

HP Operations Orchestration Software Studio

Software Version: 7.10

Author's Guide

Document Release Date: March 2008

Software Release Date: March 2008



Legal Notices

Warranty

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

The information contained herein is subject to change without notice.

Restricted Rights Legend

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

Copyright Notices

© Copyright 2008 Hewlett-Packard Development Company, L.P.

Trademark Notices

All marks mentioned in this document are the property of their respective owners.

Where to find Help, tutorials, and more

The HP Operations Orchestration Software (HP OO) documentation set is made up of:

- Help for Central
Central Help provides information to the following:
 - Finding and running flows
 - For HP OO administrators, configuring the functioning of HP OO
 - Generating and viewing the information available from the outcomes of flow runsThe Central Help system is also available as a PDF document in the HP OO home directory, in \Central\docs.
- Help for Studio
Studio Help instructs flow authors at varying levels of programming ability.
The Studio Help system is also available as a PDF document in the HP OO home directory, in \Studio\docs directory.
- Animated tutorials for Central and Studio
HP OO tutorials can each be completed in less than half an hour and provide basic instruction on the following:
 - In Central, finding, running, and viewing information from flows
 - In Studio, modifying flowsThe tutorials are available in the
 - Studio Welcome pane
 - HP OO\Studio home directory, in the Tutorials subdirectory
 - The Opsware Network
- Self-documentation for HP OO operations, flows, and Accelerator Packs
Self-documentation is available in the descriptions of the operations and steps that are included in the flows.

Updating documentation

Documentation enhancements are a continual project at Hewlett-Packard Software. You can update the documentation set at any time using the following procedure (which is also available in the HP OO readme file).

To obtain HP OO documentation

1. On the web site <https://support1.opsware.com/support/index.php>, log in with account name and password that you received when you purchased HP OO.
2. On the **Support** tab, click the **Product Docs** subtab.
3. Under **Quick Jump**, click **Operations Orchestration** (or **Process Automation System**).
4. Under **Operations Orchestration**, click **ZIP** beside **HP OO 7.10 Full Documentation Set**.
5. Extract the files in the .zip file to the appropriate locations on your system:
 - For the tutorials to run, you must store the .swf file and the .html file in the same directory.
 - To obtain the repository that reflects the state of the flow at the start of the tutorial, unzip the file Exportof<preceding_tutorial_name>.zip.

- To obtain the scriptlet for the tutorial that includes using scriptlets, click the scriptlet .txt file name.
- To update your Central or Studio Help:
 - a. Under **Help Files**, click **Studio Help File Bundle** or **Central Help File Bundle**.
 - b. In the **File Download** box appears, click either **Open** or **Save**.
 - c. Extract the files to the Hewlett-Packard Software\HP OO home directory, in either the **\Central\docs\help\Central** or **\Studio\docs\help\Studio** subdirectory, overwriting the existing files.

Support

For support information, including patches, troubleshooting aids, support contract management, product manuals and more, visit one of the two following sites:

- <https://support1.opsware.com/support/index.php>
- http://www.hp.com/go/hpsoftware/DCA_support

This Help system and Guide

Help for Studio (reproduced in StudioAuthorsGuide.pdf) provides an introduction to Studio and detailed procedures that you will use to create flows.

The Help breaks out into three sections, as follows:

- Quick View
- Basic flow authoring

This section introduces you to Studio and covers the basic tasks involved in creating flows, including:

- Importing flows and Accelerator Packs
- Finding and using flows and operations
- Creating flows and flow steps
- Testing flows
- Making flows available to Central users

A very effective way to learn basic flow authoring tasks is to complete the Studio tutorial on modifying a flow before consulting this Help system.

- Advanced flow authoring

This section goes into more depth in creating operations and steps, including defining inputs, moving information throughout a flow, and creating rules that logically determine the course of a flow based on each step. This section also includes an introduction that explores the architecture of and data flow in an operation and discusses the uses of the various kinds of operations

- Creating IActions for flow operations

This section introduces IAction programming and its uses for extending flow functionality to integration with other systems and remote execution of flows.

Introduction to flows

A flow is a set of linked actions that automate health checks, troubleshooting or any other repetitive IT support tasks. Say you want to verify that a page on your Web site contains the correct, current data, such as a certain piece of text. If the desired data is not on the Web page, you want to push new content to the site.

Without Operations Orchestration Software, your options might be:

- Assign someone to look at the Web site every 15 minutes and, if necessary, manually publish content to the site.
- Program a monitoring system to check the site and raise events or alerts if the content isn't correct, with manual content publishing. One of your technical support personnel would have to see what is going wrong.

Or, with Studio, you could author a flow to do all those tasks automatically: check the Web site periodically, create an event or alert, publish content to the site, troubleshoot and repair the underlying problem, and document the steps taken.

Let's use this example to learn about the basic concepts of a flow. A manual runbook or procedure for a person to do this site check might read something like this:

Step 1. Browse to `mysite.com/mypage.htm`. If it does not contain the text "needed text" then do Step 2.

Step 2. Copy `mypage.htm` from `server development1` to `server production1`.

Step 3. Keep track of how often the Web page has the correct text and how often you need to fix the problem

Studio is where you create the flows that automate the triage, diagnosis and resolution of problems, perform system and application health checks and handle repetitive maintenance procedures in your operations or data center. Just like the manual procedure, a flow has steps that gather data or take actions. Each step accepts data and responds to it, the step's response differing according to what it detects. Steps can also provide complete tracking of their actions by recording the data used in each step and what took place.

Key parts of a flow

Steps are the basic unit of a flow.

In addition to steps, flows have several other key elements:

- **Operations** do the actual work of the flow. Step 1 in our example above uses an operation that checks a Web page to see whether it contains specific text. Step 2 uses an operation to copy a file. Steps are created from operations, which are the templates for steps. Thus, operations are the building blocks of any flow.
- **Inputs** give the operation the data that they need to act upon. Our operation to check a Web page needs to know which page to check (`mysite.com/mypage.htm`) and what text to look for ("needed text"). Our copy file operation needs a source location and a destination. Inputs can be entered by the person running the flow, be set to a specific value, or obtained from information gathered by another step.
- **Responses** are the possible outcomes of the operation. Our get Web page operation has three responses: "page not found", "text not found", and "success" (if we got the page and it has the desired text). Our copy file operation might have just "success" and "failure".
- **Transitions:** Our read Web page operation may need to respond differently if the Web page can't be found, if the page is there but the text isn't present, or if the page is there and the

desired text is present. A transition leads from an operation response to one of the possible next steps, so that the operation's response determines what the next step.

Let's sum up the picture so far:

- A flow step's operation uses input data to perform a task, from which it obtains results.
- The operation has several possible responses, one of which is chosen depending on what its results were.
- Each response is connected by a transition to one of the possible next steps in the flow.

Thus the choice of response determines which the next step is in a particular run of the flow.

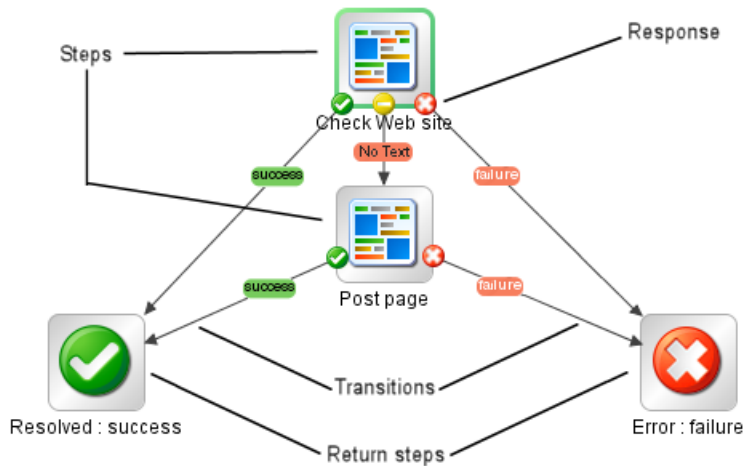


Figure 1 - Parts of a flow

Now let's look at the main parts of an operation.

What a flow needs to be valid

To be valid, a flow must have the following:

- At least one step, with one of the steps designated as the start step.
- Transitions that connect each operation response to a subsequent step.
- A way to for each step in the flow to be reached in one run or another.
- A return step to return a value and end the flow.
- Assignment (definition) of how each input gets its value.

Those are the essential features of a flow. Provide them, and you will have a productive flow. To see how you can increase a flow's value, see the next topic, [Taking a flow beyond the basics](#) and other topics referenced therein.

Taking a flow beyond the basics

There are many dimensions that you can add to a flow's usefulness. The following list includes some of the initial ways you can enhance a flow's capabilities, robustness, and ability to return more information to the user. Some of these subjects, such as inputs and transitions, are listed in the preceding topic, but you get more out of them according to how you use them.

- Enable the flow to adapt to conditions in real time by how you assign data to inputs.
For information on the different ways you can assign data to inputs, see [Inputs: Providing data to operations](#).
- By storing data in flow variables for passing between steps in the flow or from a subflow to its parent flow (the flow in which the subflow is a step).

For creating flow variables that you can use in other steps in a flow, see [Creating a flow variable](#).

For passing flow variables to another flow, see [Passing data from a subflow to a parent flow](#).

- Capture and manipulate information that you can then use elsewhere.
For capturing data as a result and using filters to refine, manipulate, or format the data that you capture, see [Outputs, responses, and step results](#).
- Tell Central users what happened in each step of the flow, presenting them with information that the flow captured.
For telling Central users what happened in each step, see [Adding a transition](#).
- Record key information for charting in Central Dashboards that the Central users define.
For recording information for Dashboard reporting, see

Operations: Models for steps

In addition to its responses, an operation is made up of the following:

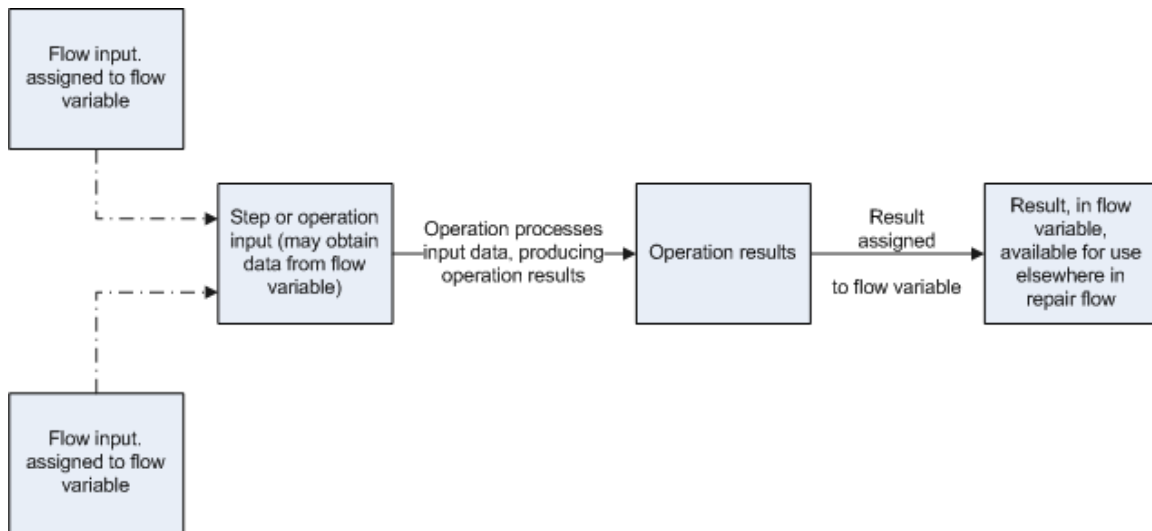
- **Command:** This dictates most of what the operation actually does. (The operation may also contain a scriptlet, which processes data.) The command may be any of several types of commands, including command-line, an http operation, or a script to run. For example, an operation might get a directory listing, check to see if a service is running, execute a Web service, or run a Unix vmstat command.
- **Results:** When a command runs, it returns data, called results. For example, the results returned by a dir command include a file list. A ps command results are a list of processes. Most commands have more than one result, returning data such as a return code, standard output (stdout), and error output (stderr). Our get http page operation needs results for the http return code (200, 302, 404, etc) and the data on the page.
Some commands can return a lot of data that we may want to use later on in our flow. Using a single command, an operation to get memory statistics on Linux can tell us how much memory is free, how much total memory is available, how much swap memory is being used and much more.
- **Flow Variables:** You can store the results data in *flow variables* that you create. You can use this data as inputs to other steps later in the flow.
- **Filters:** Figuring out what response to take based on the data that a command returns may require filtering a key piece of data out of a result or creating a scriptlet that manipulates the data.
- **Rules:** Rules evaluate the operation's results to determine which operation response to take when the step runs. Rules can evaluate any of the results fields, the data strings, return codes, or error codes.

A given response for an operation is selected when the conditions described in the response's rule match the data that you have specified in or extracted from the one of the results fields. The rules that you create are comparisons or matches between a string of text that you describe in the rule and the results field that you select for the rule.

Consider the get Web page operation in our example:

- The "page not found" response would be selected if the http return code were 404.
- The "text not found" response would be selected if text to check were not contained in the data on the page.
- **Scriptlets:** Scriptlets (written in JavaScript or Perl) are optional parts of an operation that you can use to manipulate data from either the operation's inputs or results for use in other parts of the operation or flow.

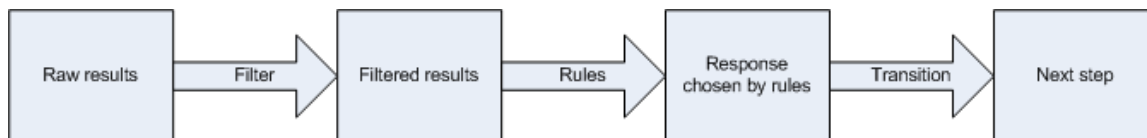
The following diagram shows how operations can get values from flow, step, and their own operation inputs, and how data in operation results can be passed to flow variables, thus becoming available to other parts of the flow.



Apply this diagram to a flow that runs the Windows command-line dir command on a directory:

- Flow inputs could provide two needed pieces of data: the host computer and the name of the directory to run the dir command against
- These two input data items could be stored in flow variables named "host" and "directory," respectively.
- The operation inputs could obtain the values from the flow variables.
- After the operation performs its task based on the inputs, the step could assign the operation results to another flow variable, which another operation could use in another step in the flow.

When you create a step from an operation, each of the operation's responses must be the starting point for a transition to another step. Thus during a particular run of the flow, the rules that evaluate the operation's results determine the response of the operation, which determines the transition that is followed and therefore the path that the run follows through the steps.



Note: You can also set the operation's result to supply values in fields to the result of a step created from that operation, then in turn set that step result to supply those values in fields to the flow's result. You can use the flow result to pass the values to other flows.



Advanced flow architecture and concepts

The Quick View introduced you to basic flow concepts and information on how to make simple modifications of existing flows. This chapter explores the following concepts in greater depth:

- The order of actions performed in the execution of a step
- Operation architecture and information flow
- Step and operation requirements such as defining data sources for inputs and defining responses
- Flows, steps, and operations as HP OO objects
- Creating scriptlets that make possible the additional manipulation of an operation's output data.

Scriptlets are JavaScript or Perl scripts contained within an operation. Scriptlets are good for programming quickly, with relatively little customization of the operation.

In addition to writing scripts in JavaScript or Sleep, this level of authoring requires that you understand:

- Input and flow variables
- Distinctions between steps and operations
- Distinctions between results, raw results, and responses
- Data filters

Step execution order

When a step is carried out, the following actions are carried out, in this sequence:

1. Input values are obtained from the collection of flow variables and global data values and applied to the inputs, making them available to the step's operation.
2. Any changes to the input values are applied to the collection of flow variables and global data values.
3. The step's operation is carried out.
For details on how information is obtained by, flows through, and can be modified within an operation, see the following section, *Operations: architecture and information flow*.
4. If the operation has a scriptlet, the operation's scriptlet is executed.

The operation's scriptlet can do the following:

- a. Select the operation response.
- b. Set the operation's primary output.
The primary output is the result that supplies a value to an input whose assignment is **Previous Step's Result**.
- c. Make changes to local and global flow variables and data values.

- d. Read the operation response value if there's a value present to be read, such as if the operation contains an IAction that uses a RAS. That IAction has a response, which the operation scriptlet can get and read.
5. If the operation's scriptlet has not already set the operation's primary result, the operation's primary result is set now.
6. If the operation's scriptlet has not already selected the operation's response, the response is selected now, using the operation's evaluation rules for selecting a response.
7. If the step has a scriptlet, that step scriptlet is executed.
The step scriptlet can do the following:
 - a. Select the operation response.
 - b. Make changes to local and global flow variables.**Note:** The step scriptlet can not set the primary result.
8. The transition that is associated with selected response is selected.
9. The next step is executed, using this same sequence.

Operations: architecture and information flow

An operation is a defined sequence of actions associated with a step in a flow. When we consider a step that has a flow associated with it as a subflow, we say that the subflow is a type of operation.

Context is a key concept for controlling how data moves in and out of operations. The *context* is a container that holds various values that can be exchanged with a step at various points (see the following diagram). There are two kinds of context: *local* and *global*. Local context exists for the duration of the step; global context exists for the duration of the flow. You can pass values to and from the local or global context.

Now let's look part by part at how a single operation (that is, an operation that is not a subflow) works:

- Core functionality (called the *core*), which encapsulates the business logic of the operation
For Web operations, the core functionality is the IAction interface execute method and the method's parameters, which provide data to and influence the behavior of the execute method.
All input values are copied to the local or global context before execution of the operation. Input values can also be bound to the local or global context.
The core might map input onto raw results.
- Further processing of raw results by a scriptlet (optional)
- Determination of a response

In information flow through an operation, as shown in the following diagram, these parts of an operation play roles:

- Raw results
If the IAction interface is part of the core functionality, the raw results are the data and state.
- Scriptlet
An optional interpretive program that may be executed at the end of a step. The scriptlet often evaluates the raw results of the operation and produces the output data of the operation.
- Output data
The data produced by the operation, if any.
- Response

The evaluation of the operation's output and the resulting determination of the transition from among the possible transitions for the operation's step.

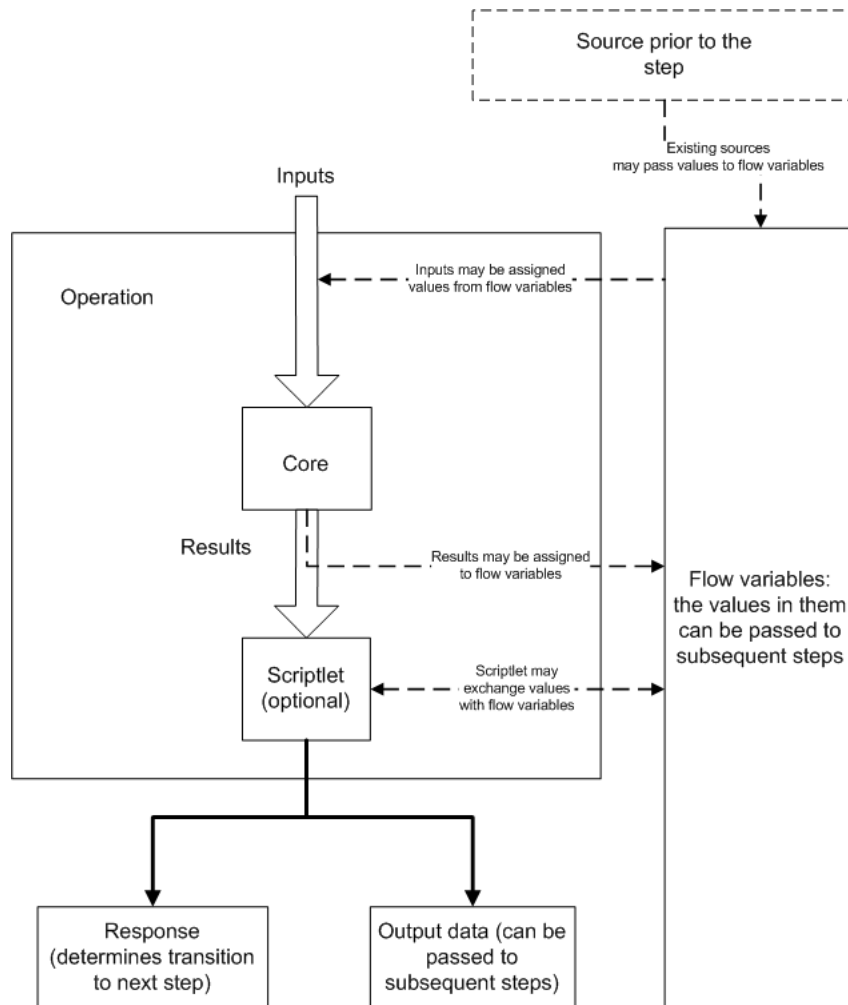


Figure 2 - Information flow through an operation

Note: Operations and flows in **Integrations, Operations, and Utility Operations** folders are sealed—that is, you cannot modify them other than to supply fixed, specific values for inputs.

Advanced flow, step, and operation concepts

The following concepts are important to successfully authoring flows.

Local vs. global variables, in operation, step, and flow inputs

You can use local and global variables to move data among steps within a flow.

- Global variables are available to all the components of a flow within its run, including any subflows and parent flows. When you supply a value to a global variable in a subflow, the value is available to any parent flows in that run.
- Local variables are available only for the flow within which they are defined.

Steps vs. operations

The basic difference between steps and operations is that steps are instances of operations.

- Operations and flows tell HP OO how to do something. This is why a flow is a kind of operation (remember this particularly when you use the flow to create a step in another flow).
- A step is always an instance of an operation or flow. Thus, an operation is a template from which you create a step when you drag the operation onto a flow's authoring canvas.

There are significant differences between the operation and the resulting step:

- You modify the step (for instance, define inputs for the step) on the step's Inspector, not by accessing the operation from the step.

If you modify the operation, you are modifying the template that is the basis for all the steps associated with that operation. This means that you are changing all the steps associated with that operation, regardless of which flows the steps are part of. Thus, unless you are careful, making changes to the operation can break flows that use that operation.

- There are also distinctions between scriptlets that are created on the step and on the operation. For instance:
 - Operation scriptlets cannot read the value of the response of the operation.
 - The operation scriptlet does not appear in the Scriptlet tab of the step created from the operation.
 - A step's scriptlet result cannot be passed to the step's result. (You can get around this limitation by performing the task you want in a scriptlet filter for the step's result rather than in the step's scriptlet.)

Flow, step, and operation inputs

Each input is mapped to a variable, whose value can come from a variety of sources.

Which element you add an input to can determine when the input's value is obtained:

- An input for a flow obtains a value before the first step runs.
It is best practice to set any inputs that the flow needs and that are not produced by processing within the flow in the flow's properties, thus making these input values available to the flow before it begins to run.
- An input for a step obtains a value before the step's operation runs.

Inputs also reflect the distinctions between elements:

- An input that you create for a step (on the step's Inspector) is not an input for the operation associated with the step. Its value is obtained before the operation runs.
- An input that you create for the instance of an operation that is associated with the step is specific to that instance. Changing the input for the instance does not change the input for the operation in the Library. That is, if you right-click a step and choose **Open Operation**, then modify the input, only this instance of the operation is affected, not the operation in the Library.
- When you change an input for the operation in the library (on the Properties sheet that you open by right-clicking the operation in the Library), all instances of the operation that you subsequently create reflect the change that you made.

For information on defining the data source for an input, see [Inputs: Providing data to operations](#).

Outputs and results

Operations (including flows) generate that are the sources for operation and flow *outputs* that the author creates. From the outputs, authors define step *results*.

There are two kinds of outputs:

- The **primary output**, which is the output used to populate a step's result when the author specifies that the step get its result from the previous step's result.
The operation's author specifies the primary output by selecting a source field from the **Extract Primary Output from Field** drop-down list (You can create a filter for it, if you wish.)

There are three kinds of step results:

- **Raw result**, which the step obtains from the collection of key value pairs representing the raw data that was returned from an operation executed in the context of a flow.
For example, if you execute the ping operation on a windows XP machine, you will get the following results:

```
{
  Code = "0"
  Error String = ""
  Output String =
  "Pinging apple.com [17.254.3.183] with 32 bytes of data:

  Reply from 17.254.3.183: bytes=32 time=24ms TTL=244
  Reply from 17.254.3.183: bytes=32 time=24ms TTL=244
  Reply from 17.254.3.183: bytes=32 time=25ms TTL=244
  Reply from 17.254.3.183: bytes=32 time=26ms TTL=244

  Ping statistics for 17.254.3.183:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
  Approximate round trip times in milli-seconds:
    Minimum = 24ms, Maximum = 26ms, Average = 24ms"
}
```

You can filter the values of the raw results to create a more refined result. For example, if in the above data you are interested only in the average latency of the ping operation, you can select the Output String field of your operation and specify a series of filters which extract the '24ms' token at the end of the string. For more information, see [Filtering outputs and results](#).

- **Other results**, which are the results that the flow author has created on the **Results** tab of the step's editor.
- **Other results**, which the flow author creates and to which he or she assigns data from the operation.

Responses are the outcomes that are connected, each response by one transition, to a following step. Because you link a response to a subsequent step, the response is the determination of what the flow does next. Return steps are an exception: Return-step responses return an outcome for the entire flow.

The following table shows which which of the above pertain to operations, steps, and flows, respectively. Note, however, that you cannot change the handling of outputs for sealed operations.

	Raw Outputs	Outputs	Results	Responses
Operations	✓	✓		✓
Steps			✓	
Flows	✓	✓		

Starting HP OO Studio

To start Studio

- In the HP OO home directory (by default, C:\Program Files\Hewlett-Packard\Operations Orchestration), in the Studio subdirectory, click or double-click Studio.exe.

OR

Click the Windows **Start** button, point to **All Programs, Hewlett-Packard**, and then click **Operations Orchestration**.

OR

If there is a Studio shortcut on your desktop or in the programs above the **Start** button, click the shortcut.

Visual overview of Studio

The main elements of Studio are:

- The *Library pane*, which shows the repository you're working in.
- The *Authoring pane*, where you do the substance of your work on the flow diagram; the Properties sheets for flows, steps, operations, and transitions; and editors for a number of system objects and flow objects such as selection lists, domain terms, filters, and scriptlets.
- The *Bookmarks pane*, where you can store shortcuts to favorite operations and flows.
- The *Icons pane*, which contains several collections of icons you can drag new icons from onto operations or steps

Library pane

Expanding the Library folder and then the flow folders reveals flows and operations, with symbols that characterize them.

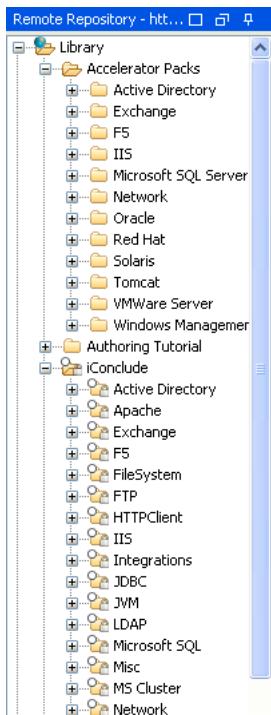


Figure 3 - Library, in Library pane

In the list of folders and flows, subflows (such as **Get User Account (Auto-logon)**) are indented from their parent flows (such as **Fix Auto-logon Account**).

As you explore the Library, the following table explains some of the icons you'll encounter.



This is a flow.



The **white circle** means that this folder and all the flows that it contains are hidden from Central users, thus do not appear in Central (no operations ever appear in Central). You can hide individual flows within a folder.

The **lock** means that the folder and all the flows and operations that it contains are sealed: You cannot modify the flows and operations in this folder. You can, however, duplicate them and then make changes to the copy.



This Warning symbol is superimposed on the symbol for a flow or operation that is incomplete or invalid.

In the **Library**, you can identify operation types by the icons that represent them.

Authoring pane

The authoring pane is the large, right-hand area in the Studio where you work on flow diagrams, adding steps and the connections between them to a flow's diagram and setting properties that determine how flows and their parts work.

The authoring pane toolbar buttons provide shortcuts for a number of tasks. The following illustration labels the buttons that are not standard application buttons (such as cut, copy, paste, undo and redo). See the list below the diagram for more information on some of the labeled buttons.

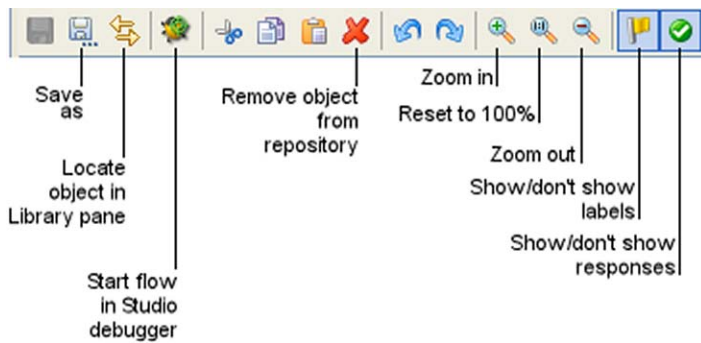


Figure 4 - Left half of flow authoring canvas toolbar

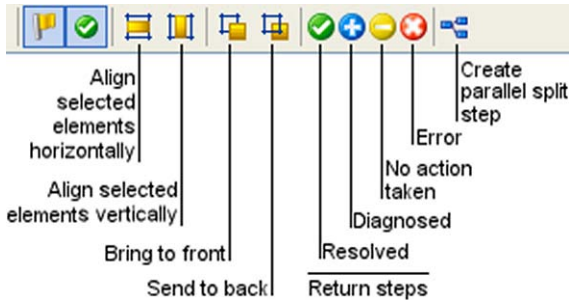


Figure 5 - Right half of flow authoring canvas toolbar

Following are some of the less familiar icons:

- **Locate in Library pane** expands the Library to show the location of the flow or operation that you're working on.
- **Run flow in Studio Debugger** opens the Studio Debugger and starts a run of the current flow in it.
- **Toggle labels** shows or hides labels of objects such as responses. A flow appears in Central with labels showing or hidden according to whether this button had toggled them on or off when you last saved the flow.
- **Show responses** toggles to represent each of a response's inputs with an icon like those of the flow return steps.
- **Align...horizontally** and **Align...vertically** respectively align flow elements that you select.
- If you have overlapping steps as you work on the canvas, the **Send to back** and **Bring to front** icons move the step that has the focus to the back or front of the stack, respectively.
- **Resolved**, **Diagnosed**, **No Action Taken**, and **Error** add those flow return steps respectively to the flow.
- **Parallel split** adds a parallel split step to the flow.

The flow diagram is also the field on which you can directly do much of your work on the flow.

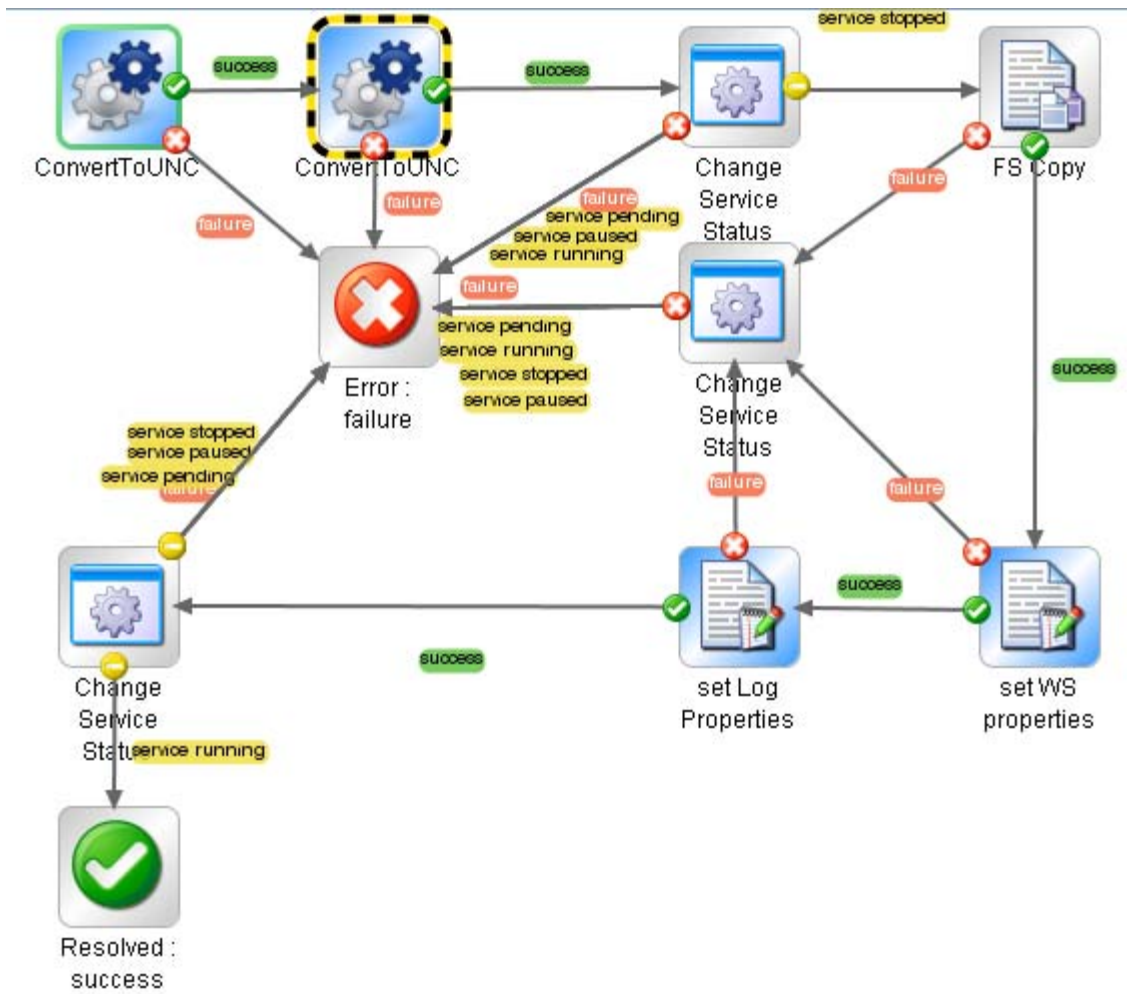


Figure 6 - Flow diagram

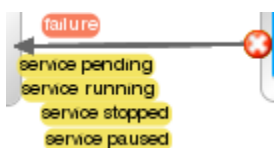
In addition to the diagram parts that you saw illustrated in *Key parts of a flow*, note also the following pieces of the diagram above:



The flow's Start Step is outlined in green.



A step that has been made a breakpoint for debugging purposes is outlined in yellow and black, as above. When you let a flow run automatically in the Debugger, it stops at each breakpoint that you have designated.



The transition above actually represents five transitions. Each transition is labeled with the response that is its source. When multiple transitions start from the same source step and go to the same destination step, they are represented visually by a single line. The names of the responses that are the sources all appear, as above. You can click the response name for a transition to move the transition or to open its Inspector.

Bookmarks pane

The **Bookmarks** pane (expandable from the **Bookmarks** tab in the upper-right of the window) contains operations and flows that you use frequently, making them more readily available.

Icons pane

The **Icons** pane (expandable from the **Icons** tab in the upper-right of the window) contains numerous libraries of operation icons that you can use to make it clear more quickly what a step's operation does. These icons replace the default operation icon on a step that you drag the icon to.

Keyboard shortcuts in Studio

The following sections list some of the keyboard shortcuts that are available in Studio:

Library pane keyboard shortcuts

To do this	Keyboard shortcut
Expand everything below selected	SHIFT + SPACE
Open editor on selected	ENTER
Delete selected	DEL

Authoring pane keyboard shortcuts

To do this	Keyboard shortcut
Delete selected step	DEL
Cut	CTRL + x
Copy	CTRL + c
Paste	CTRL + v
Undo	CTRL + z
Redo	CTRL + y
Insert callout	CTRL + t
Zoom in	ALT + F6

Zoom out	ALT + SHIFT + F6
----------	------------------

Properties Editors / Inspector keyboard shortcuts

To do this	Keyboard shortcut
Edit selected row	CTRL + <right-arrow>

Scriptlet panel keyboard shortcuts

To do this	Keyboard shortcut
Find	CTRL + f

Icons pane keyboard shortcuts

To do this	Keyboard shortcut
Remove selected bookmarks	DEL

Debugger keyboard shortcuts

To do this	Keyboard shortcut
Play	F11
Pause	ALT + p
Stop	ALT + c
Step In	F5
Step Over	F6
Step Out	F7
Reset	F12

Creating or changing a flow: overview

Let's suppose that you want to check a network connection between your computer and a server. The following topics will guide you through creating and modifying a flow that pings the server and runs a traceroute command on the network between your computer and the server.

We will change the flow inputs from user prompts to specific values, in order to make it possible for the flow to run fully automatically. Our work will follow approximately these general steps:

1. Create a folder for holding the flow and its operations.

2. Get started with one of the following means:
 - Finding the flow that you want to use and making a copy of the flow (see [Finding a flow or operation](#)).
To get the flow that you want to work on, you might need to import a repository. In our example you don't, but for information on importing a repository, see [Importing a repository](#).
 - Starting a new flow from a template (see [Creating a new flow](#)).
3. If necessary, change how values are assigned to inputs.
4. Record reporting data (identifiers for aspects of the flow) for the inputs.
5. Look at the results and filters of each operation to make sure you're getting the data you need out of the operations.
6. If you need more flow variables, create them.
7. Review the flow, step, operation, and transition descriptions for usefulness to Central users.
8. Test the flow.
9. To make the flow available to Central users, you publish your repository.

Creating a folder

Whether you create a flow or copy one for modifying, you may want to create a new folder for it.

To create a folder

1. Right-click the **Library** folder in the navigation pane and click **New Folder**.
2. In the dialog box that appears, type the name of the new folder in the text box, and then click **OK**.

Notes:

- In Studio, you cannot give two folders the same name.
- Naming in Studio is not case-sensitive.
- Names can be a maximum of 128 characters long.

Creating a new flow

The templates provided with Studio provide steps for flows that perform certain frequently used tasks. For example, there is a template (**Restart Service**) for creating a flow to restart a service, so you could start your flow that way.

And there is a template for a flow that performs our example task: pinging a server and checking the network between your computer and a server (**Network Check**).

Or, if you want to start with only the Success and Error return steps (return steps are possible last steps of a flow), the last template (**Blank Flow**) provides you with them.

To create a flow from a template

1. On the Studio Welcome page, click the **New Flow** icon.
In the list of templates, when you highlight a flow template, its description appears.

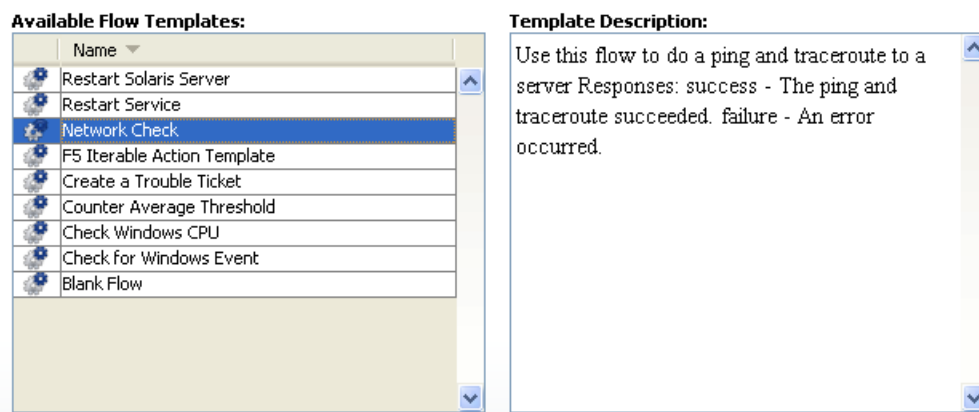


Figure 7 - Templates for commonly used flows

2. Select the flow template that meets your needs, and then click **Create**.
The flow opens in the authoring pane.

Note: Note To create a blank flow, without by right-clicking a folder, pointing to **New**, then selecting **Flow**.

Creating flows

The main steps of creating a flow, once you have created or imported the operations that you will use in it, are:

- Creating the flow, as described in this section.
- Adding operations as steps.
- Creating connections (known as “transitions”) between steps so each response for a step takes the flow to another step.
- Creating one or more return steps to end the flow and assigning each return step a flow response.
- Saving the flow.
- Debugging the flow.

For information on debugging a flow in Studio, see [Debugging a flow in Studio](#).

Tip: To help with understanding the steps, open the Restart Service – Tutorial Flow.

To open the Restart Service - Tutorial flow’s Properties sheet and authoring canvas


1. Find the Restart Service – Tutorial Flow in the Library.
2. To open the Restart Service – Tutorial Flow **Properties** sheet, right-click the flow name and then click **Properties**.
3. To open the Restart Service – Tutorial Flow canvas in the authoring pane, double-click the flow name.

To create a flow

1. Highlight the folder in which you want to create the flow.
2. From the **File** menu, point to **New** and then click **Flow**.
3. Name the flow, using standard characters.
 - In Studio, you cannot give two flows the same name.
 - Naming in Studio is not case-sensitive.
 - Names can be a maximum of 128 characters long.

A new flow diagram appears in the authoring pane.

You can also create a flow from one of several templates that are designed specifically for completely some frequently undertaken tasks. These templates give you a head start by providing the steps needed to complete those common tasks. For information on creating a flow from a template, see [Creating a new flow](#).

In the Library, the new flow is marked with the Warning symbol () and its name appears in red as long as it is incomplete. Moving the cursor over the name of an incomplete flow displays a tool tip that specifies how the flow is incomplete.

Finding a flow or operation

Flows and operations are stored in subfolders of the **Library** folder of the **Repository** pane in Studio. The main folders in the **Library** are:

- **Accelerator Packs**


The most commonly used flows, organized into subfolders by technology.

- **My Ops Flows**

Where you might want to store flows that you create. A new flow that you create from a template is automatically stored here. (For information on creating a flow from a template, see [Creating a new flow](#).)

- **Operations**

This folder contains the essential pre-configured operations and subflows that are the building blocks of the Accelerator Packs and many of the flows you will create. Many of these folders also contain a folder called "Samples" which show you examples of using many of these operations.

Note that the **Accelerator Packs**, **Integrations**, **Operations**, and **Utility Operations** folders are sealed, as indicated by the lock on the folder icon (). It contains sealed (read-only) flows and operations. Sealed flows and operations are so widely useful and fundamentally important that HP OO does not allow them to be modified. You can make copies of sealed flows and operations and modify the copies.

Tip: To provide yourself with quick access to flows and operations that you commonly use, drag them to the **Bookmarks** pane.

For finding a flow or operation, the **Search** tab is a very useful alternative to browsing the Library.

Note: If the flow you need is in a repository that is not part of either your local Studio repository or the remote Central repository, you can get the flow into Studio by one of the following, depending on where the flow and its dependent objects (operations, system accounts, and other system objects that the flow uses) reside:

- Updating the local Studio repository from the Central repository
- Importing the remote repository into your local Studio repository

Searching for a flow or operation

The search is a full-text search throughout the Library that uses the Apache Lucene search syntax. For information on constructing a search with the Apache Lucene syntax, see the Apache Software Foundation Web site.

Clicking the **Search** tab on the bottom left of the Studio window (or pressing F3 on your keyboard) opens the **Search** pane, on which you can find flows and operations by searching on the name or other field properties. You'll find examples farther down in this topic.



Figure 8 - Studio tabs

To keep the **Search** pane open, click the pin icon (📌) in the upper right hand corner of the pane.

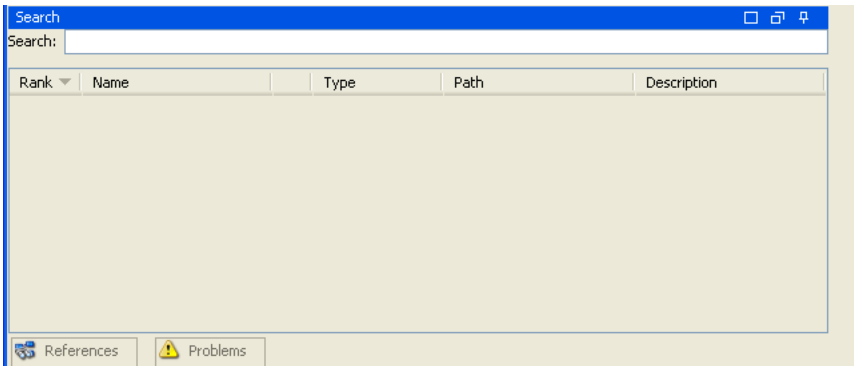


Figure 9 - Search box

For most of your searches you can simply type the search term you are interested in and press ENTER on the keyboard. For instance, to search for the Restart Service flow, in the **Search** text box, you could type **Restart Service** or just **service**. The search results are displayed with the most relevant first. You can sort by any of the columns by clicking on the column header. Note that the **Type** field will help you determine whether the search result is an operation or a flow.

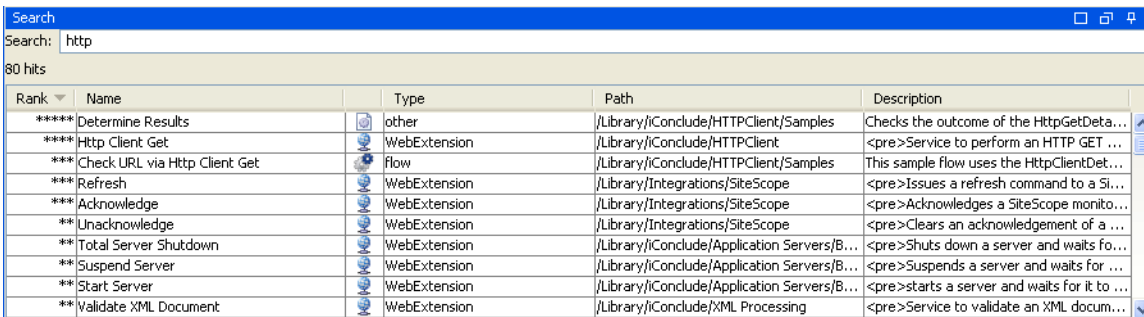


Figure 10 - Search results

Note that you can read the flow or operation's description, to see whether it's the best choice.

To quickly employ an operation or flow from the list of search results

- Open an operation's **Properties** sheet or a flow's diagram by double-clicking in the row of the operation or flow of interest.
- OR
- Drag an operation onto a flow diagram.

