HP Operations Orchestration Software Studio

Software Version: 7.50

Guide to Authoring Operations Orchestration Flows

Document Release Date: March 2009

Software Release Date: March 2009

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

Trademark Notices

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

Finding or updating documentation on the Web

Documentation enhancements are a continual project at Hewlett-Packard Software. You can obtain or update the HP OO documentation set and tutorials at any time from the HP Software Product Manuals web site. You will need an HP Passport to log in to the web site.

To obtain HP OO documentation and tutorials

1. Go to the HP Software Product Manuals web site (*http://support.openview.hp.com/selfsolve/manuals*).
2. Log in with your HP Passport user name and password.

   OR

   If you do not have an HP Passport, click **New users – please register** to create an HP Passport, then return to this page and log in.

   If you need help getting an HP Passport, see your HP OO contact.
3. In the **Product** list box, scroll down to and select **Operations Orchestration**.
4. In the **Product Version** list, click the version of the manuals that you're interested in.
5. In the **Operating System** list, click the relevant operating system.
6. Click the **Search** button.
7. In the **Results** list, click the link for the file that you want.

Where to Find Help, Tutorials, and More

The HP Operations Orchestration software (HP OO) documentation set is made up of the following:

- 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 runs

  The Central Help system is also available as a PDF document in the HP OO home directory, in the \Central\docs subdirectory.
- 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 the \Studio\docs subdirectory.
- 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 flows

  The tutorials are available in the Central and Studio subdirectories of the HP OO home directory.
- Self-documentation for operations and flows in the Accelerator Packs and ITIL folders

Self-documentation is available in the descriptions of the operations and steps that are included in the flows.

Support

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

- *http://support.openview.hp.com*

Table of Contents

Warranty ii

Restricted Rights Legend ii

Copyright Notices ii

Trademark Notices ii

Finding or updating documentation on the Web iii

Where to Find Help, Tutorials, and More iii

Support iv

This Help system and Guide 1

A good first place to go in Studio: the How Do I... folder 1

Introduction to flows 1

- Key parts of a flow 2
- What a flow needs to be valid 3
- Taking a flow beyond the basics 3
- Operations: Models for steps 4

Advanced flow architecture and concepts 6

- How steps are executed 6
- Operations: architecture and data movement 7

Advanced flow, step, and operation concepts 8

- Local vs. global variables, in operation, step, and flow inputs 9
- Steps vs. operations 9
- Flow, step, and operation inputs 9
- Outputs and results 10

Multiple authors and version control 11

Starting HP OO Studio 11

Visual overview of Studio 11

Repository pane 12

My Changes/Checkouts pane 15

Authoring pane 15

Authoring pane toolbar 15

Bookmarks pane 18

Icons pane 18

Object properties sheets 19

Keyboard shortcuts in Studio 20

Repository pane keyboard shortcuts 20

Authoring pane keyboard shortcuts 20

Properties Editors / Inspector keyboard shortcuts 21

Scriptlet pane keyboard shortcuts 21

Bookmarks pane keyboard shortcuts 21

Debugger keyboard shortcuts 21

Version control of Library objects 21

Checking out and checking in a Library object 22

Checking out a Library object 24

Checking in a Library object 24

Resolving conflicts between two authors' versions of an object 25

Forcible checkins by an Administrator 26

Managing versions 27

Abandoning changes 27

Opening an earlier version of an object 28

Restoring an object to an earlier version 28

Creating a folder 29

Creating flows 29

Creating a flow 30

Creating a flow from a template 31

Adding callouts to a flow 33

Finding a flow or operation 33
Searching for a flow or operation 34
Descriptions: Finding the right operation 36
To see a folder's description 37
To see a flow's description 37
To see an operation's description 38
Viewing many operation and flow descriptions 38
Enabling Central users to generate documentation 41
Generating documentation in a custom format 43
.vm template file descriptions 44
Structure of Generate Documentation output 46
Modifying Generate Documentation templates 47
Finding out which flows use an operation 49
Copying flows and operations 49
Renaming flows or operations 50
Bookmarking flows and operations 51
Bookmarks: Adding and removing flows and operations 51
Bookmarks pane shelves: Adding, removing, and renaming 51
Bookmarks pane shelves: Showing, hiding, and moving 52
Restoring deleted Library objects 52
Creating steps 53
Adding steps to a flow 54
Creating a step from commonly used flows and operations 56
Changing the start step 56
Copying steps 57
Modifying a step 57
Re-arranging steps in the flow diagram 58
Prompting the user before running the step 58
Transitions: connecting steps 58
Adding a transition 60

