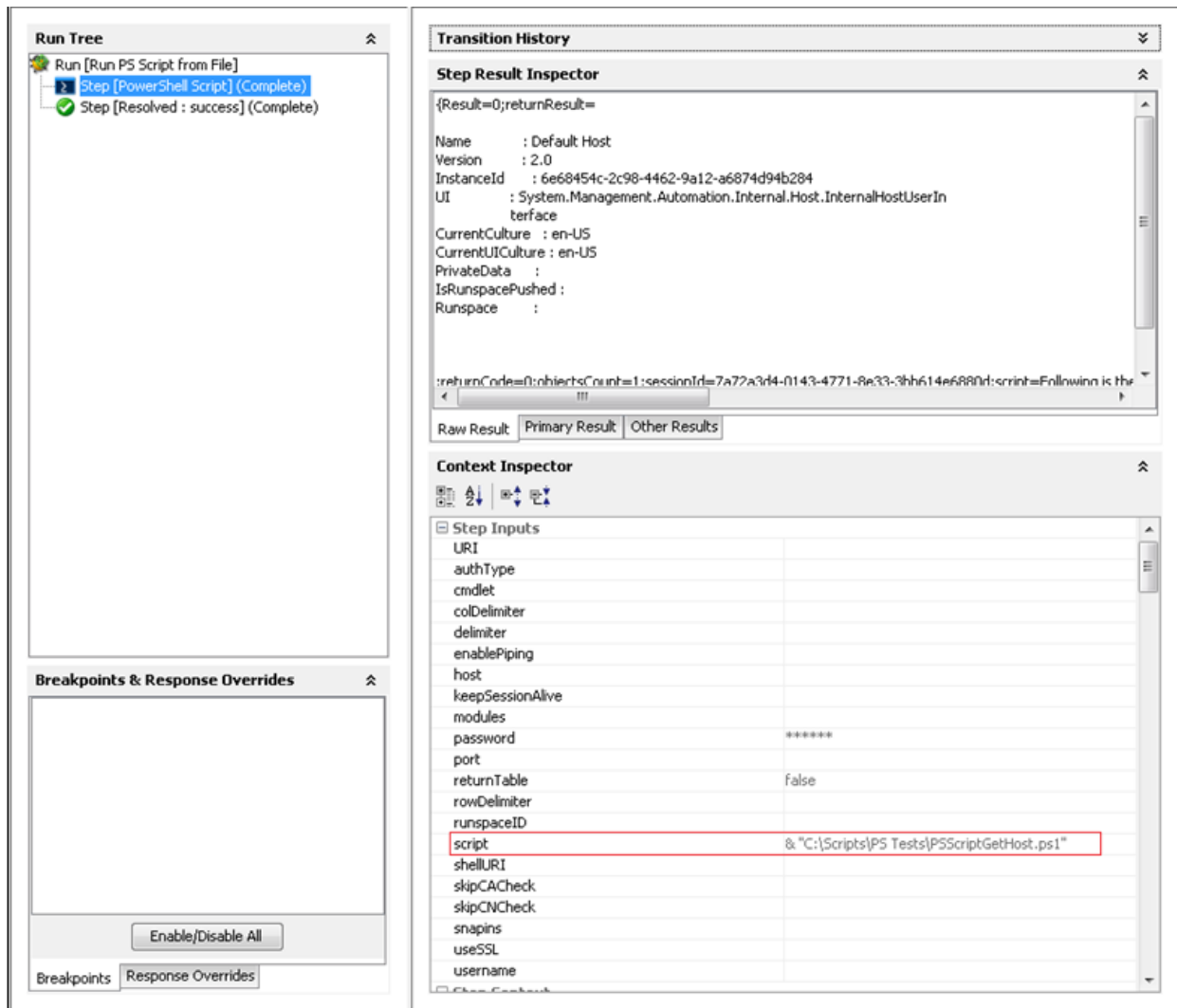


Figure 12: Running the PowerShell script operation from a file

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 like in the picture above.

## Loading PowerShell Functions from Files

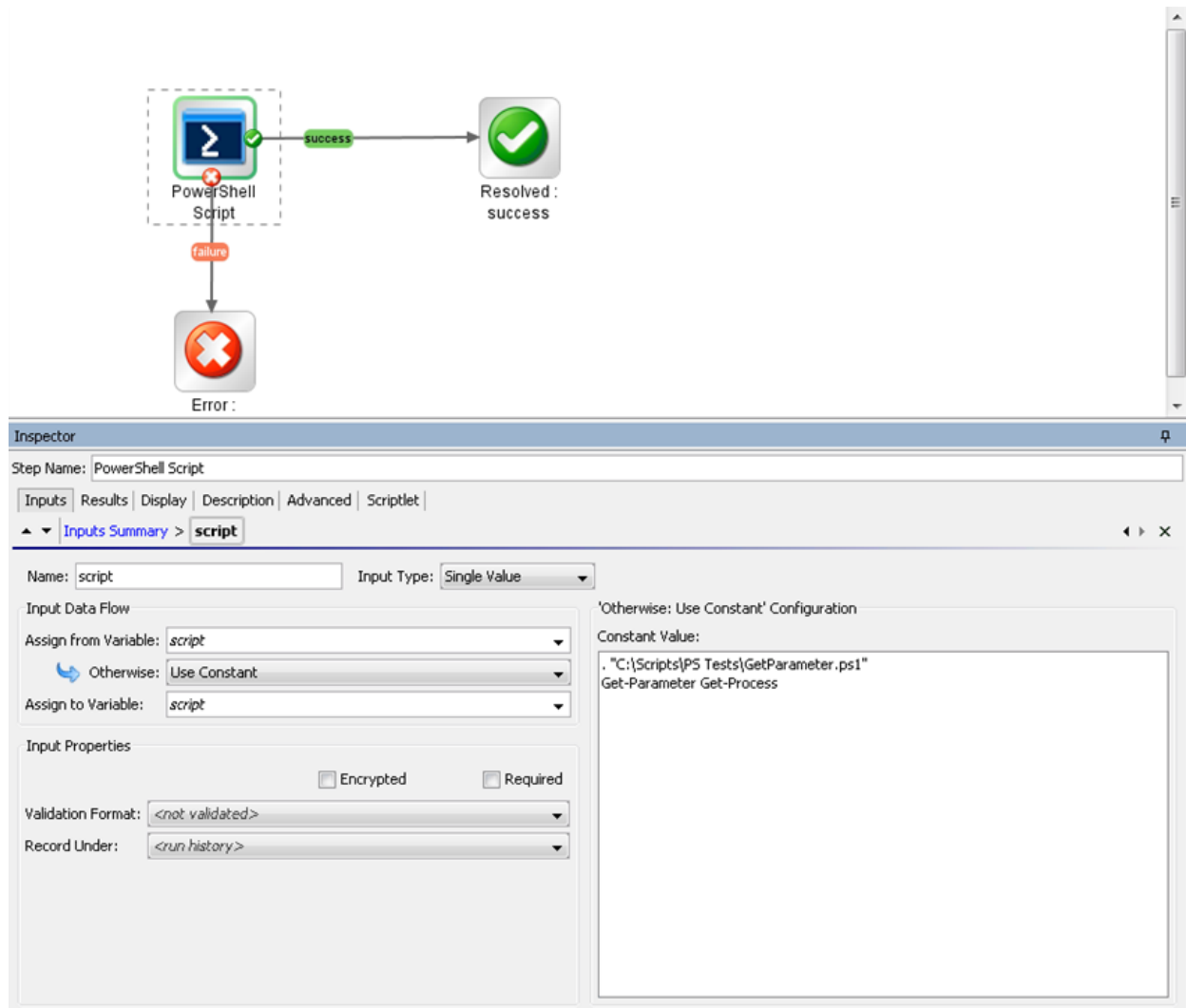


Figure 13: Using a script to enable all functions from a local file

In certain cases, a PowerShell script depends on functions from other file from the disk. The script from the picture above 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.



## Running a PowerShell Script on a Remote Host

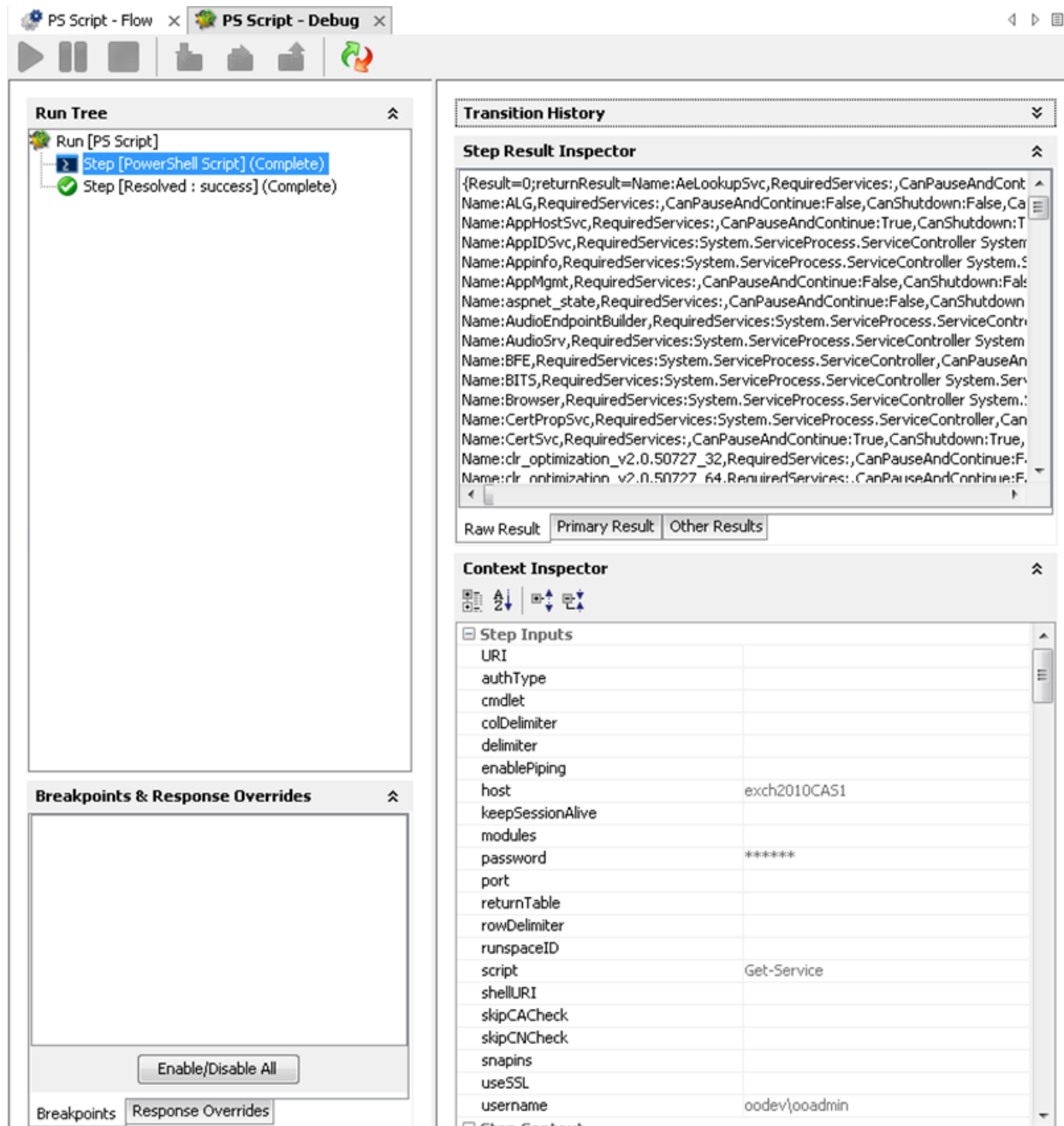


Figure 14: Running the PowerShell script operation on a remote host

First, execute the script providing only the required inputs: host, user name, password and the script.