To narrow your search down to particular fields

- Use the search syntax:
`<searchable_field_name>:<string to search for>`

Tip: The search uses a Boolean AND. If you type two words, the search returns only operations or flows that contain both words.

You can search for the following field names. Note that this list includes sample search strings.

- Flow or operation name

Examples:

`name:Get Temp Dir`

`name:Clear Temp Dir`

- Operation type

Examples:

`type:cmd`

- Category

Example:

`categories:network`

- Input name

Example:

`inputs:server`

- Flow or operation ID

Example:

`id: 1234-3453-3242-32423`

- String contained in flow or operation descriptions

Example:

`description:clear`

- RAS over which the operation or flow runs

Example:

`ras:RAS_Operator_Path`

Tips:

- When you have found an operation or flow that interests you, you can learn more about how to use it by locating it in the Library, right-clicking it, and then clicking either **What Uses This** or **What Does This Use**. For more information, see [Finding out which flows use an operation](#).
- Before you search for an operation or flow from which a step was created, you can save yourself time by checking out the **Advanced** tab on the step's Inspector. The **Advanced** tab shows which operation or flow a step is associated with and the location of the flow or operation.

Descriptions: Finding the *right* operation

Suppose you want to test connectivity with a server. You do a search on the Search tab using the term "connectivity," and come up with the Connectivity Test and Iterative Connectivity Test flows. Which one is really the right flow? Let's take a look at the description for each of the two flows.

To see a flow's description

- Open the flow in the authoring pane and click **Properties** (at the bottom of the pane), then click the **Description** tab.

Note: Besides describing the flow, the description also tells you about the flow inputs, giving you hints to the kind of values to provide the inputs.

To see an operation's description


- Open the operation in the authoring pane, then click the **Description** tab.

OR

Double-click a step that was created from the operation and in the Inspector that opens, click the **Description** tab.

Note: Besides describing the flow, the description also tells you about the flow inputs, giving you hints to the kind of values to provide the inputs.

Tip: You can see operation and flow descriptions in the results area of the **Search** tab. For information on searching for an operation or flow, see [Searching for a flow or operation](#).

If you want to further explore an operation, highlight the step whose operation you're interested in, open its Inspector (by clicking the  icon), and click the **Description** tab. Because the step was created from its operation, its description is the description of the operation. The description tells you about the operation's (and step's) inputs and the operation's responses and results, explaining each one. It also includes notes and tips on usage, as in the following (because the operation is sealed and so is read-only, the description is gray).

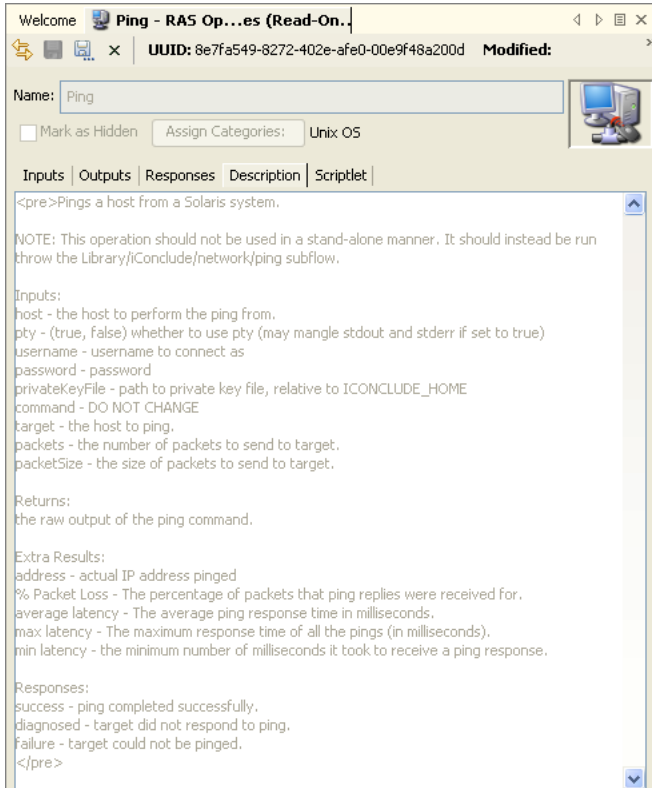


Figure 11 - Operation description

Viewing many operation and flow descriptions

You can consolidate all the information that operations and flows provide in their descriptions and in the definitions of their elements (such as inputs and results) into one place, in a coherent, linked list of HTML files. You can generate this self-documentation for individual operations or flows, or for the contents of folders. The self-documentation is generated in the form of HTML files, which are stored in a folder that you specify.

You can also customize the format of the self-documentation that you generate. For more information, see [Generating documentation in a custom format](#).

Note: You cannot generate documentation on folders that are hidden in Central or on their contents.

To generate documentation of the flows and operations in a folder

1. Right-click the folder whose contents you want to see information about.
2. From the context menu that appears, point to **Generate Documentation**, and then to **Standard Format**.
3. In the **Choose an output directory** dialog, specify where HP OO should put the HTML files.

The folder content for which you have generated documentation appears in your Web browser.

The screenshot shows the HP Operations Orchestration web interface. The top navigation bar includes the HP logo and the text 'Operations Orchestration'. The main heading is 'Documentation Overview'. On the left, there are two sections: 'Folders in Accelerator Packs' with links like '/Active Directory' and '/Active Directory/Deprecated', and 'All Operations' with a list of tasks such as 'DNS Event Check', 'Is Account Locked', and 'Verify SRV Record'. The main content area is a table titled 'Folders in Accelerator Pa' with a 'Folder path' column. The table lists various folder paths, including '/Active Directory', '/Active Directory/Deprecated', and '/Application Servers/BEA WebLogic/Health Check'.

Figure 12 - Generated documentation for the F5 Accelerator Pack

4. To view the operations in a folder within the folder that you selected, click the folder.

This screenshot shows a detailed view of a folder. The left sidebar lists folder paths, with '/Active Directory/Deprecated/Health Check' selected. The main content area shows the heading 'Health Check' and a link for 'Verify SRV Record'.

5. To view the documentation for a single operation or flow, do one of the following:
 - Under **All Operations**, click the operation you're interested in.

- In the list of subfolders (under **Folders in <foldername>**), click the name of the subfolder, then in the lower-left pane, click the operation or flow of interest.
- In the following screen shots, the operation whose documentation we've obtained is a flow.

The screenshot shows a web browser window titled "Documentation for folder: My Ops Flows - Microsoft Internet Explorer provided by Opsware Inc.". The address bar shows the file path: "C:\Documents and Settings\jthomas\Documentation Supplemental\New Folder (2)\index.htm". The browser displays the "Operations Orchestration" interface for the "Ops Flow: good pingtest".

The left sidebar contains a navigation menu with the following items:

- All Operations in My Ops Flows
- Folders in My Ops Flows
 - /testflow
- All Operations
 - bad pingtest
 - dirtest
 - filtered pingtest
 - form single input extractor (Copy 2)
 - getglobal
 - good pingtest
 - ping
 - Random Number Generator
 - testflow

The main content area displays the following information:

- Ops Flow: good pingtest**
- [/testflow](#)
- Description**
- Diagram**

The diagram illustrates a flow starting with a "Ping" operation. It branches into two paths: "Success Send Mail" and "Failure Send Mail". The "Ping" operation has a "group done" output. The "Success Send Mail" operation has a "success" output. The "Failure Send Mail" operation has a "success" output. Both paths lead to a "Resolved: success" state. The diagram also shows "failure" and "host not found" paths from the "Ping" operation.

Inputs

List depicting all inputs which the flow specifies.*

- host : Value: 10.51.0.36,10.51.0.37,10.51.0.38

Raw Results

List depicting all raw result fields which the operation supports.*

Figure 13 - Some of the information available for an operation

The illustration above shows the **Description** and **Diagram** that Generate Documentation displays for an operation that is a flow. You can scroll down for information on the operation's:

- Inputs
- Raw results
- Available outputs
- Responses

- Scriptlet
- If the operation is a subflow, information for each step in the subflow, including:
 - The operation from which the step is created
 - Values assigned to the step's inputs (these are called bindings in the **Generate Documentation** Web page)
 - Transitions leading away from the step
 - Outputs

Following is an example of those parts of an operation's self-documentation.

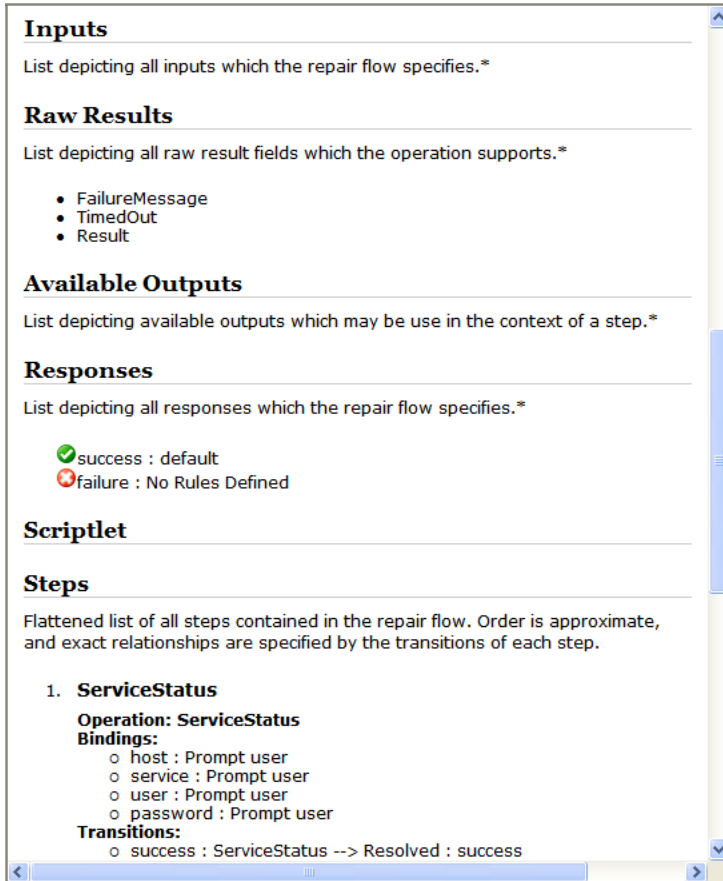


Figure 14 - Detailed information about a flow or operation

Generating documentation in a custom format

Studio ships with .vm templates that control the appearance of the HTML files that present the generated documentation. To customize the presentation of the documentation generated for flows and operations, you can modify the templates.

The templates are HTML files written in the Apache Velocity template language. The following shows the basic structure of the HTML page, which the frame definitions reference.



Figure 15 - The Generate Documentation frameset

In this topic's descriptions of the templates, note the following definitions of the frames:

- overview-frame – upper-left frame

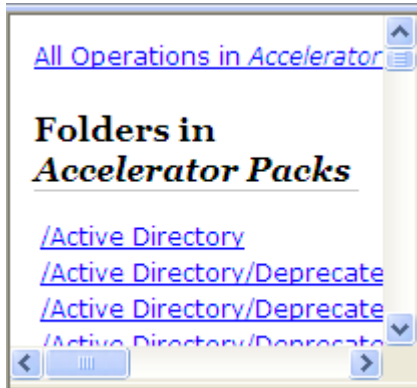


Figure 16 - Overview frame

- folderFrame – lower-left frame

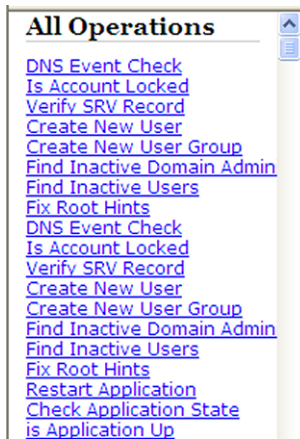


Figure 17 - Folder frame

- headerFrame – upper-right frame

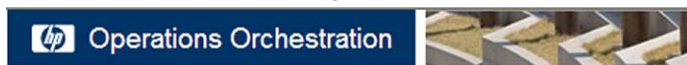


Figure 18 - Header frame

- opFrame – lower-right frame



Figure 19 - Operation frame

Template file descriptions

Folder_template.vm

The root template, which generates a frameset and calls the following to populate it:

- All_folders_template.vm
Generates a list of the subfolders of the folder Pa and places it in the overview frame (upper-left).
- All_ops_template.vm
Generates a list of all operations and places it in folderFrame (lower left).
- Header.html

Places the header in headerFrame (upper right)

- Folder_overview_template.vm
Generates information about one or more operations and places it in opFrame (lower right)

All_folders_template.vm

Generates a table of contents for the folders.

- Header.css
See definition below.
- All_ops_template.vm
Generates a list of all operations and creates a link to display it in folderFrame (lower left).
- Folder_contents.vm
Generates a list of the clicked folder's contents and creates a link to display it in folderFrame (lower left).

All_ops_template.vm

Generates a table of contents for all operations and the documentation for every child operation.

- Header.css
See definition below.
- Op_template.vm
Generates and creates link to display it in opFrame (lower right).

Folder_overview_template.vm

Generates a tabular summary describing the contents of a folder.

- Header.css
See definition below.
- Folder_contents.vm
Generates and creates link to display it in folderFrame (lower left).

Op_template.vm

Generates documentation for a single operation

- Header.css
See definition below.
- Folder_template.vm
Generates and creates link to display it in same frame (up to parent folder).
- Folder_contents.vm
Displays folder contents in folderFrame.

Flow_template.vm

Generates the documentation for a single flow.

- Header.css
See definition below.
- Flow_template.vm
Generates and creates link to display it in same frame (up to parent folder).
- Folder_contents.vm
Generates a list of the folder contents and creates link to display it in folderFrame (lower left).
- Op_template.vm

Generates and creates link to display it in opFrame (lower right).

Folder_contents.vm

Generates a table of contents for a single folder.

- Header.css
See definition below.
- Op_template.vm
Generates and creates link to display it in opFrame (lower right).

Header.html

The Hewlett-Packard banner

Header.css

Stylesheet used for general fonts, colors, etc.

Hp_rockwell.css

Stylesheet for the Hewlett-Packard banner

Hp_steps_307x39.jpg

Graphic for the Hewlett-Packard banner

Logo_hp_smallmasthead.gif

Logo for the Hewlett-Packard banner

Generation Structure

The templates' hierarchy is as follows, with each template calling the ones below it in the hierarchy.

- Folder_template.vm
 - All_folders_template.vm
 - All_ops_template.vm
 - Op_template.vm
 - Folder_template.vm
 - ...
 - Folder_contents.vm
 - Op_template.vm
 - ...
 - Flow_template.vm
 - Flow_template.vm
 - ...
 - Folder_contents.vm
 - ...
 - Op_template.vm
 - ...
 - Folder_contents.vm
 - Op_template.vm
 - ...
 - Flow_template.vm
 - ...
 - All_ops_template.vm
 - Op_template.vm
 - ...
 - Flow_template.vm
 - ...
 - Folder_overview_template.vm
 - Folder_contents.vm

• ...

To modify the .vm templates for generated documentation

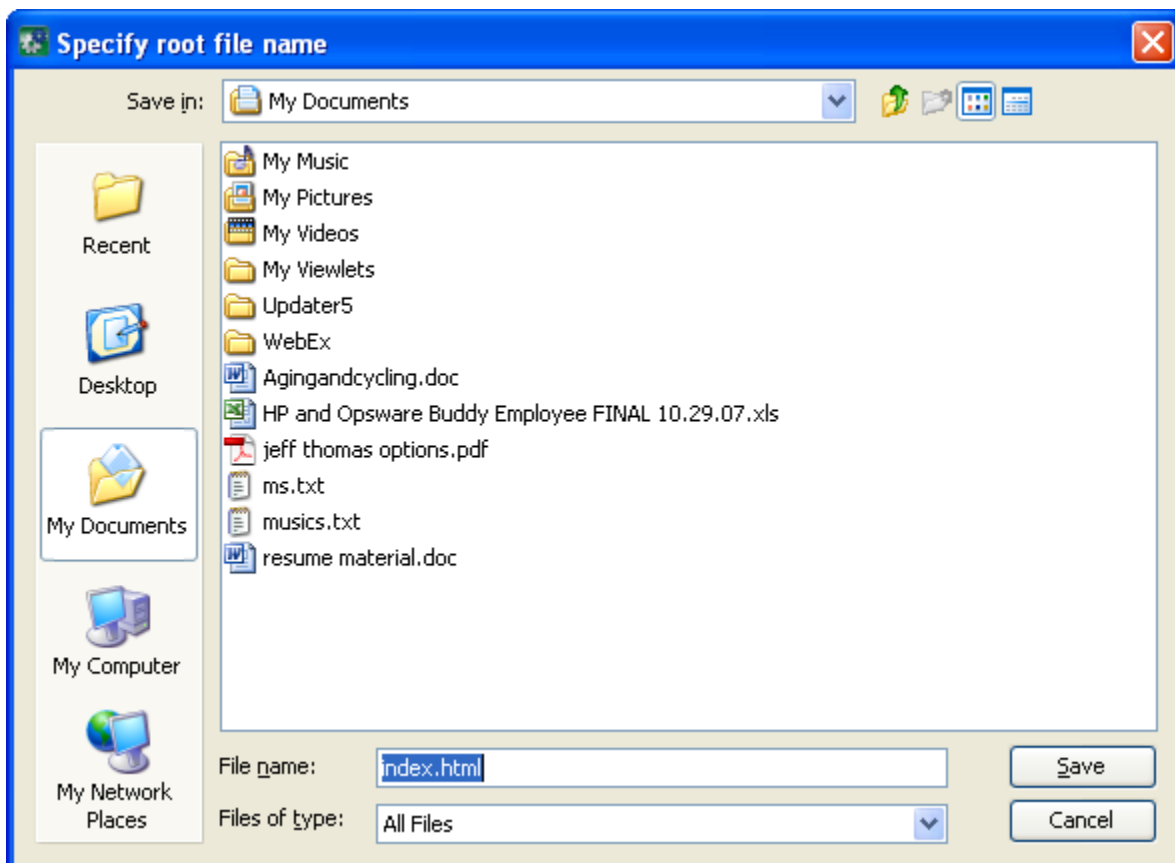
1. Navigate to ..\Hewlett-Packard\Operations Orchestration\Studio\extra\template.
2. Make a backup of any template that you wish to change.
3. Open the template that you want to customize in a text editor, and make your changes.
4. Make another backup, of any template that you have changed.

When you upgrade Studio, the templates will be overwritten. You can use the backups of your changed templates to restore your changes to the upgraded templates.

To generate documentation in a custom format

1. If you wish to customize the .vm templates used to format the generated HTML pages, use the procedure before this one, "To modify the .vm templates for generated documentation."
2. Right-click the folder whose contents you want to see information about.
3. From the context menu that appears, point to **Generate Documentation**, and then to **Custom Format**.

The **Specify root file name** dialog appears.



4. Navigate to the folder in which you want the generated HTML files to be placed.
5. In the **File name** box, type the name of the start-page file, and then click **Save**.
By default, the name of the start file is index.html, which automatically opens in your default Web browser. If you give it a different name and/or extension, you will have to manually choose the program to use for opening the start page.
6. In the **Select Documentation Generation Template** dialog, select the template to use for the generated HTML files, and then click **OK**.

The documentation is generated.

Finding out which flows use an operation

You can learn more about ways to use and implement an operation or flow by looking at how it is used in existing flows.

Important: After you have created your own operations and flows, you should not change them until you have found out which other flows use them.

There are two kinds of references:

- **References to** the operation or flow. These are the flows that have a step created from the operation or flow.
- **References from** the operation or flow. These are the objects, such as selection lists, permissions assigned to groups, system filters, etc., that the operation or flow makes use of. In the case of flows, these are the operations (including subflows) from which the flow's steps were created.

These referenced flows and operations are valuable as samples that you can copy, paste, and modify.

To view an operation's references

1. In the Library, right-click the operation or flow.
2. To view the **references to** the operation or flow, click **What Uses This**.

OR

To view the **references from** the operation or flow, click **What Does This Use**.

Tip: To bring the last obtained references back up after you closed the References panel, click the **References** tab.

Copying flows and operations

When you've found the flow or operation that you want to use, best practice is to make a copy and then work on the copy. Flows and operations in the **Accelerator Packs, Integrations, Operations,** and **Utility Operations** folders of the Library are sealed and cannot be modified, so the only way to create modified versions of them is to create a copy, then make changes to the copy.

Here's why: Suppose that the operation you want to change has an input, InputA. And suppose that the step that you have already made from the operation is used in a flow, FlowAlpha that has a flow input that supplies InputA with a value. Now suppose you add InputB to the operation. The step that you already created from the operation doesn't have any way of obtaining a value for InputB. When a way of getting a value is not defined for an input, the input presents a prompt to the user to get its value. However, if a Central user has created a schedule for automatic runs of FlowAlpha, FlowAlpha will break during those runs, because a requirement of running fully automatically is that a run not require any user inputs.

Note: If the changes you want to make are necessary only for a few uses and can be made in a step that was created from the operation, you may not need to modify the operation, but instead to make the necessary changes in the step. For more information on the differences between steps and operations, see [Advanced flow, step, and operation concepts](#).

For information on modifying steps, see the following topics:

- [Inputs: Providing data to operations](#)
- [Outputs, responses, and step results](#)
- [Changing which operation a step is based on](#)

You can copy, duplicate, and rename flows and operations with the context (right-click) menu.

To copy or duplicate flows or operations

1. In the Library, find and right-click the flow or operation.
2. Point to **Edit**, then point to either **Copy**, **Copy Deep**, or **Duplicate**.
 - If you duplicate the operation, the duplicate is automatically placed in the same folder as its original.
 - If you copy the operation, you can paste it (CTRL+V) in any folder that is not sealed.
 - If you click **Copy Deep**, not only the flow but also all the operations that the flow uses are copied.
3. If you clicked **Copy**, navigate to the location where you want to place the copy and then click CTRL+V.

You can rename the copy.

Now you're ready to work with the flow.

Renaming flows or operations

To rename flows or operations

1. In the Library, find and right-click the flow or operation.
2. Point to **Edit**, then click **Rename**.
3. Type the new name in the text box, then press ENTER and save your work.

Bookmarking flows and operations

Once you've found a flow or operation that you want to use again, you can make it easier to get to by adding it to the **Bookmarks** panel. The flow or operation is still available at its normal location in the Library. Adding a flow or operation to the **Bookmarks** panel makes it available also from the right-click menu in the flow canvas.

To add a flow or operation to the Bookmarks panel

- Drag the flow or operation from the Library or **Search** box to the **Bookmarks** panel.

Creating steps

There are several simple ways to create a step from an operation or flow (remember that when you create a step from a flow the flow is treated as a kind of operation).

When you create a step from an operation, the step is an instance of the operation and so inherits the operation's inputs, results, references, etc. You can change those elements of a step without affecting the operation from which the step was created.

Important! Changing inputs or results on the operation that underlies the step also changes the inputs or results for every step that is an instance of that operation. Making changes to operations can therefore break any step (and its flow) was created before you changed the inputs, etc. in the operation.

Operations should be generic enough to base many particular steps on them. Then you can make changes in the step to respond to particular situations.

Adding steps to a flow

The first place to look for operations or flows that you can use to make a step are the following folders, which contain flows and operations provided by HP OO. You will usually need to modify the contents of these folders in order to use them, but they are sealed, meaning that you can only modify copies of them. Therefore, to use one of these flows and operations, you make a copy of it, then modify and create steps from the copy. Important! Best practice is to follow the same procedure with flows and operations that you create. For information on copying a step, see [Copying steps](#).

- **Accelerator Packs** - Flows designed to solve common IT problems. These flows should work in most datacenters.
- **Integrations** – Operations and samples for using them that integrate HP OO with other enterprise software products, such as Hewlett-Packard Network Node Manager and BMC Remedy. Using these may require a creating a custom flow, because of the level of customization typical of enterprise software products that are used in a given datacenter.
- **Operations** - Flows and operations for interacting with common protocols and software systems, such as LDAP and linux.
- **Utility Operations** - Flows and operations that provide utility functions across nearly the entire spectrum of technologies. These flows and operations provide capabilities such as calculating and filtering date and Time, and performing mathematical manipulations and comparisons.

Note: When you create a step in a flow by dragging a flow from the Library to the current flow canvas, the flow that you drag becomes a subflow of the flow into which you drag it.

The first step that you drag to the flow's canvas automatically becomes the Start Step of the flow, which is signified by a green outline of the step.

Important: The step that you create should not be confused with the operation or subflow that the step is associated with. Steps are particular instances of operations or flows. If you make changes to make the effect of the operation specific to your flow's current needs, you should make these changes on the step.

For instance, to specify a host machine for the operation to run against, you should specify the host in an input to the step rather than to the operation. If you were to specify a particular machine in one of the operation's inputs, the operation would break any flow that the operation was part of and that could not properly run against that machine.

To add a step to a flow

1. From any one of the following, drag an operation or flow onto the authoring canvas:
 - **Library** pane
 - **Search** results
 - **Bookmarks** panelYou probably want to rename the step to reflect its function within the flow (operation names are more generic than their use in a particular step).
2. To rename the step, in the flow canvas, right-click the step, click **Edit Name**, and then type the new name and press ENTER.
3. Configure the step as needed:
 - Adding and defining data sources for inputs
For more information, see [Inputs: Providing data to operations](#).
 - Adding results, defining and filtering their data sources, and storing their values in flow variables, and adding responses that determine the next step in the flow
For more information on these items, see [Outputs, responses, and step results](#).
 - Adding transitions

For more information, see [Transitions: connecting responses to steps](#).

In addition to creating a step by dragging an operation or flow onto a flow's authoring canvas, you can create a step by copying an existing one.

Creating a step from commonly used flows and operations

Studio makes some of the most often-used operations and flows the easiest to create steps from.

These operations and flows are:

- Favorite flows
 - Windows Diagnostic
 - Solaris Health Check
 - URL Test
 - Exchange Server Health
 - Iterative Connectivity Test
 - Outlook Web Access Health
 - Restart Service
 - Get Status of a Host's Virtual Machines
 - MS SQL Server Connectivity
 - Restart Windows Server
- Popular Operations
 - manual
 - SSHCommand
 - Counter
 - SendMail
 - Iterator
 - Ping
 - RunScript
 - Restart Windows Service
 - EnableNode
 - Http Client Detailed Return
 - JRAS Command
 - Release Lock
 - Acquire Lock

To create a step from a Favorite Flow or Popular Operation

1. Open your flow in the flow canvas of Studio.
2. Do one of the following:
 - In a blank area of the flow's authoring canvas:
 - Right-click, point to **Insert**, and then to either **Favorite Flows** or **Popular Operations**.
 - In the menu that appears, click the operation or flow that you want to add to the flow.

OR

- Click the **Bookmarks** tab and then, in the **Bookmarks** pane that appears, drag either a flow from **Favorite Flows** or an operation from **Common Operations** to the flow diagram.
3. Modify the step as needed, and connect it to its preceding and succeeding steps in the flow.

Changing the Start Step

To change which step is the Start Step

- Right-click the step that you want to start the flow and, from the context menu that appears, select **Set As Start Step**.

Copying steps

You can use an existing step as the basis for creating a new step in the same or a different flow, by copying the step, then renaming and modifying the copy.

Doing so, you can save yourself time defining inputs, and creating filters, results, and other flow objects.

Tip: In addition to the following two methods, you can also copy a step by holding down CTRL and dragging the step where you want a copy of it.

To copy a step

1. On the canvas, right-click the step and then click **Copy**.
2. Right-click anywhere on the canvas and click **Paste**.

To copy a step from one flow to another

1. Open the **Design** tab of the flow that has the step you want to copy.
2. Highlight the step, and then copy it using the right-click menu or the standard Windows accelerator keys for Copy (CTRL+c).
3. Open the **Design** tab of the flow where you want to use the step.
4. Paste the step where you want it.
5. Save your work.

Modifying a step

To change a step's inputs, results, or anything else about it other than where its transitions go, you open the step's Inspector.

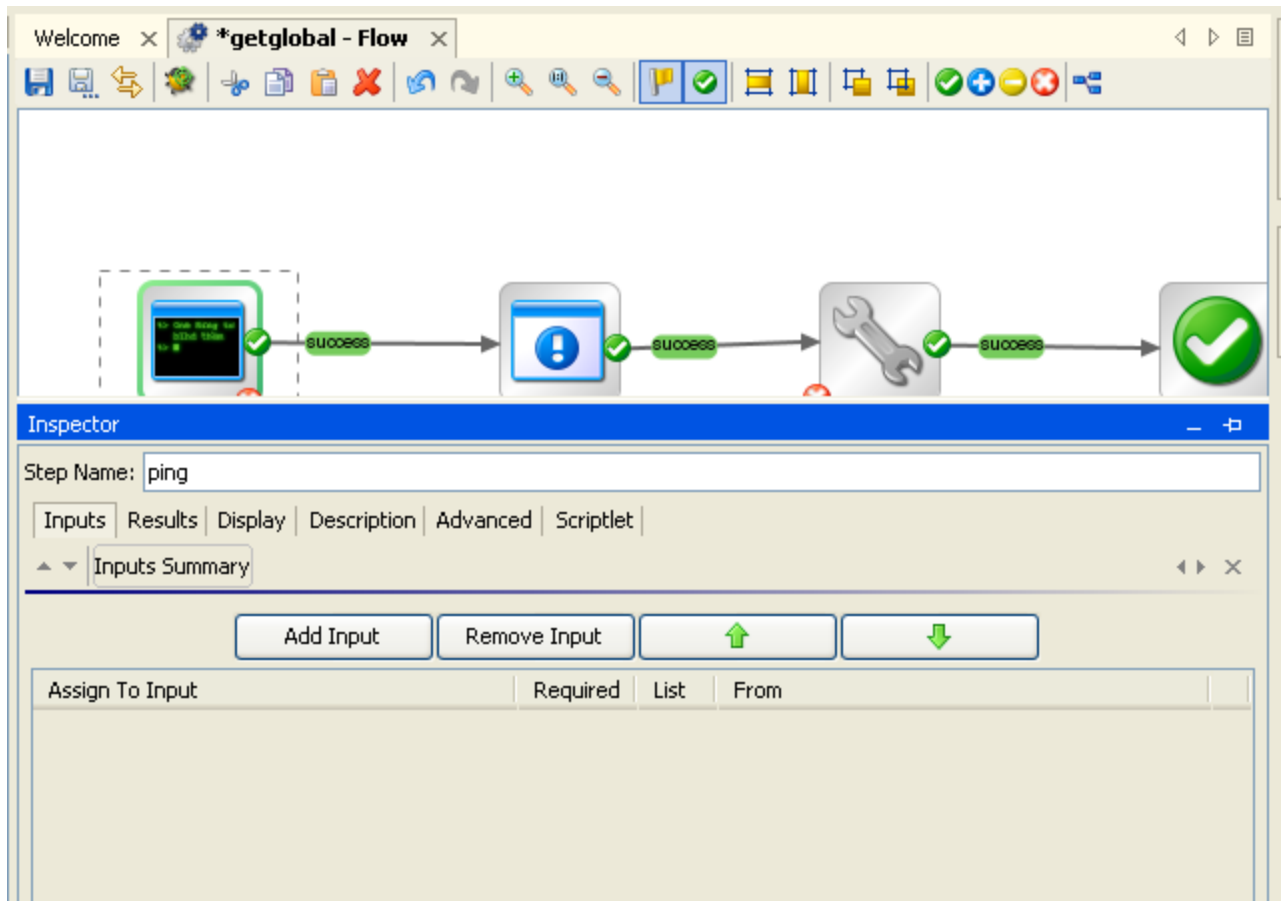
To open the step's Inspector

1. Double-click the step.

OR

Right-click the step and, in the right-click menu, click **Properties**.

The Inspector opens, looking something like the following:



2. To keep the Inspector open so that you can shift its focus from step to step or to a transition without having to close and reopen the Inspector, click the pin icon (📌) at the right end of the Inspector's title bar.

Re-arranging steps in the flow diagram

In addition to dragging steps, you can move steps as a group by using SHIFT + click (then dragging) or CONTROL + click (then dragging).

Prompting the user before running the step

Even if the step does not require user input, you can create a prompt that requires the user's consent before running the step.

Tip: For any step that requires information of the Central user, you can supplement the flow description and help the user by creating a prompt that tells what he or she will need to know for this step.

To create a user prompt for the step

1. Open the step's Inspector, and then click the **Display** tab.
2. To prompt the user, select the **Always prompt user before executing this step** checkbox.
3. Create the prompt in the following boxes:
 - Create a label for the prompt in the **Prompt Title** box.
 - Specify the size of the prompt in pixels in the **Prompt Width** and **Height** boxes.
 - Provide a message to the user in the **Prompt Text** box.

- Click **OK**, and then save your changes.

Transitions: connecting steps

You connect any two steps with one or more *transitions*. A transition starts from one of a step's responses and goes to another step. Every response in a flow must have a transition either to another step or to a return step that ends the flow.

More than one response can be connected to a given step. For example, several failure responses often are connected to a single failure return step.

The lines in the following diagram represent transitions. The names of a response by default becomes the name of the transition that connects it to a succeeding step.



Figure 20 - Transitions in a flow

In addition to simply connecting steps, transitions are valuable for:

- Control who can continue with the flow beyond the transition.
You might want to limit who can continue executing a flow beyond a certain transition for security reasons or because of who has the necessary knowledge to continue using the flow.
You can exercise this control with *gated transitions*, or transitions that require membership in a certain HP OO role for the account that executes the flow.
Gated transitions are colored red in the flow diagram in both Studio and Central.
- Provide a basis for calculating the value of a run of the flow.
When you assign a value to a transition, then if the transition is followed during a run of the flow, its value is added to the value of the flow for that run. The total value of the flow for that run is the sum of the values that have been assigned to the transitions that were followed.
Central users can see this value in the Central dashboard.
- Providing flow users with information on what happened in a step.

Because what takes place in a step determines which transition is followed, the transition's descriptions appears in the **Results Summary** area of Central as the **Message** describing what happened in each step.

The following screen shot from Central shows four transition descriptions, as they appear in the **Results Summary** area. Note that they describe what happened in the step from which they originated.

▼ Results Summary		
Step	Response	Message
Get Stopped Services	✓	Retrieved a list of services which are currently stopped.
Select a Service	✓	Selected Adobe LM Service to restart.
Restart Service	✓	Restarted service Adobe LM Service
Resolved : success	✓	Return step - Restart Service - Tutorial Flow

Figure 21 - Central Results Summary

In the transition description, you can include dynamically changing data that came from the step's operation or elsewhere in the flow run. You do so by storing the data in a flow variable, then, in the description, including a reference to the flow variable.

For example, a step that carries out a ping command can place the host machine's name into a flow variable called "host". To use this value in the transition description you can reference it with the syntax `${host}`. A description from the success response might read "Successfully pinged `${host}`". When this is run in Central against a host named "server1", the summary description will read "Successfully pinged server1."

Adding a transition

To add a transition

1. On the flow diagram, click a response and drag to the step that that response should lead to.
2. To open the inspector for the transition, select the transition by single-clicking it and then click the Inspector tab at the bottom of the flow canvas (or double-click the transition).

Figure 22 - Transition Inspector

By default, the transition name is the same value as the name of the response where it originates, but you can change the transition name to any text you want.

3. To change the transition's name, in the **Name** text box, type the new name.
4. To limit who can run the step beyond the transition, in the **Gated Transition** area:
 - Select the **Check user's roles before proceeding** check box.
 - In the **Required Role** drop-down list, select the role that the user should be a member of in order to continue executing the flow.
5. To specify that the run be handed off following the transition, click **Hand-off flow run after this transition**.
6. To assign a value to the transition, in the **Flow Transition Value** box, type a value for the transition
7. In the **Description** text box, type a description.

Note: To show the flow user (in the **Summary Information** area of the **Ops Flows** tab) with a description of what happened in the preceding step that caused this transition to be followed, this is where you type a description of the step's action and outcome. You can use flow variables to store changeable information for using here.

For example, to identify a server whose name is stored in the servername flow variable, you could type, "Server \${servername} is available for connection."

8. Save your work.

Re-arranging transitions

You might want to move and reshape transitions to tidy up your flow or to separate transitions that are in a stack. (By default, transitions that have the same origin and destination steps are stacked on each other.) You can move and reshape transitions and move transition names with two versions of clicking and dragging.

To add curve-defining points and change a transition's shape

- Position your mouse over the transition, hold down SHIFT, and drag until the transition curves the way you want it to.
- If the cursor is not positioned on an existing curve-defining point, SHIFT+dragging the transition adds a new curve-defining point as well as moves the transition.

For instance, in the following flow, we'd like to move the second Iterator step's **failure** transition so that it doesn't cross any transitions.

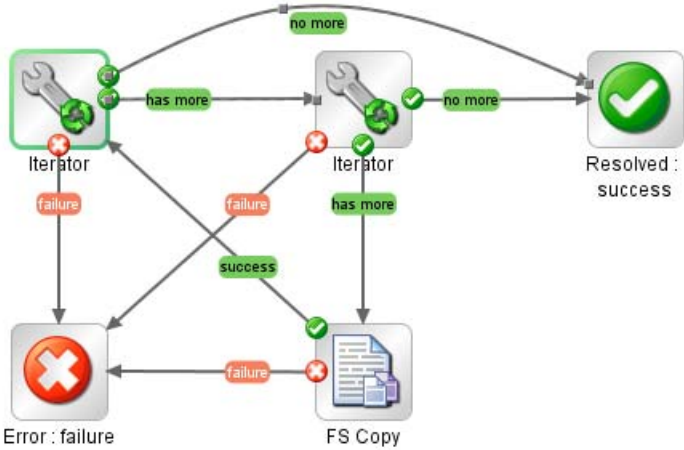


Figure 23 - Before SHIFT+dragging a transition

Using the procedure described above, we drag the transition to a new location. Note the curve-defining point that was added.

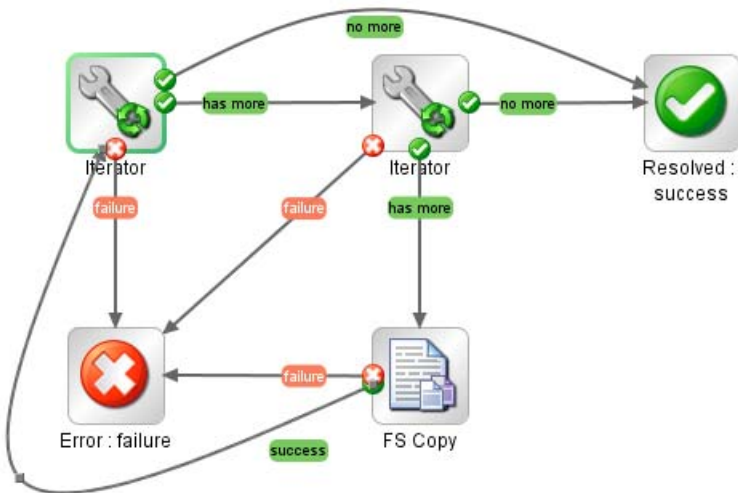


Figure 24 - After SHIFT+dragging the transition

To move a transition name

- Click the name and drag it where you want it to be.

To add a transition between two steps

- On the step that you want to connect to the next step, click the **x** that represents one of the responses, and drag to the step that that response should lead to.

When you move the mouse over the **x**, the response is labeled with the response.

The transition that you create is also labeled with the response from which it originated.
- In the flow diagram, select the transition (click its name) and click **Inspector**.

The transition's Inspector appears. The following example is the Inspector for the **success** transition that connects the **Select a Service** step's **success** response to the next step.

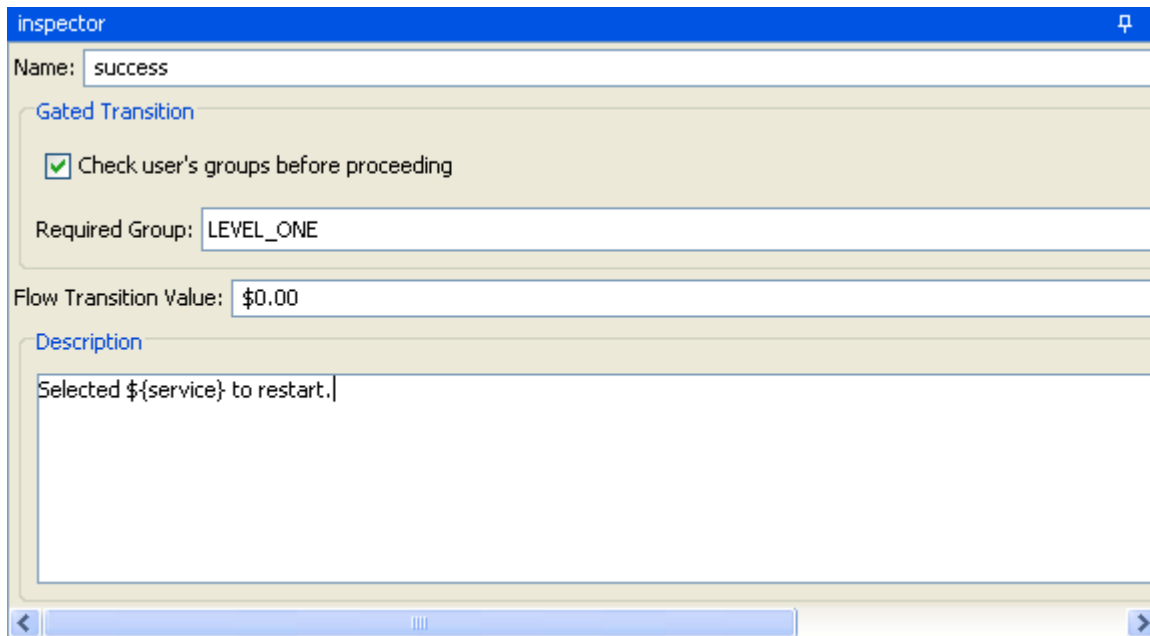






Figure 25 - Transition inspector

3. To limit execution of the flow beyond this transition to a specific role or to assign a value to this transition, under **Gated transition**, select **Check user's groups before proceeding**, and then select a group in the list box beside **Required Group**.
In the example above, flow user's must be a member of the LEVEL_ONE group.
4. To count completion of the transition in the flow's value, type a value (in American dollars, with two decimal places) beside **Flow Transition Value**.
5. Type a description in the **Description** box, then click **OK** and save your work.

Note: The description in the above example refers to the flow variable **\${service}**. For information on using flow variables, see [Flow variables: Making data available for reuse](#).

Return steps

Return steps are the step representation of flow responses. There are four kinds of return operations, which indicate four primary possible end states to the flow.

- Resolved: 
- Diagnosed: 
- No Action Taken: 
- Error: Failure: 

These return steps are sealed, so you cannot modify them.

To add a return step to the flow

1. From the flow diagram toolbar, drag the appropriate return operation to the flow canvas.



Figure 26 - Return step icons

You can change the response of a return step to more accurately reflect the outcome that led to the return step. For example, if the outcome that led to an Error:Failure return step were not a failure in an operation but a result that did not meet a required threshold, then you might want to create a new response for the Error:Failure step that reflects this outcome.

2. To change the response of a return step, in the flow diagram, right-click the return step, point to **Select Response**, and click the desired response.

OR

To create and assign a new response to the return step, right-click the return step, point to **Select Response**, and click **Add New Response**, then name the new response.

Inputs: Providing data to operations

Inputs are the means by which you specify how the operations in a flow obtain the data that they need, and when the data is obtained.

For example, in the Network Check flow, the start step's operation pings a server, so needs the IP address of the server to ping. You provide the IP address in an input.

You can create an input to supply the IP address:

- On the flow (that is, in the flow's properties).

The value of an input added to the flow can be used throughout the flow and is available as soon as the flow starts. This is recommended for data that is provided from outside the flow.

Flow inputs are very useful for providing data to steps (typically return steps, at the end of the flow) that report data for recording in Central charts.

- On the step that is associated with the operation.

A step input supplies its value to the step's particular application of its operation. As a step input's value, you can assign data obtained from earlier steps in the flow.

- On the operation itself.

Data assigned to an operation input is used in each of the steps created from the operation.

Operations are the classes of which steps are instances, so step inputs are created from the inputs of the operation from which the step was created. In a flow, a step's input provides the value that the operation input uses in that particular instance. Thus, if you change a step input's data assignment, the step input's data assignment is used. This tailors the step's inputs to the needs of a particular flow.

For instance, let's return to the Network Check flow: The ping operation gets the target IP address from the **host** input (which is defined in the operation as a user prompt). When we create a step in a flow from the operation, we can change the step's host input data assignment to a specific value: the target IP address for that particular flow.

If a flow input's value is assigned to a flow variable and the step input is configured to get its value from that flow variable, then the flow input determines step input. Otherwise, the step input gets its value from the assignment (user prompt, assignment from another flow variable, etc.) that is defined for it.

Note: If you don't supply a flow or a step with an input that it needs, the input is by default converted to a user prompt. This requires, of course, that the user have the information necessary for the flow to do its work.

Creating an input

The procedure for creating an input is essentially the same, regardless of whether you create the input on a flow, a step, or an operation.

To create an input for a flow or operation

1. To open the flow or operation's **Properties** sheet, in the **Library** pane, navigate to and double-click the flow name and then click the **Properties** tab at the bottom of the authoring pane.
2. On the **Inputs** tab, click **Add Input**, and then name the input.

Warning: Do not name the input "service" or "sp". Doing so can create errors in flow runs in certain situations.

The new input appears in its own row.