Re-arranging transitions 61

Return steps 62

Inputs: Providing data to operations 63

Creating an input 64

Specifying how an input gets its value 65

Creating a single constant input value 67
Creating a single input value from what the flow user types 68
Creating a single input value from a flow user selection 68
Creating a constant list of input values 69
Creating an input value list from text the flow user types 70
Creating an input value list from flow user selections 70
Creating the input's value from the previous step's result 71
Assigning credentials as the input's value 71

Inputs and flow run scheduling 72

Removing an input 72

Flow variables: Making data available for reuse 72

Creating a flow variable 73

Creating local flow variables 73

Reserved flow variables 74

Concurrent execution: Running several threads at the same time 77

Parallel split steps 78

Moving data into and out of a parallel split step 79
Creating parallel split steps 80
Moving or copying parallel split steps and their components 81
Adding a lane 82
Duplicating a lane 82
Deleting a lane 82
Resizing a lane 82
Renaming a lane 82
Changing a lane's start step 82
Debugging a parallel split step 83

Multi-instance steps 83

Using multi-instance steps in flow design 83
Creating multi-instance steps 85
Moving data into and out of a multi-instance step 86
Throttling a multi-instance step 88
Debugging a multi-instance step in a flow 88

Making steps nonblocking 88

Making a step single response 89

Checkpoints: Saving a flow run's progress for recovery 89

Scriptlets 90

Creating a scriptlet 91

Debugging a scriptlet 92

Saving a scriptlet for use elsewhere 93

Outputs, responses, and step results 93

Operation outputs 94

Responses 95

Step results 95

Adding and removing outputs 95

Changing the source of an output 96

Adding and removing step results 96

Changing the source of a result 97

Filtering outputs and results 98

Creating a filter 98

Filter details 100

Diff Case 100

Extract Number 100

Format 100

Line Count 101

Regular Expression 101

Remove Duplicate Lines 101

Replace 101

Round Number 102

Scriptlet 102

Select Line 103

Select Range 103

Sort 104

Strip 104

Strip Whitespace 105

Table 105

Saving and re-using filters 106

Responses: Evaluating results 106

Adding responses to the flow 109

Changing a step's icon 109

Creating system properties 110

Flow design 111

Using subflows to simplify flow design 111

Passing data from a subflow to a parent flow 112

Changing which operation a step is based on 112

Step descriptions 114

Running flows automatically 114

Debugging flows 114

Debugging a flow 115

Changing values of flow variables within the debugger 118

Working with breakpoints 119

Overriding responses in a debug run 120

Logging on as another user 121

Repositories: Libraries for flows and their objects 121

Adding and opening a repository for authoring 121

Moving flow elements between repositories 123

Setting a target repository 123

Publishing to and updating from the public repository 123

Publishing a repository: how to 124

Updating from a repository: how to 126

Rolling back a publish or update 129

Exporting a repository 129

Importing a repository 130
Validating repositories 133
Encrypting repositories 133
Encrypting a repository 133
Opening an encrypted repository 134
Decrypting a repository 134
Creating a second encrypted copy of a repository 134

Backing up and restoring repositories 135

Creating operations from Web services 135

Operating outside Central with Remote Action Services 138
How Central runs RAS-dependent operations 139
Checking the availability of a RAS 140
Adding an existing RAS 140
Adding a RAS reference 140
Reconfiguring an existing RAS reference 142
Changing RAS references 143
Creating operations that access Web services 144
Troubleshooting the running of RAS-dependent operations 145
Studio example: A flow that runs in any of multiple domains 145
Creating operations from RASes 146
Creating operations from a RAS 146
Adding an operation that is not in the default RAS 147
Removing a RAS reference 148
Importing RAS content 148

Evaluating data for correct format 148
Creating an evaluator 149
Editing an evaluator 150
Deleting an evaluator 150

Recording values for reporting in Dashboard charts 150

Domain terms for Dashboard charting 151
Adding domain terms or domain term values 152