Negotiate is the default authentication type if the host is provided as an IP address and Kerberos for host names. See PowerShell Remoting and Authentication Types if the connection fails.

## Formatting the Result

The result can be formatted as a table or the same way it will be displayed in the PowerShell console. The format is decided by the **returnTable** input described above. If **returnTable** is set to false the operation will return the result as in the PowerShell console and like the old operation.



Figure 15: The result if returnTable=false

The result is human 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 as 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 Inputs section for more information.

For example Get-Service returns the following result:

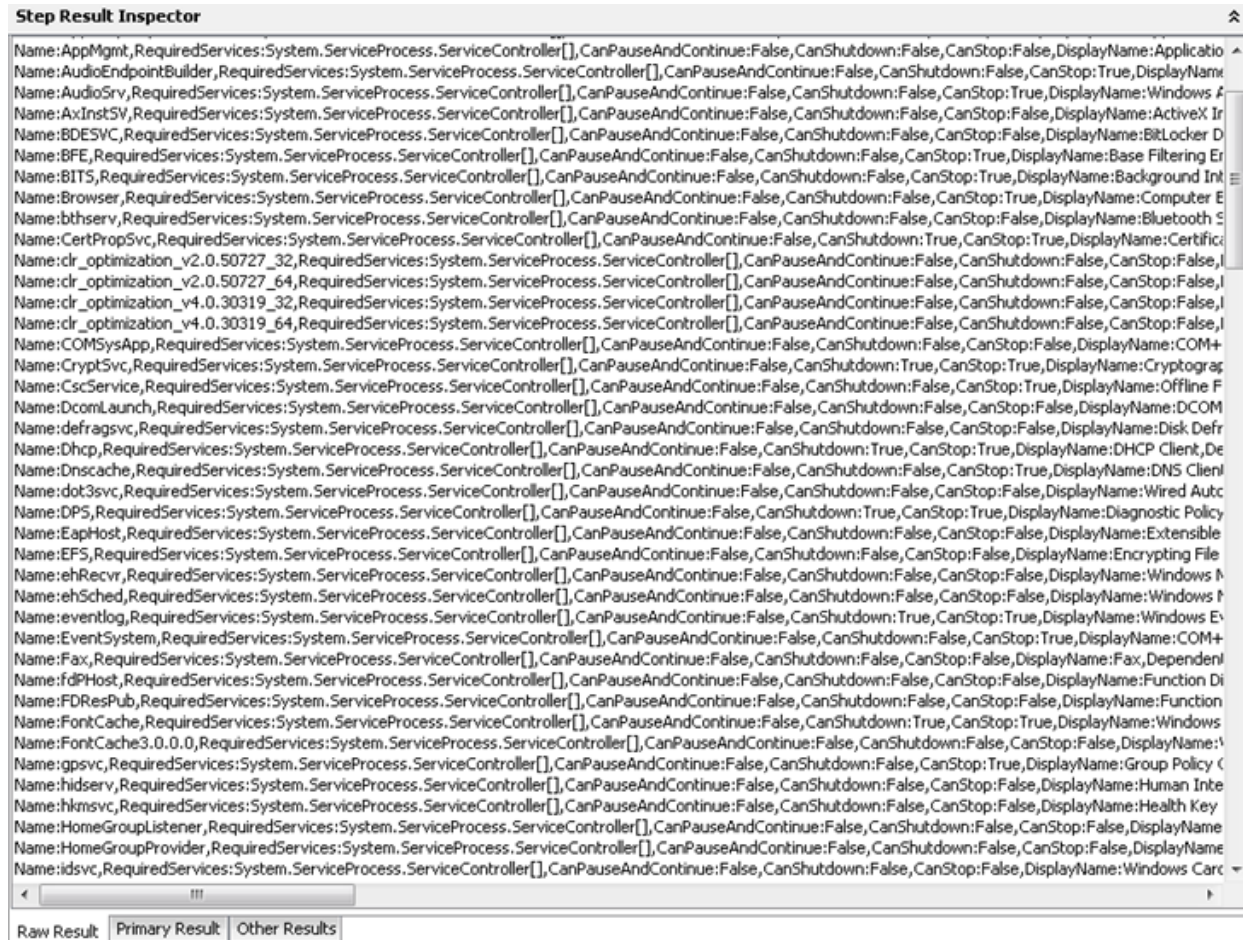


Figure 16: Raw result of the Get-Service flow

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 writing two types filters on the result.

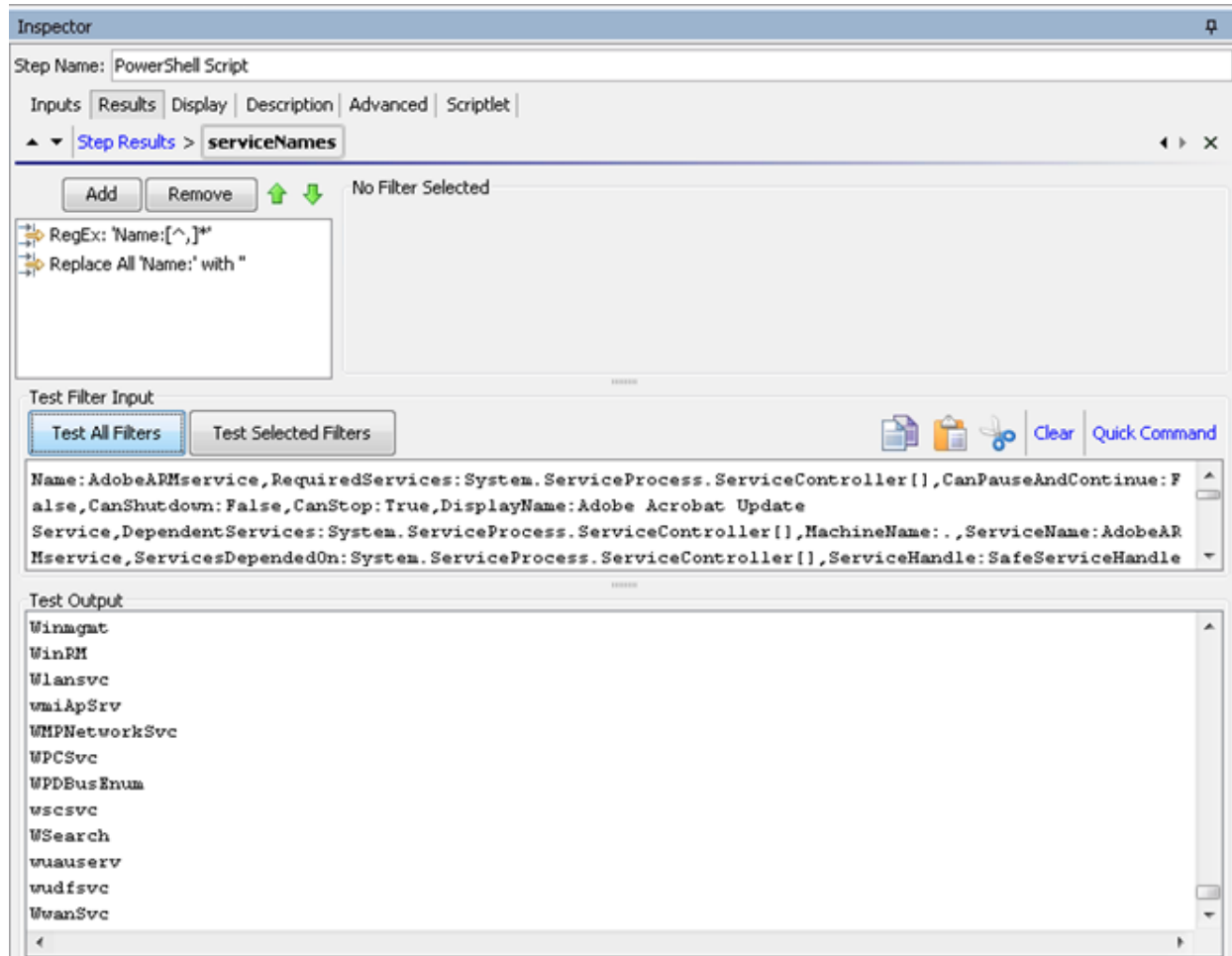


Figure 17: Filtering the results

## Running 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 from the user 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, one uses the PowerShell Wizard, connects to a host which has PowerCLI installed on it and follows the wizard steps to generate HPE OO flows for the PowerShell cmdlets to execute VMWare tasks. Suppose that after the wizard finishes, the user wants 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 VServer 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 VServer before Get VM.