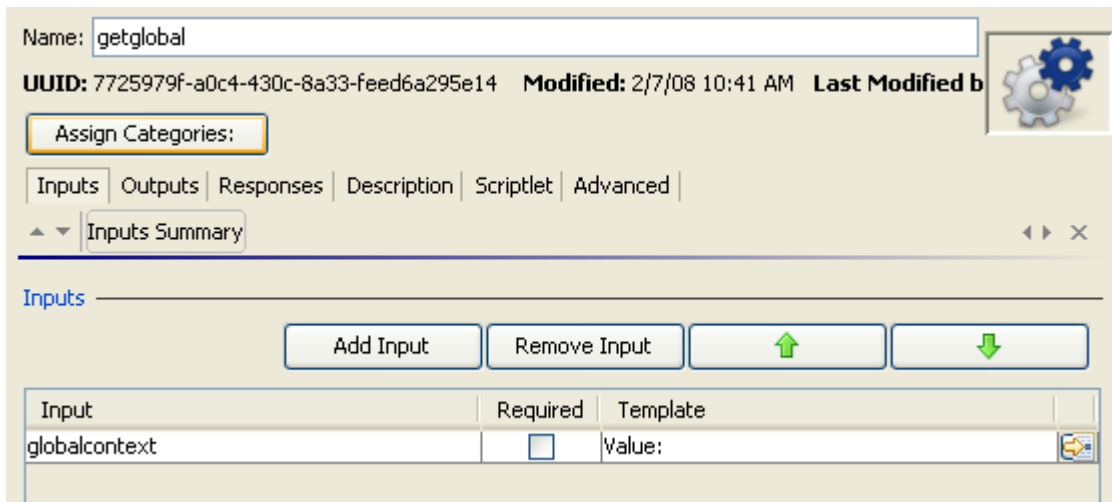



Figure 27 - Flow Properties sheet with a new input

3. If the input's data is required for the step (or later steps in the flow) to function, select the **Required** check box.
4. To open the input editor for defining how the input gets its data and how it behaves otherwise, click the right-pointing arrow at the end of the input's line (.

The input editor looks like the following:

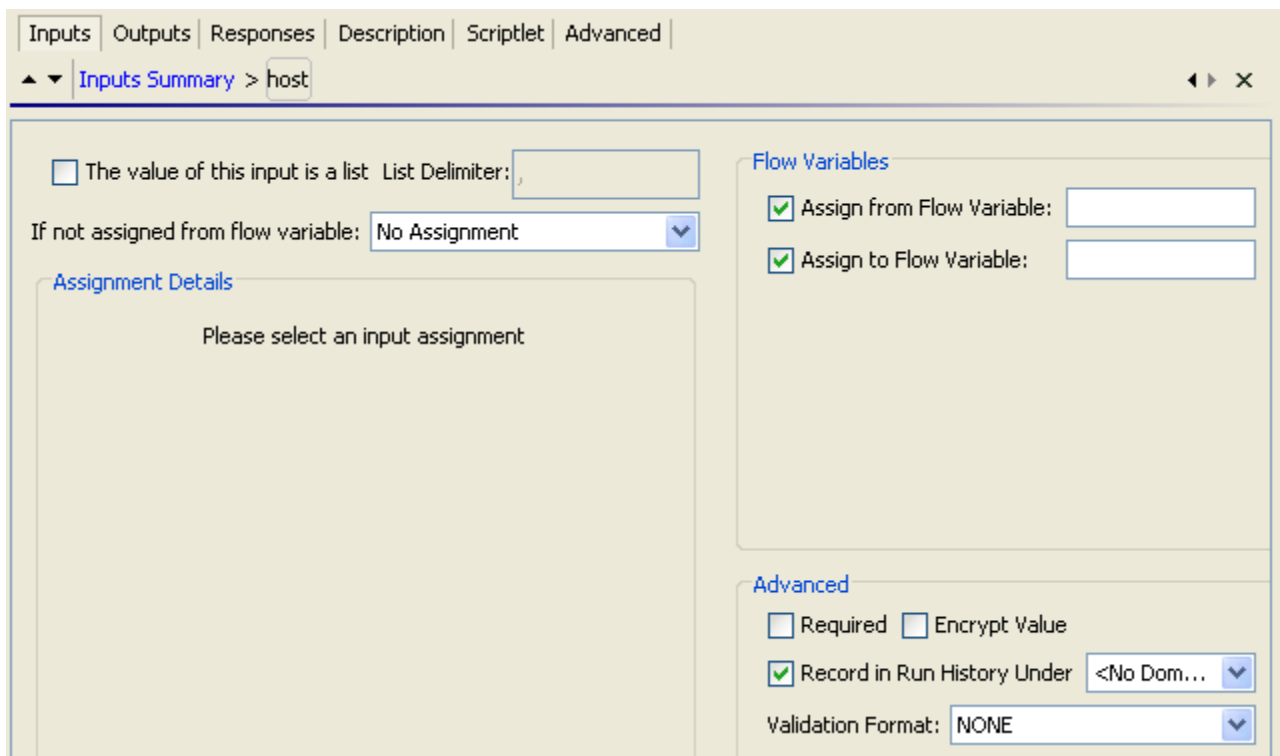


Figure 28 - Input editor

For information and procedures on defining how the input works, see the following:

- On assigning a data source for the input, see the following topic, [Assigning a data source to inputs](#).
- On assigning the input's value to a flow variable, see [Creating a flow variable](#).
- On recording the input's data for use in reporting charts in Central, see [Recording values for reporting in Dashboard charts](#).

5. To modify another input without closing the input editor, scroll through the step's inputs by clicking the up or down arrow beside **Inputs Summary**.
6. After making changes, save your work.

Creating an input for a step is slightly different, due to differences in the step Inspector.

To create an input for a step

1. Open the step's Inspector, and then click the **Inputs** tab.
For information on opening the step Inspector, see [Modifying a step](#).

2. Click **Add Input**, and then name the input.

Warning: Do not name the input "service" or "sp". Doing so can create errors in flow runs in certain situations.

The new input appears in its own row.

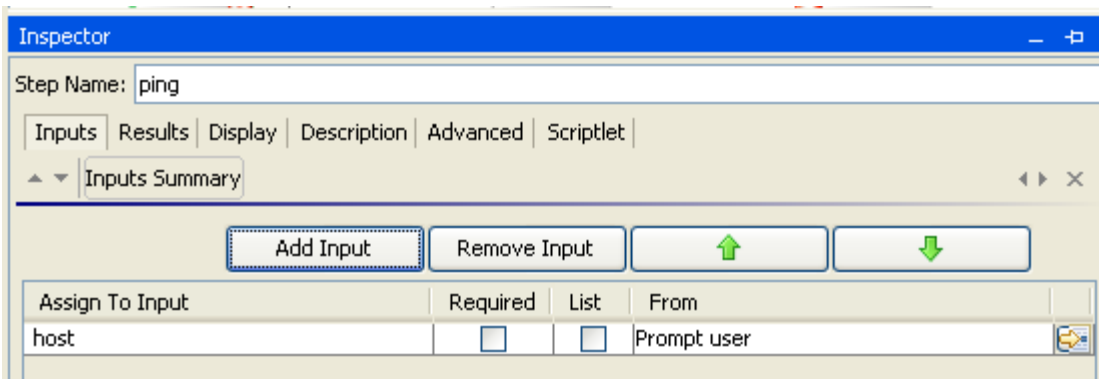


Figure 29 - A new input

3. If the input's data is required for the step (or later steps in the flow) to function, select the **Required** check box.
4. If the input's data is a list, click the **List** check box.
To supply inputs with lists of values, you will do more in the input editor. For more information, see the following topic, [Assigning a data source to inputs](#).
5. To open the input editor for defining how the input gets its data and how it behaves otherwise, click the right-pointing arrow at the end of the input's line (🔗).
The input editor looks like the following:

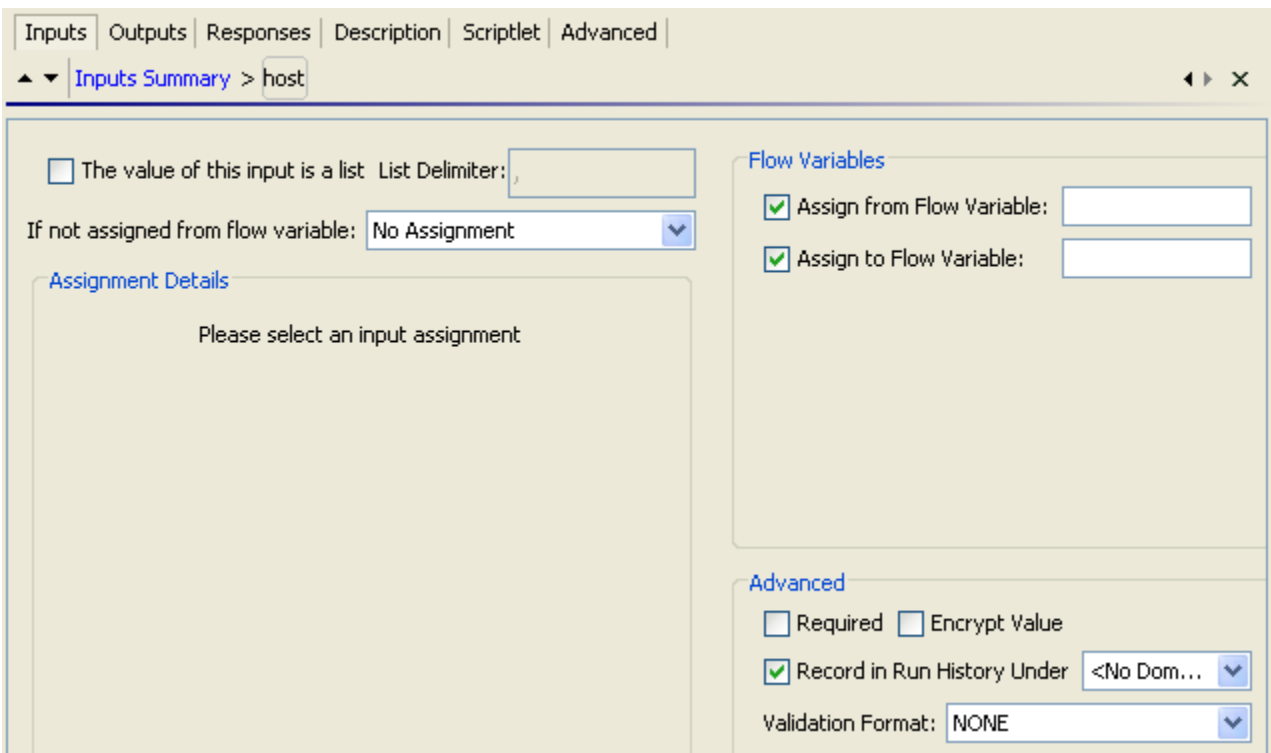


Figure 30 - Input editor

For information and procedures on defining how the input works, see the following:

- On assigning a data source for the input, see the following topic, [Assigning a data source to inputs](#).
- On assigning the input's value to a flow variable, see [Creating a flow variable](#).
- On recording the input's data for use in reporting charts in Central, see [Recording values for reporting in Dashboard charts](#).

- If you need to modify another input, scroll through the step's inputs by clicking the up or down arrow beside **Inputs Summary**.
- After making changes, save your work.

Assigning a data source to an input

There are several types of data sources you can assign to inputs:

- **User prompt**
This is the default data assignment when no other data assignment has been defined or is possible.
- **From a flow variable**
If there is a flow variable with the same name as the input and that flow variable stores a value, that value is assigned to the input. For example, if the input is named **list** and there is a flow variable named **list** that contains a value—say, a list—the **list** input is populated with the list flow variable's list.
- **A specific value**
You can reference a flow variable as providing the specific value for the input. Doing so enables you to set up schedules that each run the flow with a different value for the input.
- **No assignment**
If you do not assign a data source for the input, HP OO treats it as a user-prompt assignment in order to obtain a value when the flow is run.
- **The previous step's result**
- **A system account**
- **The logged-in user's credentials**

To assign a data source to a flow, step, or operation input

- On the **Inputs** tab, click the right-pointing arrow on the input's row to open the input editor.

The screenshot shows the 'Inputs' tab in the HP OO interface. The breadcrumb path is 'Inputs Summary > host'. The editor is divided into several sections:

- Top Section:**
 - The value of this input is a list. List Delimiter:
 - If not assigned from flow variable:
- Assignment Details:**
 - Please select an input assignment
- Flow Variables:**
 - Assign from Flow Variable:
 - Assign to Flow Variable:
- Advanced:**
 - Required Encrypt Value
 - Record in Run History Under:
 - Validation Format:

Figure 31 - Input editor for “host” input

2. To assign the input its value from a flow variable, make sure the **Assign from Flow Variable** check box is selected, and then type the name of the flow variable in the text box.
If the name of the flow variable from which you want to get the input’s value is the same as the name of the input, you can leave the text box blank.

If the flow variable that you specify as a data source for the input doesn’t exist or has no value stored in it, the input gets its value from the source selected in the **If not assigned from flow variable** drop-down list.

Note: Among the flow variables whose values you can assign to the input are reserved flow variables, whose values are always available for referencing anywhere in the flow. For more information, see [Reserved flow variables](#).

Creating an input always creates a flow variable of the same name.

3. To make sure that the input’s value is not assigned from a flow variable, unselect the **Assign from Flow Variable** box and then select a source from the **If not assigned from flow variable** box.

Tip: If you assign an input a specific value, then you might want to also select the **Assign from Flow Variable** check box and leave its text box blank, so that the input tries to obtain its value from a flow variable of the same name. This way, a Central user who creates a schedule for the flow can assign a different value (in the **Scheduler** dialog box) to the input. If you do not enable the input to get its value from a flow variable of the same name, then even if the Central user specifies a different value when scheduling the flow, the scheduled flow run(s) will still use the specific values that you have assigned the input in Studio.

The input’s value can be a list, such as the values you provide to a multi-instance step.

4. If the input’s value is a list:
 - Select **The value of this input is a list**.
 - In the **List Delimiter** box, type a character (or character sequence) that separates the elements in the list.
5. To change the data source when there is no flow variable that is assigned as the data source, select a data source type from the **If not assigned from flow variable** list.

- **Prompt the User**

The data source is the user’s response to a prompt.

- a. In the **UI Component Type** drop-down list, select a user interface:

- **Plain Text Field**

In the **Prompt Label** box, type the prompt message for the user.

- **Selection List**

When you define a **Selection List** type of user prompt, the **Assignment Details** area of the input editor looks like the following:

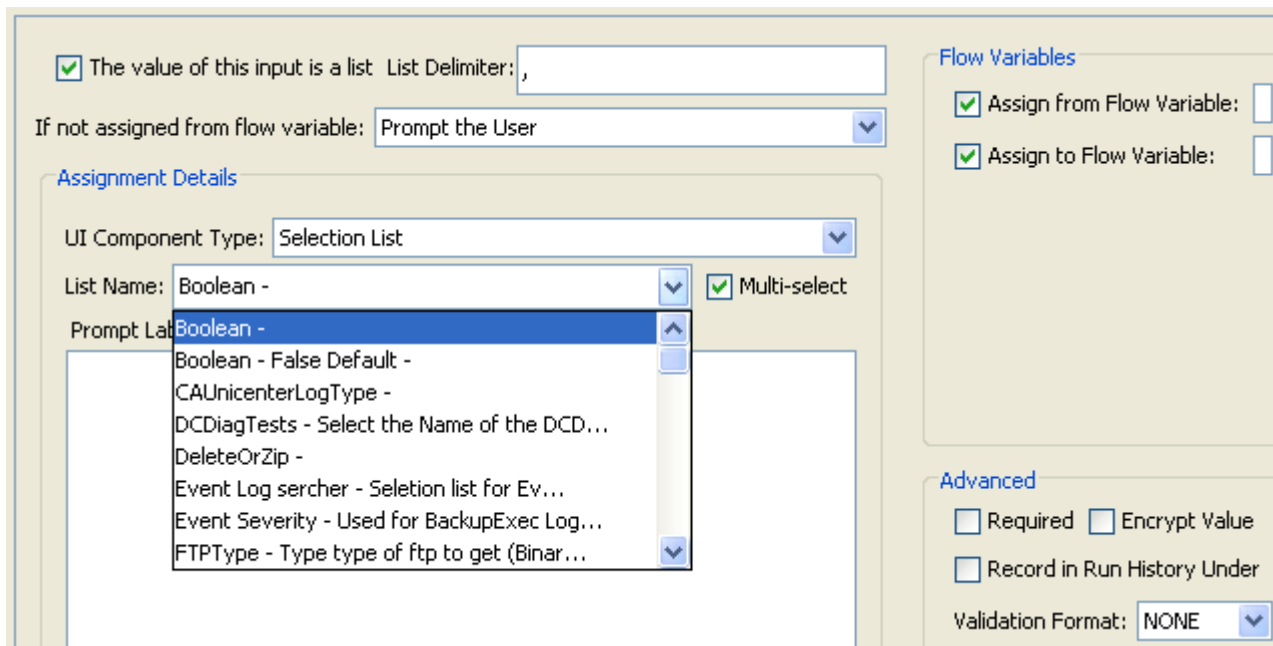


Figure 32 - Defining a selection list user prompt

In the **List Name** drop-down list, choose a selection list. These lists are stored in the **Library** pane, in the **Configuration\Selection Lists** folder.

Note: You can add more selection lists to the **Configuration\Selection Lists** folder or add items to existing lists. For information on how to do so, see [Selection lists for user prompts](#). To enable the user to choose several elements in the list, select the **Multi-select** check box.

- **Selection List From Flow Variable** – a selection list that is provided by a flow variable.

In the **Variable Name** text box, type the name of the flow variable.

Note: In the flow variable, the selection list elements are separated by line breaks.

To enable the user to choose several elements in the list, select the **Multi-select** check box.

- **Domain Term**

From the **Domain Term** drop-down list, select a domain term from which to get the value.

To enable the user to choose several elements in the list, select the **Multi-select** check box.

- **User-entered Credentials**

These are a name and password that that the user supplies to the flow in response to the prompt.

b. In the **Prompt Label** box, type the prompt message that you want to show the user.

- **A Specific Value**

The data source is an unchanging value or list of values that the step acts on.

In the **Assignment Details** area, in the **Value** box, type the text of the specific value(s). You can reference a flow variable to obtain part or all of the value. For example, the value might be **\${productName}**.

- **The Previous Step's Result**

The data source is the result of the previous step as modified by any filters you have created on that step. There are no details to specify for this data source type.

- **A System Account**

The data source is a system account, which references user credentials. System accounts enable the flow to perform tasks that require account credentials, while protecting the credentials from exposure by keeping them hidden behind the system account name. The system accounts that are available in the list are those that you have created. For information on creating system accounts, see [System accounts: secure credentials](#).

In the **Assignment Details** area, in the **System Account** drop-down list, select the name of the system account that you want to use for the input.

- **The Logged-in User's Credentials**

This data source provides credentials when the log-in account on which the flow is being run has the required access permissions to perform the tasks required by the flow.

For the **Prompt the User**, **A Specific Value**, and **The Previous Step's Result** data source types, you can encrypt the input's value, so that it appears as a row of asterisks when the flow is run in Central.

6. To encrypt the input's value, under **Advanced**, select the **Encrypt value** checkbox.

Important: When the input's assignment is **The Logged In User's Credentials**, its value is always encrypted. However, the encrypted state does not carry over if you change an input's assignment from **The Logged In User's Credentials** to **Prompt the User**. For the value to be encrypted, you must select the **Encrypt Value** checkbox.

7. To make the input required, select the **Required** box.

There are two places where you can select a **Required** check box to make the input required: In the input editor or in the input's row on the Inputs tab. Selecting or unselecting either check box is reflected in the other.

For the **Prompt the User**, **A Specific Value**, and **The Previous Step's Result** data source types, you can record the value of the input in a domain term that you choose. The value will be available for recording in the flow's run histories in Central. This means that the input's value can be used when the flow is used for diagnostics or auditing.

8. To record the input's value in the flow's run histories in Central:

- Select the **Record in Run History Under** check box.
- In the drop-down list, beside **Record in Run History Under**, select the domain term under which you want to record the input's value.

You can make sure that the value fits a certain format, such as email, filename, IP address, alphanumeric, a phone number, etc.

9. To validate the input's value with a format, in the **Validation Format** drop-down list, select the desired format.
10. Save your work.
11. To work with another input or another step without closing the Inspector, click the up or down arrow (▲ ▼) to the left of **Inputs Summary**.

Providing a list in a user prompt input

The following procedure refers to steps that are described in detail in [Assigning a data source to an input](#).

To provide a list to the flow user in a user prompt input

1. On the **Inputs** tab, add an input.

2. Open the input editor.
3. In the **If not assigned from flow variable** drop-down list, select the **Prompt the User** data source type.
4. In the **UI Component Type** drop-down list, select one of the types that can offer a selection list to the user.
 - **Selection List**
 - **Selection List from Flow Variable**
 - **Domain Term**
5. Select the specific list source that you want to present to the user, according to the UI component type that you selected:
 - If **Selection List**, select the list from **List Name**.
 - If **Selection List from Flow Variable**, select the flow variable from **Variable Name**.
 - If **Domain Term**, select from **Domain Term**.
6. To allow the user to make multiple selections in the list, select the **Multi-Select** check box.

Inputs and scheduling flow runs

To create a schedule that automatically starts runs of a flow, the flow must be able to run without needing user initiation or input. You need to change any inputs that are user prompts to specific values.

In addition, when you create schedules for a flow, the scheduling box in Central enables you to store a different value in the flow variable for each schedule. For instance, you can create several schedules for the Network Check flow and point it at a different server for each schedule, by supplying a different server IP address to the host input on each schedule.

For information on creating a schedule for a flow, see Help for Central.

To enable schedules to define different specific values for the input

- In the input editor, select **A Specific Value** as the data source for the input and, in the **Value** box, reference the flow variable with the syntax `${flowvariablename}`
For information on assigning a data source to an input, see [Assigning a data source to an input](#).

Removing an input

To remove an input from a step

1. Open the Inspector for the step you're interested in or the flow or operation's Properties sheet.
2. On the **Inputs** tab, on the list of inputs, highlight the input that you want to remove, and then click **Remove Input**.

Flow variables: Making data available for reuse

You can use flow variables to move data within and between flows. You do so by referencing a flow variable containing the data:

- In different steps in a flow or in a different flow from the one in which you created the flow variable.

Once you create a flow variable, you can then reference it (and the data that it stores) in another flow, step, or operation input.

- In flow, step, and transition descriptions.

For example, the Ping Latency operation filters out the average duration of the ping. A step associated with this operation could save the average duration as the flow variable “latency,” then the transition that follows this step could report that value to the user.

- Within a lane in a parallel-split step.

A lane step can use a flow variable’s value that was written to the flow variable by an earlier step in the same lane.

Note, however, a step in one lane cannot use a flow variable’s value that a step in a different lane wrote to the flow variable.

- As part of the data you are testing with a response rule

To see whether an output string or error string contains a value that you have stored in a flow variable.

- In scriptlets

- In operation parameters

If an operation parameter takes a value, you can access that value by referencing a flow variable that contains it.

Creating a flow variable


Creating a step or flow input automatically creates a flow variable with the same name that has the input’s value stored in it. However, you can specify that the input’s value be saved to or that the input get its value from another flow variable. You can take advantage of this feature by adding several operations that act on a server and having the first operation prompt the user for the server. All the rest of the following operations that have inputs of that name will automatically use that server name.

You can create flow variables from operation, step, or flow inputs or results. The method for doing each is the same.

Note: The following procedures assume a basic knowledge of how to create inputs and step results.

- For information on creating inputs, see [Inputs: Providing data to operations](#).
- For information on creating operation outputs, see [Operation outputs](#) and [Filtering outputs and results](#).
- For information on creating step results, see [Step results](#) and [Filtering outputs and results](#).

To create a flow variable from an input

1. Open the operation’s or flow’s Properties sheet or the step Inspector.
2. On the **Inputs** tab, select an input or create a new one.
3. On the input’s row, click the right-pointing arrow ().
4. In the editor, select the **Assign to Flow Variable** box.
5. Type a name for the flow variable in the text box.

OR

To name the flow variable the same as the input, leave the text box blank.

For instance, if you leave the text box blank and the name of the input is **list**, the name of the flow variable will also be **list**.

6. Save your work.

To create a flow variable from a result

1. Open the operation's or Properties sheet or the step Inspector.
2. On the **Results** tab, create a new result.
3. In the new result's row, under **Assign To**, select **Flow Variable**.
4. Under **Name**, specify the name of the new flow variable.
5. Under **From**, specify the source of the value for the flow variable.
6. If necessary, create one or more filters for the result.

For more information on creating a filter, see [Filtering outputs and results](#).

7. Save your work.

Creating global flow variables

You create a global flow variable with a scriptlet.

For example, to create and assign a value to a global flow variable in a Rhino script, you would use a command with the following syntax:

```
scriptletContext.putGlobal("<globalflowvariablename", <value>);
```

For more information on creating scriptlets, see [Scriptlets](#).

Reserved flow variables

In addition to flow variables that you create, the following reserved global flow variables are always available for use in any flow or run. These flow variables can be particularly valuable for using within a scriptlet.

Reserved flow variables

Reserved flow variable	Possible values, initial values
Boolean	True False
Boolean – False Default	False True
CAUnicenterLogType	CB RS LOG
DCDiagTests	All Intersite Connectivity Replications NCSecDesc CutoffServers Advertising KnowsOfRoleHolders FsmoCheck RidManager MachineAccount OutboundSecureChannels ObjectsReplicated frssysvol systemlog Kccevent
DeleteOrZip	Delete Zip
Event Log sercher	* System Security Application
Event Severity	Error AttentionRequired Warning Information All

execution_userid	The value is the user name of the user who is logged in and running the flow.
FailureMessage	Initial value: null
FTPType	Binary Ascii
HPSM ResolutionFixTypes	permanent temporary
HPSM Ticket Types	Change Problem Incident
Host Type	Local Host Remote Host
HostType	local remote
IISiteStatus	Running Stopped
Job History Status	All Successful Failed Completed With Exceptions
LinuxLogRotatorOption	Archive Delete
LinuxLogSeverity	DEBUG INFO NOTICE WARNING ERR CRIT ALERT EMERG
List Type	ClearedAlertList ActiveAlertList
LocalOrRemote	Local Host Remote Machine
MailBodyType	html text
NASDiagnostics	Memory Troubleshooting NAS Detect Device Boot NAS Device File System NAS Duplex Data Gathering NAS Flash Storage Space NAS Interfaces NAS Module Status NAS OSPF Neighbors NAS Routing Table NAS Topology Data Gathering Hardware Information
NNM - Lifecycle State	Registered In Progress Completed Closed
NNM - Node Management Modes	Out of Service Managed Not Managed
NNM – Priority	None Low Medium High Top
NetDiagTests	All Autonet Bindings Browser DcList DefGw DNS DsGet Dc IpConfig IpLoopBk IPSec IPX Kerberos Ldap Member Modem NbtNm Ndis NetBTTransports Netstat NetWare Route Trust WAN WINS Winsock
Notification Option	Email Display Write to File None
Radia Priorities	Must Should May MayNot ShouldNot MustNot
RemedyCaseStatuses	New Assigned Work In

	Progress Pending Resolved Closed
RemedyUrgency	High Medium Low Urgent
RemoteOrLocal	remote local
RS_Previous_Response	The response of the previous step
RS_Previous_Transition	The transition from the previous step
RS_Previous_Transition_Annotation	The annotation of the transition from the previous step
run_id	Initial value: -1
scp_copyAction	To From
ScriptLanguages	vbscript jscript
SelectionListDeleteOrZip	LocalDelete LocalZip RemoteDelete RemoteZip
Service Desk Object	Servicecall Incident Change Workorder Problem Person Workgroup Organization Service ServiceLevelAgreement Project Configuration MainCon
ServiceStatus	Running Stopped Paused
SFTP_operations	cd ls pwd lpwd lls chgrp chmod chown compression symlink get lcd lstat mkdir put rm stat exit
SiteScopeAckType	Enable Disable
SNMPVarbindType	TIMETICKS COUNTER GAUGE INTEGER OID STRING ADDRESS HEX
SqlAuthentication	Windows Sql
SQLDBType	Oracle MSSQL Sybase
SQLReportingServerFormat	HTML3.2 HTML4.0 HTMLLOWC MHTML IMAGE EXCEL CSV PDF XML
String Comparater Match Type	Exact Match Contains Contains Once Does Not Contain Match All Words Match No Words RegEx
TimedOut	Initial value: null
TomcatActions	Start Stop reload undeploy
VMWareActions	Start Stop Enumerate Status
WMIQueryFormat	csv text
WindowsClusteringStates	Online Offline

WindowsRegistryValueType	REG_SZ REG_EXPAND_SZ REG_BINARY REG_DWORD REG_MULTI_SZ
WindowsScriptLanguages	VBScript Jscript
WindowsServiceStartupMode	Boot System Auto Manual Disabled
WindowsServiceStatus	Running Stopped Paused
WMIQueryFormat	csv text
Yes-No	Yes No

Concurrent execution: Running several threads at the same time

Within a flow, you can enable *Concurrent execution*, also called *parallel execution*. Let's look at what this can mean for you in a few scenarios:

- To diagnose a problem, you need to run health checks against a database server, an application server, a web server, and a router. You could find your diagnosis much more quickly if you could run the four health checks simultaneously.
- You have a software upgrade to install on 100 servers. If you could install it on 25 servers simultaneously, you could upgrade all the servers in a quarter of the time it would take to upgrade them all one at a time.
- One of the steps in a diagnostic flow creates a trouble ticket or creates and sends an email. It would speed up resolution if the flow could keep running with subsequent steps while the one step is creating the ticket or email.

HP OO provides multiple ways to achieve such parallel execution.

- To run **entire flows** in parallel, you can do either of the following:
 - On the **Schedule** tab in Central, schedule simultaneous runs of a flow.
 - Specify parallel automatic runs of a flow by a URL.

Within a flow, you

The three kinds of steps that have concurrent processing are:

- **Parallel split step**, in which one step contains several series of steps that execute simultaneously.

Parallel split steps are intended for performing dissimilar and separate series of steps at once. Each series of steps is called and represented visually in the flow diagram as a **lane**. The steps contained in each lane are called **lane steps**. The lane's series of lane steps are much like a subflow and you might think of them as such, but in several respects, including movement of data into and out of them, they are very different from a subflow or flow.

In the scenario in which you want to run several different kinds of health checks simultaneously, you could create a parallel split step with four lanes. In one lane, you would create the lane steps necessary to run a health check on the database server; in the next the steps necessary to run a health check on the application server, and so forth.

For more information, see [Parallel split steps](#).

- **Multi-instance step**, which executes simultaneously with multiple members of a list provided for a value in an input of the step's operation. You can **throttle** a multi-instance step, which means limiting how many values it can process at once.

In the example of upgrading many servers at once, let's assume that you have a step that is associated with the subflow that performs the upgrade and that this step takes the target input that specifies which server the subflow runs against. You could turn this step into a multi-instance step and supply it with a list of the servers you want the subflow to upgrade. The step would then run the check against all those servers simultaneously.

If running the subflow against too many target servers simultaneously would bog down your infrastructure, you could throttle the multi-instance step by specifying that it would only process, say, 25 target inputs at once.

For more information, see [Multi-instance steps](#).

- **Nonblocking step**, which allows the flow to continue with subsequent steps while the nonblocking step is still executing.

In the case of a step that opens a trouble ticket, you could make the step a nonblocking step, thus allowing the flow to proceed while the step creates the ticket.

For more information, see [Making steps nonblocking](#).

Parallel split steps

A parallel split step is a set of step sequences that are carried out simultaneously. Each sequence is called a *lane*, and the steps within the lane are called *lane steps*. The collection of flow variables for a lane are collectively called the *lane context*.

Parallel split steps are best used for doing dissimilar things simultaneously and independently of each other. Note the contrast with multi-instance steps, whose instances do the same thing with multiple variations of a single input.

For example, you might use a parallel split step for installing a patch and sending email about the installation to the appropriate person in IT:

- One lane could include a step made from a flow that installs the patch.
- The other lane could send the email.

When working with parallel split steps, also keep the following in mind:

- Data cannot be exchanged between lanes, because the steps in each lane only have the values from available that were available when the parallel split step (all its lanes) started.
- There is only one output/response for a parallel split step: **All the lanes have completed**.
- If steps in two lanes write to a single flow variable, the value that the last lane to complete writes is the value that remains in the flow variable after the parallel split step has completed. Note: This is not necessarily the same as the value of the last write operation to complete.

For example, consider the following sequence:

- In Lane A, step A1 passes "42" to **threshold**.
- In Lane B, step B2 passes "34" to **threshold**.
- Lane B completes.
- Lane A completes.

The value in **threshold** would be "42", even though step A1 completed after step B2.

- For parallel split steps, permissions for execution of flows and operations and for use of other elements (system accounts, etc.) are determined by the permissions granted for the parent flow.

- Gated transitions in lanes can't be handed off. If the user does not have the required group membership, the run fails, with an error message that tells the user that he or she is not a member of the required group.
- Similarly, transitions in lanes cannot be configured as handoffs.
- Parallel split steps and their lane steps are all checkpointed, and you cannot remove the checkpoints. This means that if a run is interrupted while the parallel split step is running, then when you restore the run, it is restored to the point just before the parallel split step started. For more information on checkpoints, see [Checkpoints: Saving a flow run's progress for recovery](#).

Lane order: starting and finishing

Lanes start and run simultaneously in Central, but are executed as a series in the debugger. You cannot control the order in which lanes are run in the Debugger, but by giving them unique names, you can see the order in which they ran.

Moving data into and out of a parallel split step

When a parallel split step starts, each of its lanes obtains a copy of the flow variables in the global context, so they have the values associated with those flow variables available for any of its steps to use.

Steps within a parallel split step's lanes can both obtain data from the local and global contexts and save data to local context. Like any other steps, lane steps can only write to the global context by means of a scriptlet that uses the `scriptletcontext.putglobal()` method. The data that lane steps write to the global context are not merged back to the global context until all the lanes have finished. (For the syntax for using `scriptletcontext.putglobal()`, on the Scriptlet tab of an operation or step, insert the JavaScript template.)

Important notes:


- If two or more lanes have steps that write values to the same flow variable, the value written to the flow variable by the last of those lanes to complete is the value that that variable has after the parallel split step completes.
- In a parallel split step, a step within a lane cannot pass values to a step in another lane.

Creating parallel split steps

When you create parallel split steps, keep the following in mind:

- You cannot pass data between lanes (the order in which steps in any two lanes are carried out relative to each other cannot be controlled or predicted).
- A parallel split step has only one response: Success (meaning that all the lanes have completed).
- A parallel split step cannot contain another parallel split step, but it can contain a subflow that contains a parallel split step.

To create a parallel split step

1. On the Studio toolbar, click the parallel split step icon () and drag to the flow design. The step appears with the minimum number of lanes (two) by default.
2. Create the step sequence you want within each lane. You create a lane's step sequence by creating steps within each lane, using the same techniques and following the same rules that you do when creating a flow, with the following considerations:

- Lanes cannot contain return steps, so you connect all step responses either to the next step or to the exit point of the lane. You can find information on success, failure, and other responses and results of the lane steps in the debugger, and Central users can find this information in the flow's run-history reports.

It doesn't take long for the lanes' default size to be over-crowded by the steps that you create.

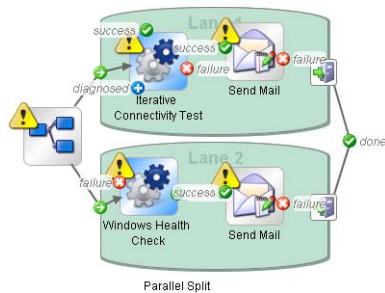


Figure 33 - Crowded lanes

To give your steps more room, you can move the parallel split step and expand each lane.

3. To move the parallel split step as a whole, click the icon that represents the entire step...




...and drag the step where you wish.

4. To expand a single lane, select the lane by clicking in a blank part of the lane and drag the side or corner handles until it's as large as you want.

5. After expanding the lanes, you can connect the steps within each lane.

As noted above, all step responses must be connected either to a subsequent step or to the

lane-end icon ().

6. To connect the parallel split step to the rest of the flow:

- If the parallel split step is not the start step, connect the step that precedes the parallel split step to the parallel split step icon.
- Connect the parallel split step's **done** response to the flow's next step.

At this point, a flow with a very simple parallel split step might look like the following:

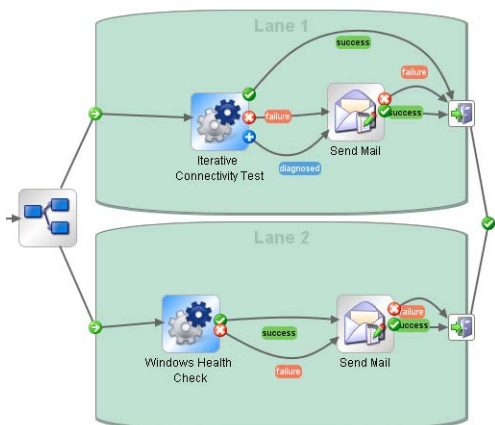


Figure 34 - Parallel split steps

Debugging a parallel split step

The key to remember when debugging a parallel split step is that the debugger processes parallel split steps serially rather than in true parallel execution. This is one way in which the debugger

does not precisely reproduce the behavior of a flow in a production environment. On the other hand, serial execution in the debugger of parallel split steps enables you to perform controlled tests for various conditions.

You debug a flow containing a parallel split, multi-instance or nonblocking step the same way you debug a flow without such steps.

For information on how to debug flows, see [Debugging a flow in Studio](#).

Changing the start step of a lane

As soon as you add a step to a lane, the lane-start is connected to the step.

To change the start step of a lane

- Right-click the lane step that you want to be the lane's start step, and then click **Set Start Step**.
OR
Drag the lane-start icon's connector from the lane step that is its current target to the step that you want to be the lane's start step.

Renaming lanes

To rename a lane

1. Right-click the lane and, in the drop-down menu, click **Edit Name**.
2. In the text box that appears, type the new name of the lane.

Rearranging lanes

You can move lanes up or down in the parallel split step. Note, however, that all the lanes begin at the same time, and that their graphical order does not affect the order in which their processing occurs.

To rearrange lanes

- Right-click the lane that you want to move and in the drop-down menu, click either **Move Lane Up** or **Move Lane Down**, as appropriate.

Creating lane steps: tips

There are several ways to save yourself work when adding steps to a lane:

- If you have several steps outside the lane that you want to add to the lane, you can select the steps as a group and drag the group into the lane.
- You can copy or move a step from one lane to another using the same methods that you use outside a lane:
 - To copy a step: CTL+C, CTL+V or CTL+drag
 - To move a step: Drag the step or CTL+X, CTL+V

Multi-instance steps

Suppose you want to run the Windows Diagnostic flow on 100 servers. Rather than creating a step that runs the flow on a single server, you can make the step that is associated with the flow into a multi-instance step and run the flow on all 100 servers at once.

A multi-instance step is a step that you can supply with many values for an input, which values are used by multiple, simultaneously running instances of the step. This enables the flow that contains the step to simultaneously run, for instance, many instances of the same operation on different targets.

However, if running the flow on 100 servers simultaneously will slow down your system's performance, you can, in effect, set a throttle level for the flow by specifying how many servers the multi-instance step should simultaneously start the flow for.

Multi-instance steps are always checkpointed, and you cannot remove the checkpoint. This means that if a run is interrupted while the multi-instance step is running, then when you restore the run, it is restored to the point just before the multi-instance step started (for more information on checkpoints, see [Checkpoints: Saving a flow run's progress for recovery](#)).

Using multi-instance steps in flow design

For each response in the multi-instance step other than the **group done** response, all the transitions of the last step in the path that follows the response must connect back to the multi-instance step. This is so important that we'll put it another way: Only the **group done** response should lead to a flow return step, that is, to the end of the flow.

In its simplest form, this is the paradigm for how a multi-instance step should be used in flow design:

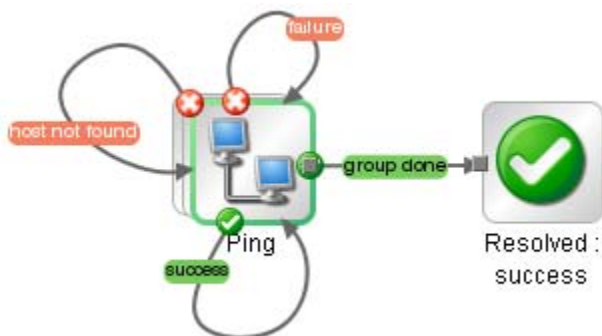
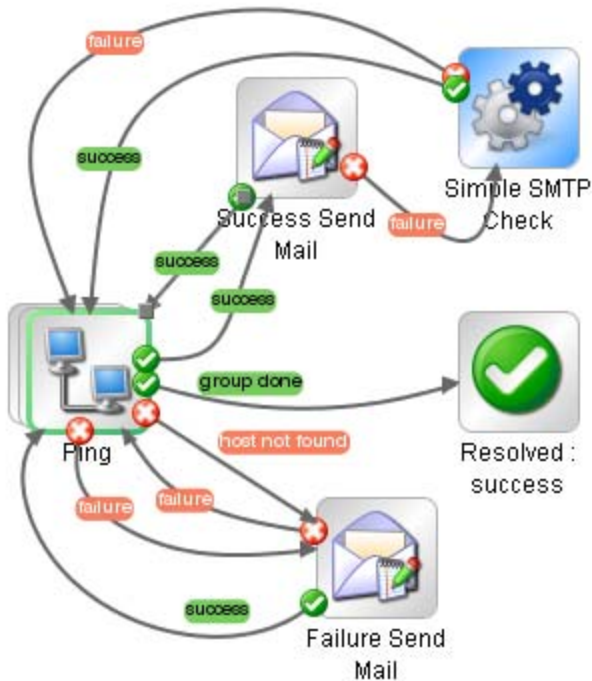


Figure 35 - The essentials of using a multi-instance step in flow design

More realistically, you probably want to have some other operations follow from the multi-instance step's various responses. Note that the following example still follows the essential pattern:

- Of the Ping step's four responses (**success**, **failure**, **host not found**, and **group done**), only the **group done** response goes to a return step.
- The **failure** and **host not found** responses each lead to an email step both of whose responses then connect back to the (multi-instance) Ping step.
- The **success** response leads to a chain of steps, but all the responses in that sequence sooner or later lead back to the (multi-instance) Ping step.

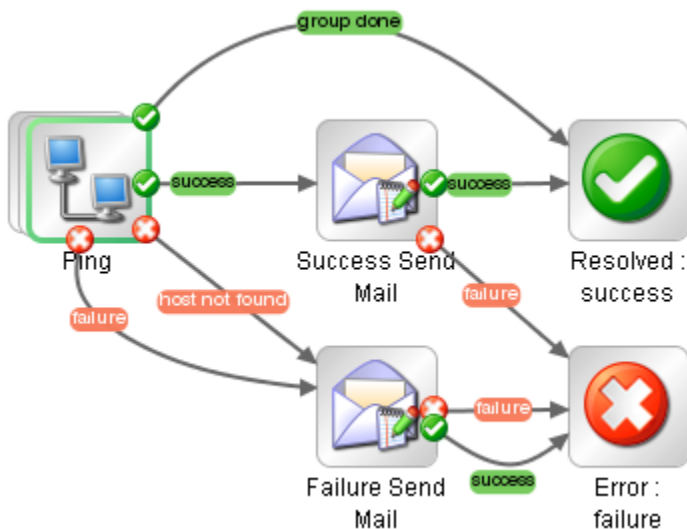


Good design: Connecting response destinations back to multi-instance step

There are two reasons that all the responses in a path of steps that follows from a multi-instance step should ultimately connect back to the multi-instance step:

- In the multi-instance step, the result of each instance is evaluated as it becomes available and the response for that result is chosen without waiting for the other instances' results to become available for evaluation. Thus if steps that follow the multi-instance step's responses do not loop back to the multi-instance step but go to return steps that complete the flow's run, the run cannot discover whether there are other results to evaluate.

As a result, in the following example, only one email will be generated by the flow, for the first instance to return a result.



Bad design: Response destinations NOT connected back to multi-instance step

- The other reason to connect all response destinations back to the multi-instance step is that if you use the bad design in a subflow, a run of the parent in Central cannot be handed off or interrupted subsequent to the step that is made from this subflow, because some results of the multi-instance step's instances remain unprocessed.

This way of processing multiple instances of a step and their results relates to one of the benefits of using the multi-instance step as opposed to the parallel split step: While the parallel split step waits for all of its lanes to complete processing before the flow can move on to the next step, in a multi-instance step, the flow continues executing with subsequent steps (based on the response chosen) for the multi-instance step's first-completed instance while the other instances are still processing. Thus the multi-instance step can work more efficiently for its purpose (to run the same operation with multiple targets) than the parallel-split step.

1. As soon as one of the instances of the multi-instance step completes processing, its response is chosen by the evaluation rules and the run proceeds with subsequent steps for that instance.
2. Meanwhile, the other instances continue processing. As each instance's processing within the multi-instance step completes, its result is stored to await its turn for further processing. At this point, the run begins to handle the multiple instances of the step serially.
3. When the first instance and any subsequent steps have completed, and you have connected this last step back to the multi-instance step, then the run returns to the multi-instance step and repeats the steps in the sequence for the result of the next instance to have completed. (If no other instance has completed processing within the multi-instance step yet, the run waits for the next instance result to become available.)

Creating multi-instance steps

To make a step into a multi-instance step, you change the step itself, and then provide a list to the input to which you want to supply more than one value. To provide the input with multiple values, you specify that the input's value is a list, then provide the list to the input.

A multi-instance step can also be nonblocking. As with any other step, the multi-instance step loses all its responses and gains a single response, the **done** response.

To make a step into a multi-instance step

1. Right-click the step and, in the context menu that appears, click Toggle Multi-instance.

The step changes to the following appearance (in which the multiple instances of the step are suggested by the layered outlines):



Multi-instance step

Note that the step now has the additional response **group done**. For more information on using the **group done** response, see the preceding topic.

2. Open the step inspector, and then open the input editor for the input to which you want to supply multiple values.
3. To specify that the input's value is a list, select the **The value of this input is a list** checkbox, and then, in the **List Delimiter** text box, type a delimiter (a character that separates the elements in the list).

By default the delimiter is a comma.

Important Notes:

- Your delimiter must accurately reflect the character or combination of characters that separates the list elements.

For instance, if your list includes a space between list elements, you must include the space in the delimiter that you specify. Consider the following list:

10.51.0.36, 10.51.0.37

If you specified only the comma (,) as the delimiter, the value for the input in one of the step's instances would be " 10.51.0.37", which could cause an error.

- Central users who schedule flows that contain multi-instance steps must know what the delimiter is, so be sure to let them know.
 - When you make a step into a multi-instance step, you cannot either add or remove specification of the step as a checkpoint.
4. Provide the input (or the flow variable from which the input gets its value) with a list that uses the same delimiter as that which you specified.
How you supply the list depends on how you obtain the list. For instance, you might get it from an integration with another program.
From within the flow, you can supply the list to the multi-instance step directly, in the **Value** text box (when you select **A Specific Value** for the input's assignment); or you can get the list from a flow variable.
 5. To revert the step to a single instance step, right-click the step and, in the context menu that appears, click **Toggle Multi-instance** again.