Displaying messages to users 153

Categories: classifying flows 153

Selection lists for user prompts 154

System accounts: secure credentials 155

Creating a system account 155

Editing a system account 156

Deleting a system account 156

Controlling access to OO objects 156

Setting permissions for folders and HP OO objects 158

Setting default access permissions for groups 160

Creating a new operation 161

Types of operations: setting properties 162

Cmd (command-line) operation 162

Flow Run Summary Report operations 163

HTTP operation 164

Perl script operation 166

RAS operation: IAction programming for a RAS 167

Scriptlet operation 169

Secure shell (ssh) operation 170

Telnet operation 171

Shell operation 173

Shell Wizard: creating a flow that uses a shell operation 173

Using a shell operation 178

Working with regular expressions 179

Testing and deploying flows 181

Restricting use of flows 181

Controlling who can see a flow in Central 182

Creating IActions for operations 182

Overview of creating operations that implement IActions 183

Troubleshooting 183

I am logged into two instances of Studio connected to the same public repository, and my changes in my workspace in one instance of Studio are not reflected in the other Studio. 183

A flow, operation, or system object that I know exists does not appear in my Library. 183

The scriptlet in my operation does not run correctly. 184

An error reports that I have lost the lock on the public repository 184

After deleting a step then undoing the deletion, changes are not saved 184

Index 185

This Help system and Guide

Help for HP OO Studio (reproduced in Studio_AuthorsGuide.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 data 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 movement in an operation and discusses the uses of the various kinds of operations
- Creating IActions for operations

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

A good first place to go in Studio: the How Do I... folder

If you want to create and run a productive flow with a minimum of study, open the **How Do I...** folder in Studio.

The flows in this folder are annotated with callouts, descriptions on the **Description** tab, and other information to tell you how to use the 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 units 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.
  - Set to a specific value.
  - 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 will be.

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 the next step in a particular run of the flow.

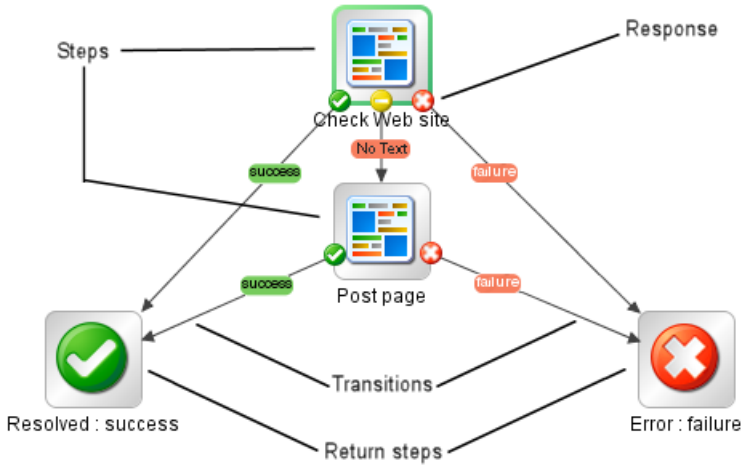


Figure 1 - Parts of a flow

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.
- For each step, from each response of the step's underlying operation, a transition connecting the step to a subsequent step. Most operations have more than one response, so most steps must have more than one transition connecting them to one or more subsequent steps. A step's operation responses are represented in flow diagrams by their names, attached to the step, as in the following:

Figure 2 - A step and its operation's failure and success responses

- A way 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 *Recording values for reporting in Dashboard charts*.

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.
- Data in operation results can be passed to flow variables, thus becoming available to other parts of the flow.