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

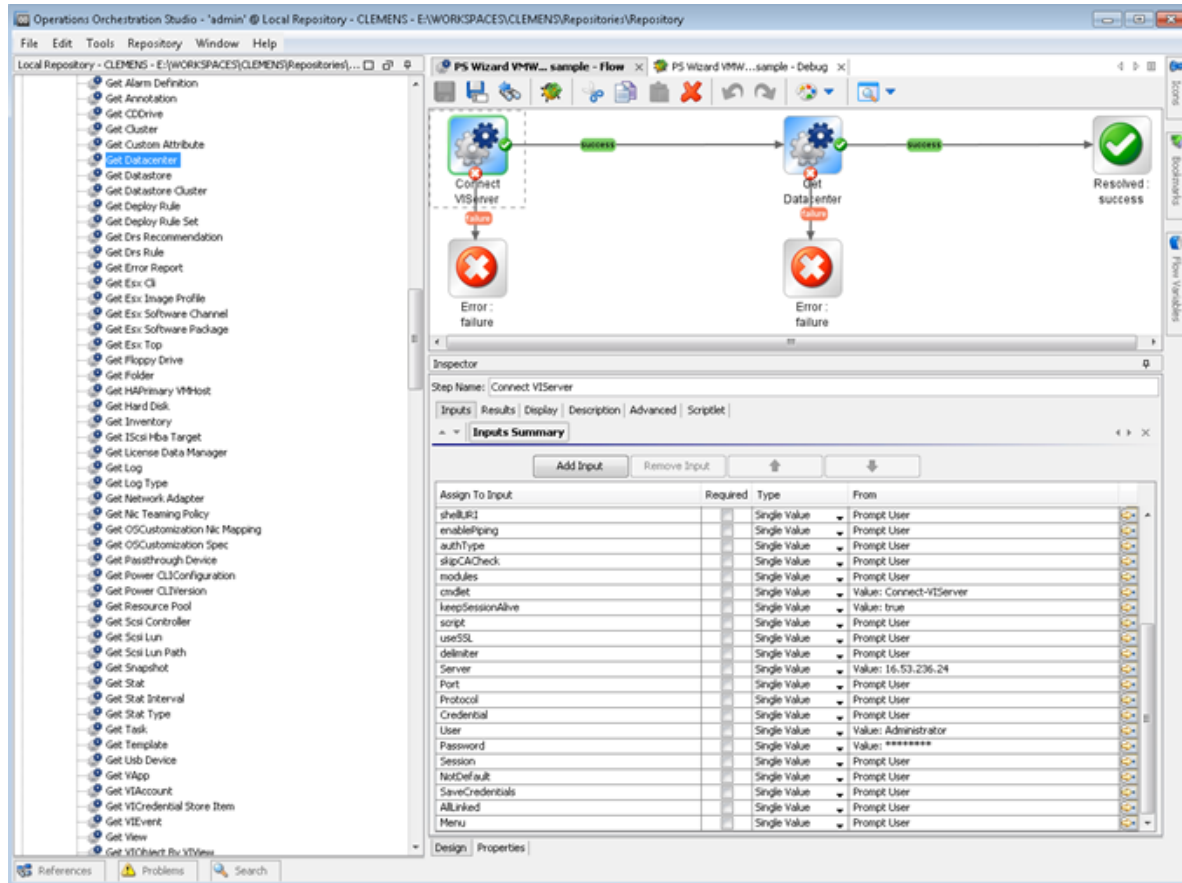


Figure 18: Adding values to specific parameters of the cmdlet

The user runs the flow and provides inputs for the server, user name and password (Connect VServer). No inputs are required for the Get Datacenter. The Connect VServer succeeds, however, the Get Datacenter fails.

The Get Datacenter fails for the following reasons:

- Connect VServer passes successfully and the connection to the VMWare server was established.
- Connect VServer 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 VServer 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 VServer; the runspaceID must be added to the results of the Connect VServer flow:

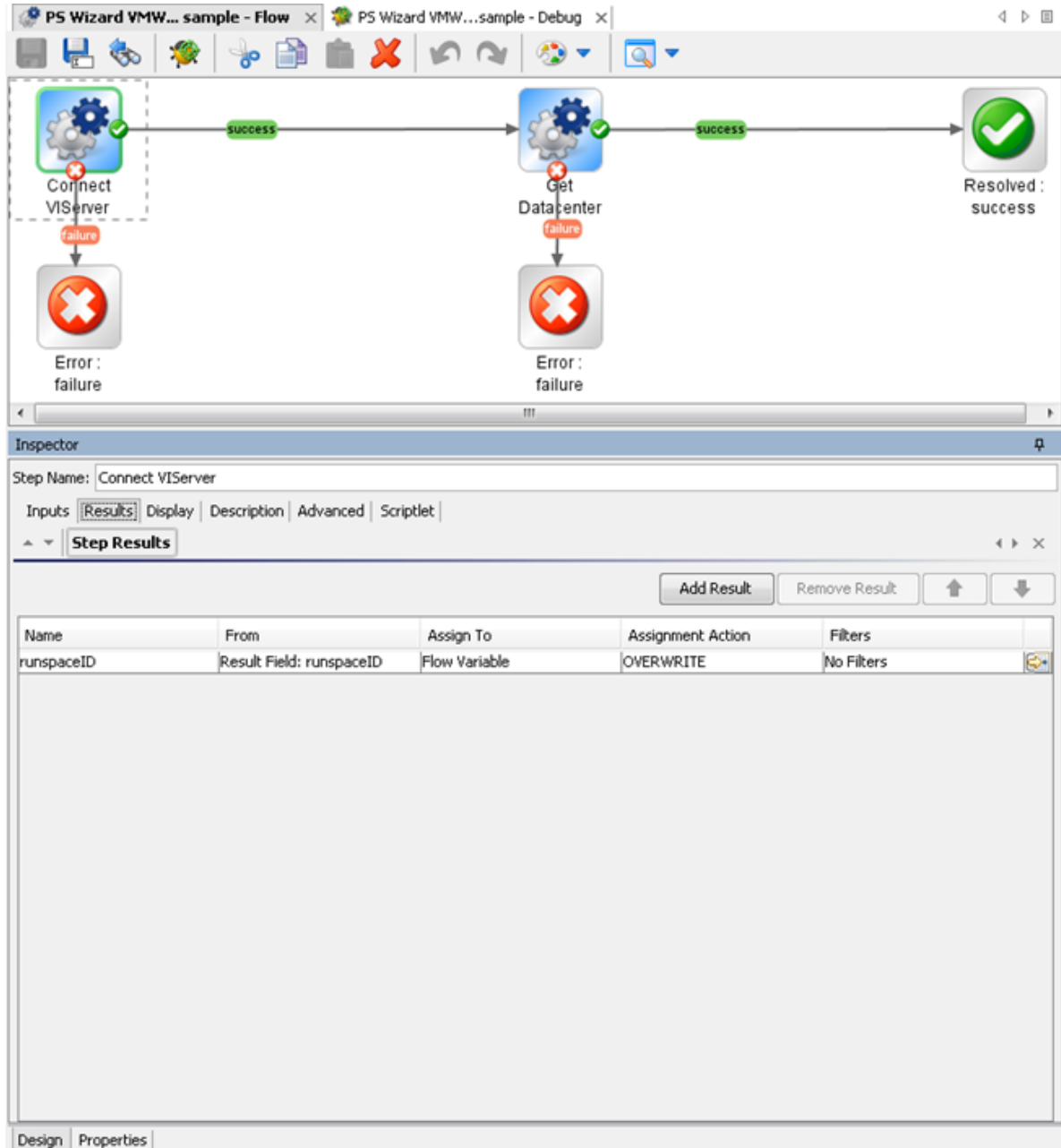


Figure 19: Adding a result to the Connect VServer flow

4. The runspaceID of the Get Datacenter flow must get its value from the result of the Connect VServer. This happens automatically because runspaceID input assigns its value from the flow variable. At this point the flow completes successfully.

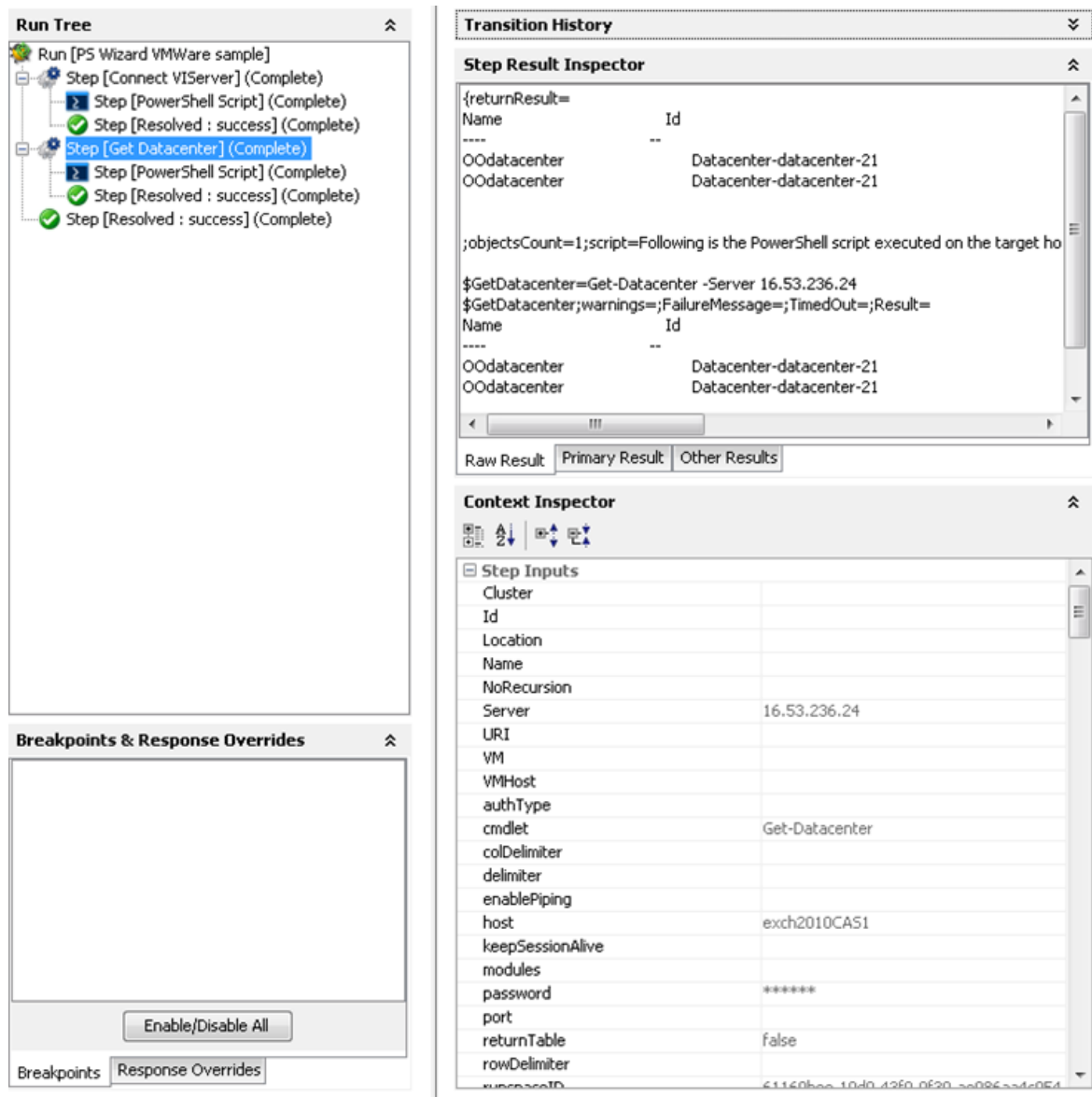


Figure 20: The execution of the PSWizard VMWare sample flow

## Assigning 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 from the user. But there are some cmdlets, probably the “new” cmdlets, which require as parameters the result of another cmdlet. For example, one would like to create a new virtual machine using the generated flow New VM.