Moving data into and out of a multi-instance step

Values are passed to the multi-instance step as to any other step, with global flow variables of the parent merged into the global flow variables of the child runs (the runs of the instances). Then the multi-instance step's inputs become global flow variables of the child run.

Any flow variables that are created in an instance of the multi-instance step are local to the instance in which they are created and populated. Only global flow variables (created by scriptlets) continue after the instance's run completes.

You make data generated within the step available outside the step the same as you do for any other type of step: by creating a global flow variable in a scriptlet or (when the step is a subflow) by creating a flow output field.

Throttling a multi-instance step

A multi-instance step that launches 1000 concurrent processes could overwhelm many systems. To match the concurrent processing capability of a multi-instance step with the capacity of your system, you can limit the number of input values that the step processes at once.

To throttle a multi-instance step

1. Open the Inspector for the step, and click the **Advanced** tab.
2. Under **Execution**, select the **Throttle parallel execution...** check box.
3. In the text box, type the maximum number of instances of the step that should run at once.

Debugging a multi-instance step in a flow

There is an important difference between how a multi-instance step is executed as part of a flow run in Central and how it is executed in the Studio Debugger.

- In Central, the multiple instances of a multi-instance step run concurrently, and the flow continues with the steps that follow one instance's response while the other instances are processed.
- In the Debugger, the instances are processed serially.

For information on how to debug flows, see [Debugging a flow in Studio](#).

Making steps nonblocking

Suppose you want your flow to send a notification of some kind to one or more people after a certain outcome such as a failure response of a Network Diagnostic step. (If you notify more than one person, you might choose to convert this step into a multi-instance step as well as making it nonblocking.)

Your flow's subsequent actions might not depend on the outcome of this step, so it would be handy if the flow could continue to carry out this step while the step goes through the work of sending the notification(s).

Nonblocking steps are steps that complete while the flow run continues to carry out steps that come after the nonblocking step. Because the flow continues on, the outcome of the step must not have any impact on the path that the flow follows. Therefore, nonblocking steps have only a single response, **done**.

When you make a step nonblocking, it is automatically checkpointed. This means that when you restore a run in a flow in which there are no further checkpoints after the nonblocking step, the run is restored to the point just before the nonblocking step started. For more information on checkpoints, see [Checkpoints: Saving a flow run's progress for recovery](#).

To make a step nonblocking

1. With the flow open on the authoring canvas, right-click the step, and then click **Toggle Nonblocking**.

An orange lightning bolt appears on the step's leading icon, and the step automatically acquires a single response, **done**.



2. Connect the **done** response to the next step.

Note: When you add concurrent processing to a step, you cannot either add or remove specification of the step as a checkpoint.

Making a step single response

If all of a step's responses go to the same destination step and the transitions all have the same properties (such as the same description and ROI value), you can collapse all these responses into a single response (which you connect to the destination step).

To convert a step's responses to a single response

- With the flow diagram open, right-click the step and, in the drop-down menu, select **Toggle Single Response**.

To restore a single-response step's multiple responses

- With the flow diagram open, right-click the step and, in the drop-down menu, select **Toggle Single Response**.

Checkpoints: Saving a flow run's progress for recovery

A checkpoint is like a bookmark in a flow. During a run of the flow, each checkpoint gathers the state of the flow and saves it to the Central database. The flow's state comprises all the data necessary to completely reconstruct the run in case the flow is interrupted. If you have configured a failover/run recovery cluster for Central and the Central server running a flow fails, other nodes in the cluster resume the flow runs that were taking place on the failed Central server. If a flow has no checkpoints, the run is resumed at the start of the flow. But if there are checkpointed steps in the flow, the run is resumed from the last checkpoint that was reached when the server failed. By default, checkpoints are created for a step when you create the step.

Note: When a user pauses or interrupts a run, the state of the run at the time of the interruption is saved to the database, so checkpointing is not necessary for resuming the run at the point at which the user paused or interrupted it. Checkpointing is for resuming runs that were lost due to a system failure.

Because a checkpoint is set by default for each step that you create, the amount of information in the flow state can become quite large. Frequent writing of such large amounts of data to the database can affect the flow's performance, so you might want to reduce the number of checkpoints in the flow. For instance, you might remove checkpointing from some steps in loops. If you remove a checkpoint from a step, a run of the flow will be resumed from the last checkpoint before the one you removed.

Notes:

- If you remove a checkpoint from a step that was created from a subflow, none of the steps in the subflow (or the steps in the subflow's steps' subflows) are checkpointed, regardless of whether they have had checkpoints set for them or not.
- Steps with concurrent processing—that is, parallel split, multi-instance, and nonblocking steps—are checkpointed, and you cannot remove the checkpoint from the step. When you restore a run that was interrupted while a concurrent-processing step was running, the checkpoint means that the run is restored immediately prior to the step.

Also keep the following in mind about checkpointing and concurrent-processing steps:

- All the lane steps in a parallel split step are checkpointed, and you cannot remove the checkpoints.
- You can remove the checkpoint from the steps that process each of a multi-instance step's instances.

To create a checkpoint in a step

1. Open the flow diagram in Design view, and open the Inspector for the step to which you want to add the checkpoint.
2. On the **Advanced** tab, select the check box by **This step saves the whole run state**.

To remove a checkpoint from a step

1. Open the flow diagram in Design view, and open the Inspector for the step to which you want to add the checkpoint.
2. On the **Advanced** tab, remove the selection from the check box by **This step saves the whole run state**.

Scriptlets

When you have obtained data from an operation such as ping, traceroute, or the http operation get, you probably want to test, format, manipulate, or isolate a particular piece of the results. Or

perhaps you want to use a value from a flow variable to compare your results against. Using a scriptlet, you can perform these tasks and anything else that is possible with JavaScript or Perl.

Scriptlets are also uniquely suited to obtain various side effects subsequent to the operation's core actions and to otherwise extend the operation's capabilities.

A scriptlet is a Sleep or JavaScript (Rhino) script that is contained in an operation. Scriptlets provide more control than operation output filters and response, for evaluating output, determining results and responses, or adding variables to the flow.

You can use JavaScript or Perl scriptlets:

- In the body of an operation, step, or flow (on the **Scriptlet** tab of each).
- To filter a flow, step, or operation's results.
- As a rule for an operation's response.

For instance, in the Network Check flow, the TraceRoute operation counts how many network links, or hops, there are between the computer running the flow and the target computer. In addition being able to test the number of hops in the operation's rules to determine the operation's response, you could add the scriptlet that is used in the Evaluate Expression operation, which compares two values. With this script, you could test the number of hops with a threshold and pass the evaluation to another step or flow, for evaluation of the system's health or for communication to the user.

Tip: If you know how to script in either the two scripting languages used in HP OO, JavaScript (Rhino) or Sleep, you can use the HP OO scriptlet templates for those languages to learn the syntax and objects that HP OO requires for exchanging information with the scripting language. The scriptlet templates are available on the scriptlet tab of an operation's Properties sheet or step's Inspector.

Following are some considerations to remember when using a scriptlet:

- As with other uses of operations, you create a scriptlet on an **operation** for a generic use that you can apply over and over.
- You create a scriptlet on a step when the scriptlet is specialized to the conditions of that step. For instance, you might add a scriptlet to a step to evaluate and/or format data that is returned from the step's operation. If you then store the data in a flow variable and pass it to one of the fields in the flow's result and repeat this with other results, you can incrementally create a trail of the data that you've discovered.
- You might add a scriptlet to a flow (on the **Scriptlet** tab of the flow's **Properties** tab) when the flow will be used as a subflow within another flow and thus will function as an operation. After the subflow has obtained some data that you have passed to a field in the flow result, you might manipulate the data in the flow result field before passing it to the parent flow.
- HP OO automatically creates from each input a variable of the same name. Therefore, a scriptlet can access values by using variables with the same name as the inputs.

Creating a scriptlet

To view a scriptlet or the scriptlet template

1. On an operation's Properties sheet or step's Inspector, click the **Scriptlet** tab.
2. If the operation does not already contain a scriptlet, then to view the scriptlet template, click **Insert Template** and select the language in which you will write your script.

Tip: For examples of existing scriptlets, look in operations in the content that ships with HP OO (such as the operations under the Operations\Linux\Red Hat folder).

To filter step or flow results with a scriptlet

1. On the **Results** tab, open the filter editor of the result that you want to filter with a scriptlet by right-clicking the right-pointing arrow on the result's row.
2. In the filter editor, click **Add** to create a new filter.
3. In the **Select Filter** dialog, select **Scriptlet**.
4. Create the scriptlet, test the filter, and save your work.

To see the syntax for functions that help perform common tasks, click **Insert Template**. The scriptlet template provides commonly used functions and their syntax.

For more information on creating filters, see [Filtering outputs and results](#).

To create a scriptlet rule for an operation response

1. On the **Responses** tab of the operation's Properties sheet, click **Add** to add a response.

OR

Right-click the right-pointing arrow on the row of the response of interest.

2. In the **Rule Type** drop-down list, select **Scriptlet**.
3. Create the scriptlet.

To see the syntax for functions that help perform common tasks, click **Insert Template**. The scriptlet template provides commonly used functions and their syntax.

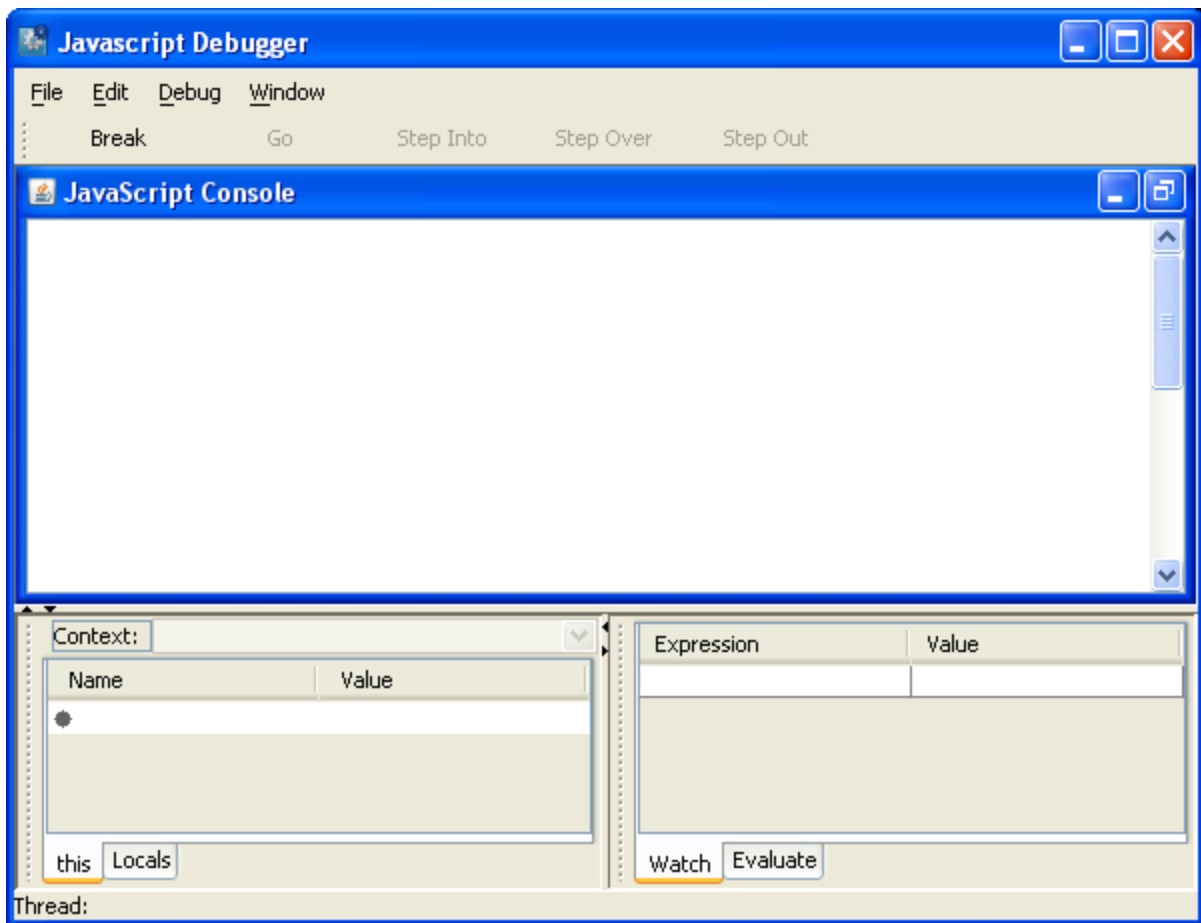
4. Save your work.

Debugging a scriptlet

To debug a scriptlet

1. Before starting to debug the flow, from the **Tools** menu, click **JavaScript Debugger**.

The JavaScript Debugger starts.



2. From the **Debug** menu, select **Break on Function Enter**.
All the scriptlets that the flow, its steps, and its operations contain loads into the Scriptlet debugger.
3. To set a breakpoint on one or more lines in a scriptlet, in the Javascript Debugger, click the line of interest.
A red dot appears beside the line.

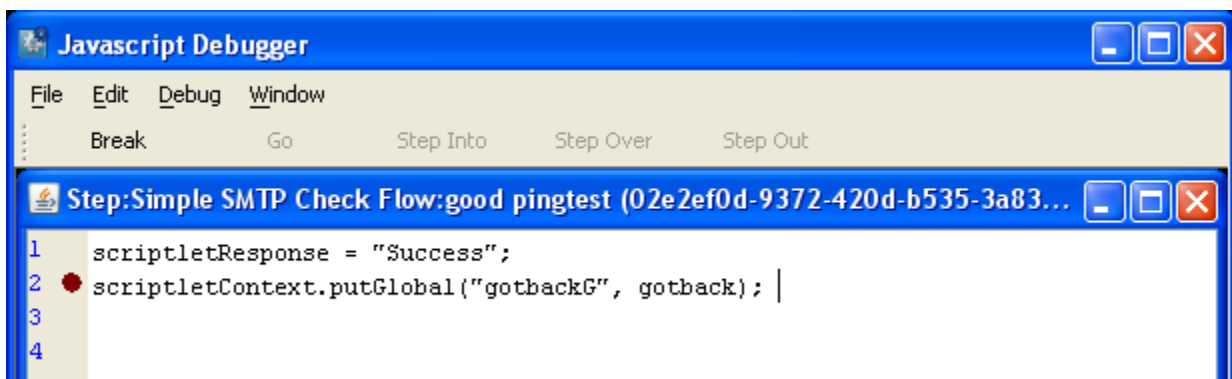



Figure 36 – Javascript Debugger breakpoint set for scriptlet line

Note: If you have created a breakpoint on a scriptlet line, when the Javascript Debugger is closed, it does not stop the script at the breakpoint.

4. In Studio, to set a breakpoint on the step, right-click the step, point to **Debugging**, and then click **Set Breakpoint**.
5. To start debugging the flow, in Studio, click the Execute Flow button ().

The Studio Debugger starts.

When the Studio Debugger reaches a step with a script in it, focus switches to the Javascript Debugger.


As the scriptlet runs, the current values of local and global flow variables in the context appear in the **Value** column of the **Context** panel.

When the Javascript Debugger reaches the breakpoint that you set, you can set a particular flow variable's value to watch.

6. To watch a particular flow variable, with the scriptlet paused, in the lower-right panel, type the name of the flow variable to watch in the **Expression** box.

Saving a scriptlet for use elsewhere

To save a scriptlet to the Configuration\Scriptlets folder

1. In the scriptlet editor, select the scriptlet, then click the Scriptlet icon () and drag to the **Configuration\Scriptlets** folder.
2. In the **Configuration\Scriptlets** folder, name the scriptlet.
3. Save your work.

To use a scriptlet from the Configuration folder

1. In the Library panel, open the Configuration\Scriptlets folder.
2. Drag the scriptlet you want to use from the folder into the scriptlet editor.
3. To change from a Configuration\Scriptlets scriptlet to one that you write yourself, click **Switch to a custom scriptlet**.

Outputs, responses, and step results

Outputs, responses, and step results are related to each other in flow authoring.

- Outputs are all or part of the output of an operation.
Remember that a flow is a kind of operation. This is particularly apparent when a step in a flow is associated with another flow. Thus flows also have outputs.
- Responses are the outcomes of evaluations of operation outputs.
For information on defining rules for testing results to determine responses, see [Responses: Evaluating results](#).
- Step results are analogs to operation outputs and obtain their content from outputs.
In addition, you can pass step results to the flow as fields in the flow result.

You can filter outputs and results in order to fine tune the data available for testing or for passing to another operation. For information on creating filters, see [Filtering outputs and results](#).

Operation outputs

Operation outputs used to be called results.

- The raw output is all of the operation's return code, data output, and error string.
The raw output is not directly visible in Studio, except as the raw result of a step that was created from the operation.

- The primary and other outputs are portions of the raw output—for example, success code, output string, error string, or failure message—that you specify as an output. You can narrow a primary or other output to a more highly focused selection by creating one or more filters for the output.

For more information on filtering outputs, see [Filtering outputs and results](#).

Besides evaluating outputs, you can pass them as data to other steps in the flow or to other flows by storing their values in flow variables. You can create filters to extract and modify parts of the result.

Note, however, that you cannot pass values to the global context except with a scriptlet. For information on using a scriptlet to pass values to the global context, see [Scriptlets](#).

Most operations have outputs that are specific to the operation. Even so, you will frequently encounter the following outputs when working with operations in the Library's **Accelerator Packs**, **Integrations**, and **Operations** folders:

- `returnResult`
The primary output of the operation. When you see "returns:" with no field named, this is usually the output. The primary output is also accessible via `Result` with a capital R (Which is universal).
- `response` (or `returnCode`)
A code or string used to determine the response the operation will take.
- `failureMessage`
An internal output provided by the infrastructure. If an operations returns a failure, this output provides the exception. Note that many operations do not use this output.

Responses

Responses are the outcomes of operations. In a step created from an operation, each response is the starting point from which a transition may be connected to another step (or back to the same step). Rules that evaluate the operation's output fields determine which of the several available responses will be the operation's actual response for the current run. Thus the outcome of the evaluation of one step's operation outcome determines which will be the next step in the flow run.

For information on creating responses and their rules, see [Responses: Evaluating results](#).

Step results

Step results are the step's analog to the outputs of the operation from which the step was created. You create the step results

You can pass step results as data to other steps in the flow or to other flows. You can create filters to extract and modify parts of the operation's output.

For example, suppose you only want the maximum, minimum, and average round-trip times for a ping operation to a certain server. You could extract all three pieces of information from the ping operation's raw results into several filtered results for the operation by filtering the raw results into three filtered results. Then you can use those filtered results to do the following:

- Create response rules (evaluators) that test one or more of the filtered data results to determine the next step of the flow.
- By passing the filtered data as values to flow variables, make them accessible to operations and transitions later in the flow, and through prompts that reference the flow variables, to the users of the flow.

- If the flow is a step in another flow (that is, a subflow of a parent flow), pass the data in the filtered results to fields in the flow's result, thus making the properties available to operations, steps, and transitions in the parent flow.


For more information on filtering outputs, see [Filtering outputs and results](#).

You can pass the step result's value to either:

- A local flow variable.
- A flow output field for the flow.

Adding and removing outputs

To add an output to an operation

1. To add a primary output, in the **Outputs** tab of the operation, from the **Extract Primary Result From Field** drop-down list box, select one of the operation's output fields.
2. To create a filter or a series of filters for shaping the result as needed, click **Edit Filters**.
The filter editor opens for you to create the filter or sequence of filters you need to isolate the data you need. For information on creating filters, see [Filtering outputs and results](#).
3. To add a secondary output (another output in addition to the primary output), click **Add Output** and then, in the dialog that appears, type the name of the output.
Note: You can change the output's name or source field by changing those values in the **Name** and **Output Field** columns of the output's row under **Available Outputs**.
4. To create filters for the output data in the secondary output, click the right-pointing arrow () at the end of the row.
The filter editor opens for you to create filters. For information on creating filters, see [Filtering outputs and results](#).
The **Filters** column in the output's row reports the filter that you have created or, if you've created more than one, how many filters you have created.
5. Save your work.

Note: Once you have created a primary output, you can change its source, but you cannot return to having no primary output.

To delete an output from an operation

1. On the **Outputs** tab of the operation, select the output you want to delete, and then click **Remove Output**.
2. Save your work.

Changing the source of an output

To change the field from which an output gets its data

1. To change the field for the primary output, click the downward-pointing arrow to the right of the **Extract Primary Output From Field** box, and then pick the desired field from the list.
OR
To change the field for a secondary output, click in the **Output Field** column of the output's row, and then pick the field you want from the list.
2. Create any filters you want for the output.


For information on creating filters, see [Filtering outputs and results](#).

3. Save your work.

Adding and removing step results

The step's raw and primary results come from the underlying operation's raw output and primary output. In the step's Inspector, you can create and specify secondary results.

To add a result to a step

1. To add a secondary result, open the step Inspector and, on the **Results** tab, click **Add Result**.
In the new row that appears in the list of results, the name of the result is by default named for the source of the result.
2. To make changes to the result's definition, click in the row in one of the following fields and do the following:
 - In the **Name** field, type the new name.
 - In the **From** field, from the list that drops down when you click the field, select the source for the result.
 - In the **Assign To** field:
 - To store the value in a flow variable, select **Flow Variable**.
 - To make the value available to a parent flow, select **Flow Output Field**.
 - From the **Assignment Action** list, select the appropriate action:
 - **OVERWRITE** – Replace the current value of the flow variable or flow output field with this value.
 - **PREPEND** – Place this value in front of the current value of the flow variable or flow output field.
 - **APPEND** - Place this value at the end of the current value of the flow variable or flow output field.
 - There are also four arithmetic assignment actions that you can select: **ADD**, **SUB**, **MULTIPLY**, and **DIVIDE**. These specify that you use this value to arithmetically modify the current value of the flow variable or flow output field. For example, if the step result were 3.14, and you selected **MULTIPLY**, effect would be to multiply the current value of the flow variable or flow output field by 3.14. Or suppose you have a timeout flow variable. If you wanted to decrement the value of the timeout flow variable by 1, you could specify that the result (assuming its value is 1) be subtracted from "timeout" by choosing **SUB** as the **Assignment Action**.
3. To create filters for the output data in the secondary output, click the right-pointing arrow () at the end of the row.
The filter editor opens for you to create filters. For information on creating filters, see [Filtering outputs and results](#).
4. After creating the filter or filters you need, close the filter editor.
The **Filters** column in the output's row reports the filter that you have created or, if you've created more than one, how many filters you have created.
5. Save your work.

To delete a result from a step

1. On the **Results** tab of the operation, select the result you want to delete, and then click **Remove Result**.
2. Save your work.

Changing the source of a result

To change the field from which a result gets its data

1. To change the field from which a secondary result gets its data, click in the **From** column of the output's row, and then pick the field you want from the list.
2. Save your work.

Filtering outputs and results

If you're diagnosing a network connectivity problem, your goal could be to extract for use elsewhere, either in or outside of the flow, pieces of data such as the maximum, minimum, and average round-trip times, or the percent of packets that were lost.

Suppose you only want the maximum, minimum, and average round-trip times for a ping operation to a certain server. You could isolate and extract all three pieces of information from the ping operation's raw output by filtering the raw output into three outputs. Then you can use those outputs to do the following:

- Create response rules (evaluators) that test data in the outputs to determine the next step of the flow.
Note: You can create filters in the response rule, too.
- By passing the filtered data as values to flow variables, make them accessible to operations later in the flow, and through prompts and transition descriptions that reference the flow variables, to the users of the flow.
- If the flow is a step in another flow (that is, a subflow of a parent flow), pass the data in the filtered outputs to fields in the flow's result, thus making the properties available to operations, steps, and transitions in the parent flow.

Note: To work with a ping operation, you can either create your own ping operation or copy the existing Ping operation and work with the copy. If you are working with a copy of the existing Ping operation, you'll note that some results that are available by default extract the data that you would otherwise isolate with filters.


Creating a filter

Frequently, you will create a series of filters to extract exactly the data that you want.


To filter an operation output or step result

1. To create one or more filters for an operation's primary output, with the operation's **Properties** sheet open on the **Outputs** tab, click **Edit Filters**.

OR

To create filters for one of the operation's secondary outputs, at the right end of the output's row under Available Outputs, click the right-pointing arrow (.

OR

To create filters for a step result, open the step's Inspector and, on the **Results** tab, at the right end of the result's row, click the right-pointing arrow (.

The filter editor appears.

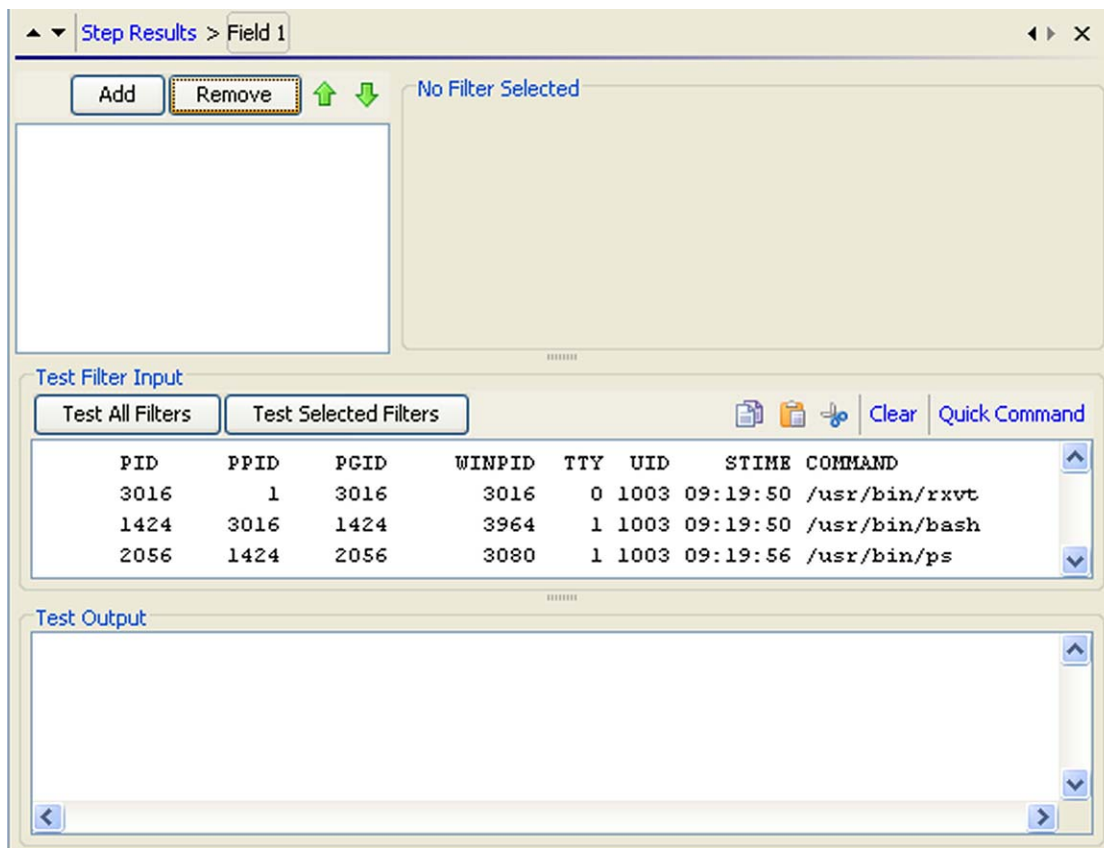


Figure 37 - Filter Editor

- The upper-left box builds a list of the filters as you create them.
 - When you create a filter and have selected a filter type, the upper-right box (labeled **No Filter Selected** in the above screen shot) contains controls for modifying filters depending on the kind of filter you select.
2. To add a filter, click **Add**, and then select the type of filter from the drop-down list in the **Select Filter** box that appears.
 3. Define the filter's particulars in the **Details** for area at the upper-right of the filter editor. When you create multiple filters for a result, they appear in a list.

Tip: While you have the filter editor open, you can click the upward- or downward-pointing arrows (▲ ▼) to change which result you're creating filters for.

Although filters are frequently applied to operation outputs or step results, when you filter an output's or result's data, the data is the filter's input. Thus, in the filter editor, you put your test data in the **Test Filter Input** box.

4. To obtain realistic data for testing, click **Clear** to empty the **Test Filter Input** box, and then do one of the following:
 - If the data can be generated by a local command-line command, click **Quick Command** and then, in the text box that appears, type a command that generates the desired data.
 - If the data is produced by means that you cannot reproduce with a simple command-line command, you can:
 - a. Run the flow in the Debugger.
 - b. Highlight the relevant step.
 - c. In the **Step Result Inspector**, copy the contents of the **Raw Result** tab.
 - d. In the filter editor, paste the contents into the **Test Filter Input** box.

5. In the filter editor, click **Test All Filters**.

OR

Select the filters you want to test and click **Test Selected Filters**.

The filters are applied (in top-to-bottom order) to the data in the **Test Filter Input** box, and the filtered results appear in the **Test Output** box.

For specifics on the kinds of filters you can create and on saving filters, see [Filter details](#).

Filter details

The following are the kinds of filters you can create and how you define particulars of each kind. For information on creating filters, see [Creating a filter](#).

In the following descriptions, remember that the data on which the filter operates is the filter's input, even though the data may have come from an operation output or step result. Thus, in the filter editor, you put your test data in the **Test Filter Input** box.

- Diff Case

A Diff Case filter changes all the characters in the string either to upper case or to lower case. If you leave the **To Upper Case** check box unchecked, the filter changes all the characters to lower case.

- Extract Number

An Extract Number filter extracts the first number found in the result. The filter treats an unbroken series of integers as a single number. For instance, the Extract Number filter would extract the number "123" from the strings "123Test" or "Test123".

- Format

A Format filter attaches text to a result or output or replaces the original content with the text.

- In the **Text** box, type the text you want to attach to the result or use to replace the result.
- In the **Place Input At** list, select **Beginning** or **End** respectively to prepend or append the text

OR

To replace the output with the text, select **Replace**.

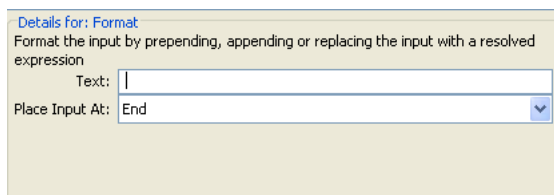


Figure 38 – Format filter definition

- Line Count

A Line Count filter outputs the total number of lines of the result. There are no settings to edit for this filter.

- Regular Expression

Filters the raw results using a regular expression (regex). For more information on regular expressions, see [Working with regular expressions](#).

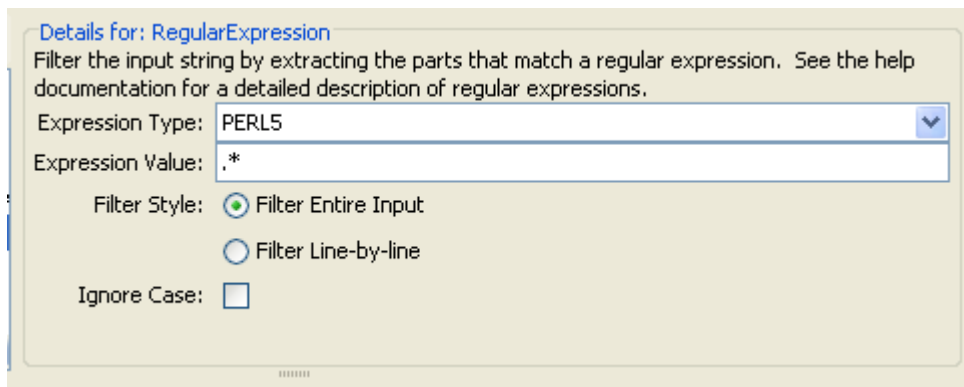


Figure 39 - Regular Expression filter definition

- In **Expression type**, select the type of regex to apply.
- In **Expression value**, type the regex.
- Select **Filter Entire Input** or **Filter line-by-line**, according to how you want the filter applied to the raw results.
- To make the regex not case-sensitive, select **Ignore case**.
- Remove Duplicate Lines
Finds lines that are identical and removes all but one of them.
 - To apply the filter only to duplicate lines that follow each other directly, select **Consecutive**.
- Replace
Replaces the first or last instance or all instances of one string with another string.

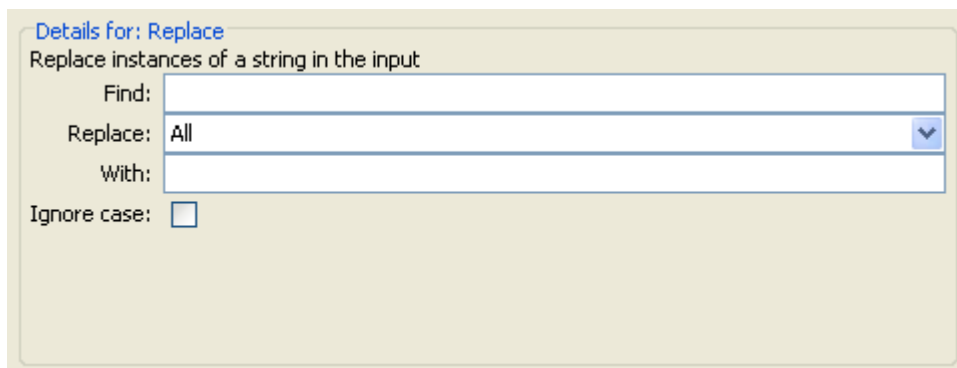


Figure 40 - Replace All filter definition

- In the **Find** box, type the target string (the string to search for and replace).
- From the **Replace** drop-down list, select **First**, **All**, or **Last**, depending on which instances of the target string you want to replace.
- In the **With** box, type the string to replace the target string with.
- To make the search not case-sensitive, select the **Ignore case** checkbox.
- Round Number
Rounds numbers as you specify in the filter editor.

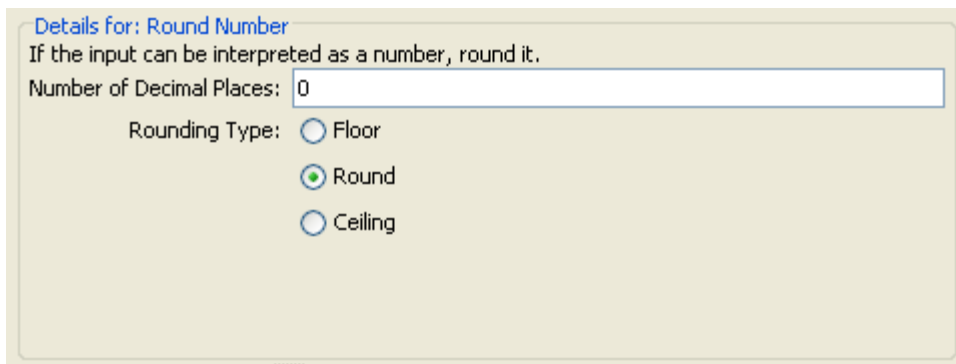


Figure 41 - Round Number filter definition

- To specify the accuracy of the rounding, type the number of decimal places the number should be rounded to in the **Number of Decimal Places**.
- Select either **Floor**, **Round**, or **Ceiling**, to specify how the number should be rounded: **Floor** always rounds the number down and **Ceiling** always rounds the number up. **Round** rounds the number up if the last meaningful place is 5 or more, and down otherwise.

- **Scriptlet**

Filters data with a scriptlet that you create, with the same sort of scripting environment that you use when creating a scriptlet for an operation or step.

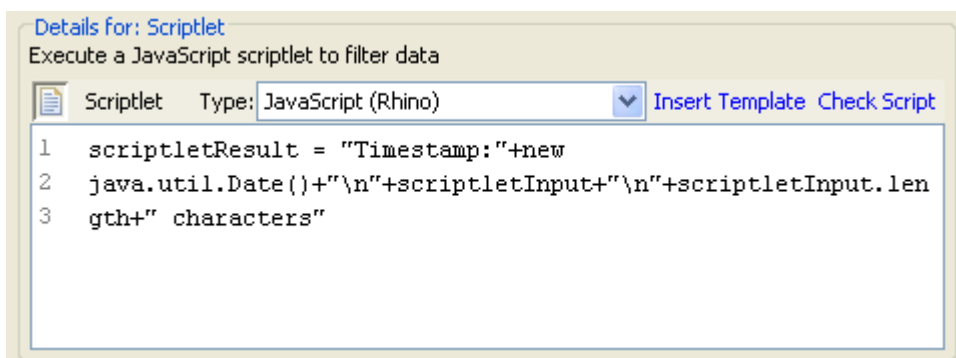
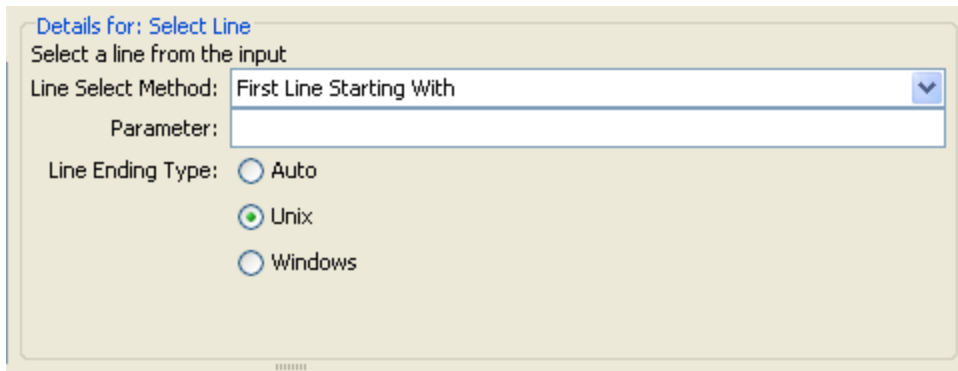


Figure 42 - Scriptlet filter definition

- From the **Type** drop-down list, select the JavaScript (Rhino) or the Sleep scriptlet language. If you choose JavaScript (Rhino) as the type of script, the text box starts you out.
- To get started with lines that you will need for the scriptlet to work as a filter, click **Insert Template**.
The template that is inserted is specific to the language that you chose and includes the most commonly used commands for accessing flow variables (whose values are *context data*), operation results, and inputs and setting and manipulating flow variable values and results.
- To debug the script, click **Check Script**.
- **Select Line**
Defines a line that you want to extract from the raw results.



Details for: Select Line

Select a line from the input

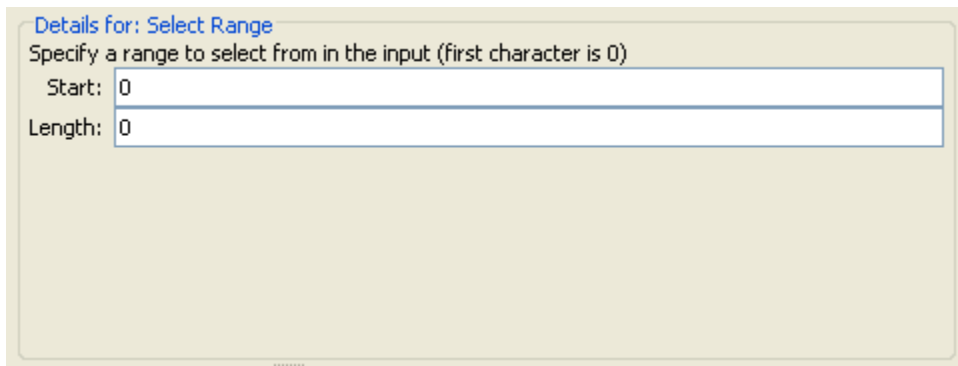
Line Select Method: First Line Starting With

Parameter:

Line Ending Type: Auto
 Unix
 Windows

Figure 43 - Select Line filter definition

- From the **Line Select Method** list, select a criterion for the line that you're interested in.
- In the **Parameter** text box, type a string that the string contains.
- From the **Line Ending Type** group, select the type of line ending according to whether the text that you're filtering was generated on a Unix (which ends lines with LF) or Windows (which ends lines with CR/LF). **Auto**, the default selection, accepts both Unix and Windows type line endings.
- **Select Range**
 Defines a string that you want to extract from the input data. The two criteria for defining the string are: length in characters and position of the first character from the start of the input data.



Details for: Select Range

Specify a range to select from in the input (first character is 0)

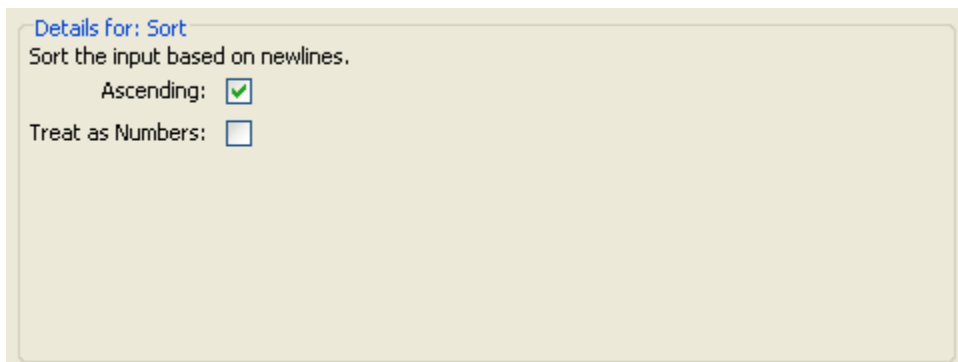
Start: 0

Length: 0

Figure 44 - Select Range filter definition

In the **start** and **length** boxes, type the zero-based start position and the character length of the string that you want extract from the raw results.

- **Sort**
 Sorts the input data by line.



Details for: Sort

Sort the input based on newlines.

Ascending:

Treat as Numbers:

Figure 45 - Sort filter definition

Specify the direction of the sort.

If you choose to treat the data as numbers, the lines that do not begin with numbers are sorted to the bottom.

- Strip

Strips characters from the raw results.

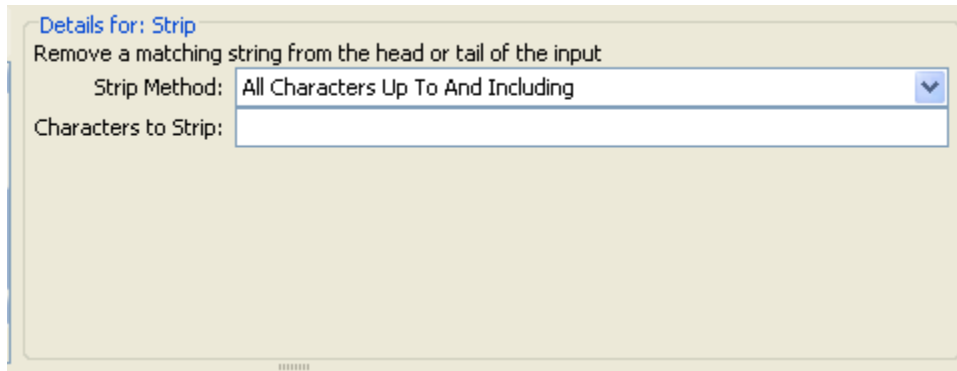


Figure 46 - Strip filter definition

From the **Strip Method** list, select how you want the filter to strip the raw results. You can specify stripping all characters up to or up to and including, or all characters after, or after and including the string that you specify in the **Characters to strip** text box.

In the **Characters to strip** text box, type the string to find.

- Strip Whitespace

Removes all the whitespace characters from the front and the end of the raw results. There are no settings to specify for this filter.

- Table

A Table filter does not convert the raw results into a table, but enables you to manipulate the raw results as if they were a table, including sorting columns and selecting columns, rows, and blocks.

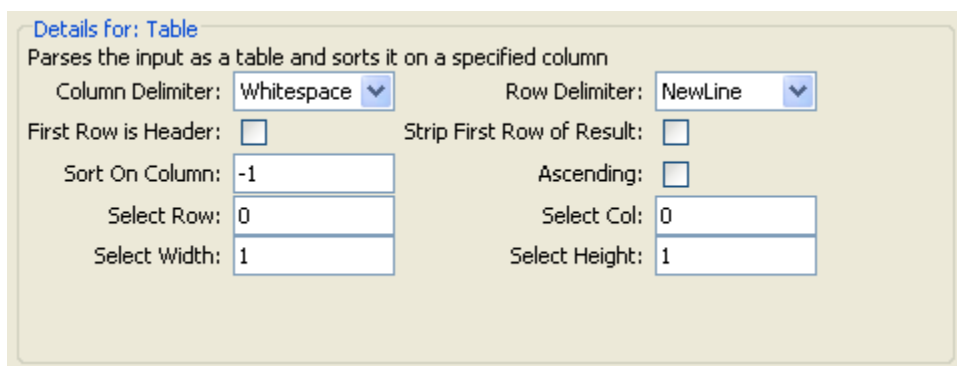


Figure 47 - Table Sort filter definition

Note: Row numbering is 0-based, and column numbering is 1-based.

- In the **Column Delimiter** list, choose the character that will serve to divide the data into columns in a meaningful way.
- In the **Row Delimiter** list, choose the character that will serve to divide the data into rows in a meaningful way.

Note: Two or more consecutive white spaces count as a single white space, so a column may be occupied by data that you expected to find in a column to the right. For example, this behavior will appear if you apply this filter to the output of a “dir” command-line command with whitespace specified as the column delimiter.

- To treat the members of the first row as column headers, select **First Row is Header**.
- To remove the first row, select **Strip First Row of Result**.
- To sort on a column, type the column number in the **Sort On Column** box. Column numbering is 0-based.
- To specify ascending order, select the **Ascending** box. By default, the sort order is descending.
- To select a row you want the filter to extract, type the row number in **Select Row** and in **Select Width** type the number of columns in that row that you want extracted. Remember that row numbering starts with 0.
- To select a column you want the filter to extract, type the column number in **Select Col** and in **Select Height** type the number of rows in that column that you want extracted. Remember that column numbering starts with 1.

For example, to extract the first 5 rows of the 2nd through 4th columns, you would specify the following. In these settings, the first two settings define the rows selected, and the second two settings define the rows selected.

- In **Select Row**: 0
- In **Select Height**: 5
- In **Select Col**: 2
- In **Select Width**: 3

Saving and re-using filters

You might want to save filters that you create for one operation's result and re-use them in another operation. For instance, the filters in RAS Ping might be useful for other ping operations. You can save them as system filters, which are stored in the **Configuration\System Filters** folder.