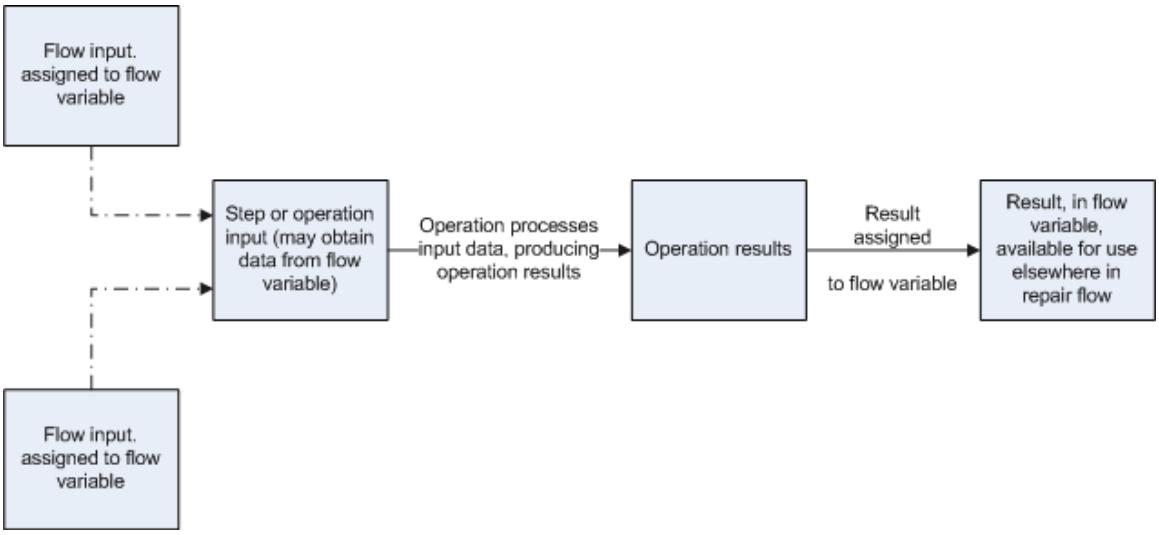


Figure 3 - How operations get data

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.

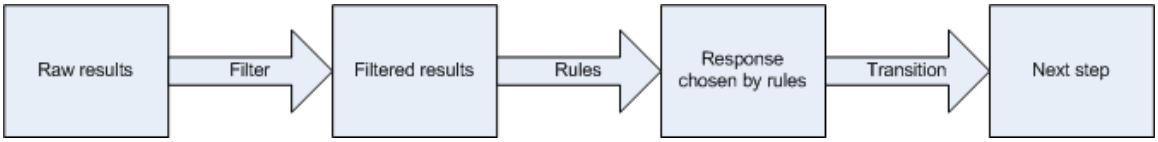


Figure 4 - How an operation's raw result can determine the next step

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.

Figure 5 - How an operation's result can become the flow result

Advanced flow architecture and concepts

Introduction to flows introduced you to basic flow concepts and information on how to make simple modifications to 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

How steps are executed

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, moves through, and can be modified within an operation, see the following section, *Operations: architecture and data movement*.
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 cannot set the primary result.

8. The transition that is associated with the selected response is selected.
9. The next step is executed, using this same sequence.

Operations: architecture and data movement

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 data movement 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.