Even if the user follows all the steps described in the previous section, the flow can not be executed. The PowerShell Wizard generates the flow, but the user 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:

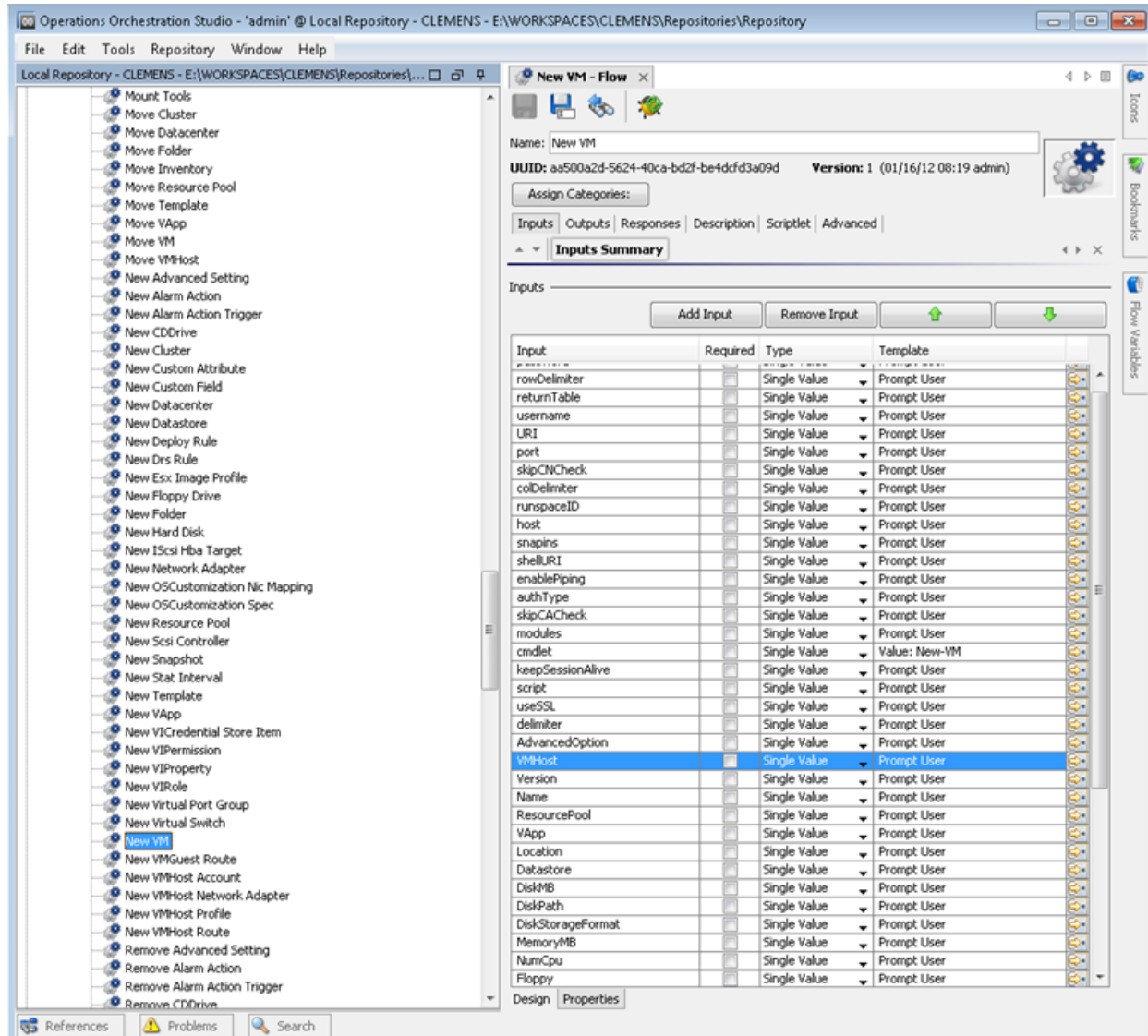


Figure 21: The inputs of the New VM flow

The New VM flow 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 from 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 the user can find detailed information about the cmdlet.



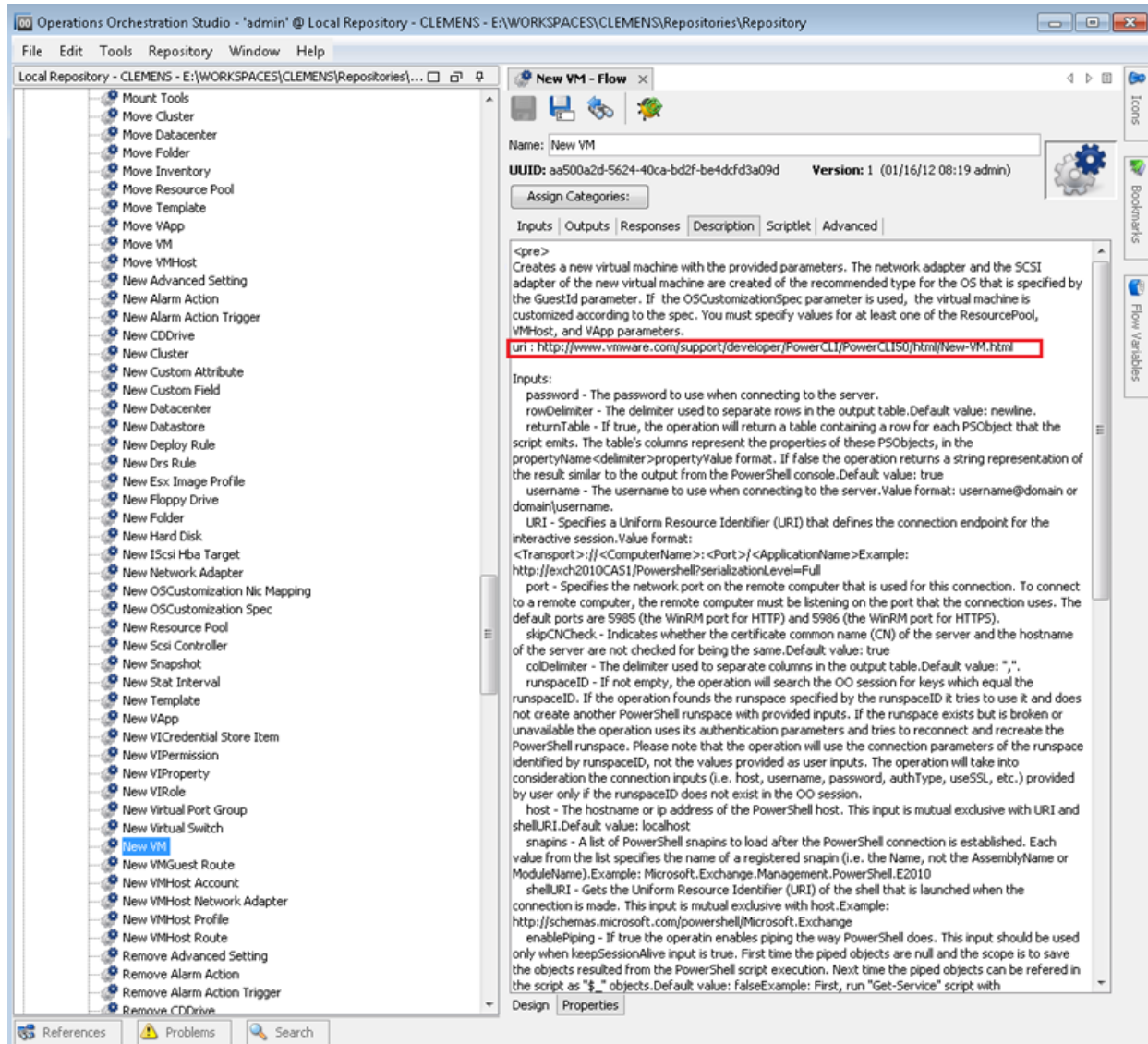


Figure 22: The description of the New VM flow

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

### Parameters

NAME	TYPE
<b>VMHost</b>	<a href="#">VMHost</a>
<b>Template</b>	<a href="#">Template</a>
<b>AdvancedOption</b>	<a href="#">AdvancedOption[]</a>
<b>AlternateGuestName</b>	String
<b>CD</b>	SwitchParameter
<b>Confirm</b>	SwitchParameter
<b>Datastore</b>	<a href="#">StorageResource</a>

Figure 23: The VMHost object: parameters

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 the user can find which cmdlets return VMHost objects as it can be observed below. In our case the type suggests that a cmdlet like Get-VMHost would return this kind of objects.

---

#### vSphere PowerCLI Reference

##### VMHost - Object

---

**Property of**

[VMHostProfileInput](#), [HostVMKernelVirtualNic](#), [EsxCli](#), [VirtualMachine](#), [VMHostFirewallDefaultPolicy](#), [Log](#), [VMHostPatchResult](#), [HostService](#), [VMHostAutonomousUpdate](#), [HostVirtualNic](#), [VMHostDiagnosticPartition](#), [VirtualSwitch](#), [VmHostModule](#), [VMHostNetworkInfo](#), [HostNic](#), [IScsiHba](#), [VMHostProfile](#), [VMHostPatch](#), [HostNicTeamingPolicy](#), [VMHostProfileIncompliance](#), [PciPassthroughDevice](#)

**Parameter to**

[Start-VMHost](#), [Restart-VMHost](#), [Get-VMHostHba](#), [New-VMHostNetworkAdapter](#), [Get-PassthroughDevice](#), [Get-Datacenter](#), [Get-Cluster](#), [Remove-Datacenter](#), [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-VMHostDatastore](#), [Get-VMHostStorage](#), [Get-VMHostNetwork](#), [Get-VMHostFirmware](#), [Get-VirtualSwitch](#), [Test-VMHostProfileCompliance](#), [Get-VirtualPortGroup](#), [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](#)

Figure 24: The VMHost object : additional information

In the previous section, we explained how to execute in the same PowerShell session multiple HPE OO flows generated with the PowerShell wizard. At this point we can imagine the following chain of cmdlets which need to be executed to create a new vm:

- **Connect-VIServer** – this must be executed before any VMWare cmdlet;
- **Get-VMHost** – we 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, but we are going to explain how to solve the VMHost parameter, because the process is the same for the other parameters, too.