Note that by saving a scriptlet filter as a system filter, you can save a scriptlet specifically for use in a filter.

To save a filter in the System Filters folder

1. Open the appropriate operation and, in the filter editor, select the filter you want to save.
2. In the **Repository** pane, expand the **Configuration** folder.
3. Drag the filter from the operation's filter editor to the **System Filters** folder.
4. To rename the new system filter, right-click it, click **Rename**, and give it a more descriptive name.
5. Save your work.

To use a filter in the System Filters folder

1. Open the editor in which you want to use the system filter.
2. In the Library, open the **Configuration\System Filters** folder.
3. Drag the filter that you want to use from the folder to the filter list box in the result editor.

Responses: Evaluating results

A response is one of an operation's possible outcomes. For example, an operation that runs an SQL query against a database should distinguish between its outcomes like this:

- **Failure** response if the database isn't running or can't be reached
- **No rows returned** response if the query ran but no data was returned

- **Rows returned** response if the query successfully retrieved data.

A particular response is selected when a rule or set of rules that describes a particular condition of the operation's result is true. A rule compares a value that you specify with a value in a field of an operation's raw results:

Response	Default	On-Fail	Response T...	Rules
port open	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	2 Rules.
port listening	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	1 Rule [Source: returnCode, No Filters, No Filters]
host not found	<input checked="" type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	1 Rule [Source: returnCode, No Filters, No Filters]

The above screenshot illustrates several faces of rules that you will want to remember:

- The response types are:
 - Success, or resolved: ✓
 - Diagnosed: +
 - No action: -
 - Failure: ✗
- If you create more than one rule for a response (such as the **port open** response), then all the rules for that response must evaluate to true for the response to be chosen.
- Responses are evaluated in the order in which they are listed on the operation's **Responses** tab. The first response whose rule or rules evaluate to true is the response chosen. So if the **port open** response's rules evaluate to true, then that is the response chosen, even if the rule for **port listening** would also evaluate to true. The order of responses can be very important for obtaining the most helpful outcome for your flow.
- If you specify a default response, that is the response chosen if none of the responses' rules evaluate to true.

To create a response

1. On the **Responses** tab of the operation, click **Add Response**.
2. To make a response the one chosen if an operation fails to execute, select the response's check box in the **On-Fail** column.
3. To create a rule for the response, at the right end of the response's row, click the right-pointing arrow (➡).
4. In the response rule editor, click **Add**.
5. In the **Apply Rule to Field** column, select the results field that has the content you want to test a rule against.
6. In the **Rule Type** column, pick the comparison or match that you want the rule to test.
7. In the **Rule Text** column, type the text to use in the test.

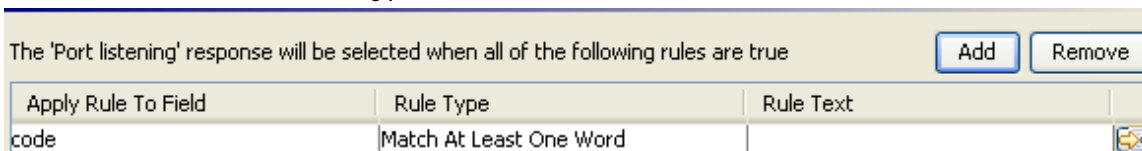



Figure 48 - Defining a rule

Using the rule editor, you can make the changes described above, or:

- Filter the operation result before applying the rule.
- Test the rule.
- Drag system evaluators (rules), filters, or a scriptlet into the rule.

Note: In a rule, when you use a mathematical comparator (such as =, !=, <, >) in an evaluation of a string that starts with a number, the comparator compares only the numerical portion of the string. For example, if you compare "123" with "123Test" using != (does not

equal), the evaluation would be “false”, although “123” is clearly not the same as “123Test”. You can work around this issue, however, by comparing the strings with the Not Exact Match evaluator.

8. To open the rule editor, at the right end of the rule’s row, click the right-pointing arrow ().

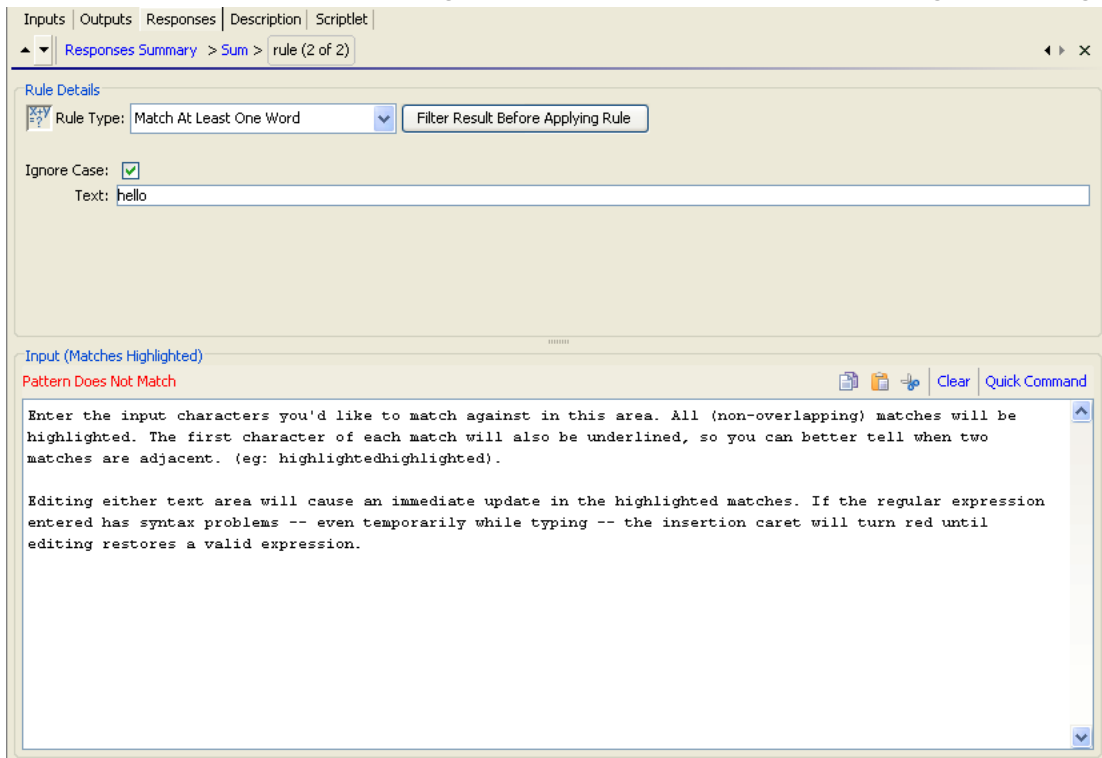


Figure 49 - Rule editor

Note: If you choose **Scriptlet** as the rule type, the rule editor changes to a scriptlet editor, as you use elsewhere. For more detailed information on creating and using scriptlets (including saving scriptlets for re-use), see [Scriptlets](#). For information on creating scriptlet filters, see [Filtering an operation’s results](#).

Now you can test the rule and/or create a filter for filtering the results in the field before applying the rule.

9. To test or refine the rule, click the right-pointing arrow to open **Rule Details**.
The rule type that you chose in the rule editor appears in **Rule Details** also, already selected in the **Rule Type** drop-down list.
10. In **Rule Details**, to choose another rule type, select a different one in the drop-down list.
11. For most of the rule types, in the **Text** box, type the text that you want to test comparison with and, if you want to ignore case, select the **Ignore Case** check box.

OR

For Regular Expression rules, specify the regular expression and its application as you do when creating a Regular Expression filter for operation results. For information on creating Regular Expression filters, see [Filtering outputs and results](#), and for details on filter types, see [Filter details](#).

12. To work on a rule for another of the operation’s responses, click the up or down arrow beside **Responses Summary**.

Tip: Response rules are evaluated in the order in which they appear in the operation’s **Responses** tab. When you run a flow, the response of the first rule that evaluates to true is selected as the

response of the step. So the order you provide to the responses can determine whether users obtain the desired results from the flow.

Adding responses to the flow

Flow responses are the possible outcomes of the flow run—such as, whether the system that the flow appraised needed fixing or whether the action was successful.

For example, the Restart Service – Tutorial Flow has three responses. To see how the following responses are obtained, in Studio, find and double-click the Restart Service – Tutorial Flow in the Library.

- Failure—The flow was unable either to get a list of services, or the service that it was running against was stopped and the flow could not restart it.
- Service Already Running—The flow found the service and tried to restart it, but the service was already running.
- Started Service—The flow successfully started the service.

To add a response to the flow

1. On the flow **Properties** sheet (with the flow diagram open, click the Properties tab at the bottom of the window), click **Responses**.
2. Click **Add Response** and then, in the text box that appears, type the name of the response.
Tip: To delete a flow response, on the **Outputs** tab, click the response you want to delete and then click **Remove Response**.
3. To close the **Properties** sheet and save changes, click **OK** and then press CTL+S.

Changing a step's icon

You can change a step's icon to one that gives you a clearer visual cue to what the step does.

To change a step's icon

1. To open the **Icons** panel, click the **Icons** tab on the right side of the flow canvas.

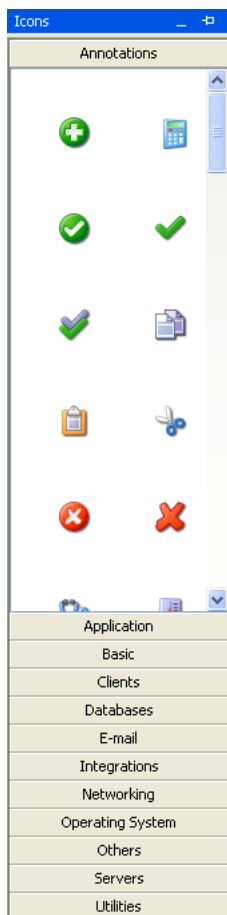


Figure 50 - Icons panel

- In the **Icons** panel click the subpanel name that contains the icon you want, find the icon, and drag it onto the step.

Flow Design

Many flow design issues are those that you will conclude on your own. However, those included in this section are important to keep in mind.

- Do not create flows that create unlimited growth in memory.
For example, a flow that runs an infinite loop, in which the flow sleeps, performs some tasks, then goes back to sleep. In this case, the Run History grows until the system runs out of memory.
- You can *use flows to simplify flow design*, thus making it possible to:
 - Separate the programming tasks into smaller, more manageable pieces.
 - Test parts of the flow individually.
 - Reuse the pieces that you create.

Using subflows to simplify flow design

You can simplify a flow by creating steps from subflows. This way, you can:

- Separate the programming tasks into smaller, more manageable pieces.

- Test parts of the flow individually.
- Reuse the pieces that you create.

For example, suppose you want to copy a set of files from one server share to another. You could:

1. Create a step from the Iterator operation, providing something list inputs such as the following for the Iterator step (and specifying a semicolon [;] as the separator between the members of the list).

```
\\server1\share1\fileAAA.zip,\\server2\share2\fileAAA.zip;
\\server1\share1\fileBBB.zip,\\server2\share2\fileBBB.zip;
```

On the first iteration, the Iterator step extracts

```
\\server1\share1\fileAAA.zip,\\server2\share2\fileAAA.zip.
```

2. To use `\\server1\share1\fileAAA.zip` as the source pathname and `\\server2\share2\fileAAA.zip` as the destination pathname, you create two ListItemGrabber steps, one to extract the source pathname and one to extract the destination pathname.

Alternatively, you could simplify the flow and create a reusable piece by doing the following:

3. Create a flow that consists only of the two ListItemGrabber steps and the success and failure return steps.
4. Drag the flow onto the parent flow, to create a step from the subflow.

In this case, you need to enable the parent flow to use data that have been created or modified by the subflow. In this example, a copy step in the parent flow needs each source pathname and destination pathname. For the technique for passing information from a subflow to its parent flow, see [Passing data from a subflow to a parent flow](#).

Passing data from a subflow to a parent flow

Subflows often generate data that steps in the parent flow need to access. Flow variables that you create within a flow cannot be referenced outside that flow. However, you can pass values outside the flow to a parent flow by assigning a step result to a flow output field of the subflow.

To make data from a step in the subflow available to steps in the parent flow

1. On the subflow's authoring canvas, open the Inspector for the step whose data you want to make available to the parent flow.
For information on opening the Inspector for a step, see REF TK.
2. Add a result, and, in the row that appears for configuring the result, do the following:
 - Under **Assign To**, select **Flow Output Field**.
This stores the value in an output field for the subflow, which makes the data available outside of the subflow.
 - Under **Name**, type a name for the flow output field.
 - Under **From**, select **Result Field:Result**.
3. In the parent flow's authoring canvas, open the Inspector for the step that was created from the subflow.
4. Create a step result, using the procedure described in [Step results](#).
5. In the new step result's row, under **Assign To**, select **Flow Output Field**.

Changing which operation a step is based on

To switch the operation that the step is based on

Suppose that you discover that you need a different operation for some step in your flow, but you want to keep the existing transitions to and from that step. The step Inspector helps you accomplish this.

1. In the flow's design view, open the step's Inspector.

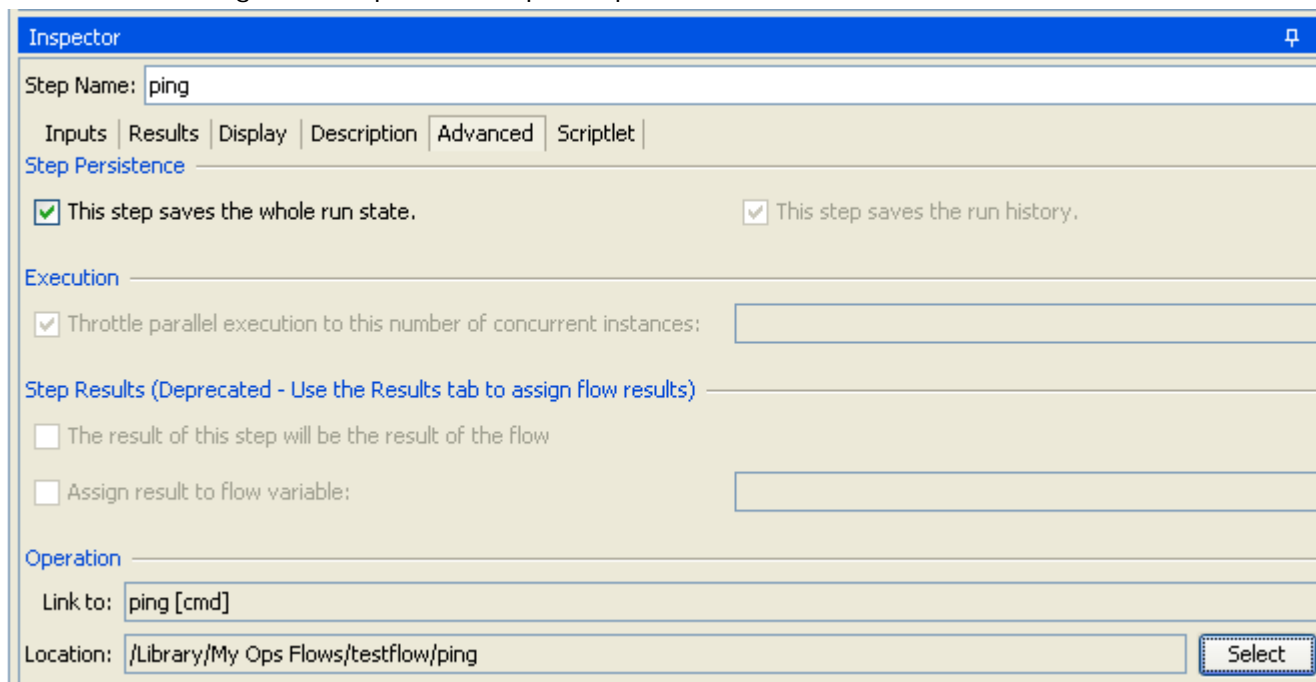


Figure 51 - Step Inspector Advanced tab

2. On the **Advanced** tab, under **Operation**, click **Select**.
The **Select Operation** dialog box appears.



Figure 52 - Select Operation box

3. Navigate to and select the operation that you want to base the step on, and then click **OK**.
Note: The step's name is unaffected by your having changed the underlying operation, so if the step was named for the previous operation, it may look as if the step had not changed. However, the **Advanced** tab, under **Operation**, reflects your change.
4. Rename the step to reflect the change in operations.
5. Save your work.

Step descriptions

A description that includes likely search words helps you or others find your flow. Suppose, for example, you modified a copy of the Network Check flow. The description "Performs a ping and a traceroute to the server," a Search tab search with the following:

```
description:ping traceroute
```

would find any flows and operations that have either or both of the words "ping" and "traceroute" in the description of the flow or one of its steps.

To create a description for the step or flow

1. On the flow's Properties sheet or step's Inspector, click the **Description** tab and type a description of the step.
2. Click **OK**, and then save your changes.

Running flows automatically

When a flow can run automatically, Central users can schedule the flow's runs, and the flow can be started from a URL. Running automatically means that a flow does not require human intervention to start or complete, which means that it cannot have any inputs that get data from user prompts. More, if the flow's inputs get their data from flow variables, the Central user who schedules the runs can control the data that are assigned to the inputs for each run.

So, to enable a flow to run automatically:

- For any inputs that get their data from user prompts, change the data source to **A Specific Value**.
- To enable a Central user to assign inputs different data on different runs, create a flow variable and reference the flow variable for the value in the input's **A Specific Value** data source assignment.

For information on changing data sources for inputs, see [Inputs: Providing data to operations](#).

Debugging a flow in Studio

The Debugger in Studio enables you to find the exact causes of errors and unexpected behaviors in flows that you test there. It does so by displaying the following information:

- A tree showing the steps executed
- Step results and operation outputs generated for each step
- Flow variable values in the various contexts current to each step
- The transition description for each transition followed

You can also set breakpoints for the Debugger and force response choices in order to zero in on the behavior you want to test.

All of this capability focuses on fixing broken flows and on the types of data described. To learn particularly how a flow that has steps that use parallel processing will behave in Central, there is no substitute for running the flow in Central in a staging environment after testing the flow in the Studio Debugger.



Key information: Parallel processing in a flow (processing carried out in one or more nonblocking, multi-instance, or parallel split steps) that takes place in parallel in Central, runs serially in the Studio Debugger.

- In a parallel split step, the lanes always execute serially. In Central, they all begin at the same time, and the order in which they finish depends on variable factors that cannot be predicted in Studio. Thus the Debugger cannot predict considerations such as in the case of conflicting writes to the same flow variable, which lane writes to the flow variable last. On the other hand, in Studio, you can manipulate the order in which the lanes will finish in the Debugger in order to test, in a controlled fashion, various scenarios.
- In a multi-instance step, the instances are executed serially. While this means that you are not testing under actual conditions, it does allow you to examine how long it takes each instance to finish.
- Nonblocking steps do not behave as nonblocking steps in the Debugger. That is, in the Debugger, steps that follow a nonblocking step do not execute until the nonblocking step has completed.

Debugging a flow


Best practice is to debug subflows before debugging the parent flows.

When debugging a flow you might find the keyboard shortcuts useful. For the Debugger's keyboard shortcuts, see [Debugger keyboard shortcuts](#).

To debug a flow in Studio


1. In the Library, right-click the flow, and then click **Debug**.

OR

Open the diagram of the flow you want to debug and then, in the authoring pane, click the **Debug Flow** button ()

The Studio debugger opens.

You can either play the flow from start to finish or one step at a time. When you play the flow from start to finish, the run is interrupted only by any breakpoints that you may have set.

2. To run the flow to its end, click the **play** icon () or press F11.

OR

To run the flow step by step, click the **step over** icon () or press F6 for each step.

Let's look at what we can learn from the Debugger with a flow that pings three IP addresses (using a multi-instance ping step).

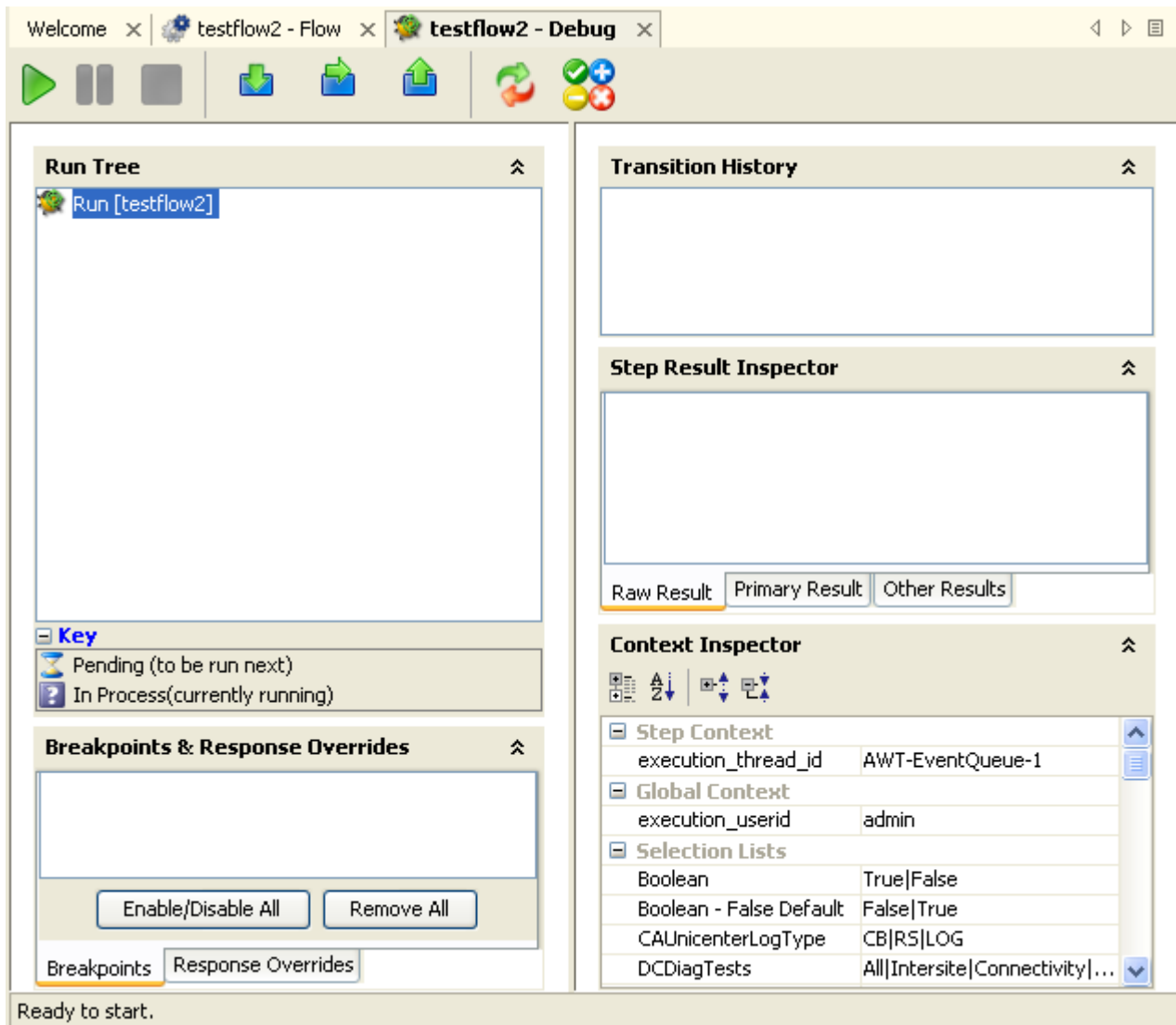
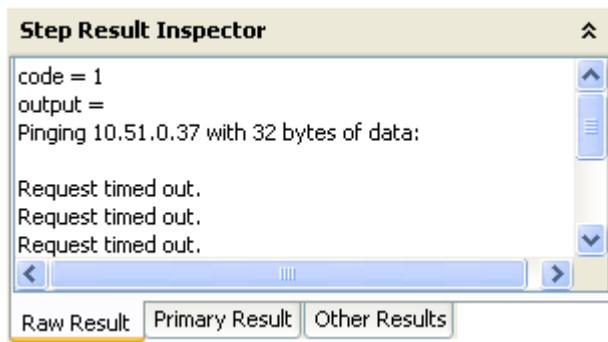
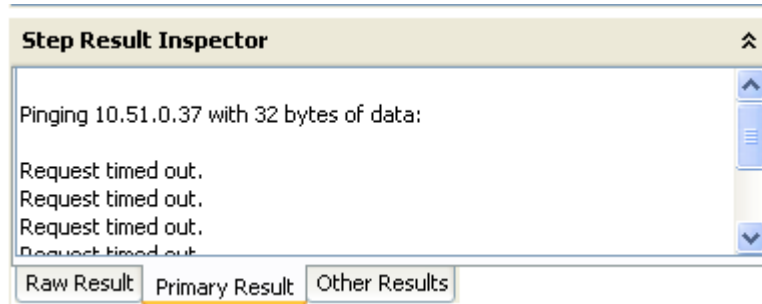


Figure 53 - Flow Debugger mid-flow

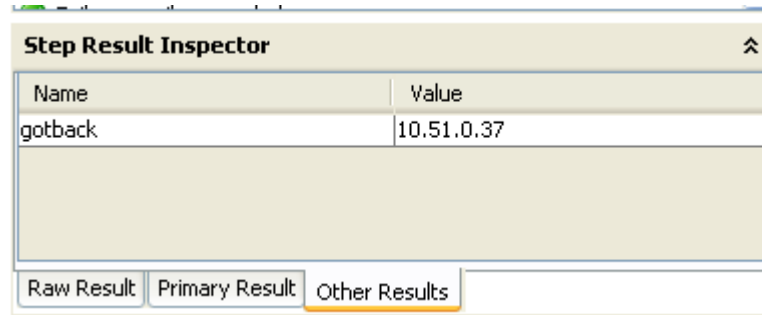
- **Run Tree** shows each step that runs, including steps in subflows of the flow. Steps that will run simultaneously in Central are run in a serial sequence in the Debugger.
 - **Breakpoints & Response Overrides** Breakpoints are flags that enable you to automatically pause a run at a certain step in order to examine the results, the path of the run, or the values in the flow variables at a that point. Response overrides force the response that you selected, regardless of the response chosen by evaluation of the relevant operation's result. This panel lists those elements and enables you to remove them or enable or disable them for this run.
 - **Transition History** lists the transitions that have been followed in the run and displays their descriptions (thus rewarding the best practice of providing a description for each transition).
 - **Step Result Inspector** shows the raw results (the results of the step's operation) and the filtered results of the step.
 - **Context Inspector** displays the values that are current to flow variables (global as well as local) for each step.
3. To see the information in each of these panels for a completed step, click the step in **Run Tree**.
 4. In the **Step Result Inspector**:
 - To see the raw result of the step, click **Raw Result**:



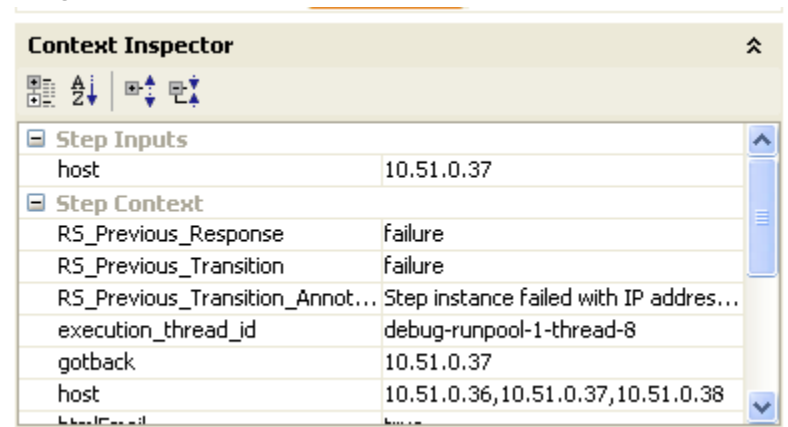
- To see the primary result, click Primary Result:



- To see other results that you may have created, Other Results:



- To see the flow variables and their values for the step's inputs and the step and global context, navigate to the appropriate section of the Context Inspector.



In the **Context Inspector**:

- Step inputs' values are the values that have been assigned to the input before the step started.
- The values in the **Step Context** section are the values that are updated after the step has begun.

A step's context is the collection of flow variables and their value assignments in the local contexts of the step's parent flow (and, if the step's parent flow is a subflow, that flow's parent flow).

For this step (one of the multi-instance ping step's instances), the **Step Inputs** include the **host** flow variable for the step, and the step context includes the **gotback** flow variable and the **host** flow variable from the context of this instance of the multi-instance step.

The text boxes that contain the values for flow variables are color coded, as in the following example.

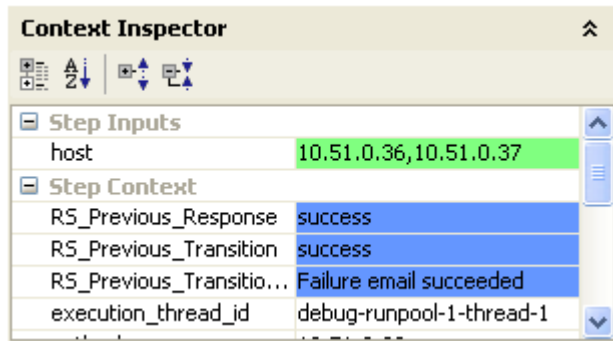





Figure 54 - Color coding of flow variable values

- Blue signifies a value that has been changed by or within the run.
- Green highlights a new variable.

To step into and out of a subflow

1. To step into a step's subflow, click the **step-into** icon () or press F5.
2. To step out of the subflow, click the **step-out** icon () or press F7.

To reset and restart the flow in the Debugger


- In the toolbar, click the **reset-flow** icon () or press F12.
Before the flow is restarted, the values of its flow variables are reset to the values that they had when you opened the Debugger.


Changing values of flow variables within the Debugger

If you want to see how a flow behaves with different values for its flow variables, you can change values for the flow variable before running the step.

To change a flow variable during a flow run in Debugger

1. Open the flow in the Debugger.
2. To reach the step at which you want to change the flow variable, do one of the following:

- Click **step over** () or F6) until the step in which you're interested is pending.

- If you have a breakpoint set on the step, click **play** ()

The run pauses with the step for which you set the breakpoint pending. For information on setting breakpoints, see [Working with breakpoints](#).

The Context Inspector shows the current values of the Step Inputs and Step Context as of the point at which the step is pending.

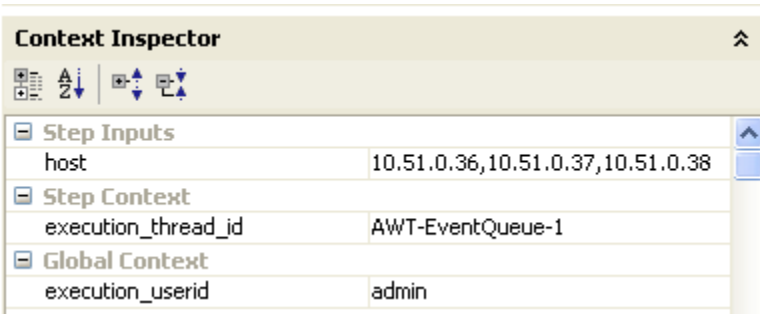


Figure 55 - The Debugger Context Inspector

Now you can:

- Change values used in the execution of this step.
 - Change values used in a later step.
3. To change the value of a flow variable used in this step, the flow variable must provide the value in a step input.

The flow variable must therefore be listed under **Step Inputs**, so you change the value for the flow variable there. In the example shown here, the step is a multi-instance step. You could add another IP address to the list in the host flow variable.


OR

To change the value of a flow variable that is accessible in this step but is used in a later step, change the value for the appropriate listing under **Step Context**.

4. Continue to play or step through the flow.

OR

To reset any flow variable values that you have changed to the values that were set the last

time you saved the flow, click the **reset-flow** icon ().

Working with breakpoints

Breakpoints provide automatic pauses in the running of a flow in the Studio Debugger, so you don't have to try to manually pause the flow. This can come in handy when you want to do either of the following at the step where you set a breakpoint in order to do, for example, one of the following:

- Examine the value of a flow variable
- Change the flow variable's value to see its effect on the flow in the rest of the run


You set breakpoints in the flow's diagram, but you can enable or disable any breakpoints that you have set from inside the Debugger.

To set a breakpoint

- With the flow open in the authoring pane, right-click the step where you want to set the breakpoint, point at **Debugging**, and then click **Set Breakpoint**.

Note: You cannot create a breakpoint from within the Debugger, although the Debugger is where you enable and disable breakpoints.

To enable or disable a breakpoint

1. To open the flow in the Debugger, click the debug icon () in the toolbar.
2. In the Debugger **Breakpoints & Response Overrides** panel, the **Breakpoints** tab shows the existing breakpoints:

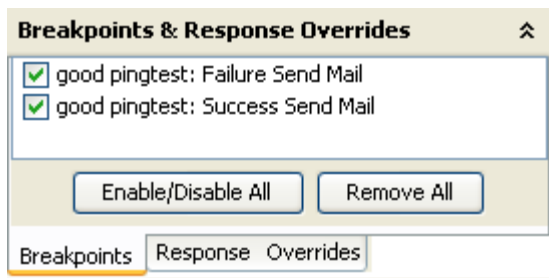


Figure 56 - Enabling and disabling breakpoints

3. Select or de-select the breakpoint's checkbox.

Overriding responses in a debug run

By overriding a response, you can test a particular path of the flow without having to exit the Debugger and change input values. For instance:


- Suppose you have a step in a flow for which you don't have the necessary information. You might want to test the rest of the flow, regardless of the certain failure of that step. You can force the run to follow the response and transition that you want, rather than the failure response that would come about without your intervention.
- Or you might choose to override a success response if you want to test the run on the flow's failure path.

To set a response override for a single step

1. With the flow open in the authoring pane, right-click the step whose response you want to override.
2. In the drop-down menu, point at **Debugging** then **Override Response**, and then click the response you want to force the step to have.

After you have created a response override here, then in the Debugger you can enable or disable the override, or choose a different response for it.

To enable, disable, or choose a different response for the response override

1. To open the flow in the Debugger, click the debug icon () in the toolbar.
2. In the Debugger **Breakpoints & Response Overrides** panel, the **Response Overrides** tab shows the existing response overrides:

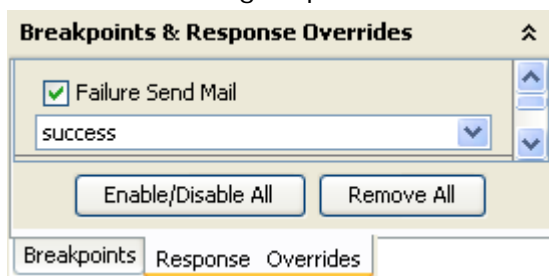


Figure 57 – Working with response overrides

3. Scroll up or down to the response override of interest.
4. To enable or disable the response override, select or de-select its checkbox.
5. To choose a different response for the override, click the down arrow and select the response.

To manually choose a response for every step in the run

1. Before the run starts or at any point at which the run is paused, click the override responses



icon () in the Debugger toolbar.

2. After each step completes, respond to the prompt to choose a response.

Logging in as another user

You can log in to Studio as a different user without stopping and restarting Studio. You might want to do so if there is only one user account that can publish to the Public repository for Central and you're working in a local Studio repository as a different user.

To log in as a different user

- On the **File** menu, click **Switch User**.

Repositories: Libraries for flows and their objects

Flows, operations, and any HP OO objects (domain terms, remote action services, scriptlets, selection lists, etc.) that they reference are stored in a *repository*, a structured set of XML files.

When Studio is installed, it is by default connected to the Central *public* repository. You can, however, work in a *local* repository—that is, a private repository that is local to Studio and external to the Central server. You develop flows in this local repository, then publish them to the public repository on the Central server.

You can exchange work with the other flow authors on your team by publishing to and updating from the Central public repository on a staging server. Or, when one of you is offline, you can export and import selected portions of a repository.

Only the author who creates a private repository has access to it. No one can either update or publish to a private repository that you created except you.

Adding and opening a repository for authoring

You might want to work in another repository besides your default public repository in order to:

- Work in a multi-author environment.
- Keep work on one version of a flow separate from the rest of your flows.

For discussion of the scenarios in which you might want to use multiple repositories, see [Repositories: Libraries for flows and their objects](#).

To work in another repository, you add it and then open it.

To add a repository

1. On the **Repository** menu, click **Add Repository**.

The following dialog appears:

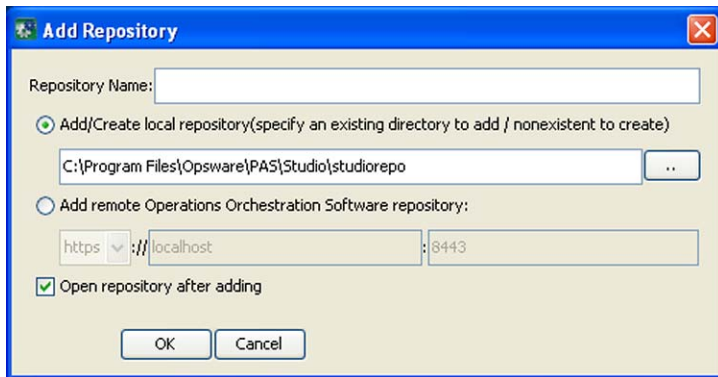


Figure 58 - Adding a repository

2. In the **Repository Name** text box, type a name for the repository
3. To add a local repository, select **Add/Create local repository** and specify a location for the repository folder. Note: if there is already a repository stored at the location you specify, you are prompted to type or browse to a different location.

OR

To add a repository on the Central server, select **Add remote Operations Orchestration Software repository** and then:

- a. In the drop-down list of communication protocols, select the protocol that Central uses for communications.
By default, Central uses the **https** protocol.
 - b. In the text boxes to the right of the list, type the Central server's name and the port that it uses for HP OO communications.
By default, the port for Central when it uses the https protocol is **8443**.
4. To open the repository after you create it, click **Open repository after adding**.
 5. Click **OK**.

The repository is added to the list of available repositories.

To open a repository for authoring

1. On the **Repository** menu, click **Open Repository**.
2. From the list of available repositories, select the one you want to work in.

Moving flow elements between repositories

There are two ways to exchange flows, operations, and other objects that flows use:

- Exporting and importing repositories or parts of them is intended for exchanging flows and related objects with authors who are outside your company.
Exporting a repository creates a standalone repository. When an author imports an exported flow, he or she must manually resolve any conflicts in his repository.
When importing a repository, you can select which of its objects you want to import.
- Updating and publishing repositories or parts of them is for making flows and related objects available to other authors who share access to the same public (Central) repository.
When you publish or update a folder, all the items within the folder and all the objects that are used or referenced by a flow within the folder are published or updated. That is to say, you can't be selective about which objects within a folder that you publish or update.

For procedures for importing, exporting, updating, and publishing, see the following topics in this section:

- [Setting a target repository](#)
- [Publishing a repository](#)
- [Updating from a repository](#)
- [Rolling back a publish or update](#)
- [Exporting a repository](#)
- [Importing a repository](#)

To publish or update flows, operations, and other HP OO objects from one repository to another, you must first set a target repository. The repository that is not the target repository is the source repository.

Note: You cannot set the open repository as the target repository.

Because storage of the repositories resides outside of Studio, you do not need to set a target repository for exporting or importing

- When you **publish**, you are publishing **from the source** to the **target**.
- When you **update**, you are updating **from the target** to the **source**.

Setting a target repository

To set a target repository

1. If the repository that is open is the one that you want to set as the target repository, open another repository.
2. On the **Repository** menu, point to **Set Target Repository**, and then click the repository that you want to be the target.

OR

To set no target repository, click **None**.

Publishing to and updating from the public repository

Whether your target Central server is in a staging environment or a production environment, you exchange flows with the Central repository by publishing to and updating from its repository. The recommended scenario for working with other flow authors is that you have Central installed on a staging server and both authors working an installation of Studio on his or her own computer. With this arrangement:

- The authors make their work available to each other by each publishing to and updating from the public repository on Central (on the staging server). Author A publishes to Central and author B acquires A's work by updating his own (B's) local repository from the public repository.
- When a flow has been sufficiently tested on the staging server, then one of the authors locks the staging server's repository and publishes the flow (or flows) to the repository that servers the production installation of Central.

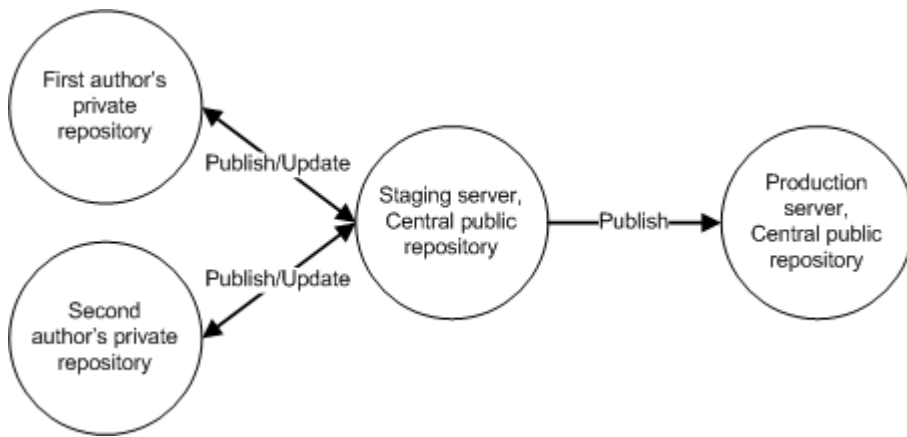


Figure 59 - Recommended arrangement for or more two authors creating flows

When you publish or update, you can pick and choose which groups of objects you publish, with the constraint that when a flow is published, all of its dependent objects – that is, the operations and system objects that that flow uses – are published or updated with the flow.

For information on publishing and updating flows, operations, and system objects, see [Moving flow elements between repositories](#).

Publishing a repository: how to

Publishing is the copying of objects that are new or have changed from a source repository to a target repository. The source repository is the repository from which you initiate a publish or update. It must be a local repository, and the target must be the public repository of the Central that Studio is connected to.

Note: When you publish, changes that you previously published to the target that are not present in the source are not overwritten by the local (source) state of the repository. Suppose that you have two local (source) repositories, A and B, publishing to a public (target) repository, in the following sequence of events.

1. The versions of Testflow1 on local (source) repositories A and B are in sync, identical.
2. Author A adds a step to Testflow1 and publishes to the public (target) repository.
3. The public repository now has the new step.
4. The two authors agree that adding the new step was a mistake.
5. To correct the mistake in the public repository, author B tries to publish the original version of Testflow1 to the public repository.

When author B does the publish, he gets the message that there are no changes, and the Testflow1 in the public repository retains the new step.

You can get the undesired step out of the public repository's version of Testflow1 in the preview stage of updating or publishing. In the **Update/Publish Preview** panel, you select which objects you want to publish or update.

Note: You can in effect undo a publish by importing the target repository's backup. For information on how to do so, see [Rolling back a publish or update](#).

This section includes procedures for both publishing and previewing a publish.

Reminders:

- Before you try to publish or update a repository, make sure you have set a target repository and that the source repository is open. For information on setting a target repository, see [Setting a target repository](#).

- When you publish, you are publishing **from the source** to the **target**.
- Your publish action cannot delete a folder unless you have write permission for every item within the folder.

To publish to a repository

1. With the local repository open, set the public repository as the target repository.
2. On the **Repository** menu, click **Publish Source to Target - Preview**.

OR

Click the **Publish/Update** tab at the bottom of the Studio window.

3. Click the pin icon (📌) to keep the **Publish/Update** panel open.

Tip: If you are publishing and realize that you might need to update from the target repository, then at the left end of the **Publish/Update** panel's toolbar, click either the Update Preview icon (🔄) or the Publish & Update Preview icon (🔄📌).

The **Publish/Update** panel lists any conflicts between the source and target versions of the two repositories' objects.

4. To completely expand each folder in the dialog box, click the plus sign (⊕).

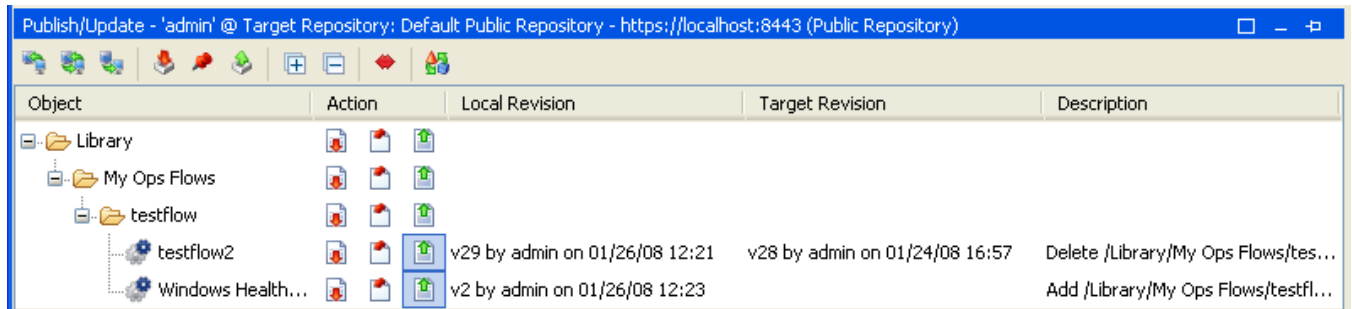


Figure 60 - Publish/Update panel

Note the following information in the dialog:

- The version information and last-modified time/dates for both the local and the target versions
- The action, **Delete** or **Add**, that will be effected by applying the changes, and the pathname of the repository on which the changes will act.
- Folder contents that you have not changed since the last time you published the repository are listed, but the action selected for them is No Change (📌).
- If there is a conflict between versions of objects, you can examine the two versions of the object individually, or side by side.



Key information: When you preview publishing, a conflict is reported only if there have been changes to both objects of the same name in the target and the source repositories. Suppose, for instance, the following:

- a. Local repository X and the public repository are in synch.
- b. You import a new version of testflow1 to local repository X.
- c. From a second local repository Y, you publish another version of testflow1 to the public repository.

Now testflow1 has changed in both local repository X and in the public repository. A conflict will be reported for testflow1 when you preview publishing from local repository X to the public repository.

In this example, there are conflicting versions of testflow1 on the local and target repositories.

- To look at either of the versions by itself, right-click in the row for the object (testflow1), and then, in the menu that appears, click **Open Local** or **Open Target**, depending on which you want to examine. The flow diagram of version that you select opens in the authoring pane of Studio, above the Publish/Update panel.