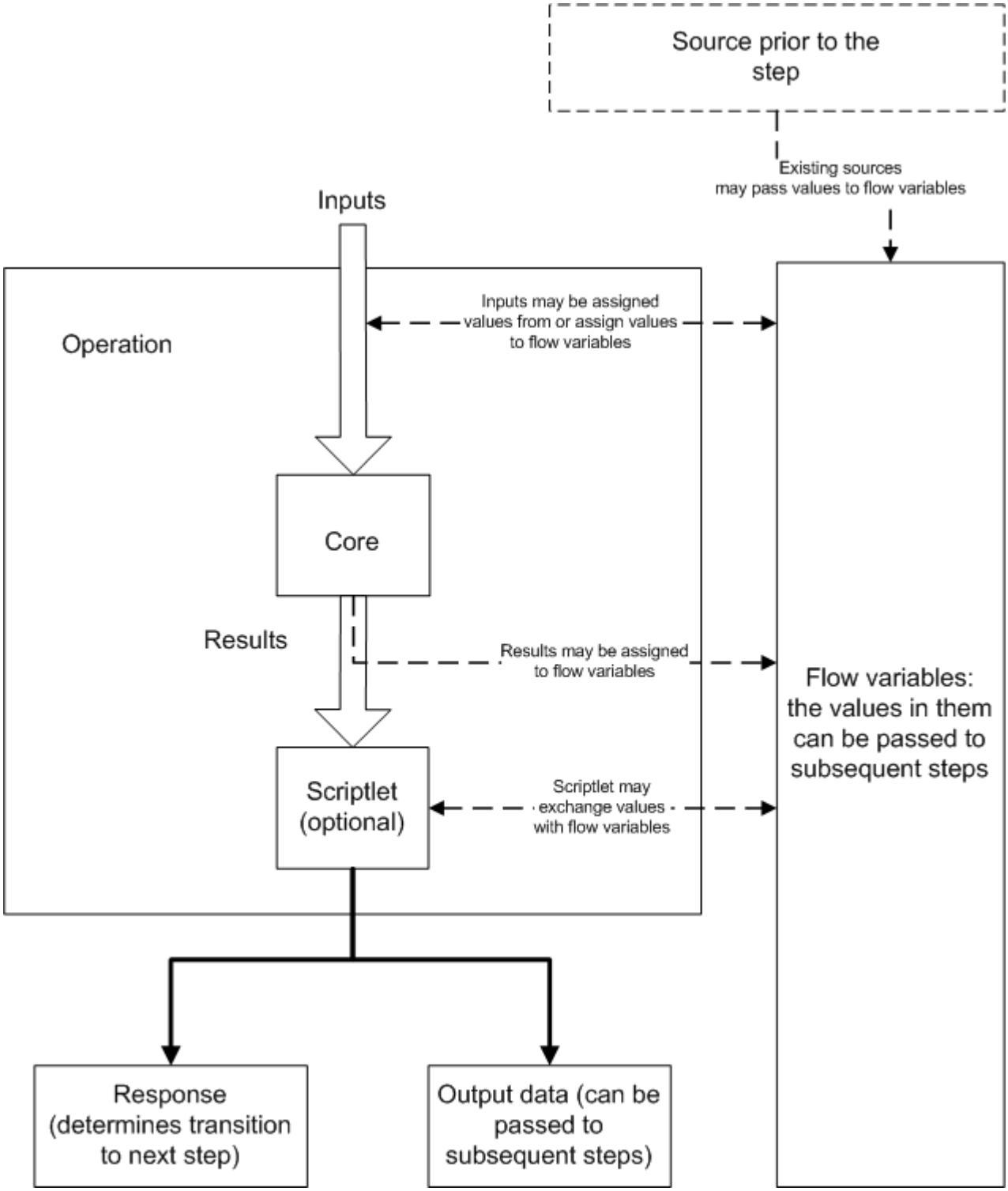


Figure 6 – Data movement through an operation

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.
- 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 outputs that are the sources for operation and flow *outputs* that the author creates. From the outputs, authors define step *results*.

One of the operation's outputs is the primary output, which the operation author can designate as such. The **primary output** is the output used to populate a step's primary result. For information on how to designate an output field as a primary output, see *Adding and removing outputs*.

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 **Primary Output From 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.

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 of the above pertain to operations, flows, and steps. Note, however, that you cannot change the handling of outputs for sealed operations.

	Raw Outputs	Outputs	Results	Responses
Operations (including flows)	✓	✓		✓
Steps			✓	

Multiple authors and version control

Multiple instances of Studio can be connected to the public repository of Central at the same time. The checkin/checkout feature avoids and mediates change conflicts when more than one author is working in the Central repository at the same time, and gives your authoring team version control. The checkin/checkout feature requires that you check out a flow, operation, or Configuration-folder object in the public repository before working on it and check it back in before other authors can see your changes to it.

For more information on checking repository objects out and what this means in practical terms, see *Version control of Library objects*.

Starting HP OO Studio

To start Studio

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

  OR

  Start Studio from the Windows **Start** button. In the list of programs, you can find Studio via **Hewlett-Packard**, and then **Operations Orchestration**, where it is listed as **HP Operations Orchestration Studio**.

  OR

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

In general, you should start and log in to a given instance of Studio as a single user, on a single computer. Make sure you understand the following discussion of this stricture:

Key information:

- **Do not** start multiple instances of Studio on the same computer, using multiple remote desktop programs from other computers, whether you log in as the same user or as different users. Doing so can result in data corruption and/or loss, or can cause unexpected behavior in Studio, including crashes.
- **Do not** connect to the public repository for the same installation of Central when logged in to two different instances of Studio as the same user. If you do, the user's workspace in the repository can become corrupted. This can also occur when Central is clustered.

Visual overview of Studio

The main elements of Studio are:

- The *Repository pane*, which shows the repository you're working in.
- The *My Changes/Checkouts pane*, where you can see the checkin/checkout status of an OO object.
- 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. You open this pane by clicking the **Bookmarks** tab.
- The *Icons pane*, which contains several collections of icons you can drag new icons from onto operations or steps. You open this pane by clicking the **Icons** tab.