You need to execute three 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**? Select from one of the following solutions:

## Solution 1: Create a New PowerShell Script Step

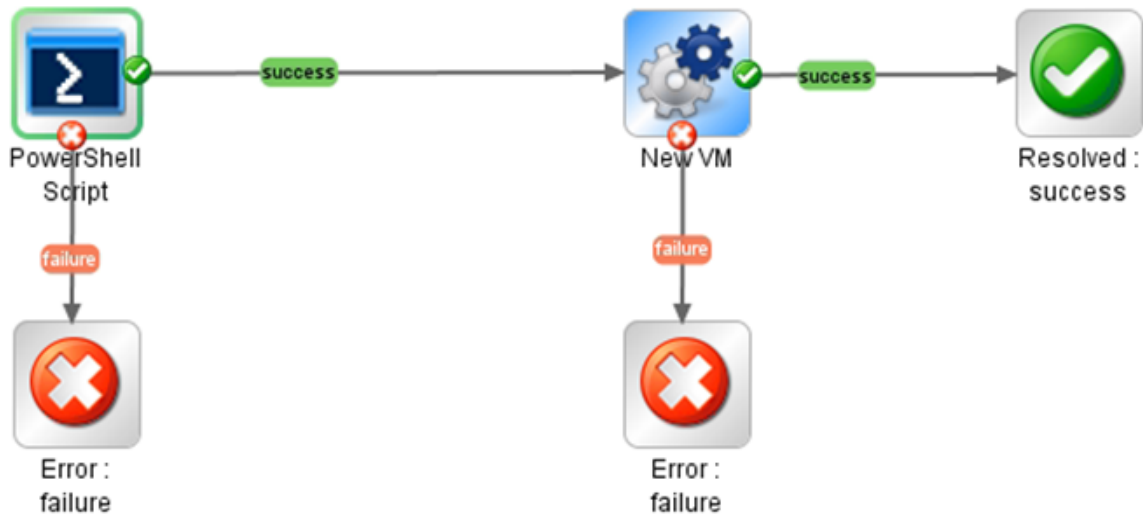


Figure 25: Add a PowerShell script step to the flow

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 2 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**.

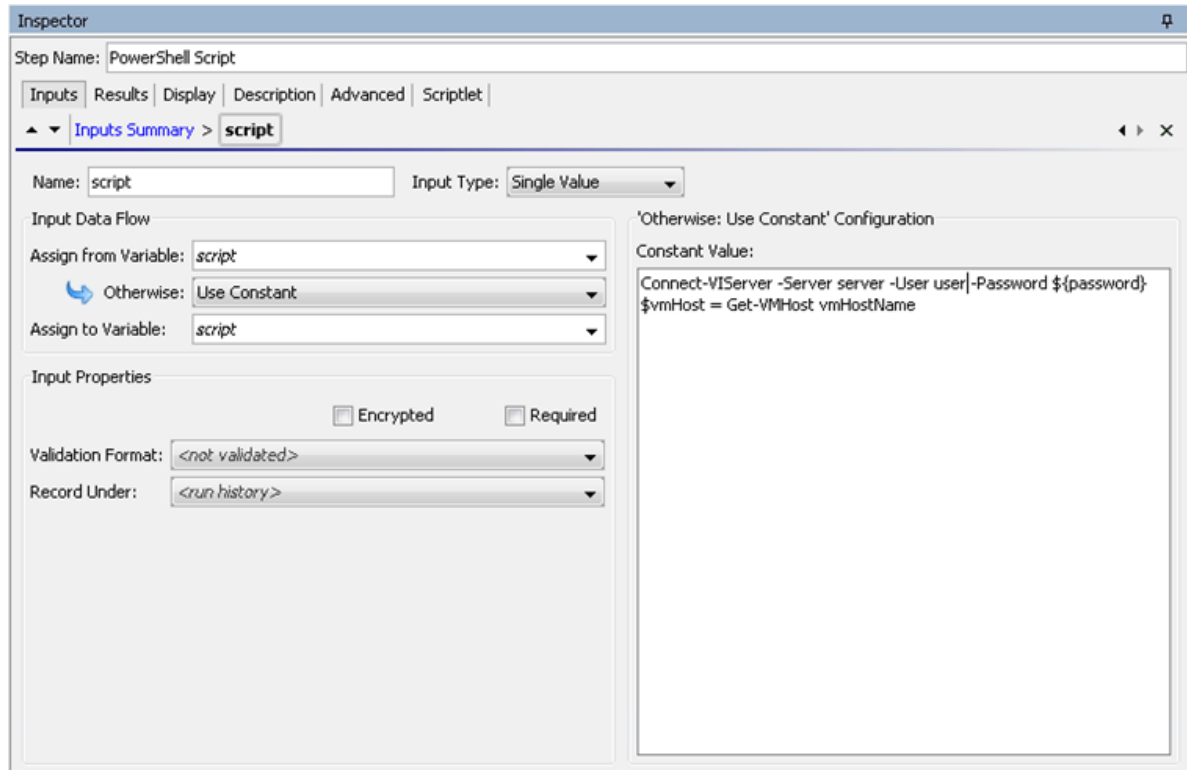


Figure 26: Use a variable to save the result of the step

2. The next step is to assign the value of the VMHost input from the **\$vmHost** variable.

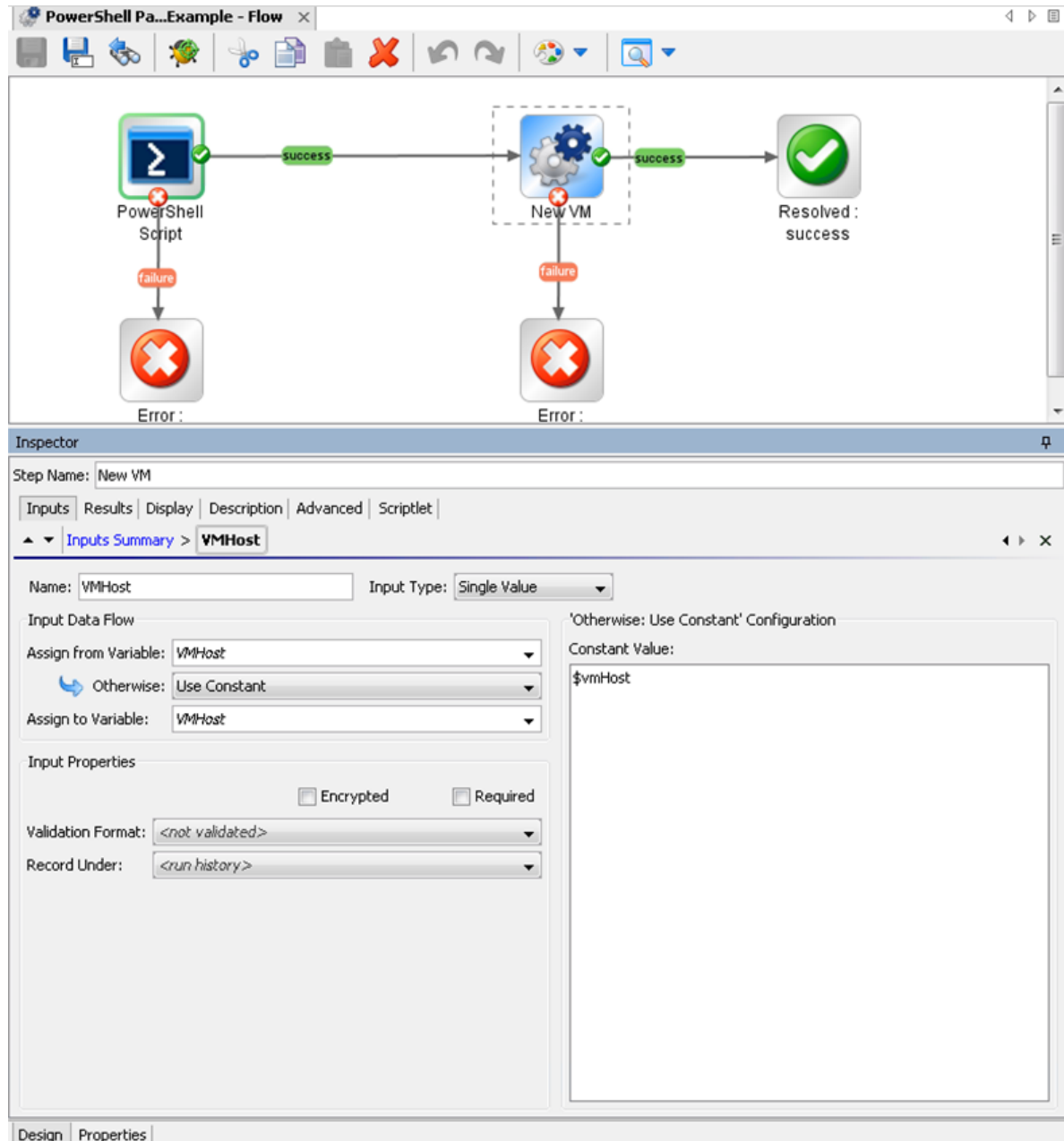


Figure 27: Assign the variable to the VMHost input of the flow

This way we managed to pass PowerShell cmdlets results between HPE OO flows generated with the PowerShell wizard.

**Note:** The `$var` refers to PowerShell variables and `$(var)` refers to HPE OO flow variables.

## Solution 2: Run a PowerShell Script in the Generated Flow Context

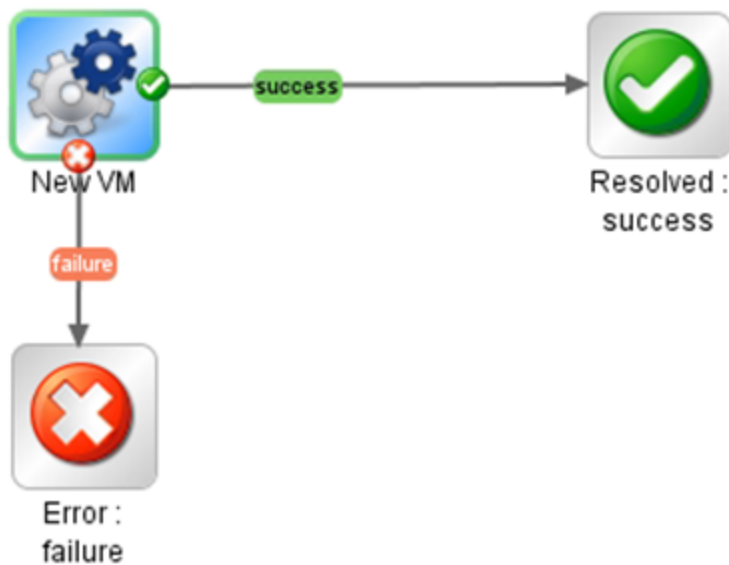


Figure 28: The NewVM Flow

The second solution is to execute the previous defined script 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;
- **Cmdlet** – the PowerShell cmdlet name. If the script input is not empty, than the PowerShell script defined by this input is going to be executed before the cmdlet. Although the script is executed in the same PowerShell runspace with no extra settings.