OR

To look at both versions side by side, , right-click in the row for the object (testflow1), and then click **Compare**.

In the flow diagram(s), you can make and save changes.

- After comparing the objects and possibly making changes, choose an action for each object or folder:


Tip: Note that in the Publish & Update Preview, you can choose to publish some objects and update others.

- To change the object in the target repository, click the upward-pointing arrow .
- To delete the object or folder from the source, click the downward-pointing arrow .
- To take no action, click the No Change icon .

OR

Right-click the object and choose the corresponding command:

- To update the object from the target, **Modify <objectname> in local**
- To publish the object to the target, **Modify <objectname> in target**
- To take no action, **No Change**

- To apply the changes that you've specified, click the Apply icon .
- When a message appears saying that all changes have been successfully applied, click **OK**.
- Save your work.

Updating from a repository: how to

Updating from a repository is the opposite of publishing to a repository. But the source repository is still the repository from which you initiate a publish or update. It must be a local repository, and the target must be the public repository of the Central that Studio is connected to. So when you **update**, you are updating **from the target** to the **source**.

When you initiate an update, you cannot stop it or choose which objects you want to copy from the target Public repository to your local source repository. However, you can preview the update, and you can, in effect, undo it after the fact.

- When you preview an update, you can select which HP OO objects that you want to update to the local repository, and update them.
- To undo an update, you need to import the source repository's backup. For information on how to do so, see [Rolling back a publish or update](#).

To update from a repository

- With the local (source) repository open, set the repository that you want to **update from** as the target repository.

For information on setting a target repository, see [Setting a target repository](#).

- From the **Repository** menu, select **Update Source from Target - Preview**.

OR

Click the **Publish/Update** tab at the bottom of the Studio window.

- Click the pin icon (📌) to keep the **Publish/Update** panel open.

The **Publish/Update** panel lists any conflicts between the source and target versions of the two repositories' objects. When updating, conflicts are reported for objects that have the same name in the same location in the target and source repositories, but that have two different IDs.

Suppose, for instance, the following:

- The local repository and the public repository are in synch, and both contain testflow1.
- In the public repository, you delete testflow1 and then add another version or instance of testflow1.

Now the ID of testflow1 in the public repository is different from the ID that testflow1 has in local repository. A conflict will be reported for testflow1 when you preview updating from the public repository to the local repository.

- To completely expand each folder in the dialog box, click the plus sign (⊕).

Important: If there is a conflict between versions of objects, you can examine the two versions of the object individually, or side by side.

In the illustration, there are conflicting versions of testflow on the local and target repositories and a new flow, testflow2, that was published to the Public repository from another local repository.

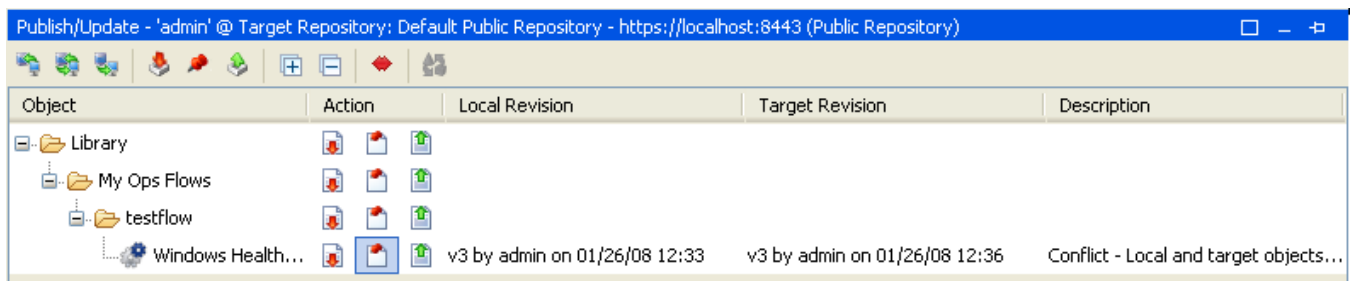
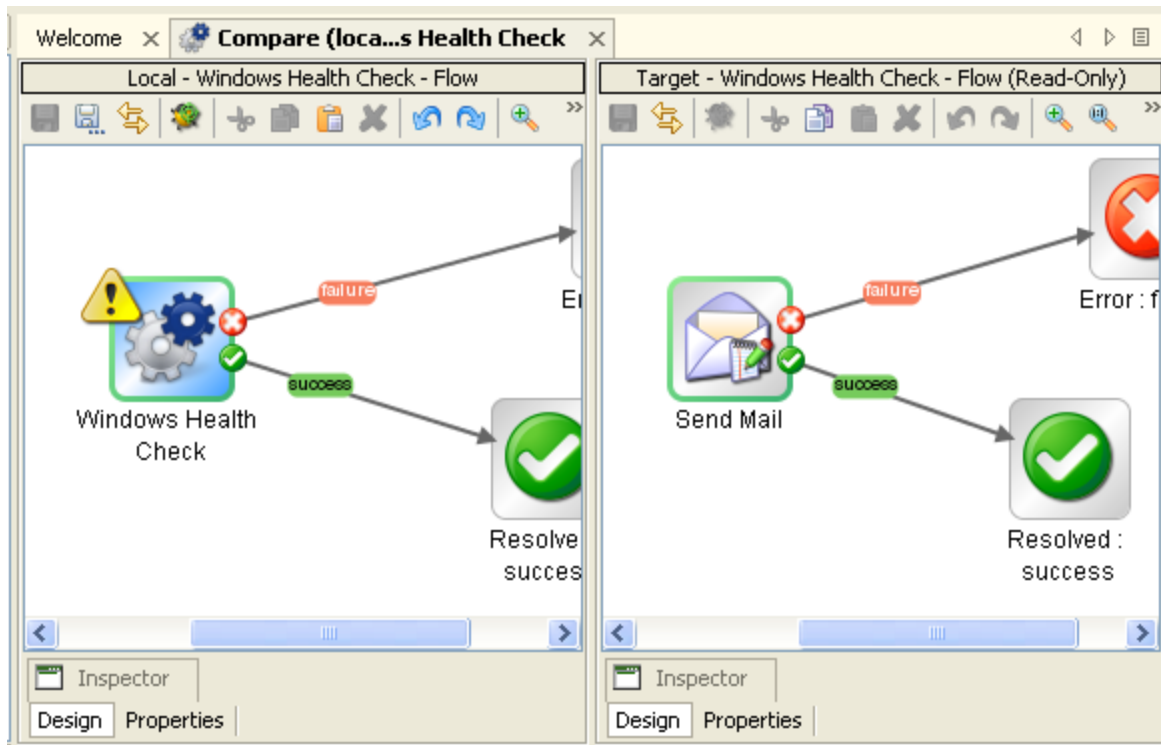


Figure 61 - Update panel with conflict

- To look at either of the versions by itself, right-click in the row for the object (testflow1), and then, in the menu that appears, click **Open Local** or **Open Target**, depending on which you want to examine. The flow diagram of version that you select opens in the authoring pane of Studio, above the Publish/Update panel.

OR




To look at both versions side by side, right-click in the row for the object (testflow1), and then click **Compare** (or double-click the row).



You can make and save changes in the local repository's flow diagram. (Note that the target [i.e., Public] repository's flow diagram is read-only.)

6. After comparing the objects, choose an action for each object or folder:

Tip: Note that in the Publish & Update Preview, you can choose to publish some objects and update others.

- If you are updating, click () to update the local repository with the object from the public repository.
- If you are updating, click the downward-pointing arrow () to copy the local version of the object to the public repository.
- To take no action, click the No Change icon ().


OR

Right-click the object's row and click either **Select Incoming** (to update the local repository's version of the conflict from the Public repository) or **Select Outgoing** (to publish the local repository's version of the conflict to the Public repository).

OR

For more detail on what it will mean to select the incoming version or the outgoing version of the conflict, highlight the object's row, right-click, and choose the corresponding command:

- To update the object from the target, **Modify <objectname> in local**
- To publish the object to the target, **Modify <objectname> in target**
- To take no action, **No Change**

7. To apply the changes that you've specified, click the Apply icon ().

8. Save your work.

Rolling back a publish or update

If you publish to or update a repository and then realize that you did not want to make those changes to that repository, you can effectively undo or roll back the changes by opening the repository and importing its backup.

A repository is automatically backed up to a .jar file 10 minutes after a change occurs, so:

To reverse the effects of a publish or update

1. Unpack the .jar and then import it.
2. Save your work.

Exporting a repository

To make flows and HP OO objects available to another author with whom you do not share a public repository, you can export the flow and its dependent objects as a repository, which other Studio authors can then import.

In addition, you can back up your flows and their dependent objects by exporting them to a safe backup location.

To export a repository

1. In the Library, right-click the folder that contains all the HP OO objects you want to export, point to **Repository**, and then click **Export as New Repository**.
2. In the **Select Repository Directory** dialog, do one of the following:
 - Navigate to an existing folder and delete the content.
 - Navigate to where you want a new folder to reside and type the folder name in the **File name** box.
3. Click **Save**.

The **Export Options** dialog appears.



Figure 62 - Options for export

4. In **Export Options**, select any items that you want not to include in the exported repository. If you are going to import the exported repository to the installation of Studio where you'll use it, you don't need to include HP OO content. If any of your flows use HP OO content and you are going to open the exported repository without importing it, then be sure to include HP OO content in the export (that is, do not exclude it).

The starred items (RASes, selection lists, system accounts, and HP OO content) are excluded from the export (even if you have not marked them for exclusion) if they are not referenced by a flow or operation that you're exporting.

If you are going to use the Shell Wizard to add an operation that runs a Windows or linux shell (command-line) command to the repository that you are exporting, it is recommended that you export the entire Studio Library, rather than a subfolder of the Library.

If you export a subfolder of the Library and are going to point the Shell Wizard or Web Service Wizard at the resulting export, be sure not to exclude OO content from the export. If you do so, the repository must include either the SSH Shell or Telnet Shell operation. SSH Shell or Telnet Shell operation are part of the default OO Content, so you must include the OO Content in an export when the Shell Wizard will add a shell operation to the resulting repository.

Among the other items you can exclude from the export of the repository, you might want to exclude any of the following if they are specific to one environment and you're exporting the repository for use in a different environment:

- Remote Action Services (RASs)
A RAS is an reference to RAS that enables a flow to carry out commands outside HP OO or on a remote computer, or to integrate with other application programming interfaces
- Selection lists
Stored lists that are presented to the user.
- System properties
Global flow variables with values that never change and so can be used in many flows, saving you the time of recreating the flow variable each time you need to use it.
- System accounts
These are user credentials that are hidden behind the system account name by which they can be referenced.

Important: Keep in mind that the items that you select are those that you want to **exclude** from the export. That is, those that you do **not** want to export.

5. To give everyone read, write, execute, and link permissions for the flows, operations, and HP OO objects (such as selection lists) that you have created or have the right to modify and are exporting, select **Give EVERYBODY group full access to exported items for which you have write access**.
6. Click **OK**.

Importing a repository

To obtain additional existing flows and operations – which may have been created by someone else or are in a new Accelerator Pack – you can import the flow as a repository or a Java archive (.jar) file, if the repository has been stored in that format.

When you import a flow, you also import the flow's configurable HP OO objects, such as domain terms and filters that have been saved as system filters, that are used by the flow or its operations.

Remember: When you import a flow or operation, you're importing the object, not a copy or instance of it. Further, the object can only exist in one place within the repository. Therefore, an operation can reside anywhere in the library relative to the flow or flows that use it.

To import a repository

1. Either select or create a folder in the Library.
2. From the **Repository** menu, choose **Import Repository**.
If you have a flow or operation open in the authoring pane, a message appears, prompting you to allow the program to close all editors.
3. If you see that message, click **OK**.
4. In the **Select Repository Directory or .jar** dialog box that opens, navigate to the directory that contains the repository (which may be an Accelerator Pack) or .jar file that you want to import, and then click **Open**.

After Studio checks for differences between the source and target repositories, the **Importing from...** dialog box opens.

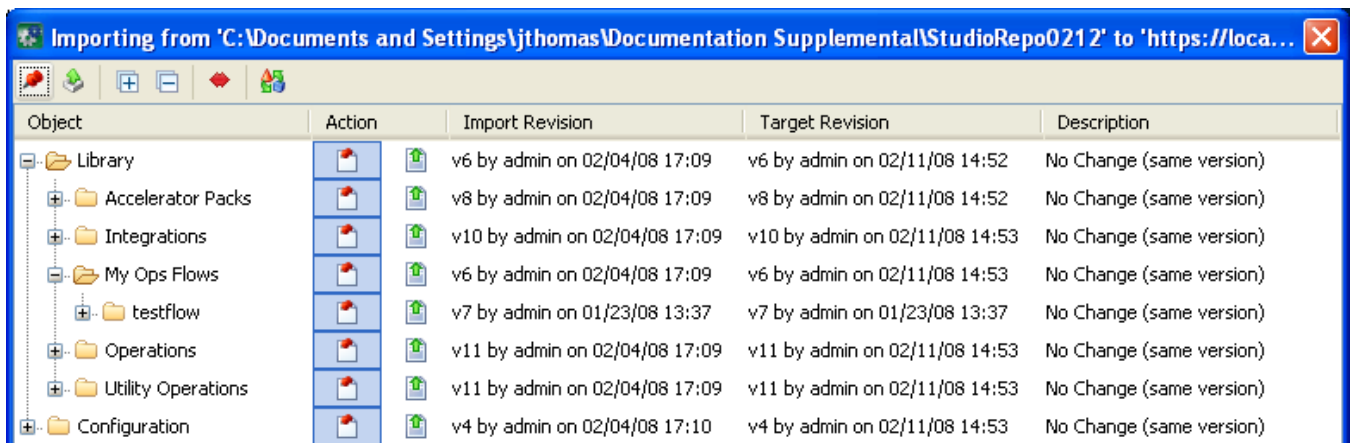


Figure 63 - Importing dialog box

This dialog lists the folders whose contents you will import.

- The **Library** folder contains the flows and operations that you're importing.
- The **Configuration** folder contains domain terms and HP OO objects.

When you expand a folder, the dialog shows this information for each object to be imported within that folder. Note that although the action shown for a folder may be No Action ([Icon]), there may in fact be changes within that folder for you to examine.

5. To completely expand each folder in the dialog box, click the plus sign ([Icon]).
When you expand the folders, the dialog shows similar information for each HP OO object.

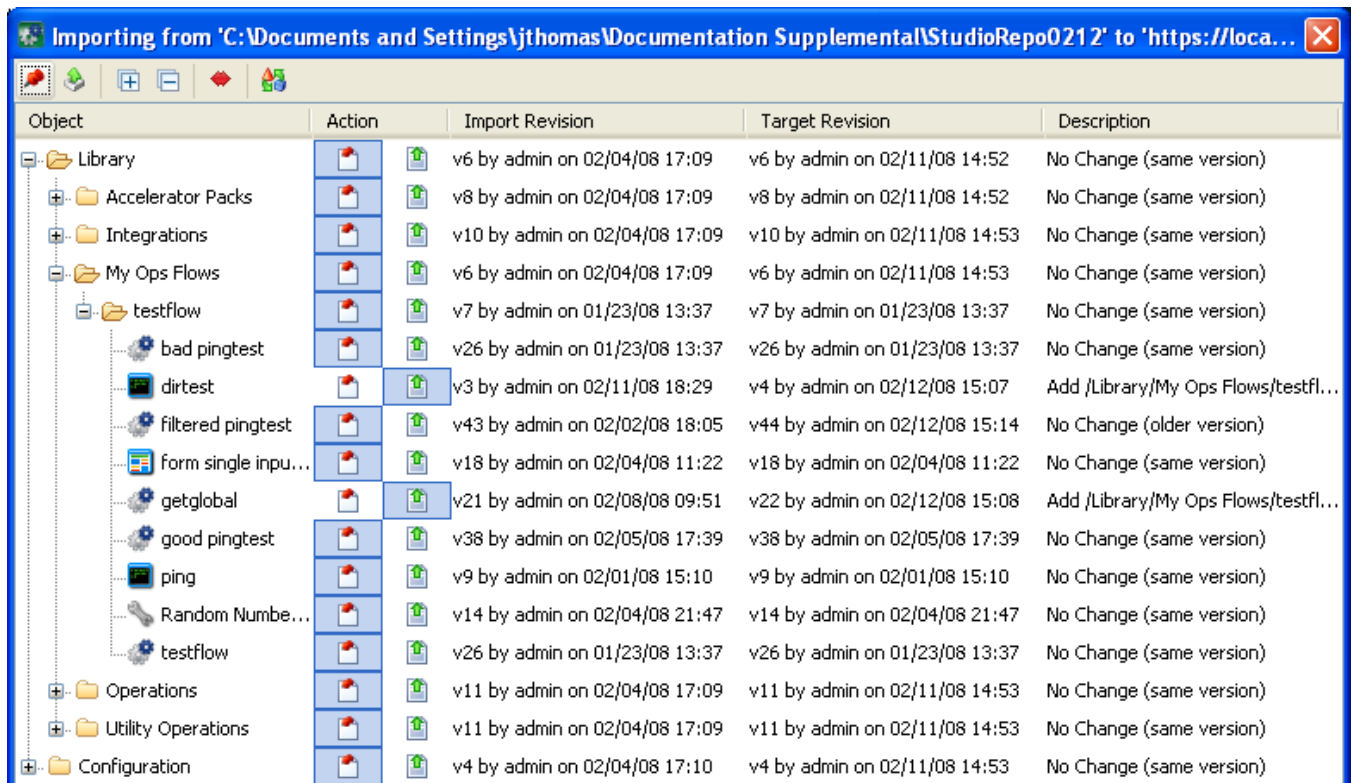




Figure 64 - Selecting objects to import

Note particularly that the dialog shows the following information for the items in the target repository that exist in both the target and the source.

- The Revision (version) number for the item in the target and in the source
- Who created or changed the item, and on what date
- A description of the change, or notation that there is no difference between the two apart from a date stamp.
- In this example, by the way, the folders and their contents have not been changed since they were exported. When you import a folder or HP OO object that you haven't changed since you exported it, the folder or object is listed, but the No Change icon () is highlighted for it. When you click the Apply icon, no action will be taken on that item.

Note: You can view only the conflicts by clicking the **View All Changes** button (.

6. To import a folder and its contents or an HP OO object to the target repository, under **Action**, click the upward-pointing arrow () for the appropriate row.

OR

To leave an object as it is in the source, click the no change icon (.

Tip: You can take the same action for all the objects within a folder by clicking the upward-pointing arrow or the no-change icon for the parent folder.

7. In the toolbar of the dialog, click the apply icon (.

Because storage of the repositories resides outside of Studio, you do not need to set a target repository for exporting or importing.

Validating repositories

For a flow to run, the flow itself, its operations, and any system accounts used in the flow must be valid. You can check an individual flow or operation for problems (by highlighting the flow or operation in the Library and then clicking the **Problems** tab). (See also [What a flow needs to be valid.](#))

Or, to discover and correct in one sweep any problems that there might be, you can validate an entire repository. This validates all the flows, operations, and system accounts in the repository.

To validate a repository

- With the repository open, from the **Tools** menu, click **Validate Repository**.
A list of any problems, with their location and description, appears, as it does when you use the **Problems** tab, to guide you in repairing the problems.

Encrypting repositories

Creating encrypted copies of local repositories protects snapshots of your work from tampering or theft. When you encrypt a local repository, you are making a copy of the repository and encrypting the copy. To modify in anyway, publish to, update from, import to, or export an encrypted repository, you must type the correct password.

Notes:

- You can only encrypt a local repository. If Studio has the public repository open, the commands related to encrypting repositories are unavailable.
- The entire repository is exported to the encrypted copy, regardless of which folder within the repository is selected.
- When you encrypt a repository, the original, unencrypted repository remains open in Studio.
- Once you have created an encrypted repository copy in a given location, you cannot re-encrypt a repository or encrypt another repository to the same location.

Encrypting a repository

To encrypt a repository

1. With the local repository that you want to encrypt open, on the **Repository** menu, click **Encrypt Repository**.

The following dialog appears:

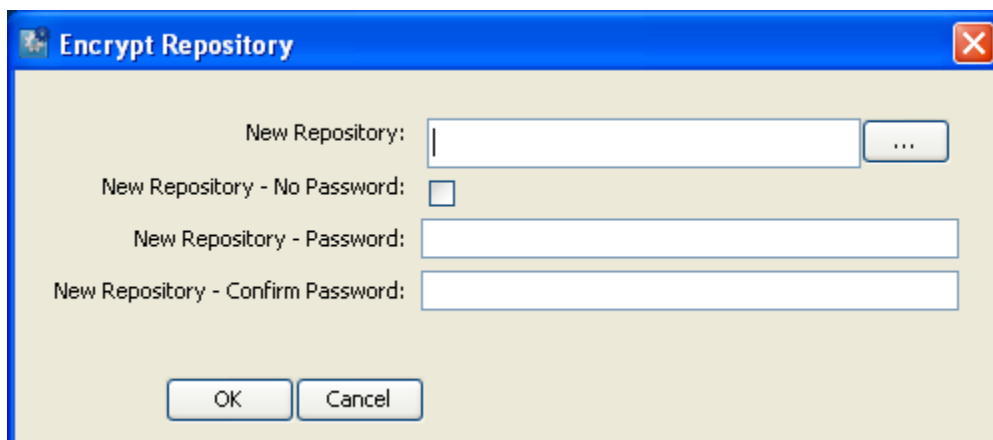


Figure 65 - Encrypting a repository

- In **New Repository**, type a path and name for the new copy of the repository.
OR
Click the ... button to navigate to the desired location or repository and name the encrypted copy.
- Type and confirm a password for the encrypted copy, and then click **OK**.

Opening an encrypted repository

To open an encrypted repository

- To open the encrypted copy of a repository, from the the **Repository** menu, click **Open Repository** and select the encrypted repository that you want to open.
OR
If you are opening the encrypted copy for the first time, add the repository: on the **Repository** menu, click **Add Repository** and navigate to the encrypted copy.
- When prompted for the password, supply the password that you specified when you encrypted the repository.

Decrypting a repository

To decrypt a repository

- Open the encrypted repository.
- From the **Repository** menu, click **Decrypt Repository**.

Suppose you want to create a second encrypted copy of the repository, with a different password. You do so by re-encrypting the repository.

Creating a second encrypted copy of a repository

To create a second encrypted copy of the repository

- Add (if necessary) and open the encrypted repository.
- From the **Repository** menu, click **Re-encrypt Repository**.

The Re-encrypt Repository dialog box appears.

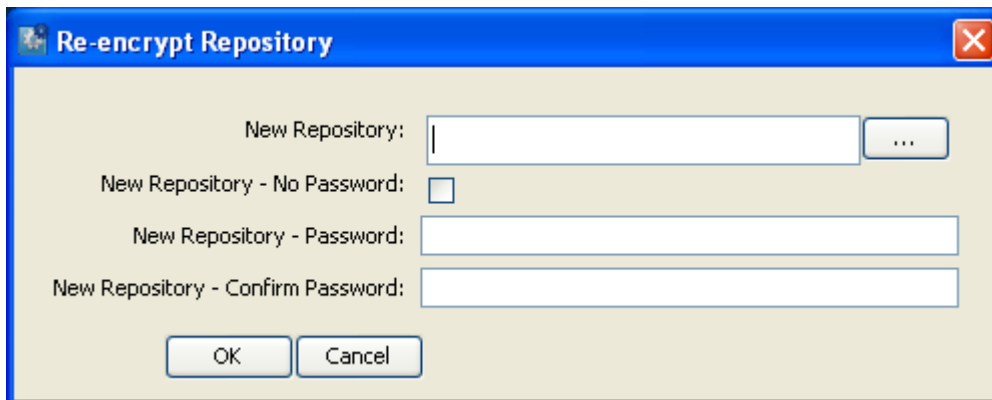


Figure 66 - Re-encrypting a repository

- In **New Repository**, type a path and name for the new copy of the repository.
OR

Click the ... button to navigate to the desired location or repository and name the encrypted copy.

4. Type and confirm a password for the encrypted copy, and then click **OK**.

Backing up and restoring repositories

The public repository (the repository that Central uses) and the Studio default local repository are each automatically backed up to a .jar file when you start Studio and when you save an object. By default, 10 backup .jar files are kept. On the 11th save, the oldest backup file is deleted. However, you can also manually back up the private repository.

To back up a repository

1. Open the repository that you want to back up.
2. From the **Repository** menu, choose **Create Backup**.

To restore a repository

1. Unpack the .jar file that was created by the backup.
2. From the **Repository** menu, choose **Import Repository**.
3. In the Select **Repository Directory** dialog box, navigate to the folder that contains the .jar file that defines the repository.

The folder that contains the .jar file contains two folders, **backups** and **data**.

4. Click **Open**.

After checking the Central public repository for version conflicts with the objects that you want to import, HP OO imports the repository.

5. If there are conflicts, resolve them as you do when you import repositories otherwise.

For more information on importing repositories, see [Importing a repository](#).

Creating operations from Web services

The Web Services Wizard creates HP OO operations based on the API in the Web Service Definition Language (WSDL) of the Web service that you identify in the wizard. (A Web service is a piece of business logic that resides somewhere on the Internet.)

When you run the Web Services Wizard, you provide it with the WSDL for a given Web service. The WSDL string you provide as a pointer can be a file's location and name or a URL.

For instance, suppose you have an application called AlertAlert that creates a ticket through a Web service and API, and you want to tell AlertAlert to create a ticket. The Web Services Wizard extracts, from the Web service's WSDL, the application's APIs 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 that each method needs, and the required format for each input.

When you provide the wizard with the WSDL (in our example, for AlertAlert) and run the wizard, it generates operations that can run against the Web service. The operations appear in the Studio Library, with the required inputs created. You can use the operations in flows. To run a flow that uses one of these operations, in Studio you'll need to create a remote action service (RAS) that points to the Web service.

To create operations for a Web service with the Web Services Wizard

1. In the HP OO home directory, in \Studio\tools\ double-click **Wswizard.exe**.

The WebService Wizard starts.

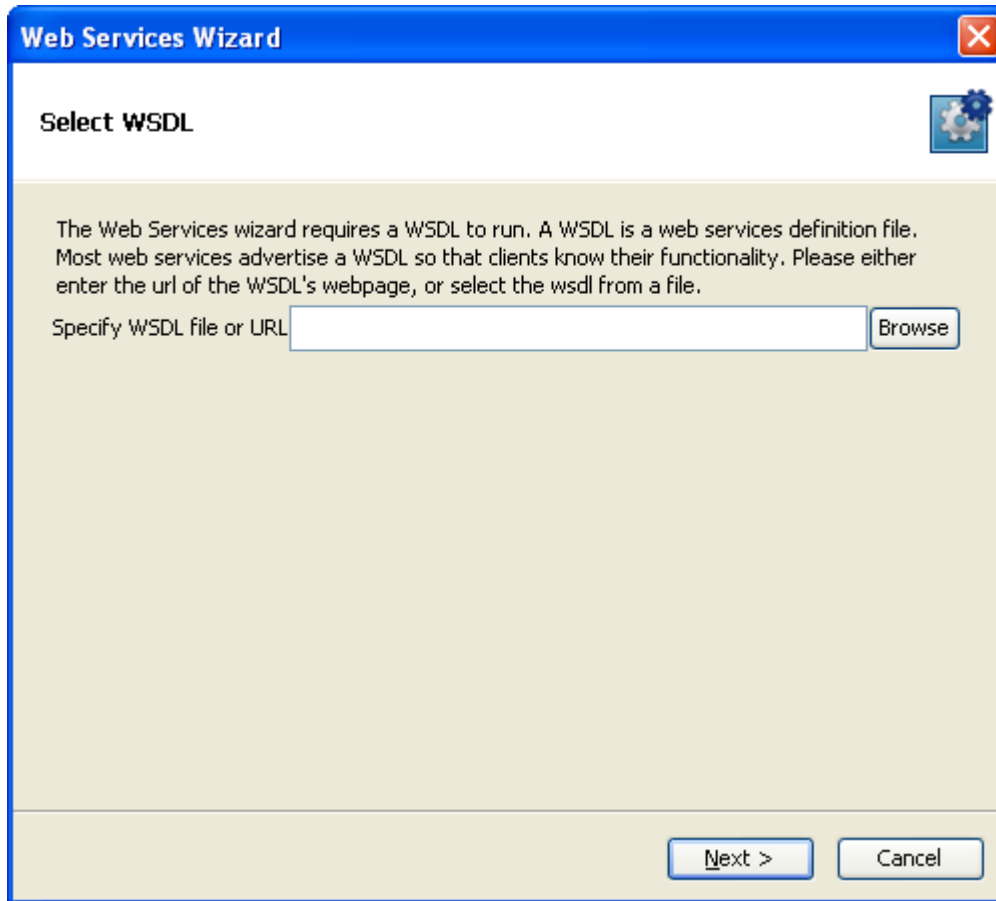


Figure 67 - WebService Wizard

2. In the **Specify WSDL file or URL** box, type the URL or file location of the WSDL.
OR

Click **Browse** and navigate to the location where you want to store the files.

The WSDL location can be the URL of a RAS reference that you created in Studio, because the RAS that the pointer references is a Web service.

3. Click **Next**.

A progress box appears while the the wizard loads the WSDLs.

The **Choose Operations to generate** pane appears.

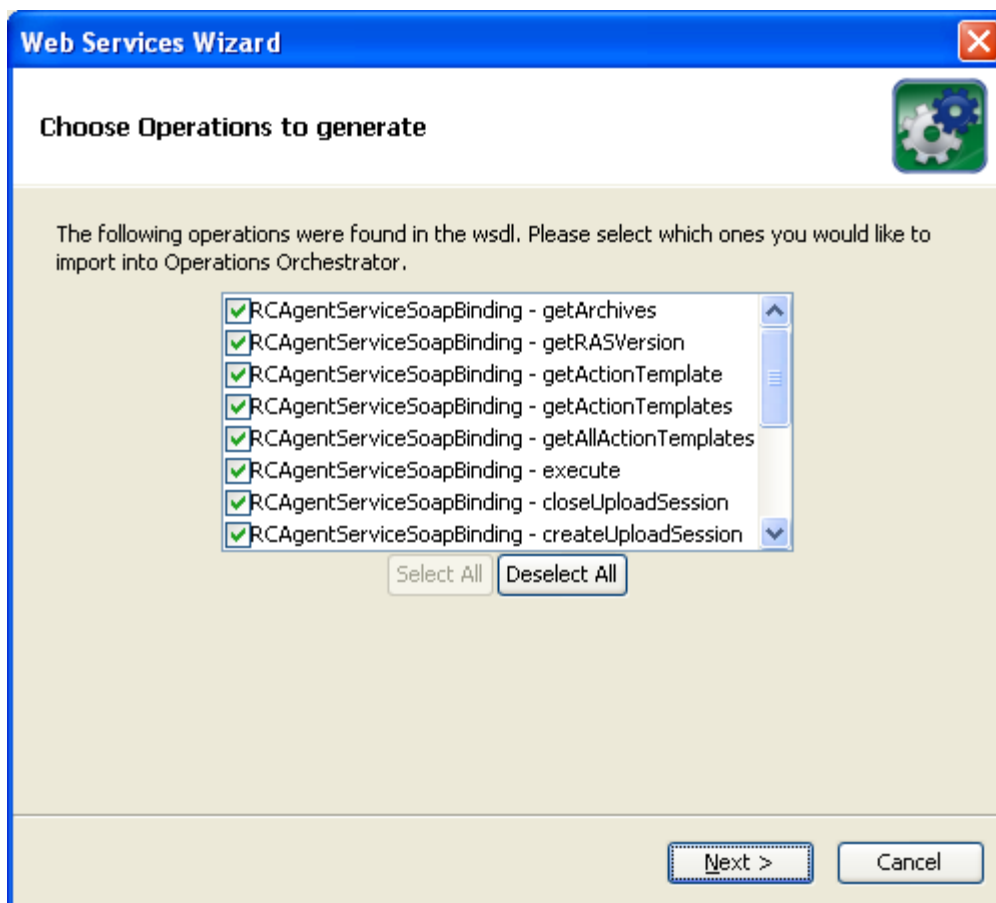


Figure 68 - Choosing operations to create

The page lists all the WSDL's methods, which can be turned into operations.

4. Deselect (remove the check from the checkbox of) the methods that you do not want to turn into operations, and then click **Next**.

The **Select Repository** page opens, in which you will specify the folder where a repository is created to contain the new operations. It is recommended that you specify a folder that is currently empty.

5. Under **Enter the repository to open**, type the path and folder where you want to create the repository containing the operations the wizard creates, and then click **Next**.

OR

Click **Browse** and navigate to the location where you want to create the repository, and then click **Next**.

The wizard completes creating the operations and placing them into the repository that you specified.

6. To create more operations, click Next and repeat the earlier steps in this procedure.

OR

If you are finished with creating new operations by this method, click **Finish**.

7. Open Studio and import the repository that you just created.

Operating outside Central with Remote Action Services

A Remote Action Service (RAS, which we will pronounce phonetically) is an instance of a service that interacts with both Java and .NET to enable a flow to run in the following situations:

- The commands are not defined in the Central repository.
- The operations run on machines that are remote from the Central server – or, indeed, anywhere on the internet, even on the other side of firewalls.
- The operations integrate with other application programming interfaces (APIs).

The RAS must be installed on the computer where it extends a flow's operations.

You could use a RAS to accomplish the following:

- Integration with the Exchange API.
- Restarting a server in a different domain from the one in which Central is installed.
If you have installed a RAS on a machine in the domain of the server you're going to restart, you can run a flow that restarts the server by pointing the flow at the RAS on that domain.
- Carrying out an operation on a Windows server when Central was installed on a Linux server.
- Distribution of the load of your automation across multiple servers.

A flow that uses a RAS requires a *RAS reference* on the Central server that points to the RAS. A complete RAS reference on the Central server has a name and a URL that accesses the RAS.

Therefore, to make a flow that works outside HP OO, you may need to install the RAS, then configure a RAS reference for your flow.

When you installed Central and Studio, the RAS reference `RAS_Operator_Path` was created on Central. If your flow can use this RAS, you only need to confirm the availability of the RAS. However, if your flow uses a RAS that has a different path from the existing RAS, you would need to add and configure a new RAS service.

When you integrate HP OO with another API, you use the Operations Orchestration Software Development Kit (SDK) and C# or Java to implement the `IAction` interface, then deploy the operation to the Remote Action Service (RAS). This level of authoring requires the ability to program with .NET or Java as well as Windows Scripting Host or Perl scripting.

How Central runs RAS-dependent operations

The RAS contains a dynamic-link library (DLL, or .dll) and Java archive (.jar) that contains definitions of operation classes. When the RAS is called, it runs an instance of the class of the required operation.

A RAS-dependent operation contains the information that Central needs to identify the RAS on which the operation must run:

- **Action Class** is the name of the operation that is inside the archive or DLL and is the operation that you want to run.
- **Archive** is a DLL (.dll) or Java archive (.jar), such as `Dotwebactions.dll`, that contains operations.
- **RAS** is the RAS reference that the operation uses.

These pieces of information are specified in the **RAS Operation fields** on the operation's **Properties** sheet. Operations in the **Integrations**, **Operations**, and **Utility Operations** folders in the Library already have this information provided. If you change the URL of the RAS, you must also change the URL of the RAS pointer in Studio. Creating a RAS-dependent operation from scratch, includes:

- Creating a DLL or .jar to contain the operation's definition.
- Specifying that DLL or .jar in the installation of a RAS service for running the operation.
- in the operation's **RAS Operation fields**, providing the name of the operation class, the DLL or .jar, and the name of the RAS pointer.

Checking the availability of a RAS

To check the availability of an existing RAS

1. In the **Library**, right-click the **Configuration\Remote Action Services** folder and click **Open**.
2. In the **Remote Action Services** sheet, highlight the RAS pointer whose availability you want to check, and then click **Check Availability**.

Name	Description	URL	Availability
JDPrivateRAS		https://blue-jdoed620.opsware.com:9004/RAS/services/R...	UNAVAILABLE
mirage			UNKNOWN
RAS_Opera...		https://blue-jdoed620.opsware.com:9004/RAS/services/R...	UNAVAILABLE

Figure 69 - Remote Action Services sheet

The RAS's availability is reported in the **Availability** column.

3. If the RAS is not available, click the right-pointing arrow at the end of the RAS pointer's line, and then correct the location of the RAS in the **URL** box.

Adding an existing RAS

When you open the **Configuration\Remote Action Services** folder, any RASes that are installed on your network are listed under **Discovered RAS**. Use the following procedure to make them available to operations that require them for operations outside HP OO.

To add an existing RAS

1. Open the **Configuration\Remote Action Services** sheet.
2. Under **Discovered RAS**, select the RAS that you want to make available to your operations, and then click the **Add** button under **Discovered RAS**.
3. In the dialog that appears, name the new RAS.
The new RAS appears in the list of RASes above.

Adding a RAS reference

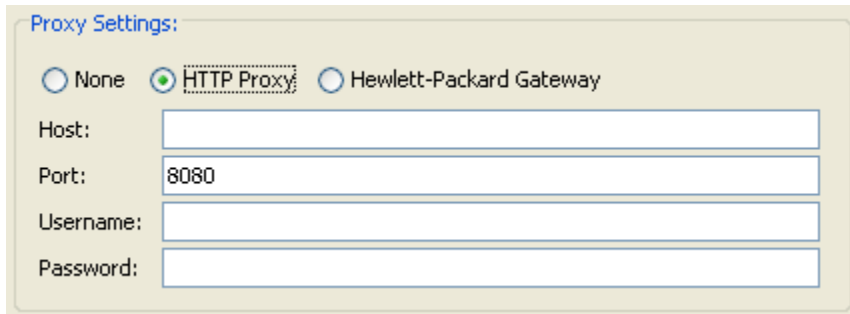
To add a new RAS reference

1. In the Library pane, right-click the **Remote Action Services** folder and then click **Add**.
The **Remote Action Services** sheet opens in the authoring pane.
OR
If the **Remote Action Services** sheet is already open, click the upper **Add** button.
2. In the box that appears, type a name for the new RAS reference and then click **OK**.
3. In the **URL** column, type a URL with the following syntax:
`http://<yourHostname>:<yourPort>/RAS/services/RCAgentService`

If you accepted the default port number in the RAS installation program, the port number is 4085.

To connect to a RAS that is on the other side of a firewall from Central, you might need to specify a proxy for communicating with the RAS.

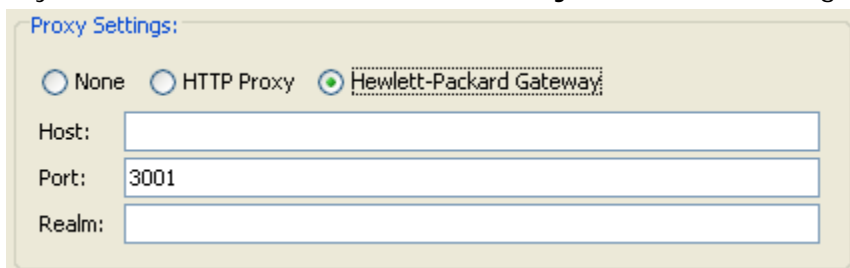
- To select a proxy, under **Proxy Settings**, select either **HTTP Proxy** or **Hewlett-Packard Gateway**.
 - If you select **HTTP Proxy**, fill in the following text boxes:



The screenshot shows a dialog box titled "Proxy Settings". At the top, there are three radio buttons: "None", "HTTP Proxy" (which is selected), and "Hewlett-Packard Gateway". Below the radio buttons are four text input fields: "Host:" (empty), "Port:" (containing "8080"), "Username:" (empty), and "Password:" (empty).

Figure 70 - Settings for HTTP Proxy

- In the **Host** text box, type the machine name or IP address of the server on which the proxy resides.
 - In the **Port** text box, type the port that the proxy uses.
 - In the **Username** and **Password** text boxes, type the credentials required to use the proxy.
- If you select **Hewlett-Packard Gateway**, fill in the following text boxes:



The screenshot shows a dialog box titled "Proxy Settings". At the top, there are three radio buttons: "None", "HTTP Proxy", and "Hewlett-Packard Gateway" (which is selected). Below the radio buttons are three text input fields: "Host:" (empty), "Port:" (containing "3001"), and "Realm:" (empty).

Figure 71 - HP Gateway proxy settings

- In the **Host** text box, type the machine name or IP address of the server on which the proxy resides.
 - In the **Port** text box, type the port that the proxy uses.
 - In the **Realm** text box, type the name of the Hewlett-Packard Gateway realm that the gateway is a member of.
- To check availability, click the left-pointing arrowhead at the top right of the RAS editor, and then click **Check Availability**.

Doing so checks for the availability of the services' URLs and displays their availability (or lack of availability) in the **Availability** column.

- Click **Check Availability**.

The availability or lack thereof appears in the **Availability** column.
- When the RAS is available, click **OK** at the bottom of the pane, and then save your changes.

Reconfiguring an existing RAS reference

To reconfigure an existing RAS reference

1. In the **Library**, open the **Configuration\Remote Action Services** folder and either double-click the RAS you want to configure or right-click the RAS and then click **Open**.

The **Remote Action Services** sheet opens in the authoring pane.

OR

With the **Remote Action Services** sheet open, select the RAS reference you want to reconfigure, and then click the right-pointing arrow (➔) at the end of the pointer's row in the table.

2. In the **URL** text box, the URL that you specify must have the following syntax:

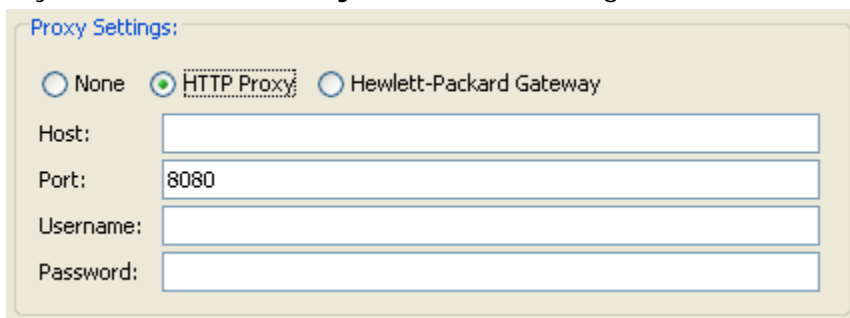
`http://<yourHostname>:<yourPort>/RAS/services/RCAgentService`

If you accepted the default port number in the RAS installation program, the port number is 4085.

To connect to a RAS that is on the other side of a firewall from Central, you might need to specify a proxy for communicating with the RAS.

3. To select a proxy, under **Proxy Settings**, select either **HTTP Proxy** or **Hewlett-Packard Gateway**.

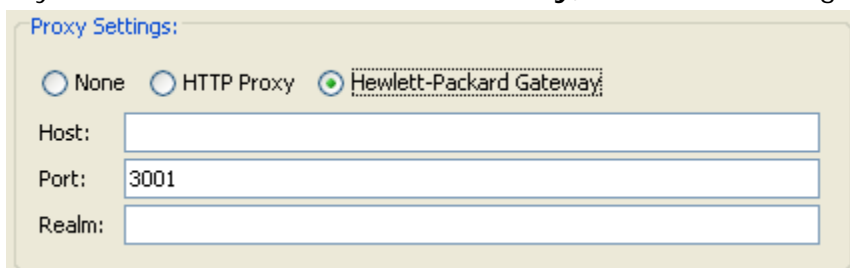
- If you select **HTTP Proxy**, fill in the following text boxes:



The screenshot shows a 'Proxy Settings' dialog box with three radio buttons: 'None', 'HTTP Proxy' (which is selected), and 'Hewlett-Packard Gateway'. Below the radio buttons are four text input fields: 'Host:', 'Port:' (containing '8080'), 'Username:', and 'Password:'.

Figure 72 - Settings for HTTP Proxy

- In the **Host** text box, type the machine name or IP address of the server on which the proxy resides.
 - In the **Port** text box, type the port that the proxy uses.
 - In the **Username** and **Password** text boxes, type the credentials required to use the proxy.
- If you select **Hewlett-Packard Gateway**, fill in the following text boxes:



The screenshot shows a 'Proxy Settings' dialog box with three radio buttons: 'None', 'HTTP Proxy', and 'Hewlett-Packard Gateway' (which is selected). Below the radio buttons are three text input fields: 'Host:', 'Port:' (containing '3001'), and 'Realm:'.

Figure 73 - HP Gateway proxy settings

- In the **Host** text box, type the machine name or IP address of the server on which the proxy resides.
 - In the **Port** text box, type the port that the proxy uses.
 - In the **Realm** text box, type the name of the Hewlett-Packard Gateway realm that the gateway is a member of.
4. To check availability, click the left-pointing arrowhead at the top right of the RAS editor, and then click **Check Availability**.

Doing so checks for the availability of the services' URLs and displays their availability (or lack of availability) in the **Availability** column.

5. When the RAS is available, save your changes and close the editor.

OR

To work on another RAS, click either the up arrow or the down arrow at the top left of the RAS tab, below the save icon.

Creating operations that access Web services

A Web service is a Web-based piece of programming logic that performs a particular set of functions. If you want to integrate a flow with a Web service, monitor it, or validate that it is running, you can access the Web service through the methods that are defined in the Web service's WSDL (Web Service Description Language) file, which defines the interface of the Web service. To access a Web service, you create a SOAP (Simple Object Access Protocol) message (written in XML) and send it to the Web service. The Web service responds to the SOAP message and returns XML.

For the purposes of a flow, the WSDL defines the methods that a flow operation can use to interact with the Web service. The Web service Wizard is a tool that displays a list of the methods in the interface of the Web service that you specify in the Wizard. Within the Wizard, you pick the methods that you want to use, and with one click, for each method you have selected, the Wizard creates an HP OO operation that can execute the method.

Or suppose that you want an operation to perform a Web search using the Google search engine. You would locate the GoogleSearch Web service WSDL and navigate to the WSDL in the Web Service Wizard. From the list of methods that appear in the Wizard, you would select doGoogleSearch and, by clicking Generate XML, create the XML that defines an operation that performs the Google search.

Note: To use RAS-dependent operations, you must also configure HP OO for extended functionality. For information on doing so, see *Administering Operations Orchestration Software* (AdminGuide.pdf).

For information on using the Webservice Wizard, see [Creating operations from Web services](#).