The solution is shown in the following flow:

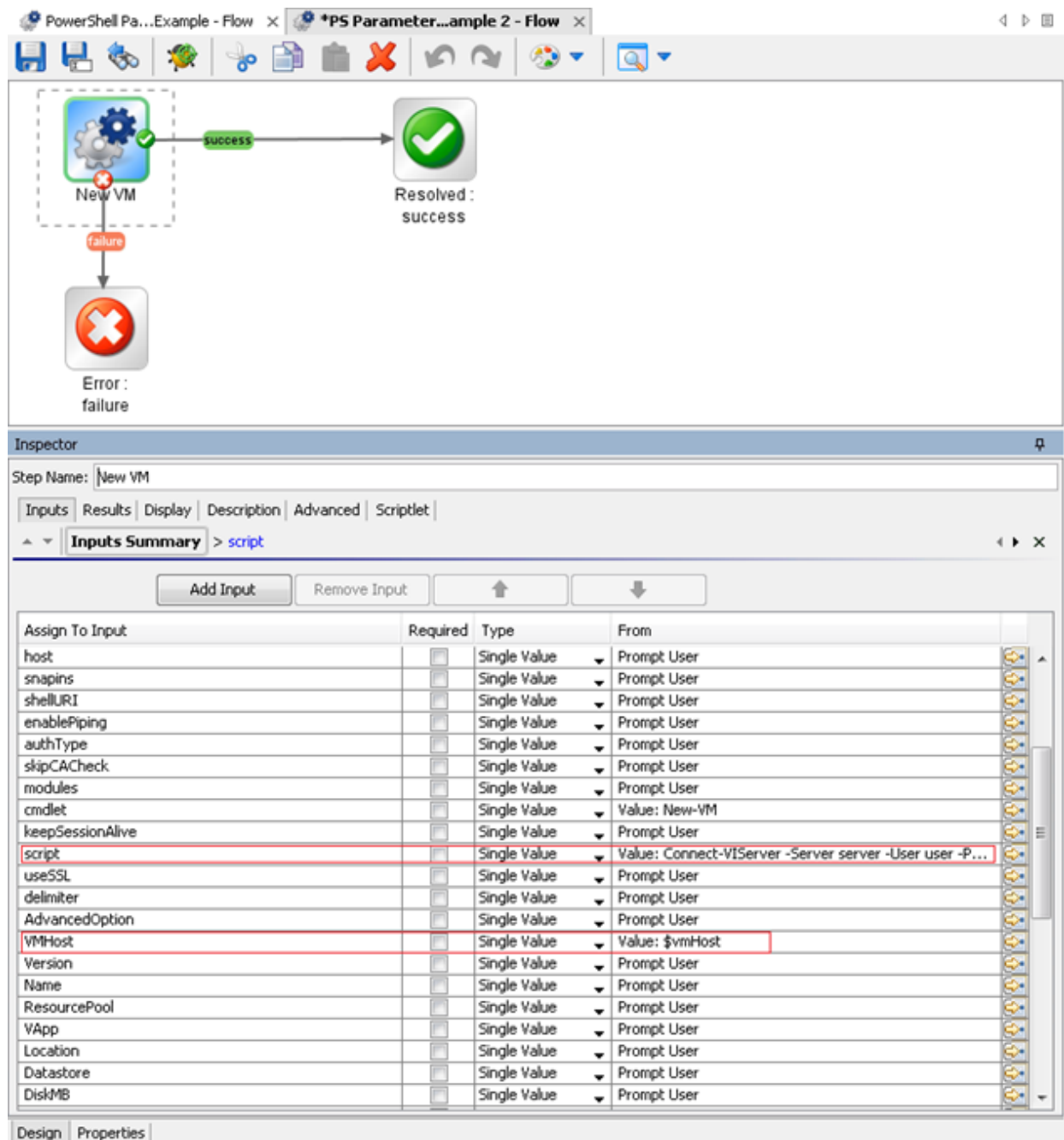


Figure 29: Add a script to the input of the flow

## Solution 3: Use Generated Flows Only and Minimize the User Effort

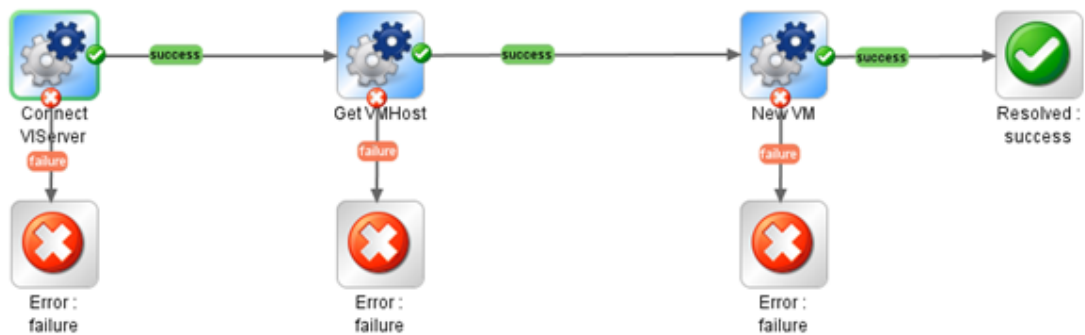


Figure 30: Using only generated flows

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, the user can execute **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.

## PowerShell Remoting

### Overview

Enable PowerShell remoting, by running the following cmdlet: **Enable-PSRemoting**.

In workgroup environments, enable classic mode authentication for network logon:

1. Open **Local Security Policy** from the Control Panel and select **Administrative Tools**.
2. Navigate to **Local Policies \ Security Options**.
3. 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. This can be done using 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).

### Enabling Remoting Using GPO (Group Policy Objects)

While remoting can be enabled manually using **Enable-PSRemoting**, it is recommended to use GPO management tools whenever it is 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**.

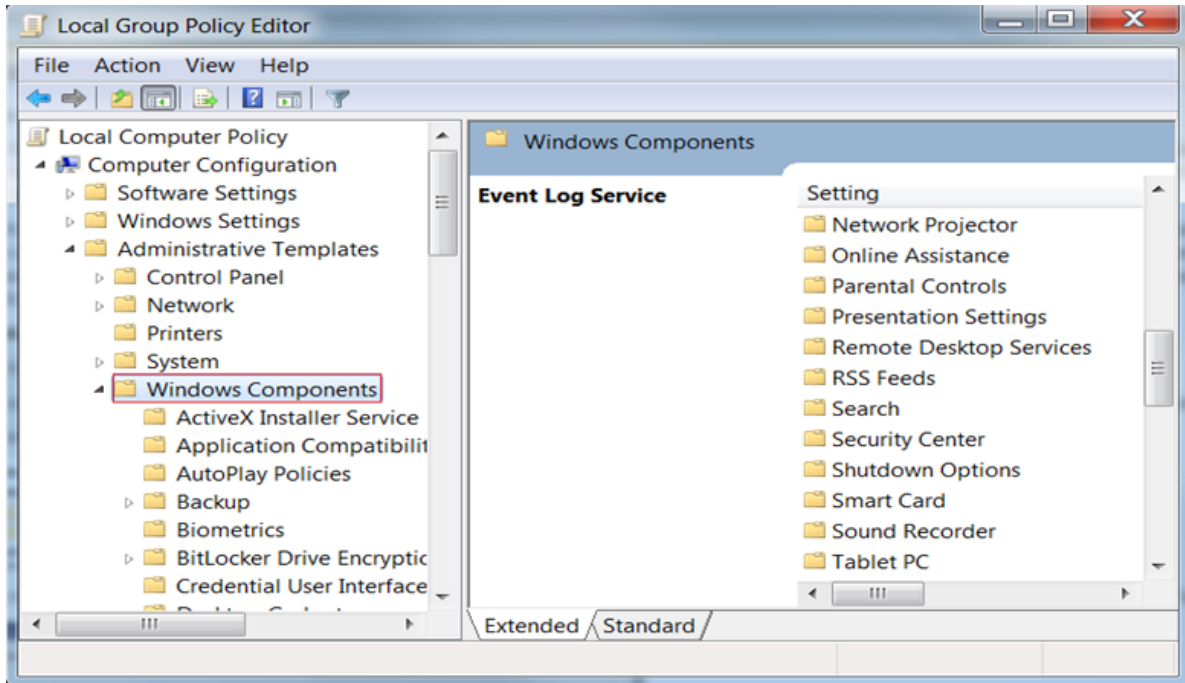


Figure 31: Group Policy Configuration : Windows components section

3. From the Windows Components listed in the right pane, we are interested in two of them. The first one is Windows Remote Management (WinRM) and the second one is Windows Remote Shell. The next step will refer to the first one, therefore browse the components and open Windows Remote Management (WinRM).

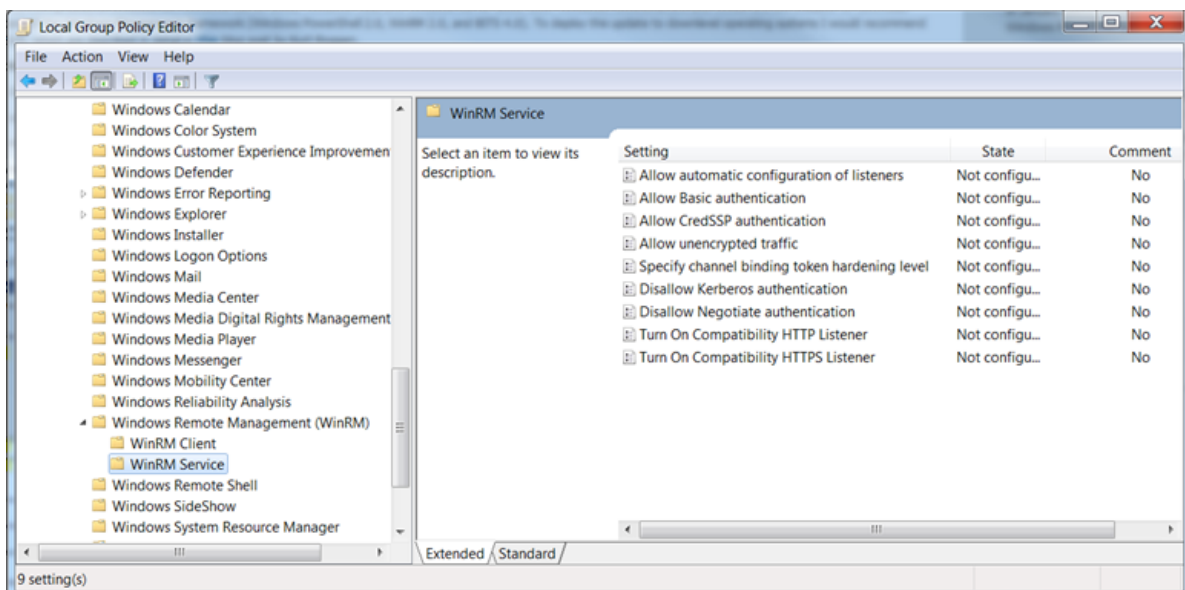


Figure 32: Group Policy Configuration: WinRM Service section

- WinRM is the service that 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 (for example, request access to execute scripts on other hosts or allow other hosts to execute scripts on the current host). At this point, you can enable different authentication types, specify the trusted hosts, enable HTTP or HTTPS listeners, and so on.
- There are some other default settings the user might want to change in a production environment. These settings can be found in Windows Remote Shell as shown below:

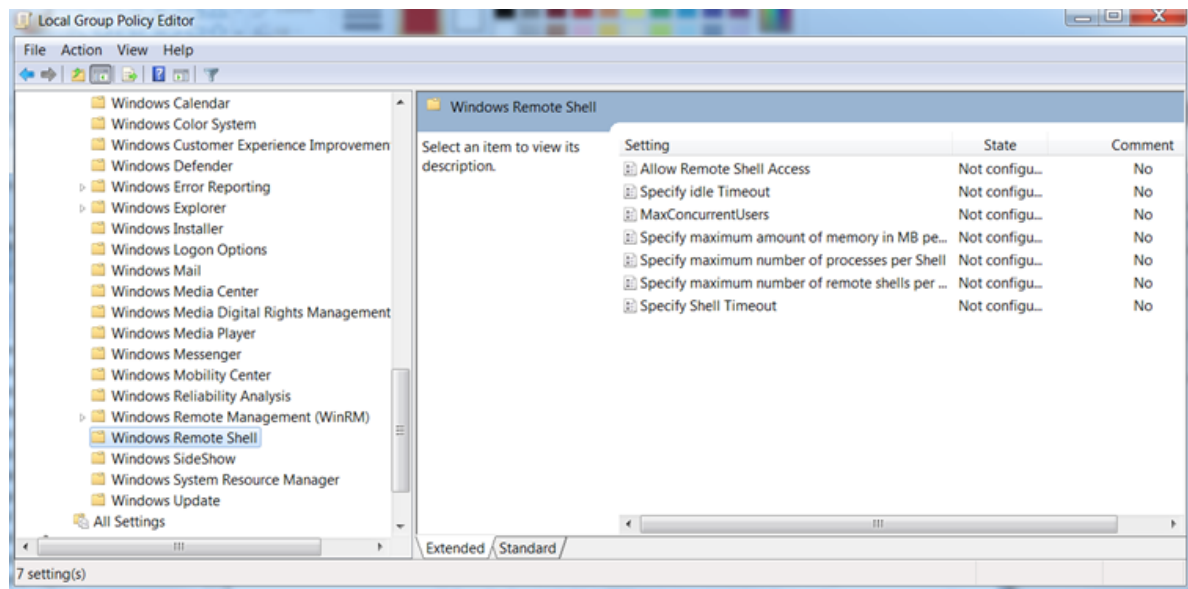


Figure 33: Group Policy Configuration : Windows Remote Shell section

- You can specify the maximum number of remote shells per user (default is 2) or the maximum amount of memory in MB for shell (the default is 150).
- 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

Sometimes the GPO policies must be applied on multiple server hosts and repeating the above steps on every server might not be the best solution. Therefore, you can create a new GPO policy, configure it and apply it on a list of servers.

- Go to the domain controller or on a server where **gpmc.msc** is available and open it.
- Right-click the **Group Policy Object** item as in the following image and choose **New**. Enter the name for the new GPO and select the policy to inherit from, and then go to the next step.

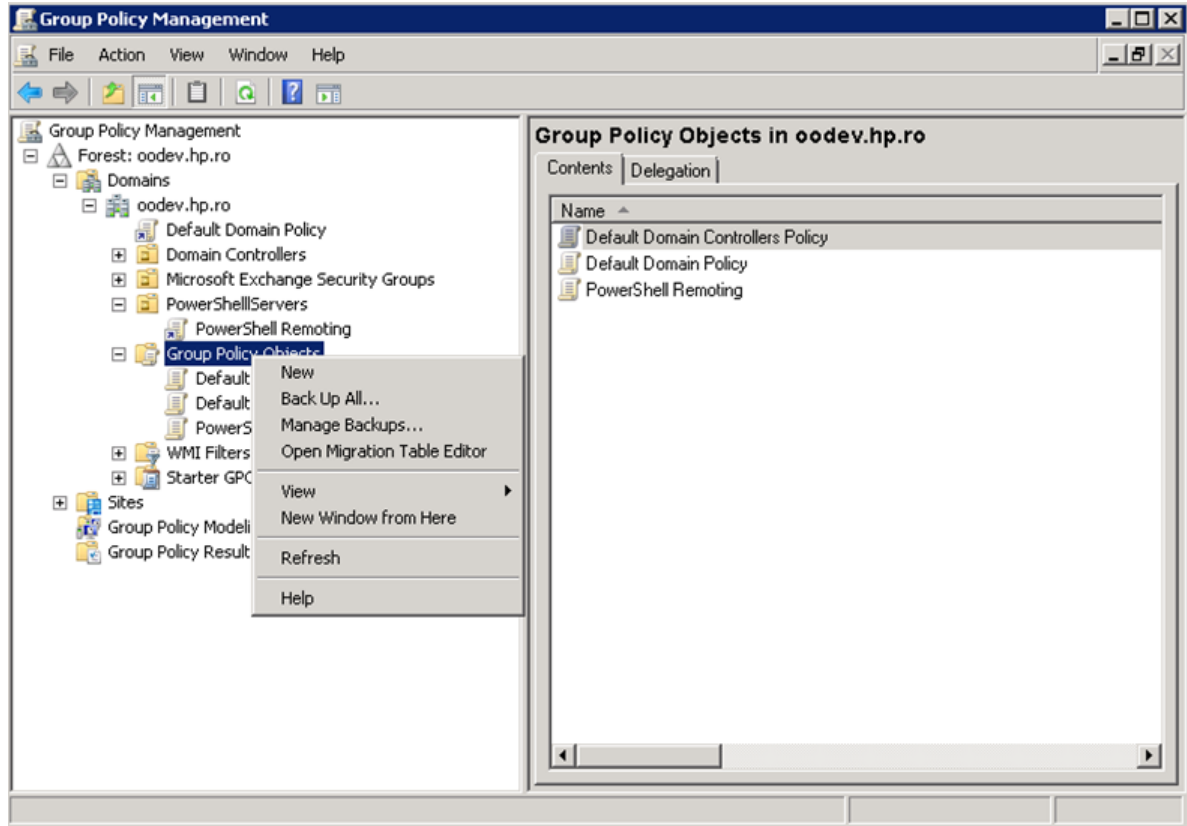


Figure 34: Using Group Policy Objects - step 1

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

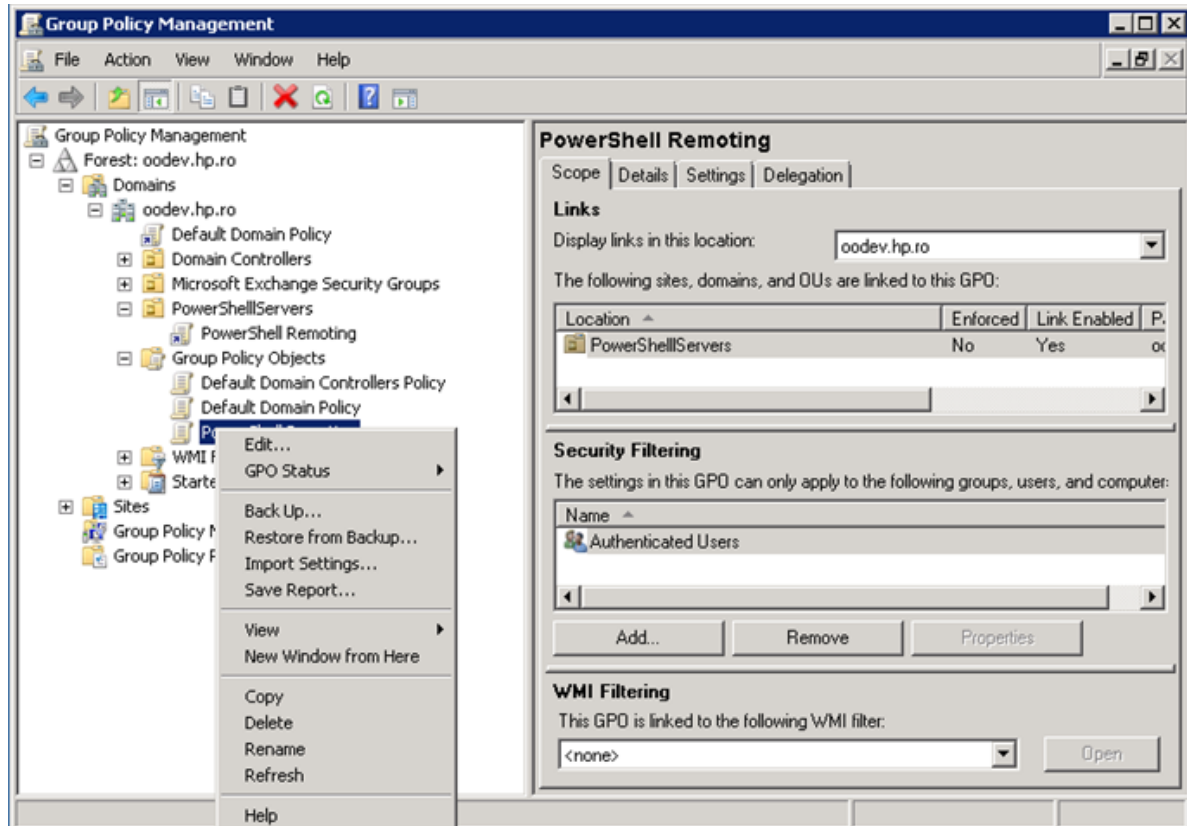


Figure 35: Using Group Policy Objects - step 2

4. Configure the GPO same way as for single hosts.
5. Now that there is a new GPO, you can apply it to a group of servers. The following example shows you how to link it on an existing OU from AD; however, this can be applied to other groups as well. 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**.