Troubleshooting the running of RAS-dependent operations

There are three reasons that Central could fail when attempting to run a RAS-dependent operation:

- The Servicename does not point at the correct RAS service (the RAS that contains the DLL or .jar that contains the definition of the operation you want to run).
- The RAS service was never installed on the target machine.
- The library is not present in the RAS service.

Studio example: A flow that runs in any of multiple domains

By using a flow variable, you can create a flow that chooses the correct RAS on which to run an operation remotely. Basically, you store the name of the desired RAS service in the flow variable, and flow logic determines which RAS service that is.

Suppose you want to restart either ServerA in DomainA, ServerB in DomainB, or ServerC in DomainC. With the following steps in the flow, the Central user can decide, when he or she runs the flow, which server to run the flow against.

1. Create a scriptlet step that prompts the user for the server name, including the domain name.

2. Write the step's scriptlet to:
 - Extract the domain name.
 - Depending on the domain name, store the name of the RAS that serves that domain in the flow variable "rasname".
3. Provide the flow variable rasname as an input to the RAS operation that restarts the server. You might want to rename the RAS operation in the RAS operation's rasname property.

Creating operations from a RAS

To create operations that use a RAS, you can

- Create your own RAS and operations.

After you import your Web service and operations, you move the operations from the location where they are originally imported in the Library. If you leave the operations in their original location and subsequently export any flows that use these operations, the operations are not exported with the flows. The exported flows will not work on any other machine. Moving the operations to another folder within the Library avoids this situation.
- Import and use HP OO-provided content that uses the Remote Action Service (pointed to by the RAS_Operator_Path RAS reference).

To import custom Web service content, you create, name, and assign a URL to a RAS reference. Identification of the RAS by primarily by name rather than by URL means that changing the URL in the Studio enables you to use the RAS in multiple locations.

The **Configuration** folder contains a **Remote Action Services** folder that holds the existing RAS pointer.

Now you're ready to import Web service content (operations and flows) from a RAS.

To configure a RAS reference using an existing, discovered RAS

1. In the **Library**, right-click the **Configuration\Remote Action Services** folder, and then click **Open**.

The **Remote Action Services** sheet opens in the authoring pane.
2. Under **Discovered RAS**, select the discovered RAS for which you are going to create a pointer, and then click the lower **Add** button.
3. To check availability, click the left-pointing arrowhead at the top right of the RAS editor, and then click **Check Availability**.

Doing so checks for the availability of the services' URLs and displays their availability (or lack of availability) in the **Availability** column.
4. When the RAS is available, save your changes and close the editor.

Creating operations from a RAS

To create operations from a RAS

1. In Studio, create or highlight a folder in which you want to create operations from the Web service.
2. From the **File** menu, point to **Create Operations from RAS**.

If any editors or Properties sheets are open, you are prompted to close them.

After you do so, the **Import RAS** dialog box appears.
3. In the drop-down list box, select the RAS that contains the operations you're interested in, and then click **OK**.

A progress box monitors creation of the operations. After all the operations are created, the folders containing the RAS operations appear in the folder that you selected in step 2.

If you have already created these operations in this folder, you are prompted to confirm whether you want to overwrite the operations that are already in the folder.

To add an operation that is not included in the default RAS

1. Create an IAction that performs the task you want an operation to accomplish and store the IAction in a dynamic-link library (DLL) or JAR file.

OR

Obtain a third-party DLL or JAR that contains the desired operation.

2. To add the DLL or JAR to the RAS, copy it to the appropriate one of the following locations within the HP OO home directory:

- A DLL (added to RAS), whether it is your own or comes from a third party to the following location:

`\RAS\.NET\Default\RCAgentService\services\bin\Actions\`

- A JAR:

`\RAS\Java\Default\repository\`

- A third-party JAR (added to RAS):

`\RAS\Java\Default\webapp\WEB_INF\lib\`

Note: Because all RAS actions have to implement the IAction interface, in order to make the third-party library work, you probably must build an IAction class that contains the library.

3. To make sure that your remote or extended RAS operations function correctly:

- After the RAS that you need is installed and configured in Studio, be sure to check its availability in Studio.

For information on checking the availability of an RAS, see [Checking the availability of an RAS](#).

- You may also want to make sure that the RAS you're using supports your operation. To do so, you can import the operations in the RAS and review them. See [Creating operations from RASs](#).

Note: By default, the operation that you create from the IAction bases any responses only on IAction return codes. However, after you have created the operation, you can create response rules that test any other output of the IAction.

Removing a RAS reference

To remove a RAS reference

- In the Remote Action Services editor, select the RAS reference you want to remove, and then click **Remove**.

Importing RAS content

To import RAS content

1. Install the RAS that contains the content you're interested in.

For information on installing a RAS by itself, see [Installing Operations Orchestration Software \(InstallGuide.pdf\)](#).

2. Open Studio.

3. To import the RAS content, from the **File** menu, choose **Import** and complete the importing process as described in "Importing a flow," as described earlier in this section.
4. To make sure that the RAS you're using supports your operation, you might want to review them.

Note: The RAS (RAS_Operator_Path) that is installed with HP OO contains all the operations used in the Accelerator Packs and standard Library content that are installed by default.

Evaluating data for correct format

Evaluators are string formats used to validate inputs for any data sources except system accounts. They can be particularly useful for validating data obtained from the following data assignments for inputs:

- User prompt
- Specific value
If the specific value contains a flow variable (such as `${domain}`), the evaluator is applied to the specific value with the variable's value (rather than the variable reference) included in the specific value.
- Previous step's result

Studio has by default system evaluators for validating the following:


- Alphanumeric
- Email
- Filename
- IP address
- No white space
- Numeric
- Phone number

Evaluators can use standard evaluators such as `=`, `!=`, `Begins with`, `Contains`, `Match All Words`, `Match At Least One Word`, etc., or they can use evaluation tools such as the following:

- Regular expressions
For information on creating regular expressions, see [Working with regular expressions](#).
- Scriptlets
- References to shared evaluators

Creating an evaluator

To create an input-validation evaluator

1. In the Library, right-click the **Configuration\Evaluators** folder and click **New**.
OR
Click **Open**, and in the **System Evaluators** sheet that appears, click **Add**.
2. In the **Select Evaluator** drop-down list that appears, select an evaluator or evaluation tool and click **OK**.
3. In the next box, name the evaluator and click **OK**.
The new evaluator appears in the list of evaluators in the **System Evaluators** sheet.
4. To complete the evaluator, click the right-pointing arrow () at the right end of the row.

The system evaluator editor opens. Its appearance depends on the kind of evaluator you

5. In the **Compare To** box, type the string that you want to use the evaluator to test the input.
6. To clear the contents of the **Test Filter Input** box, click **Clear**.
7. Test the evaluator by either:
 - Typing a sample of the text you want to validate
 - Click **Quick Command**; in the **Input** box that appears, type the command and arguments that obtain output that you want to validate or test; and then click **OK**.
8. Save your work.

For instance, to create an evaluator that validated input for URL format, you could select the evaluator **Begins With** and then, in the **Test** text box, type **http://www**. Or you could describe the start of a URL with a Regular Expression.

Editing an evaluator

To modify an evaluator

1. In the Library, open the **Configuration\Evaluators** folder.
2. Right-click the evaluator you want to modify and click **Open**.
3. Make changes as you did in the procedure for creating an evaluator.
4. Save your work.

To edit another evaluator without closing the editor

- Click **Previous Item** or **Next Item**.

Deleting an evaluator

To delete an evaluator

1. Open the **System Evaluators** sheet.
2. In the list of evaluators, select the one you want to delete, and then click **Remove**.
3. Save your work.

Recording values for reporting in Dashboard charts

Recording data from a flow for use in Central Dashboard charts takes place in the inputs for a single step in the flow. On the step, you create inputs that get their data from the flow variables where you stored the data you want to record. Because all the values must be recorded on one step, that step must follow the last step in which one of the values you want to report is generated. This is often the Success return step.

You specify a domain term to record the input's value as. An Central Dashboard bar chart then has the information it needs to record the input's value in this flow. The domain term is either the vertical or the horizontal axis of the bar chart, and the input value is charted on that axis. The chart might depict the names of flows on the horizontal axis and the servers that the flows were run against on the vertical axis.

The data that you want to correlate for reporting purposes must all be input values on a single step. Therefore, the step must follow the last of the steps that obtained the data that you want to record for reporting and have correlated with each other. This means that return steps and steps that contain remedies for a problem are frequently the steps in which all the necessary inputs for

Dashboard reporting are both defined and recorded. You can create inputs on these steps that don't do anything except record values.

For example, suppose that an IT manager wants to see, in Central, a Dashboard chart that shows which servers the Restart Windows Server was run against and what the actions were. For either the Ping or the TraceRoute step, you could record the host input as a domain term that one of the Dashboard charts in Central uses. One such domain term would be "Configuration Item," which obtains the names or IP addresses of applications, and hardware components, including servers. The IT manager could choose the **Alerts Per Configuration Item** chart, which would show the IP addresses or domain names of the servers that the flow was run against. If the manager adds the **Flows Per Configuration Item** chart to the Dashboard, he or she can see instantly which flows have been run against those servers. Further, by double-clicking a bar representing Restart Windows Server flow when it ran against a certain server, he can see the results of the flow.

There are many other possible uses for which you can use reporting. For instance, if you have a step that takes an action on the target server such as restarting the server, you could record the appropriate input as an Action. In order to see how many times a given server was restarted, in Central you could then view the **All CI's Organized by Action** chart.

Let's see how we're going to record this information for reporting. We'll record this input with the domain term Configuration Item.

In Central, you can then add a chart that charts that domain term. For information on viewing and creating Dashboard charts, see Help for Central.

To record an input for Dashboard reporting

- With the **Inputs** tab open for the input you want to record, select **Record for reporting** and in the drop-down list select the domain term under which you want it recorded.

The value of the input for this flow will be reported to Central. Any Dashboard chart that has an axis that presents data recorded for the domain term that you chose will show this inputs value on that axis.

Domain terms for Dashboard charting

Domain terms are attributes that you can assign to flows and inputs, usually to be associated with specific data that the flows obtain, the kind of IT component and the specific IT component that the flow ran against, the type of trouble ticket that prompted the running of the flow, the action that the flow took, etc. Once associated with such specific data, domain terms add to the capability of Central as a diagnostic tool for your IT infrastructure.

Suppose that you run a flow that takes an IP address as that of the server that it will run against and you assign the domain term Configuration Item to that input. After the flow has run in Central, the Central user could view charts that report other factors against Configuration Items, such as the **Flows per Configuration Item for the Last 7 Days chart**. If the flow ran against 192.118.55.109, there would be a row in the chart labeled "192.118.55.109," with bars that break down the flows that ran against that server.

In Central, on the **Dashboard** tab, users can view charts that collate different combinations of domain terms and the values of the inputs associated with them to bring to the surface analyses that reveal trends in your IT infrastructure, such as:

- Overall usage and results of flows
- What kinds of and how many alerts or events have occurred
- Configuration items (servers, applications, etc.) that have been affected.

The domain terms that you can use for reporting data to Dashboard charts include the following. Some of them have values by default; others obtain their values from the flow's inputs; yet others have the values that you create for them.

- **Action**
The action that the flow performed. You add these values to the list, then enter them as values of inputs that are associated with this term.
- **Alert**
The identity of the alert that you want to associate with the flow. The value associated with Incident can be supplied by a flow input.
- **Category**
The category of the flow. A number of categories are supplied by default; you can add to the list of categories.
- **CI Minor Type**
These terms could be the divisions of CI Types (see next domain term). For instance, if "network hardware" is a CI Type term, then CI Minor Type terms might include "router," "server," "gateway," etc. You add these values to the list, then enter them as values of inputs that are associated with this term.
- **CI Type**
These terms could be the major divisions of IT infrastructure elements, such as "network hardware." You add these values to the list, then enter them as values of inputs that are associated with this term.
- **Configuration Item**
Specific servers, routers, switchers, applications or other elements in your IT establishment. The value associated with Configuration Item can be supplied by a flow input.
- **Incident**
The incident that triggered the running of the flow. The value associated with Incident can be supplied by a flow input.
- **Problem**
The issue that the running of the flow is meant to correct. You add these values to the list, then enter as values of inputs that are associated with this term.
- **Severity**
The severity of the problem, incident, or alert that triggered the flow.

Adding domain terms and domain term categories

To add a domain-term category

1. In the **Library** pane, expand the **Configuration** folder, right-click the **Domain Terms** folder, and click **New**.
2. Name the new domain-term category and save your work.

To add a domain term

1. In the **Library** pane, expand the **Configuration** and **Domain Terms** folders, and double-click the domain term category that you want to add the term to.
2. In the domain term editor that appears in the authoring pane, click **Add**, then type the term and its description in the **Name** and **Description** columns, respectively.

Displaying messages to users

You can display informative messages to the flow user in the step as well as on transitions (for information on providing steps to users on transitions, see [Transitions: connecting steps.](#))

To display a message to the flow user on a step

1. On the **Display** tab of the step's Inspector, select **Always prompt user before executing this step.**

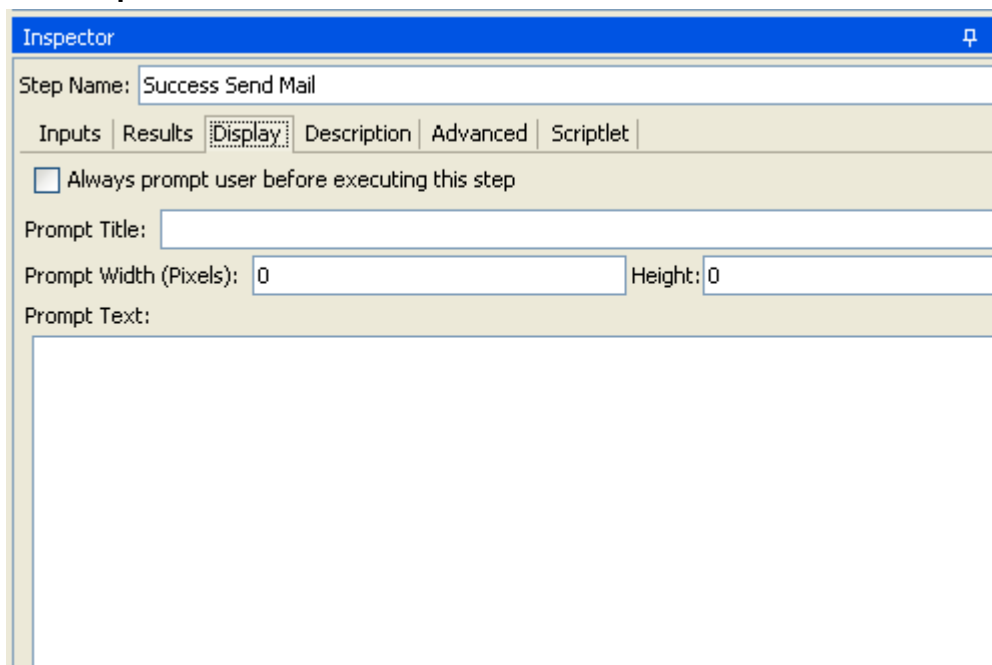


Figure 74 - Specifying a display for users

2. To provide a title for the message, in the **Prompt Title** box, type the title.
3. To specify a width and height for the display, type the width and height, in pixels, of the message or prompt of the appropriate text box.
4. In the **Prompt Text** box, type the text to be displayed.
5. Save your changes.

Categories: classifying flows

Categories are a kind of domain term that you assign to flows rather than to inputs. Assigning a category to a flow adds another factor that Central users can use in creating Dashboard charts. A number of categories are installed with Studio, but you can also create your own categories.

Central users might use categories to create reports that indicate the health of key infrastructure components. For instance, if you assign the category "Server" to all the flows that check server health, then a report that finds only flows that were assigned the Server category could highlight the health of the servers on your network.

To assign a category to a flow

1. To open the flow **Properties** sheet, click the **Properties** tab at the bottom of the flow diagram.
2. Click the **Assign Categories** button, then in the following dialog, select one or more categories, and then click **OK**.

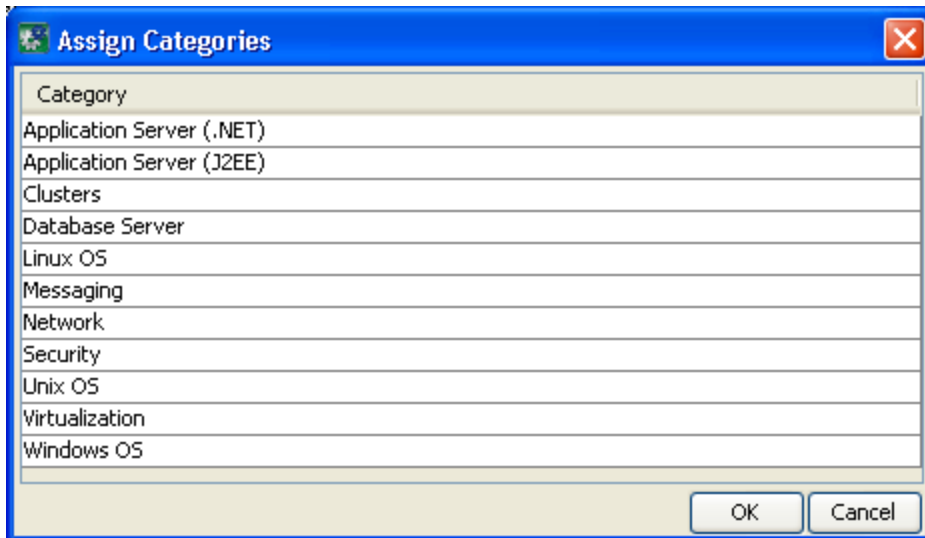


Figure 75 - Assigning a flow to a category

You can create new categories as you can any other domain term. For information on adding new domain terms, see [Domain terms for Dashboard charting](#).

Selection lists for user prompts

Selection lists are lists of items that you can provide in flow user prompts. There are a broad variety of selection lists provide by default, many of them provided specific for working with various technologies. For instance, if the flow user needs to provide a step in the flow with the service status, you can create an input whose data source is a selection list and specify the ServiceStatus selection list (whose members are Running, Stopped, and Paused). Or to capture the SQL Server authentication type, the input's data-source selection list could be SqlAuthentication (Windows, Sql).

Selection lists are stored in the Library, in the **Configuration\Selection Lists** folder.

To create a selection list

1. In the Library pane, right-click the **Configuration\Selection Lists** folder and click **New**.
2. In the box that appears, name the new selection list.

The selection list editor appears.

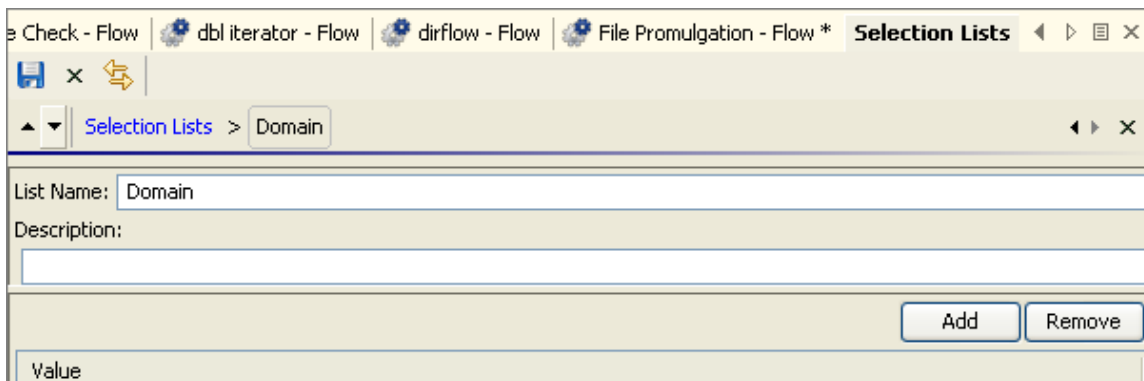



Figure 76 - Selection list editor

3. Give the selection list a description, if you wish.
4. To add an item to the list, click **Add** and name the item.

5. To remove an item, click **Remove**.
6. To close the selection list editor, click the **X** that is on the same level with **Selection Lists**.

To edit a selection list

1. In the **Library** pane, right-click the **Configuration\Selection Lists** folder and click **Open**. The list of selection lists appears.
2. To edit the list, click the right-pointing arrow () at the right end of the list's row.
3. To add an item to the list, click **Add** and name the item.
4. To remove an item, click **Remove**.
5. To close the selection list editor, click the **X** that is on the same level with **Selection Lists**.

To remove a selection list

1. In the Library pane, click the plus sign to the **Configuration\Selection Lists** folder.
2. Right-click the selection list that you want to remove, then click **Remove**.
3. Click **OK**, then save your work.

System accounts: secure credentials

A system account is an object that contains an account's credentials (user name and password), while protecting the credentials from being viewed other than in the installation of Studio on which the system account was created.

The flow author provides the system account name to the input when creating the flow; the user never sees the system account name that provides a flow with user account credentials for access to a remote machine. Also, the user cannot enter a system account name in response to a user prompt. Thus the credentials are protected from decryption, and the system account name is hidden from the user.

System accounts are stored in the Library, in the **Configuration\System Accounts** folder.

Creating a system account

To create a system account

1. In the Library pane, right-click the **Configuration\System Accounts** folder and click **New**.
2. In the box that appears, name the new system account.
The system account editor appears.

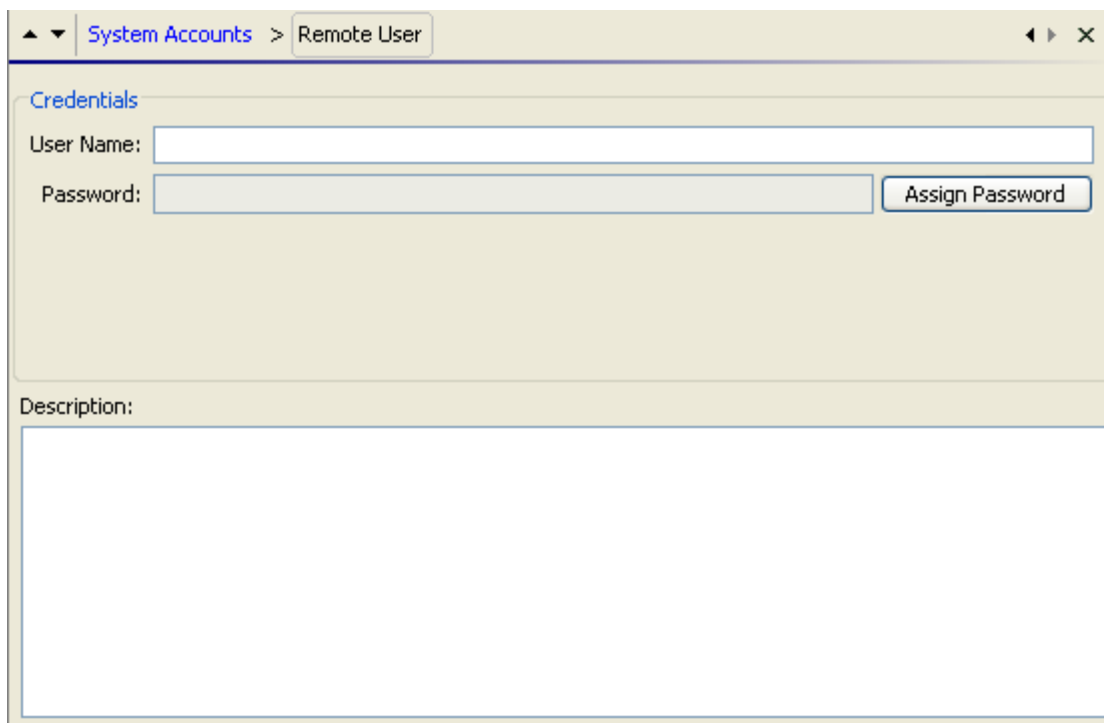


Figure 77 - System account editor

3. Type the user name of the account that the system account represents, using the following syntax:
`<domain>\<username>`
4. Click **Assign Password** and in the box that appears, type the password, then type it again when prompted.
5. Give the system account a description, if you wish.
6. Click the Save icon and then, to close the system account editor, click the **X** that is on the same level with **System Account**.

Editing a system account

To edit a system account

1. In the Library pane, open the **Configuration\System Accounts** folder, highlight the system account that you want to edit, and click **Open**.
2. Make any changes necessary and save your work.

Deleting a system account

To delete a system account

1. In the Library pane, open the **Configuration\System Accounts** folder, right-click the system account that you want to remove and click **Remove**.
2. Save your work.

Controlling access to HP OO objects

Permissions are access rights to individual objects, such as individual folders, flows, operations, or system accounts.

You assign permissions to groups (ADMINISTRATOR, AUDITOR, LEVEL_ONE, LEVEL_TWO, and LEVEL_THREE), rather than individuals. So if you publish your repository but don't want another author who updates the repository to be able to work on the flow, you could specify that that author's group does not have write permission for the flow. For information on defining HP OO roles or mapping external roles or groups to the HP OO groups described here, see the HP OO *Administration Guide* and Help for Central.

You can also set default access permissions for groups, so that they have the permissions you specify for every new object that you create. For instance, to allow a group to debug but not author flows, you could grant its members **Read**, **Execute**, and **Link** permissions, but not **Write**.

For procedures for managing group membership and managing capabilities, see Help for Central.

Note: You cannot specify permissions for the ADMINISTRATOR role, because it has all permission for all the objects in HP OO, and these permissions cannot be revoked.

Following are the permissions that are applied to objects:

Read

Only with this permission can an author see an object in Studio or a user see a flow in Central.

Write

Required for an author to modify or delete an object.

Execute

Enables an author or Central user to start a run of the flow, in either Studio or Central. This is not a recursive requirement. That is, for a Central user to run a flow or for an author to debug a flow, he or she does not have to have execute permission for every object, such as operations and configurable items such as system accounts, associated with the flow.

Link To

Allows an author to use an object in a flow, such as creating a step from a flow or operation.

Thus, to find and run a flow in Central, users must have read and execute permissions for the flow.

In Studio:

- To debug a flow, an author must have the execute permission for that flow.
- A flow author must have the Link permission for any flow or operation from which he or she creates a step in a flow.
- To change a system account, an author must have the Read and Write permissions for the system account.

The following two tables describe the permissions and which of them are needed for objects in Studio.

HP OO objects and the permissions needed to work with them

Object	Action	Necessary permission(s)
Folder		Read
	View contents	Read, Write
	Add to contents	Read, Write (also needed for all children of the folder)

	Move	Read, Write
	Rename	Read, Write
Flow or operation		
	View/open	Read
	Modify	Read, Write
	Rename	Read, Write
	Execute/Run	Read, Execute
	Use as a step or sub-flow	Link To
System accounts, selection lists, and other objects in the Configuration folder		
	View account name	Read
	Change account password	Read + Write
	Rename account	Read + Write
	Use in flow or operation	Link To
	Use at runtime	Execute

Setting permissions for folders and HP OO objects

You can set permissions directly for:

- Folders
- Flows and operations
- System accounts

How your changes to permissions are applied differs slightly, according to whether you apply them to an object or a folder:

- For a folder, you can choose to apply the permissions to all the contents of the folder, recursively.
- For a flow or operation, you can choose to apply the permissions to all the flows, operations, and system objects (domain terms and other items in the Configuration folder) that reference or are referenced by the flow or operation for which you're defining permissions. References can be indirect as well as direct.

Note: While you can set permissions directly for system accounts, you can only implicitly set permissions for all the other system objects in the **Configuration** folder (by specifying that the permissions you set for a flow or operation be applied to all the objects that the flow or operation references).

To set permissions for folders or HP OO objects

1. In the Library, right-click the operation, flow, or system account and, on the right-click menu, click **Permissions**.

A dialog box similar to the following appears.

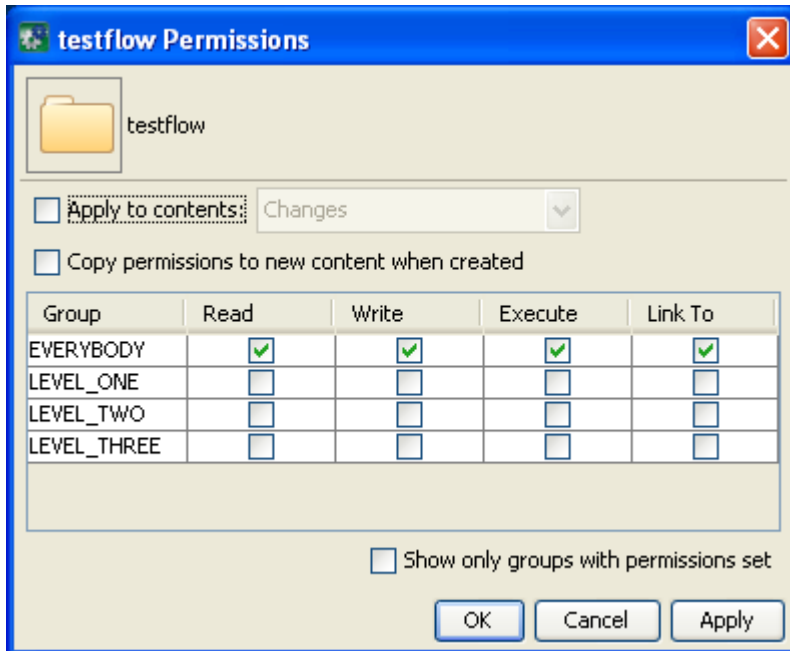


Figure 78 – Setting permissions

Tip: You can hide the groups for which there are no permissions set by selecting the **Show only groups with permissions set** box.

2. Select permissions for each group of users according to the access that you want them to have.

Notes:

- When you change a permission for a group, the box for that permission is outlined in black.
- For each parent folder of the flow, specify the same permissions that you did for the flow.

The **Apply to** box appears as **Apply to contents** or **Apply to referenced objects**, depending on what kind of object you're specifying access permissions for:

- For folders, it reads, **Apply to contents**.
- For flows, operations, and system accounts, it reads, **Apply to referenced objects**.

In this dialog, "Referenced objects" means not only flows, operations, and system objects that this object refers to, but also flows and operations that refer to this object.

3. If you're setting permissions for a folder, to apply the folder's access permissions to all the flows, operations, and subfolders (and their contents), select **Apply to contents**.

OR

If you're setting permissions for a flow, operation, or system account, to apply the object's access permissions to all the flows, operations, and other system accounts that refer to the object or that the object refers to, select **Apply to referenced objects**.

Now, do you want to apply only the changes in permissions that you have made and not yet applied (those which are bold in the grid), or all the permissions that currently appear in the matrix?

4. To apply only the changes that have not already been applied to the folder or object, in the drop-down list beside the **Apply to...** box, select **Changes**.

OR

To apply all the permissions currently specified in the grid, including the settings you have just specified, then in the drop-down list beside the **Apply to...** box, select **All**.

The following step assumes that you are specifying permissions for a folder.

5. To apply these permissions to any flows, operations, or system accounts that are subsequently created within the folder, select **Copy permissions to new content when created**.

When you select this checkbox, the permissions that you have set for the folder override default permissions that you have set for a group. For example, suppose you have:

- Set the **Write** permission in the default group mask for the LEVEL_THREE group.
- In the permissions for folder A, set only the **Read** permission for the LEVEL_THREE group.
- Specified that the folder A permissions be applied to anything created in the folder in the future.

If you then create a flow in folder A, the LEVEL_THREE group will have **Read** but not **Write** permission for the new flow. (You can subsequently change the LEVEL_THREE group's permissions for the new flow.)

For information on setting default permissions for a group, see [Setting default access permissions for groups](#).

6. Click **OK** and save your work.

Setting default access permissions for groups

You can set the access permissions that groups have by default for any new objects that are created. Sets of default access permissions are called *group masks*. Note, however, that if a group mask conflicts with access permissions that have been set for any objects as they are created in a folder, then the access-permission settings for that folder prevail.

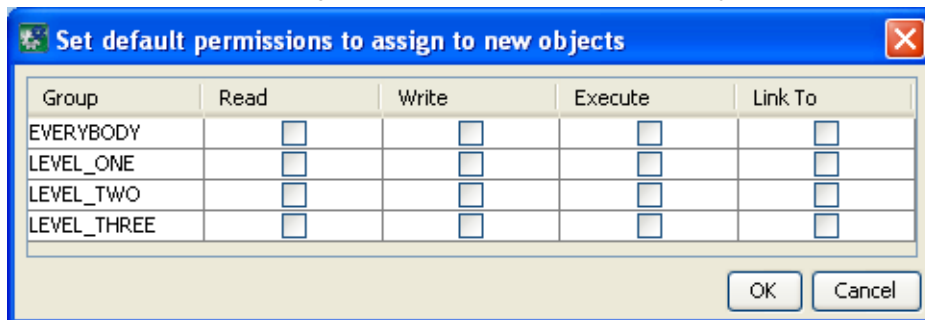
For example, suppose you do the following:

- In the default group mask for the LEVEL_THREE group, set the **Write** permission.
- In the permissions for folder A, set only the **Read** permission for the LEVEL_THREE group.
- Specify that the folder A permissions be applied to anything created in the folder in the future.

If you then create a flow in folder A, the LEVEL_THREE group will have **Read** but not **Write** permission for the new flow. (You can subsequently change the LEVEL_THREE group's permissions for the new flow.)

To set default access permissions for groups

1. On the **Tools** menu, click **Set Group Mask**.
2. In the following dialog, select the permissions for each group that you want that group to have in Studio and Central by default on all new HP OO objects, and then click **OK**.



Creating a new operation

Depending on the kind of operation you need, creating the operation involves some of the following, which are treated elsewhere in this Help system:

- Defining [Inputs: Providing data to operations](#) and [Outputs, responses, and step results](#) for steps and operations
- Writing a [scriptlet](#)
- Specifying [responses](#) that determine transitions and mapping results to the responses
- Assigning the operation's [access permissions](#) to HP OO roles as appropriate
- Storing output data in [flow variables](#)

A valid operation requires:

- One or more inputs, with a defined data source for each input.
- Responses that are mapped to valid expressions describing outcomes of the operation.
- Properties specified as necessary for the type of operation.

After you create an operation, you complete the operation's **Properties** sheet. The following procedure tells you about the Properties sheet items that are common to all the operation types. After you finish this procedure, specify the properties that are particular to the operation type that you have created. Operation-specific properties are described in the list in [Types of operations: setting properties](#), following this procedure.

To create an operation

1. Right-click the folder in which you want to create the operation, point to **New**, then to **Operation**, and then select the desired type of operation from the list that appears.
2. In the dialog box that appears, type a name for the new operation in the text box and then click **OK**.

Notes:

- Naming in Studio is not case-sensitive, and you cannot give two operations the same name.
 - Names can be a maximum of 128 characters long.
 - The **Properties** sheet for the new operation appears in the authoring pane.
3. To assign the operation to a category for search purposes, click **Assign Categories** and then select a category from the list.

Note: Because Central users never see operations, it is not necessary to select **Hide in Central**.

4. If the operation requires an input, click the **Inputs** tab and then click **Add Input**.
You can obtain input values from flow variables.
5. Add any necessary inputs and assign them their data sources.
6. In the dialog that appears, type the input name and then click **OK**.
For information on adding inputs, see [Inputs: Providing data to operations](#). For information on using flow variables when assigning a data source to an input, see [Flow variables: Making data available for reuse](#).
7. Add and define any output data.
For information on adding and working with output data (results), see [Outputs, responses, and step results](#).
8. Create any responses needed.
For information on defining rules that govern which responses are chosen for the operation, see [Responses: Evaluating results](#).

For information on the response definitions that are particular to each type of operation, see [Types of operations: setting properties](#), following this procedure.

9. To document the operation for the Central user, click the **Description** tab and write the description in the text box.
10. Finish specifying properties as described in the following section ([Types of operations: setting properties](#)), and then click **OK**.

In the Library, if the new flow is invalid or incomplete, its name is displayed in red type. Moving the cursor over the name of an incomplete operation displays a tool tip that specifies how the operation is incomplete. (To review the requirements for a valid operation, see the list that comes before this procedure.)

Types of operations: setting properties

The following list of the classes of operations that you can create includes:

- Guidelines for the using the operation in Central and Studio.
- A screenshot of each operation type's **Properties** sheet and notes on the properties that you must set for the operation.

For the command-line, http, secure shell, and telnet operations, you can add the op-timeout input, which is optional, and specify a time value for the input, in milliseconds. The default value is 2 minutes, or 120000 milliseconds. You add the op-timeout input as you do any other. For the steps to add an input, see the procedure "To create an operation."

Note: RAS operations cannot take op-timeout inputs, because they only act as proxies for IActions. Any operation timeouts must be programmed in the IAction.

For some of these operation types, there are various ways to create the operation. Where relevant, discussion of the operation type includes recommendations on which method to use.

In the following screen shots, note that the Properties sheet for an operation includes the operation's universally unique identifier (**UUID**) and last modified date.

Cmd (command-line), or shell operation

Use for existing commands and scripts that, outside of a flow, you would run from the command line.

Programming that you carry out with command operations sometimes can also be carried out with a scriptlet operation or with IAction programming.

- For more information on using a scriptlet, see [Scriptlet operation](#).
- For information on using IAction programming, see [Creating operations that access Web services](#), [Creating IActions for operations](#), and [RAS operation: IAction programming for remote action service \(RAS\)](#).

Tip: You can use the Shell Wizard to create a cmd operation. For information on using the Shell Wizard, see [Creating a cmd operation with the Shell Wizard](#).

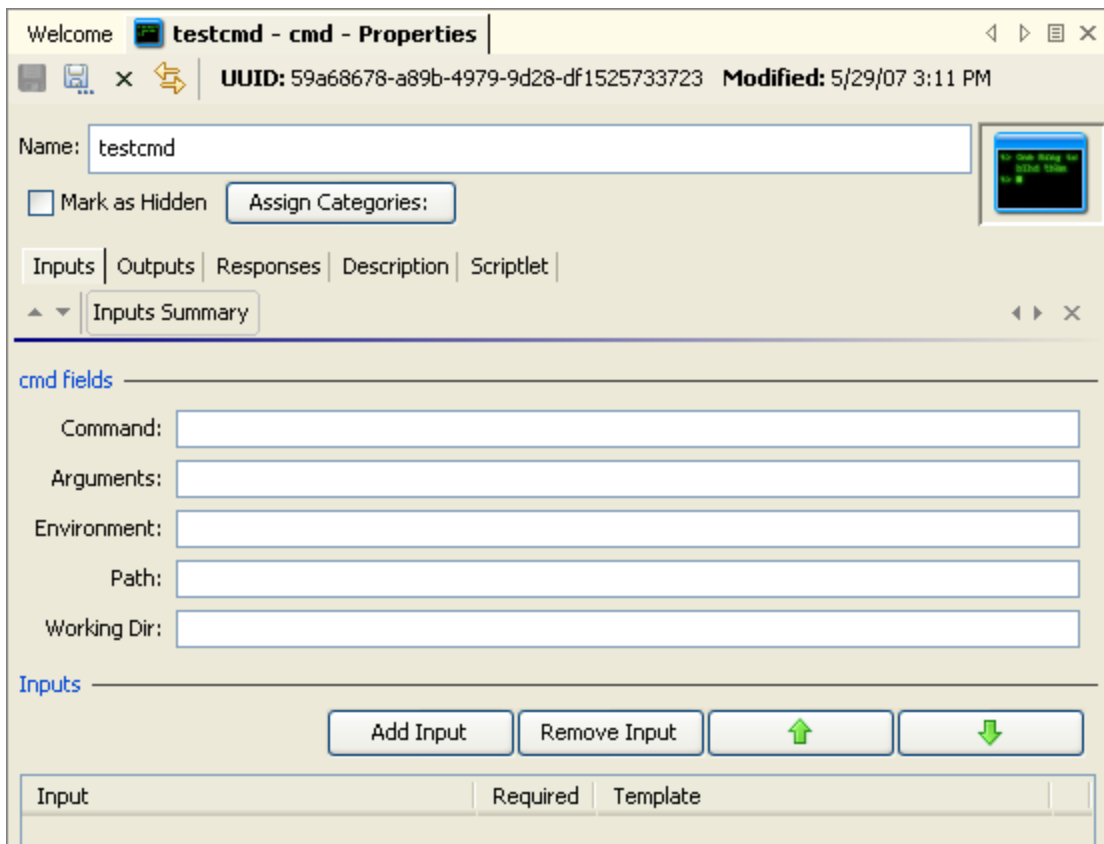


Figure 79 – Cmd (shell) operation properties

For information on the values you need to provide for the inputs, see the **Description** tab of the operation.

On the **Responses** tab, in the response editor (which you open by clicking the right-pointing arrow in the response's line), you can use the following terms in the **Apply Rule to Field** column for building statements to which a response is bound. For more information on these terms, see the Description tab of the operation, under **Extra Results**.

- Code
- Output String
- Error String
- FailureMessage
- TimedOut

Http operation

Use for simple put or get operations, retrieving docs such as log files from an intranet or extranet.

Because IAction programming would not accomplish these tasks in a significantly different way or offer advantages over a standard http operation, this is the method of choice.

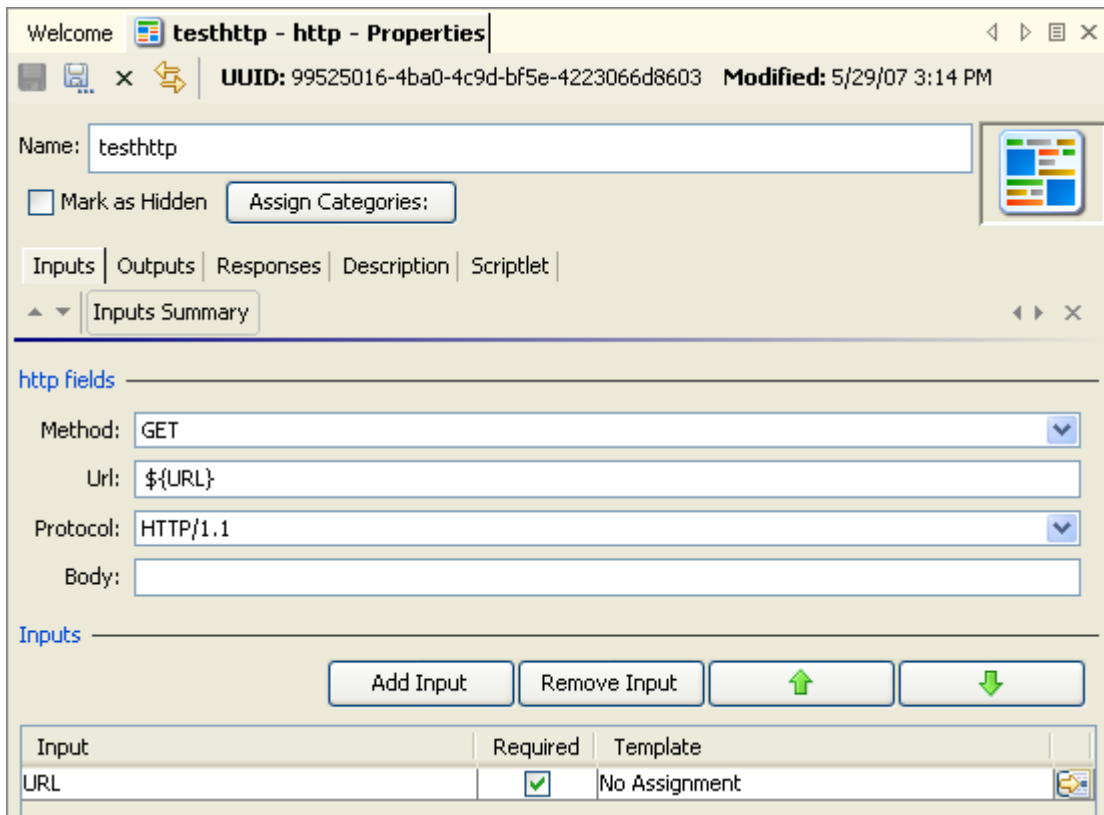


Figure 80 - Http operation properties

Following are the inputs to Http operations, and what they need for values:

Method – Http protocol methods. Pick one from the drop-down list.

Url – The URL of the target Web page.

Protocol – The version of the http protocol. Pick the version that the target server supports.

Body – If the request has a body, the data of the body, as defined in RFC 2616.

On the **Responses** tab, in the response editor (which you open by clicking the right-pointing arrow in the response's line), you can use the following terms in the **Apply Rule to Field** column for building statements to which a response is bound. For more information on these terms, see the Description tab of the operation, under **Extra Results**.

- code
Http return code as described in RFC 2616.
- reason
A text string for the return code as described in RFC 2616.
- headers
Response headers as defined in RFC 2616.
- document
The actual document that was returned by the server.
- FailureMessage
The message that you receive on failure.
- TimedOut
Compare **TimedOut** to "true" (without the quotation marks).

Perl script operation

Use for calling a Perl script that executes your Perl scriptlet. You can pass variables to and modify their values with a Perl scriptlet within the flow, then pass the modified values back to the Perl script.