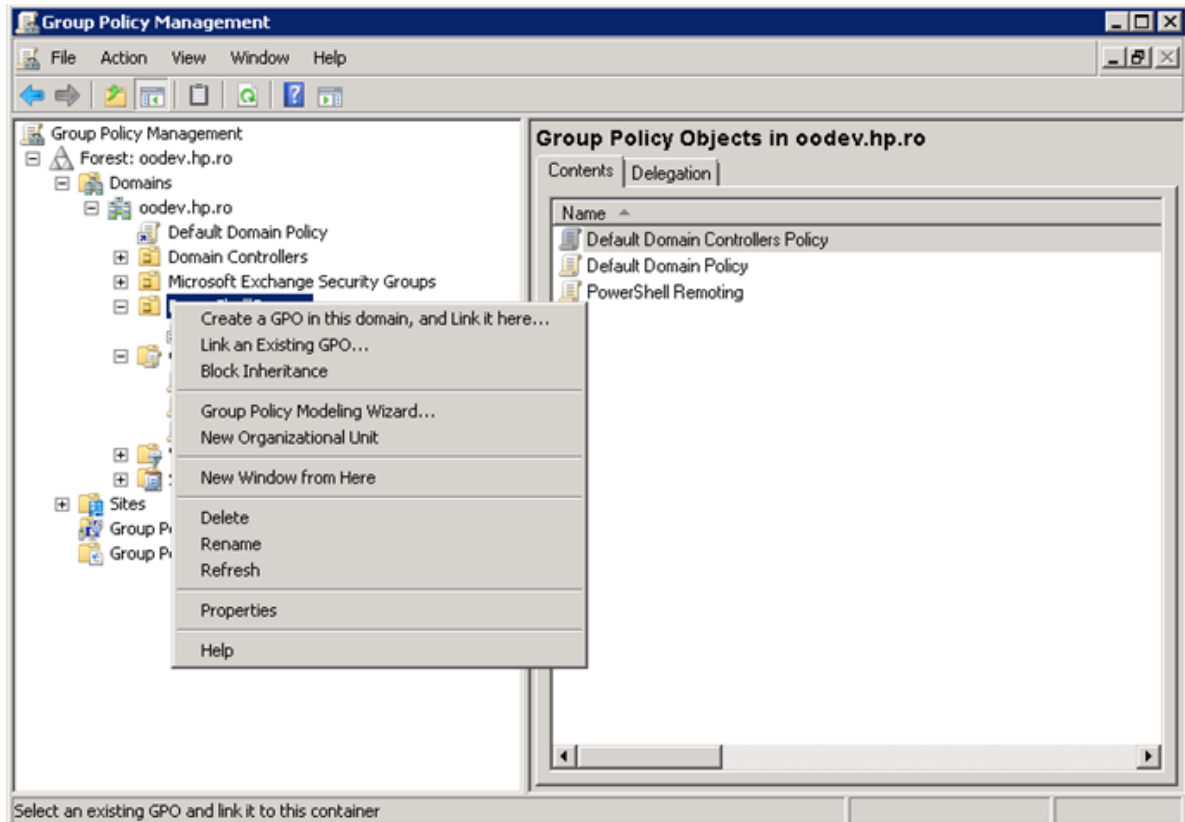


Figure 36: Using Group Policy Objects - step 3

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

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

## Enabling 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
```

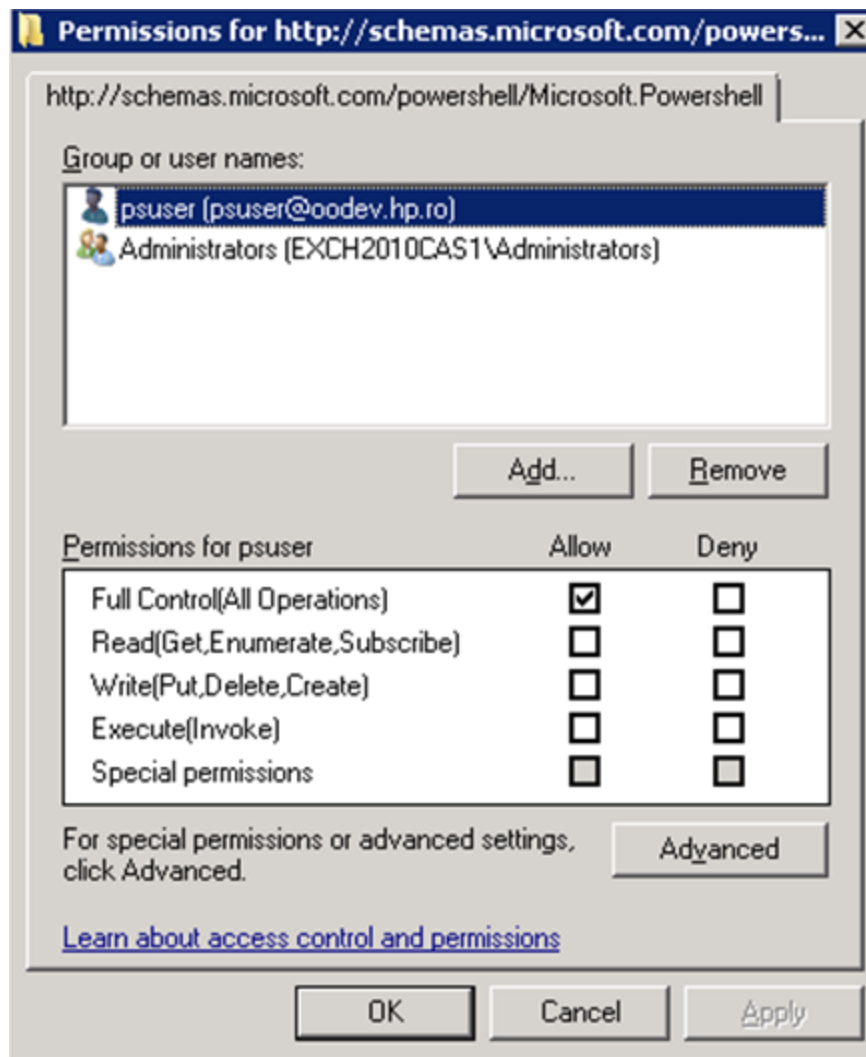


Figure 37: Adding a user to the PowerShell session configuration

## 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
```

- 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.
  - a. Open gpedit.msc.
  - b. Go to **Computer Configuration > Administrative Templates > System > Credentials Delegation**.
  - c. Enable **Allow Delegating Fresh Credentials** and add the wsman hosts to the server list.
  - d. 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 doesn't have "Validated write to service principal name", do one of the following:
  - Try running the following command:
 

```
dsacl "CN=AdminSDHolder,CN=System,DC=domain,DC=com" /G "Sn-1-5-20:WS;Validated write to service principal name"
```
  - Open ADUC, go to **Computers > DC object > Security**, select **Network Service** and 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
```

- 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
```

- 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 host names of the cluster nodes can be used for the destination host.

### NegotiateWithImplicitCredential

- 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].



## 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 see the section [Enable Remoting for Non-Administrative Users](#).
- Authentication problems (most common). See "Run a PowerShell Script on a Remote Host" in ["Using the PowerShell Wizard" on page 62](#).
- 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 (depends on the HPE OO jre), which ends up calling x32 PowerShell. The x32 version of PowerShell cannot load some modules (for example, FailoverClusters); 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 will try to authenticate the localhost like any other remote host. Note that remoting rules should be satisfied for localhost in this case. If user is left empty, the wizard will connect 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

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. See the "Enable Remoting Using GPO (Group Policy Objects)" section in ["PowerShell Remoting" on page 88](#) in order to increase the allowed number of remote shells per user.

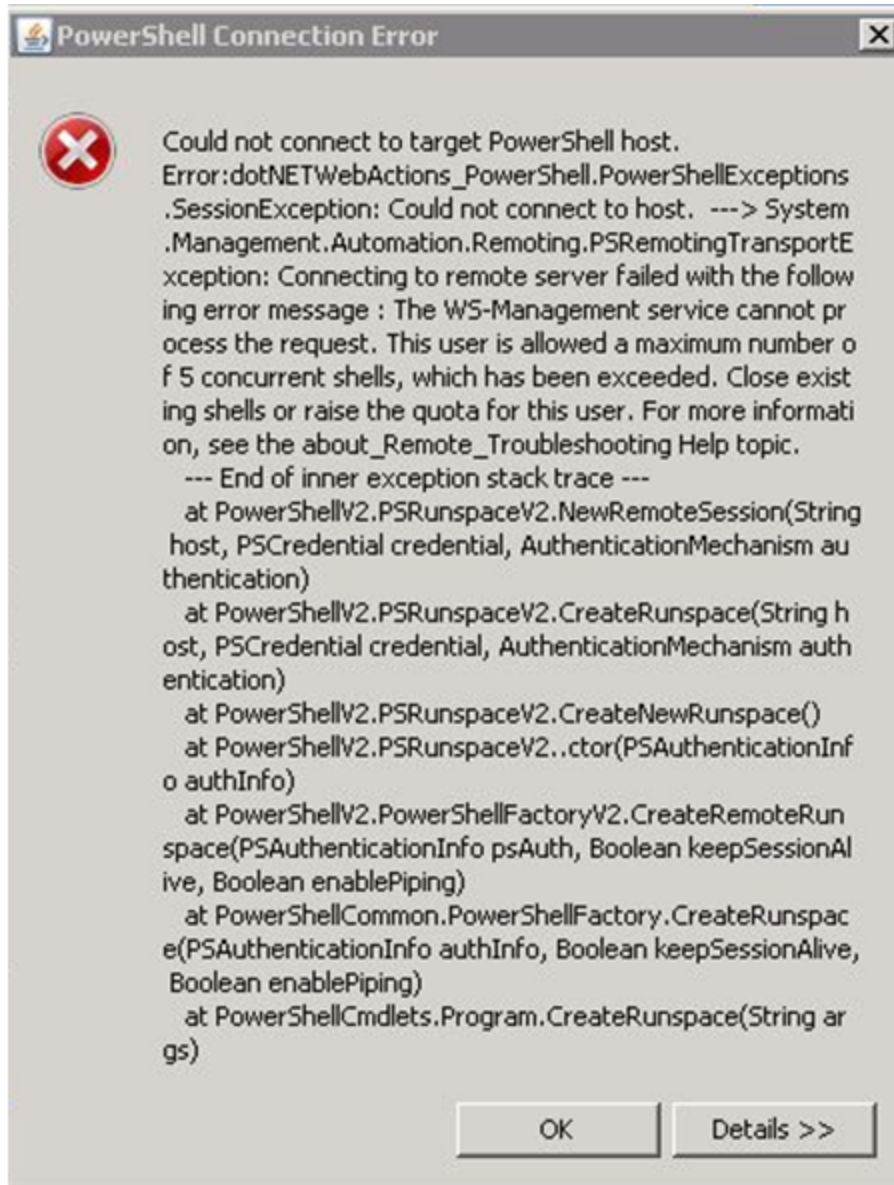


Figure 38: Connection Error - number of remote shells has been exceeded.