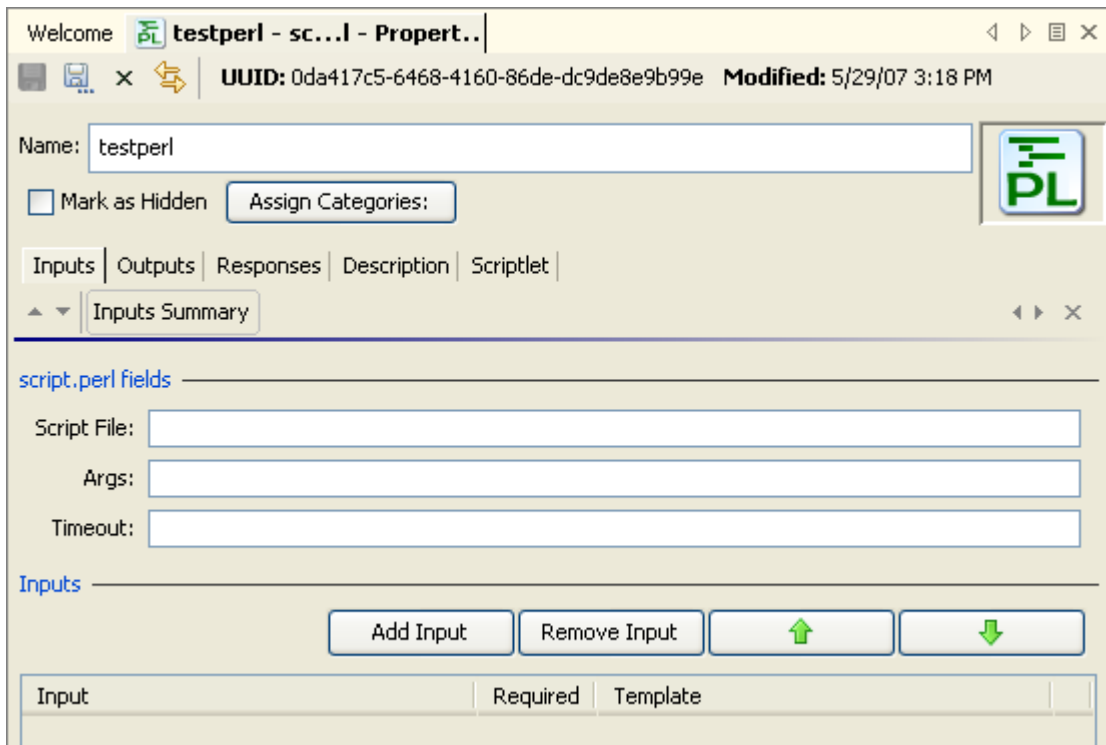


Figure 81 - Perl script operation properties

Following are the inputs to Perl script operations, and what they need for values:

Script File – The Perl script to run

Args – The arguments that you pass to the script

Timeout – The maximum time allowed for execution of the script.

On the **Responses** tab, in the response editor (which you open by clicking the right-pointing arrow in the response's line), you can use the following terms in the **Apply Rule to Field** column for building statements to which a response is bound. For more information on these terms, see the Description tab of the operation, under **Extra Results**.

- Code
The numeric return code of the operation. If the operation succeeds, this value is typically 0.
- Output String
The standard output of the operation (what the operation writes to stdout).
- Error String
The error output of the operation (what the operation writes to stderr).
- Script Response
For script operations, the string that is returned by the script.
- Script Result
For script operations, the output that is returned by the script.
- FailureMessage

The message that you receive on failure.

- TimedOut
Compare **TimedOut** to "true" (without the quotation marks).

RAS operation: Action programming for remote action service (RAS)

Use for the bulk of work in the flow, assuming that your organization has the resources for programming IAction objects.

The advantages of IAction programming include:

- More capacity for passing large amounts of data.
- Greater speed.
- Less overhead when launching processes.
- Improved performance.
- Greater scalability.
- Greater stability.
- Ease of reuse.

Programming that you carry out with scriptlet operations can also be carried out with command or IAction programming operations. For more information on when command or IAction programming operations are indicated, see their descriptions in this list.

The screenshot shows a Windows-style dialog box titled "testrasop - R...n - Propert..". The window has a standard title bar with minimize, maximize, and close buttons. Below the title bar, there is a toolbar with icons for save, print, and help, followed by the text "UUID: db19be3b-ccf7-466a-990e-b4ebf97be5c7" and "Modified: 5/29/07 3:19 PM". The main area of the dialog is divided into several sections. At the top, there is a "Name:" field containing "testrasop" and a "Mark as Hidden" checkbox. Below this is an "Assign Categories:" button and a globe icon. A tabbed interface is visible with tabs for "Inputs", "Outputs", "Responses", "Description", and "Scriptlet". The "Inputs" tab is selected, and within it, there is an "Inputs Summary" section. Below this, the "RAS Operation fields" section contains four input fields: "Action Class:", "Archive:", "RAS:" (with a globe icon), and "Override RAS:". At the bottom, there is an "Inputs" section with four buttons: "Add Input", "Remove Input", an up arrow, and a down arrow. Below these buttons is a table with columns for "Input", "Required", and "Template".

Figure 82 – RAS operation properties

Following are the inputs to Perl script operations, and what they need for values:

Action Class – The name of the action class from which the operation is to be created.

Archive – The name of the Java Archive (JAR) file or dynamic-link library (DLL) file in which the action class is contained

Service Name – The name of the RAS reference that you configured in Studio. The reference is to a default or custom RAS that you installed, on which the operation executes the action class.

On the **Responses** tab, in the response editor (which you open by clicking the right-pointing arrow in the response's line), you can use the following terms in the **Apply Rule to Field** column for building statements to which a response is bound. For more information on these terms, see the Description tab of the operation, under **Extra Results**.

- FailureMessage
The message that you receive on failure.
- TimedOut
Compare **TimedOut** to "true" (without the quotation marks).

For more information on RASs, JARs, and DLLs and how you can use them to extend the functionality of flows, see [Operating outside Central with remote action services](#) and [Creating IActions for Ops Flow Operations](#).

Scriptlet operation

Use for creating a scriptlet that you want to have available for reuse in multiple flows.

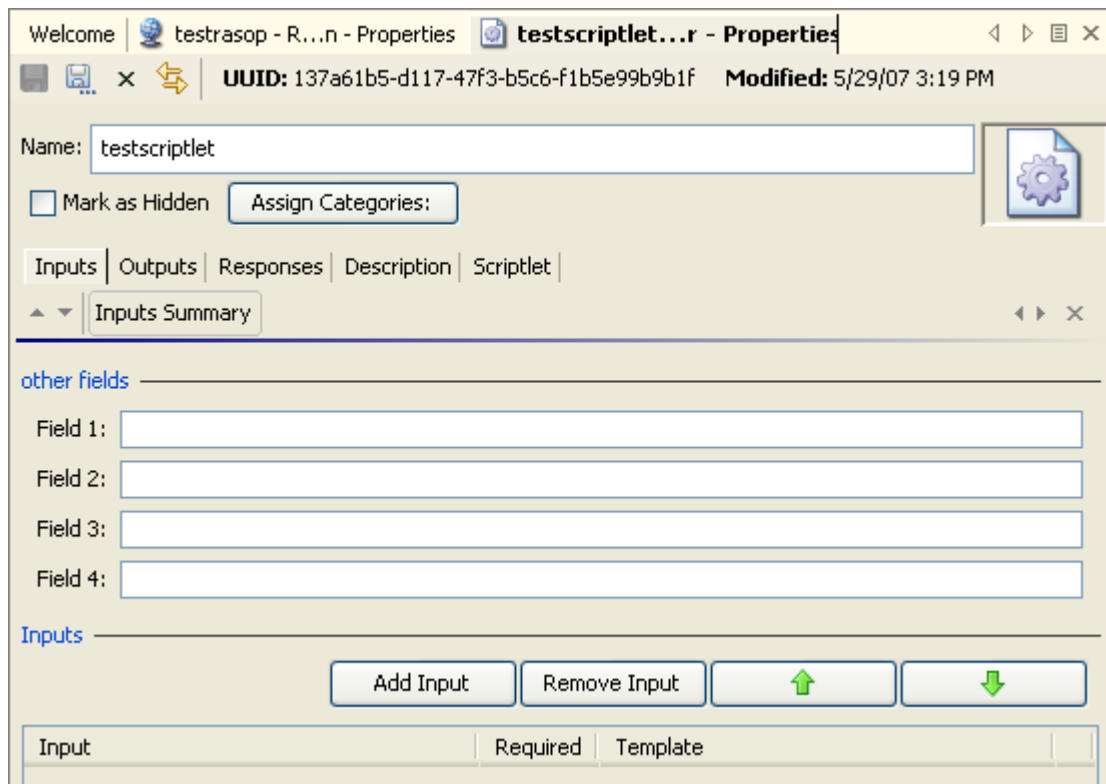


Figure 83 - Scriptlet operation properties

Use for gathering up to four pieces of data and passing them to flow variables for use elsewhere in the flow.

One of the uses of a Scriptlet operation is to provide user prompts when, in order for the flow to run fully automatically, the step must be able to alternatively obtain the prompt's input value from a flow variable.

Following are the inputs to Perl script operations, and what they need for values:

Fields 1 through 4 – The four fields that you can populate using the inputs that you define for this operation.

On the **Responses** tab, in the response editor (which you open by clicking the right-pointing arrow in the response's line), you can use the following terms in the **Apply Rule to Field** column for building statements to which a response is bound. For more information on these terms, see the Description tab of the operation, under **Extra Results**.

- **Fields 1 through 4**
The contents of the four fields in which you have stored the data from the inputs.
- **FailureMessage**
The message that you receive on failure.
- **TimedOut**
Compare **TimedOut** to "true" (without the quotation marks).
- **Result**
The output that is returned by the script.

Secure shell (ssh) operation

Use for remote commands over a secure channel.

For example, if you wanted to start a service on a machine whose operating system is a Red Hat version of linux, you could use the Start Service flow (in the Library, under Accelerator Packs\Operating Systems\Red Hat\State change\). The Start Service flow uses the Run Service operation (in the Library, under Operations\Linux\SUSE Linux\Process Operations\), which is an SSH operation. To see how the inputs, outputs, and responses for the Run Service operation are configured for this usage, open the operation.

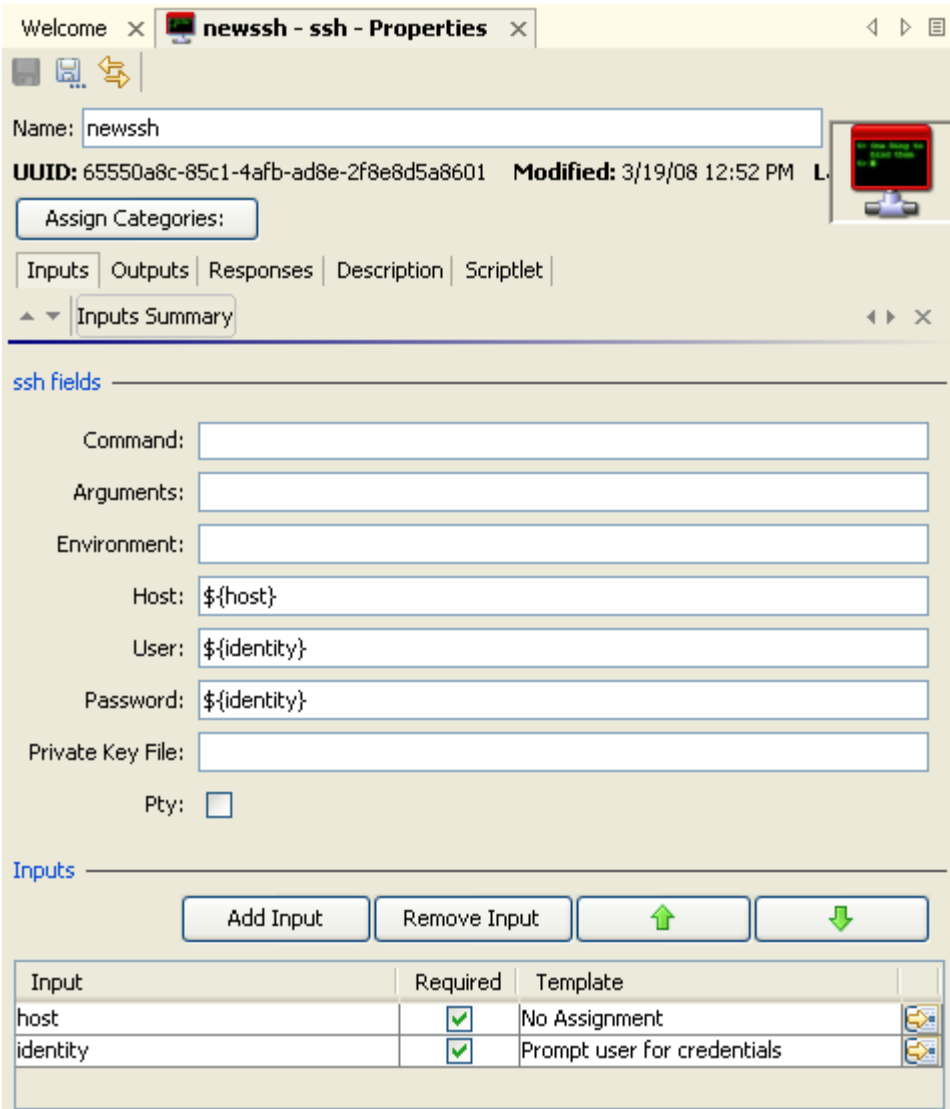


Figure 84 - Secure shell operation properties

For information on the values you need to provide for the inputs, see the **Description** tab of the operation.

Notes:

- In the Host input, to specify a particular port for the host, specify the host and port with the following syntax:
`<hostname>:<portname>`
- The **Pty** checkbox creates a pseudoterminal so that Unix operations that require a terminal can be run from a secure shell operation that you create.

Warning: A pseudoterminal does not distinguish between the standard output (or output string) and standard error (or error string) but takes both. As a result, the error string is always empty.

On the **Responses** tab, in the response editor (which you open by clicking the right-pointing arrow in the response's line), you can use the following terms in the **Apply Rule to Field** column for building statements to which a response is bound. For more information on these terms, see the Description tab of the operation, under **Extra Results**.

- Code
- Output String

- Error String
- FailureMessage
- TimedOut

Telnet operation

Use for sending messages or commands to a server using the telnet protocol.

The screenshot shows a window titled "testtelnet - ...t - Propert.." with a yellow title bar. The window contains several configuration fields and buttons:

- Name:** testtelnet
- Mark as Hidden
- Assign Categories: [button]
- Inputs | Outputs | Responses | Description | Scriptlet | [tabs]
- Inputs Summary [dropdown]
- telnet fields** section:
 - Command: [text box]
 - Arguments: [text box]
 - Environment: [text box]
 - Path: [text box]
 - Host: [text box]
 - Port: [text box]
 - Session Name: [text box]
 - Terminal Expression: [text box]
- Inputs** section:
 - Add Input [button]
 - Remove Input [button]
 - Up arrow [button]
 - Down arrow [button]
- Input table:

Input	Required	Template

Figure 85 - Telnet operation properties

Following are the inputs to Perl script operations, and what they need for values:

Command – The shell command that the operation executes

Arguments – The command-line arguments (switches)

Environment – Additions to the existing environment variables

Path – The path to the shell command

Tip: To specify a particular port for the host, specify the host and port with the following syntax:

<hostname> <portname>

Note that for telnet operations, the separator between <hostname> and <portname> is a whitespace.

Host – The target computer for the secure shell command

Port – The port through which the operation connects to the target computer

Session Name – The name for the session that you want to store in the context

On the **Responses** tab, in the response editor (which you open by clicking the right-pointing arrow in the response's line), you can use the following terms in the **Apply Rule to Field** column for building statements to which a response is bound. For more information on these terms, see the Description tab of the operation, under **Extra Results**.

- **Code**
The numeric return code of the operation. If the operation succeeds, this value is typically 0.
- **Output String**
The standard output of the operation (what the operation writes to stdout).
- **Error String**
The error output of the operation (what the operation writes to stderr).
- **FailureMessage**
The message that you receive on failure.
- **TimedOut**
Compare **TimedOut** to "true" (without the quotation marks).

Creating a cmd operation with the Shell Wizard

The Shell Wizard guides you through creating a command-line operation based on either the Secure Sockets (SSH) or telnet protocol. The wizard includes recorder-like technology that creates steps from the commands that you carry out in the shell window that the wizard opens. To run the wizard, you will need to have available the connection information on the host (the machine on which the commands you specify execute): the host name and the name and password of the user account under which the command(s) will execute on the host.

To create a cmd (or Shell) operation using the Shell Wizard

1. In the HP OO home directory, in \Studio\tools\, click or double-click **shellwizard.exe**.
The Shell Wizard starts.

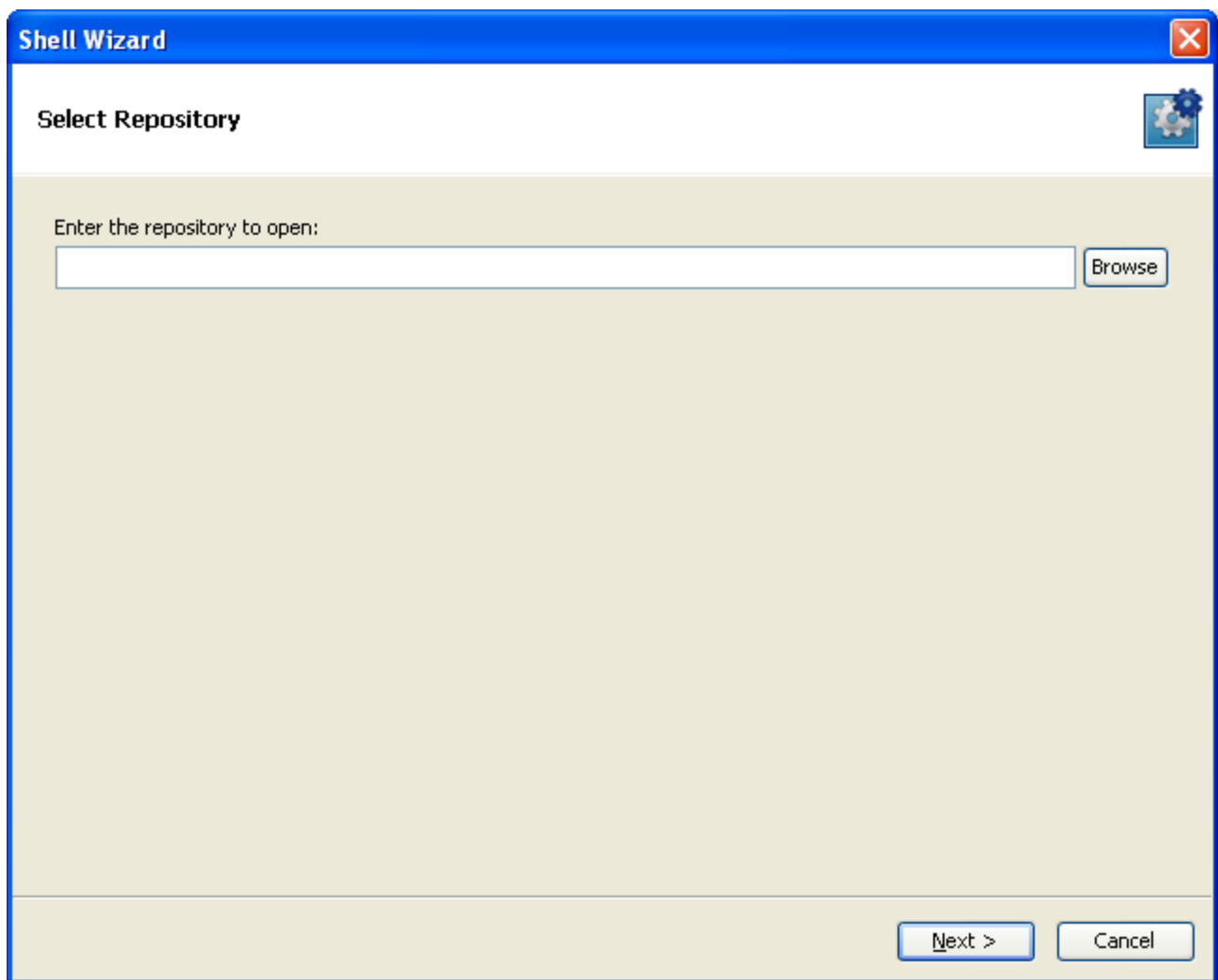


Figure 86 - The Shell Wizard

2. Type the location of the repository in which you want to create a command-line (or shell) operation.

OR

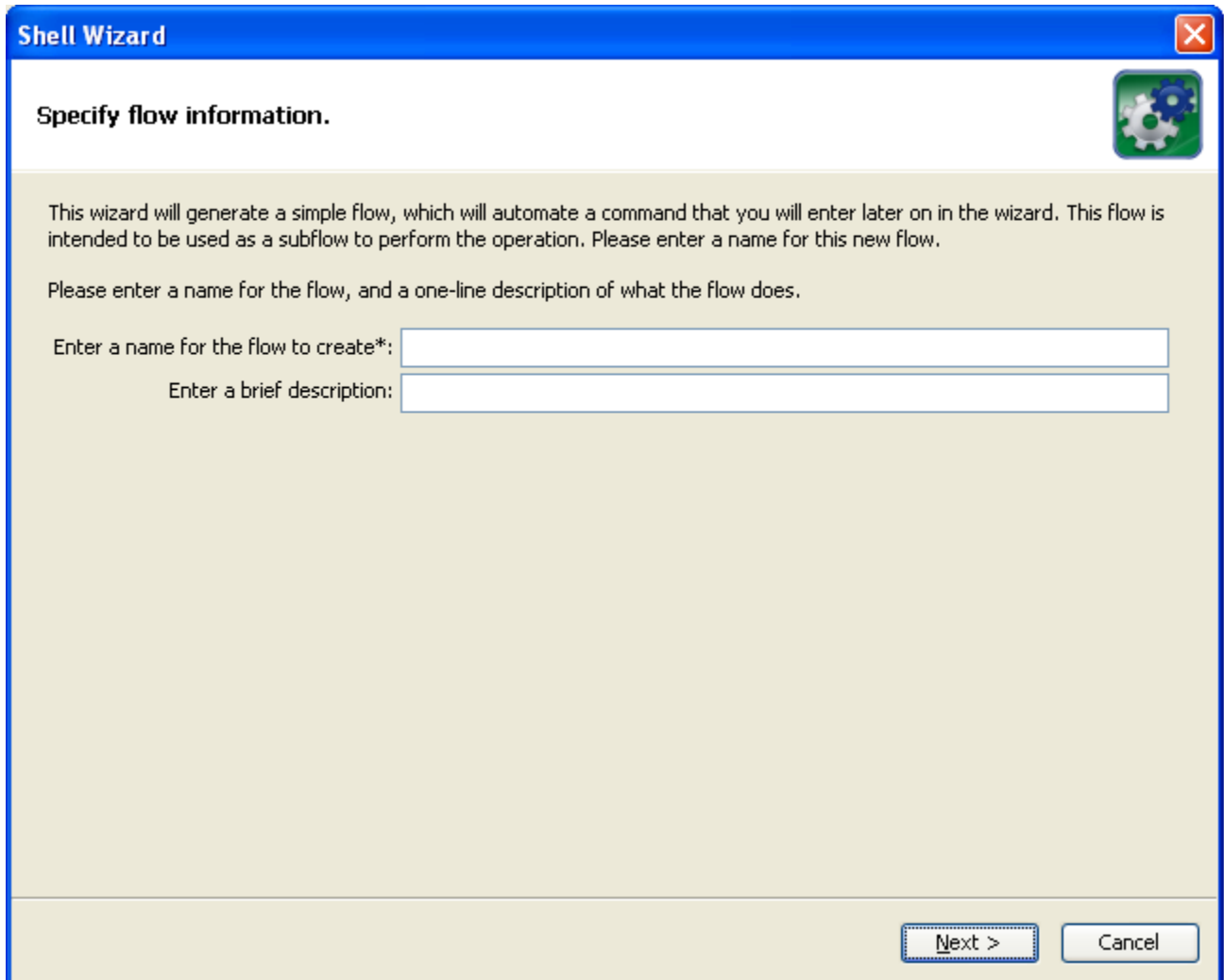
Click **Browse**, navigate to the location of the repository, and then click **Open**.

Important:

- If you point to an empty or nonexistent folder, the wizard creates a folder of that name in that location, if necessary, and creates a new repository. The new repository contains only what is needed to run the operations that you create, including either the SSH Shell or Telnet Shell operation, which enables the running of the operations that you create in this wizard.
- If you point to an existing repository, the repository must include either the SSH Shell operation (if you use the SSH protocol) or the Telnet Shell operation (if you use the telnet protocol). If it does not include the appropriate Shell operation, you will not be able to complete this wizard. The SSH Shell and Telnet Shell operations are included in Studio's default content (operations, flows, and HP OO objects). However, if you export a subfolder of the Studio Library as a repository, you must include default content in the export. If you do include default content in the export, only the default content that is used by a flow or operation in the folder that you're exporting is included in the export.

- Therefore, it is strongly recommended that you point the Shell Wizard to an empty folder (which you can then import into your Studio repository) or at an export of the entire Studio Library.

3. In the wizard, click **Next**.



The screenshot shows a dialog box titled "Shell Wizard" with a close button in the top right corner. The main heading is "Specify flow information." followed by a gear icon. Below this, there is explanatory 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. Please enter a name for this new flow." This is followed by another instruction: "Please enter a name for the flow, and a one-line description of what the flow does." There are two input fields: the first is labeled "Enter a name for the flow to create*:" and the second is labeled "Enter a brief description:". At the bottom right, there are two buttons: "Next >" and "Cancel".

4. In the **Specify flow information** page of the wizard, in the **Enter a name...** box, type a name for the flow that the wizard will create for the shell (command-line) operation.

5. Complete the **Enter a brief description** box, and then click **Next**.

Keep in mind that Central users may choose your flow based on the description that you type.

Shell Wizard

Please specify connection settings.

This wizard will create a subflow that runs through a set of steps that you provide in either a telnet or ssh session. To get started, please provide a host to connect to, credentials for that host, and select whether you wish to connect via ssh or telnet.

Connection Info

Host to Connect to*:

Username*:

Password*:

Choose a protocol

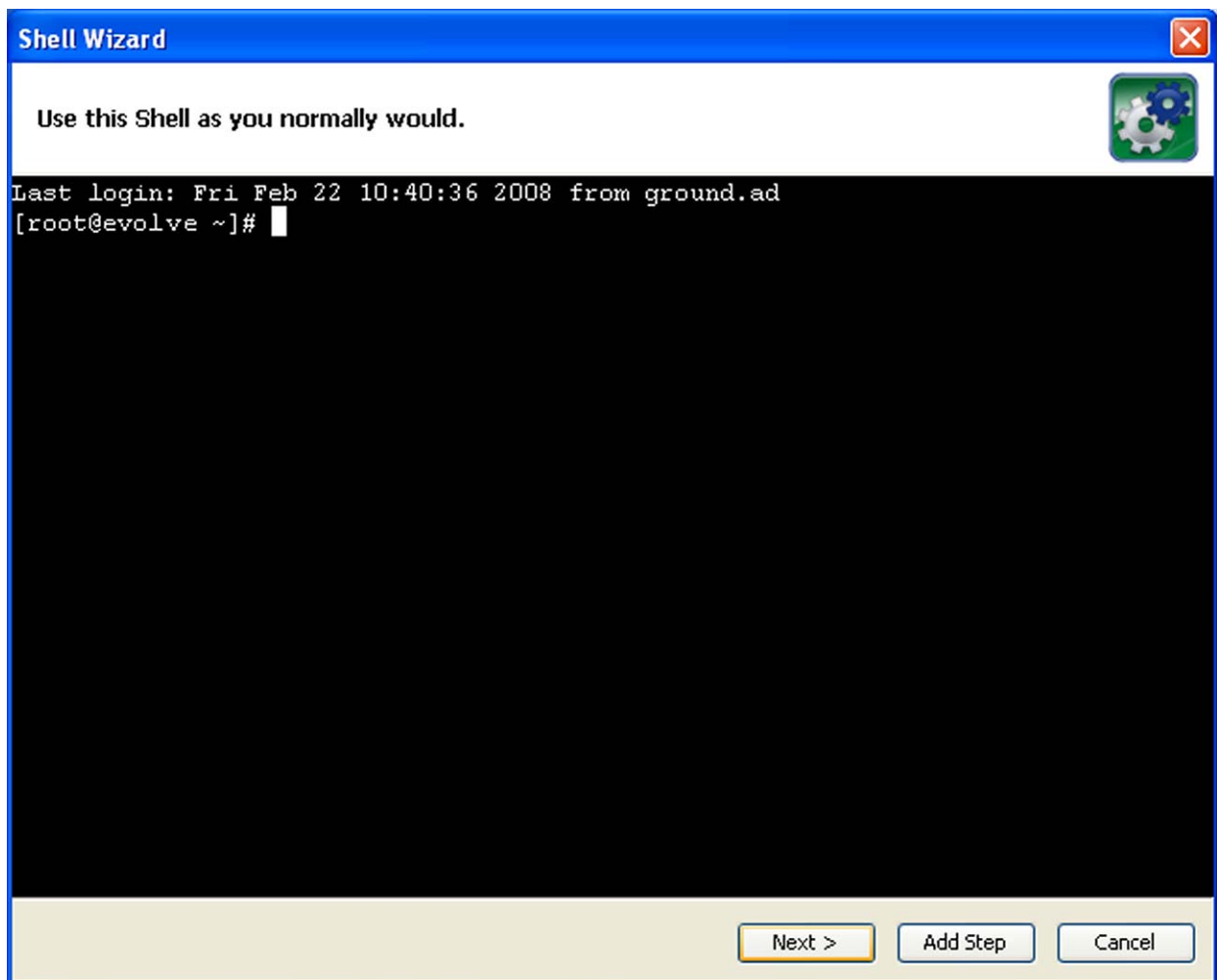
ssh

telnet

Next > **Cancel**

6. Under **Connection Info**, in the **Host to Connect to** box, type the name or IP address of the name on which you will run the command.
7. In the **Username** and **Password** boxes, type the credentials of the account name under which the command will run on the specified host.
8. Under **Choose a protocol**, select either **ssh** or **telnet**, according to which protocol you want to send the command under, and then click **Next**.

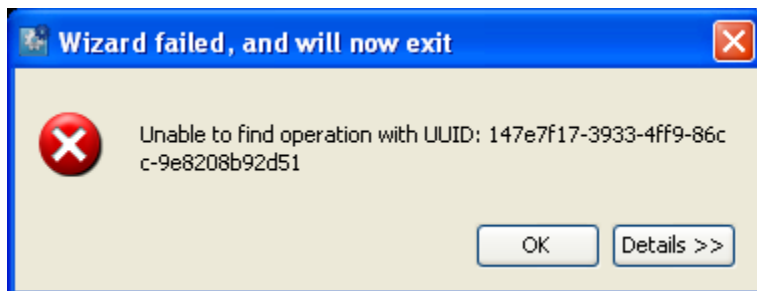
It is recommended that you select **SSH**.



9. Type the command or command sequence that you want to execute.
10. To add an operation (and step that is an instance of the operation) that executes the command sequence that you have entered, click Add Step.
You can add as many steps as you wish.
11. When you are done adding steps, click **Next**.

The wizard adds the operation and step to the flow that you specified.

If you see the following error, the cause is that you are adding the operation to an existing repository, and the repository does not include the SSH Shell operation (if you are using the SSH protocol) or the Telnet Shell operation (if you are using the telnet protocol).



Click **OK**, and then do one of the following:

- Export the default public repository (the Studio Library, not a subfolder of the Library) to the repository that you created for this wizard, and then re-run the wizard.
- Re-run the wizard, pointing it to an empty or non-existent folder.

12. In the Wizard Is Finished page, click **Finish**.

Debugging flows

Studio includes three important tools for solving problems in operations and flows:

- **Execute** flow sheet, which tracks the execution of a flow in real time and provides the raw results of the flows operations, including error strings.
- **Problems** panel, which displays problems in the currently selected flow and operations
- **Validate Repository** command (on the Tools menu), which displays, in the Problems panel, problems for all the flows and operations in the entire Library

The **Problems** panel reflects any problems that it encounters in real time, as you work.

To view problems from the entire repository

- On the **Tools** menu, choose **Validate Repository**.

As you execute a flow in order to debug it, the **Execute** sheet of the authoring pane looks like the following:

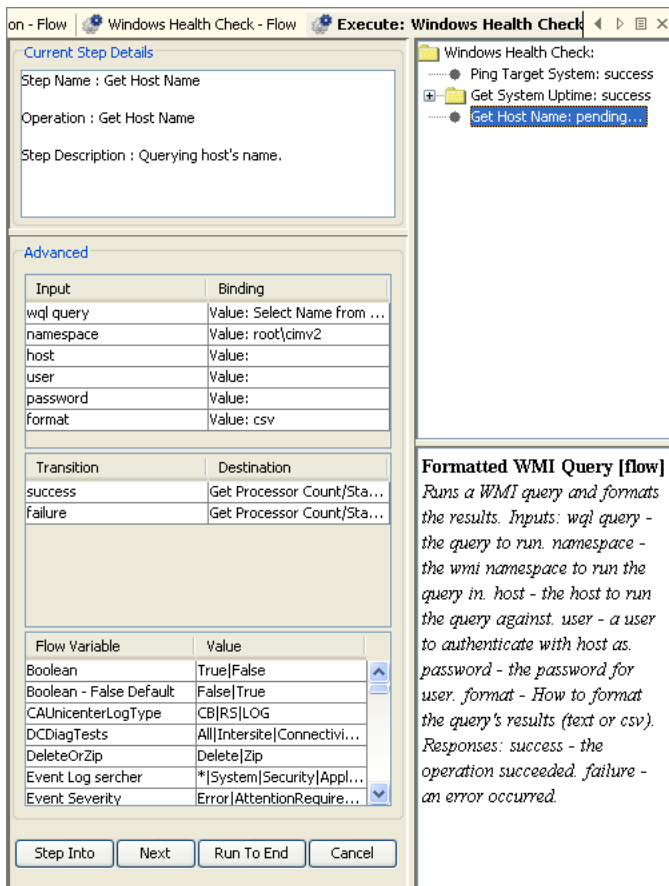


Figure 87 - Execute (debug) sheet

The **Advanced** area shows information on each step when the step is pending.

To view the raw results of a step's operation

- After the run has completed, highlight the step whose raw results are of interest. Raw results appear in the lower-right panel.

Working with regular expressions

Regular expressions (also known as regexes) are a powerful tool that you can use to:

- Create results filters that extract key pieces of data for:
 - Saving in variables for use in later operations.
 - Testing to determine a step's response.
- Validate the form of inputs.

For example, suppose the information that an input variable needs from a Ping operation is the packet loss. The output of pinging the IP address 111.111.111.111 looks like the following:

```
Pinging 111.111.111.111 with 32 bytes of data:
```

```
Reply from 111.111.111.111: bytes=32 time=27ms TTL=246
```

```
Reply from 111.111.111.111: bytes=32 time=26ms TTL=246
```

```
Reply from 111.111.111.111: bytes=32 time=29ms TTL=246
```

```
Reply from 111.111.111.111: bytes=32 time=28ms TTL=246
```

```
Ping statistics for 111.111.111.111:
```

```
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

```
Approximate round trip times in milli-seconds:
```

```
    Minimum = 26ms, Maximum = 29ms, Average = 27ms
```

To extract the number of packets lost, you can instruct HP OO to search for the string "Lost = " followed by any number. Using a regular expression will help.

A regular expression allows you to search not only for exact text but also for classes of characters as well. For example to match any digit you can use the wildcard `\d`.

Therefore, in the above example, a search using the regular expression `Lost = \d` would return the string **Lost = 0**.

The key wildcards for regular expressions are:

Wildcard	Uses
<code>^</code>	Matches the beginning of a string
<code>\$</code>	Matches the end of a string
<code>.</code>	Any character except newline
<code>\b</code>	Word boundary
<code>\B</code>	Any except a word boundary
<code>\d</code>	Any digit 0-9
<code>\D</code>	Any non-digit
<code>\n</code>	Newline
<code>\r</code>	Carriage return
<code>\s</code>	Any white space character

\S	Any non-white space character
\t	Tab
\w	Any letter, number or underscore
\W	Anything except a letter, number or underscore

Modifier	Effect
*	Match zero or more
+	Match one or more
?	Match zero or one
{n}	Match exactly n occurrences
{n,}	Match n or more occurrences
{n,m}	Match between n and m occurrences
[abc]	Match either a, b, or c
[^abc]	Match anything except a, b or c
[a-c]	Match anything between a and c
a b	Match a or b
\	Escape a special character (for example \. Means '.' not match anything)

Using these wildcards and modifiers, the following regex extracts the IP address from the ping output:

```
\d{1-3}\.\d{1-3}\.\d{1-3}\.\d{1-3}
```

When using the results filter in Studio, you can combine multiple regexes to isolate the value that you need. For example, consider the output of the Unix ps command:

```
F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD
0 S 512 21604 21603 0 75 0 - 1096 wait pts/1 00:00:00 Bash
0 R 512 2659 21604 0 76 0 - 1110 - pts/1 00:00:00 Ps
```

You could create the following filters to extract the time for the ps command:

1. In Studio, open the Unix ps operation.
2. Open the **Output String Field Filters** dialog box.
For information on opening the **Output String Field Filters** dialog box and performing the following tasks, see [Filter details](#).

Extracting the time for ps requires two regexes, one to filter the output down to the line for ps and the second to extract the time.

3. Add a new regular expression filter.

4. For the expression value, type `.*ps` (any characters ending with “ps”; be sure not to omit the leading period `[.]`) and check the **Filter line by line** box.
5. Click **Test Selected**.
In the **Test Output** box, the only output is the line containing “ps”.
6. To extract the time from the line, add a new regular expression filter.
7. For the expression value, type `\d*:\d*:\d*` (three sets of digits separated by colons)
8. Click **Test Selected**.

The test output now shows only the time from the ps line. Now you can assign this value to a variable.

For information on storing values in variables, see [Inputs: Providing data to operations](#) or [Flow variables: Making data available for reuse](#).

Testing and deploying flows

To safely make and deploy changes to flows:

1. Work in a local repository or in a public repository on a development server’s installation of Central.
2. Test your work on a staging server.
3. Deploy the changed version of the flow during non-production hours.

To deploy changes to an existing flow from a test server

1. On the development server, make changes to and test the flow.
2. After saving changes, export the flow as a repository.
Note: Unsaved changes in a flow or operation are not included in the export.
3. Publish the flow to the staging server and test it there.
4. To see whether there are any current runs of the flow:
 - Open the Central Web site, logging in with an account that has HP OO administrator rights.
 - Click the **Administration** tab.
Any current runs of flows are listed under **Run Administration**.
5. Cancel any current runs of the flow you’re going to change, and instruct Central users not to start any new runs of the flow.
6. Publish from the staging server to the production server .
7. To test the flow on the production server, set its properties so that only you have access permissions to run the flow, then test it.
8. After the flow tests successfully, grant access permissions to appropriate end-users.

Restricting use of flows

You can restrict who can use a flow by:

- Restricting access permissions, specifying which groups of users can see a flow and start a run of the flow.

For information on assigning access permissions for HP OO objects, see [Controlling access to HP OO objects](#).

For information on defining HP OO roles or mapping external roles or groups to the HP OO groups described here, see the *HP OO Administration Guide* (AdminGuide.pdf) and Help for Central.

- Hiding the flow from Ops Central users

If your organization does not use a staging server for testing flows and you publish a flow that is not ready for production use in Central, you can hide it from Central users.

For more information, see the next topic, [Controlling who can see a flow in Central](#).

- Limiting access to a flow beyond a given transition: Specifying in a transition's properties which HP OO role members can continue running the flow beyond the transition.

Suppose one part of a flow can be run by a member of the LEVEL_1 HP OO role but a subsequent part of the flow needs to be escalated because it requires user credentials that a LEVEL_1 user doesn't have. When you create the flow, you could make the transition that leads to the step where the credentials are required a gated transition: one that requires membership in a certain role for the user to continue running the flow.

For more information on creating gated transitions, see [Transitions: connecting responses to steps](#).

- Adding an Acquire Lock step to the flow that prevents anyone but a user who possesses the key to the lock from continuing the flow run beyond that point.

For more information on creating Acquire Lock and Release Lock operations, see [Types of operations: setting properties](#).

Controlling who can see a flow in Central

There are two ways to hide flows from Central users:

- Marking the flow as hidden hides it from all Central users, regardless of their access permissions or group capabilities.
- To specify who can see the flow and who can't, specify which groups have read permissions for the flow.

To mark a flow as hidden

1. Open the flow and then, to open the flow's Properties sheet, click the **Properties** tab at the bottom of the authoring pane.
2. On the **Advanced** tab, select **Hide in Central**, and then save your work.

To select which groups have the read permission

1. In the Library, right-click the flow, and then click **Permissions**.
2. In the **Permissions** dialog that appears, select the **Read** permission for the groups you want to be able to see the flow in Central and Studio, and then click **OK**.

Creating IActions for operations

In advanced operations authoring, you use the Remote Action Service (RAS) to execute the core of an operation outside HP OO.

The RAS is a service that manages remote actions. The RAS instance that you install contains the IAction interface and is an implementation of the Remote Agent Service. The Remote Agent Service communicates with the Central Web application and Studio using the SOAP/HTTP(S) protocol. The RAS manages a database repository of IAction implementations, which are contained in Java

archives (.jar files) for use in a Java environment and in dynamic-link libraries (.dll files) for use in a .NET environment.

The IAction interface uses the execute() method and methods that are specific to the Web application, standalone application, platform, or extension service on which you want actions performed. These methods map actions in the Central SDK to actions of the target system's or application's SDK. The IAction interface thus mediates between HP OO and systems external to HP OO.

Standard operations interact with the HP OO infrastructure (the database and Internet Information Services) and are limited to the actions that are possible within the infrastructure. When you program the IAction interface in the operation, you can interact (via the extension services) with entities outside HP OO. Thus you can create operations that:

- Interact with systems throughout the network or Internet.
- Integrate HP OO with other entities that RAS can interact with through IActions.

Programming operations with Web extensions requires advanced .NET or Java programming skills.

Overview of creating operations that implement IActions

Overall, the authoring process involves creating one or more IAction implementation classes and moving them into Studio:

1. Create custom IAction implementation classes in your development environment and compile them into a dynamic-link library (DLL) or Java Archive (JAR) file.
The following section describes programming the IAction implementation class execute() method.
2. Copy the DLL or JAR into your Web service (your Web server's \bin\Actions directory).
3. Import the Web service into Studio.

Programming the IAction class execute() method for Web extensions

This section assumes that you have a background in installing J2EE Web applications.

Programming IAction implementation classes (IActions) for Web extensions requires that either RAS be installed. For information on installing a RAS, see *Installing Operations Orchestration Software* (InstallGuide.pdf).

To create a new Web operation using Java IActions

1. Create a Java project in a development environment.
For instance, you might use an Ant build file.
2. Add "JRAS-sdk.jar" to classpath.
3. To create IActions, implement the IAction interface, particularly the getActionTemplate() and execute() methods.
 - To implement the IAction interface, see the HP OO SDK.
 - To implement Java IActions, interfaces, and classes, see the Java Extension Service SDK.
 - To implement .NET IActions, interfaces, and classes, see the C# SDK.
4. Compile the project.

A sample ant project (with source code and reusable build file) is available at: JavaExtensionService/samples/MailAction.

5. Create a .jar file that contains all class files.
Do not include third-party libraries in the JAR file that you create.

For an example, see the sample ant project at: JavaExtensionService/samples/MailAction.

6. Copy the .jar file into the repository folder.
7. Copy all third-party .jar files into the **lib** subdirectory of your repository folder.
8. Restart the Central Web application.

Best practices for creating custom IActions

Use Java SDK classes only in the IAction implementation class. The IAction implementation class should only translate from the RAS SDK data types to the host application's data types (or custom data types).

Debugging IActions

The most efficient means of debugging IActions is line-by-line debugging in your preferred integrated development environment (IDE). This primarily involves connecting the IDE to the RAS remote debugger.

Debugging Java based operations

The following procedure describes how to connect Eclipse or IntelliJ to the RAS remote debugger, also known as remote Java Virtual Machine (JVM) debugging.

To debug JRAS

1. Within the HP OO home directory, navigate to \RAS\Java\Default\webapp\conf\.
 2. Add one of the following lines to the end of the wrapper file (wrapper.conf), replacing **[open port]** with an open port number:
 - If the RAS was developed in the Eclipse IDE:
`wrapper.java.additional.3=-Xdebug -Xrunjdpw:transport=dt_socket,address=[open port],server=y,suspend=n`
 - If the RAS was developed within the IntelliJ IDE:
`wrapper.java.additional.3=-Xdebug -Xnoagent -Djava.compiler=NONE -Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=[open port]`
 3. Restart the RSJRAS service, using the following sequence of command-line commands:
`net stop rsjras`
`net start rsjras`
 4. With RAS running, set a break-point in the IAction code.
 5. Connect the IDE to the RAS remote debugger.
- You are ready to start debugging the IAction.

Debugging .NET-based operations

As with debugging Java-based RAS IActions, a prerequisite for debugging .NET-based RAS IActions is to connect the IDE Visual Studio debugger to the RAS service process (RSJRAS) that is already running.

To debug a .NET-based RAS IAction

1. In Visual Studio (with your IAction project open), from the **Debug** menu, choose **Processes**.
2. In the **Processes** dialog box, select the Java.exe process that corresponds to your RAS.

You can do this by noticing that one of them shows up as managed code, or just connect to the all.

3. Click **Attach**.
4. In the **Attach to Process** dialog box that appears, make sure that **Common Language Runtime** is selected, click **OK**, and then (back in the **Processes** dialog) click **Close**.
5. Set a break-point in the IAction code.
6. Run a flow that references the IAction.

The IAction should stop at the break point.

Troubleshooting

A flow, operation, or system object that I know exists does not appear in my Library.

You may not have read permission for the item. If you do not have read permission for an object, it does not appear in the Library pane.

The scriptlet in my operation does not run correctly.

By default, Central expects to find scripts in the HP OO home directory, in the \Central\scripts subdirectory. This default location is specified in the Central.properties file.

If the script that the operation uses does not reside in the \Central\scripts subdirectory, do one of the following:

1. Move the script to the \scripts subdirectory.
2. In Central.properties, find the following line:
`dharmapp.script.repository=${iconclude.home}/Central/scripts`
3. Append to the line a semicolon separator and the location of any scripts that you want to use. For example, to execute scripts residing in the c:\MyScripts directory, you would modify the line to read as follows:

```
dharmapp.script.repository=C:\Program Files\Hewlett-Packard\Operations  
Orchestration\Central\scripts;c:\MyScripts
```

An error reports that I have lost the lock on the public repository

This error might appear when you try to import a repository to the public repository. One possible cause is that Central was re-started while Studio was open. This can happen if, for instance, your computer has a power-saving scheme that causes it to hibernate after some period of inactivity.

To regain a lock on the public repository: In Studio, open a private repository and then open the public repository.

After deleting a step then undoing the deletion, changes are not saved

Steps to reproduce:

1. With a flow diagram open, delete a step.
2. Save your work.
3. Undo the deletion.
4. Make changes to the step that you deleted.
5. Save your work.

Expected result: The changes that you make to the step after undoing its deletion are saved.

Actual result: The changes that you make to the step after undoing its deletion are not saved.

Subsequent saves work correctly.

Workaround: After undoing the deletion, editing and saving, move the step again and save. Subsequent changes will be saved correctly.