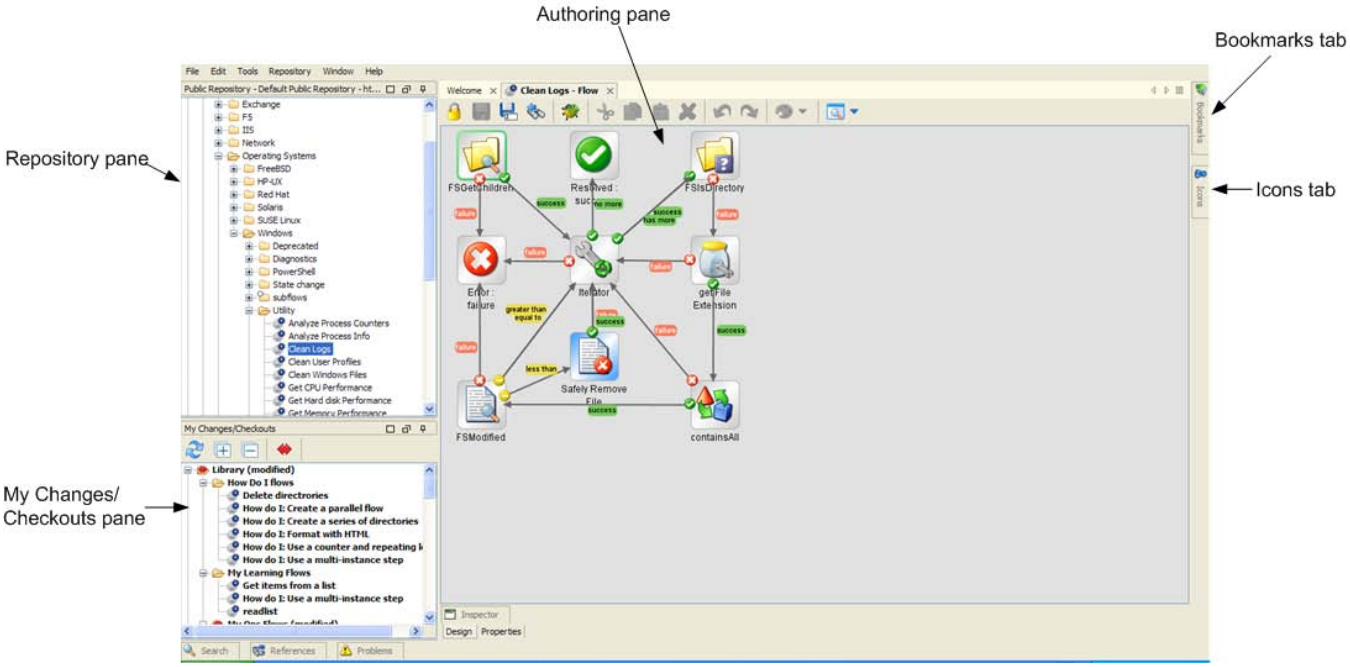


Figure 7 – Main elements of Studio

Repository pane

The content of the **Repository** pane falls within two folders:

- The **Library** folder, which holds all the flows and operations that you get when you install Studio.
- The **Configuration** folder, which holds other OO objects (also called *Library objects*) that you will use to process operation results, report results data for Dashboard charting in Central, and facilitate the running of flows both inside and outside Central, and remotely.

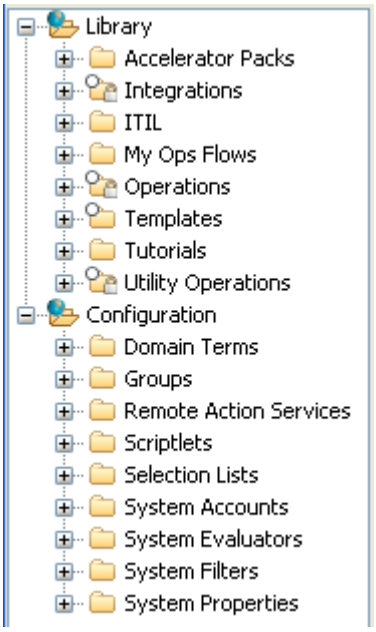


Figure 8 - Library, in Repository pane

Expanding the Library folder and then the flow folders reveals the default Studio content: flows and operations that run with various technologies. In the following screenshots, which show the content folders expanded, you can get an idea of the flows and operations that are available for building flows.

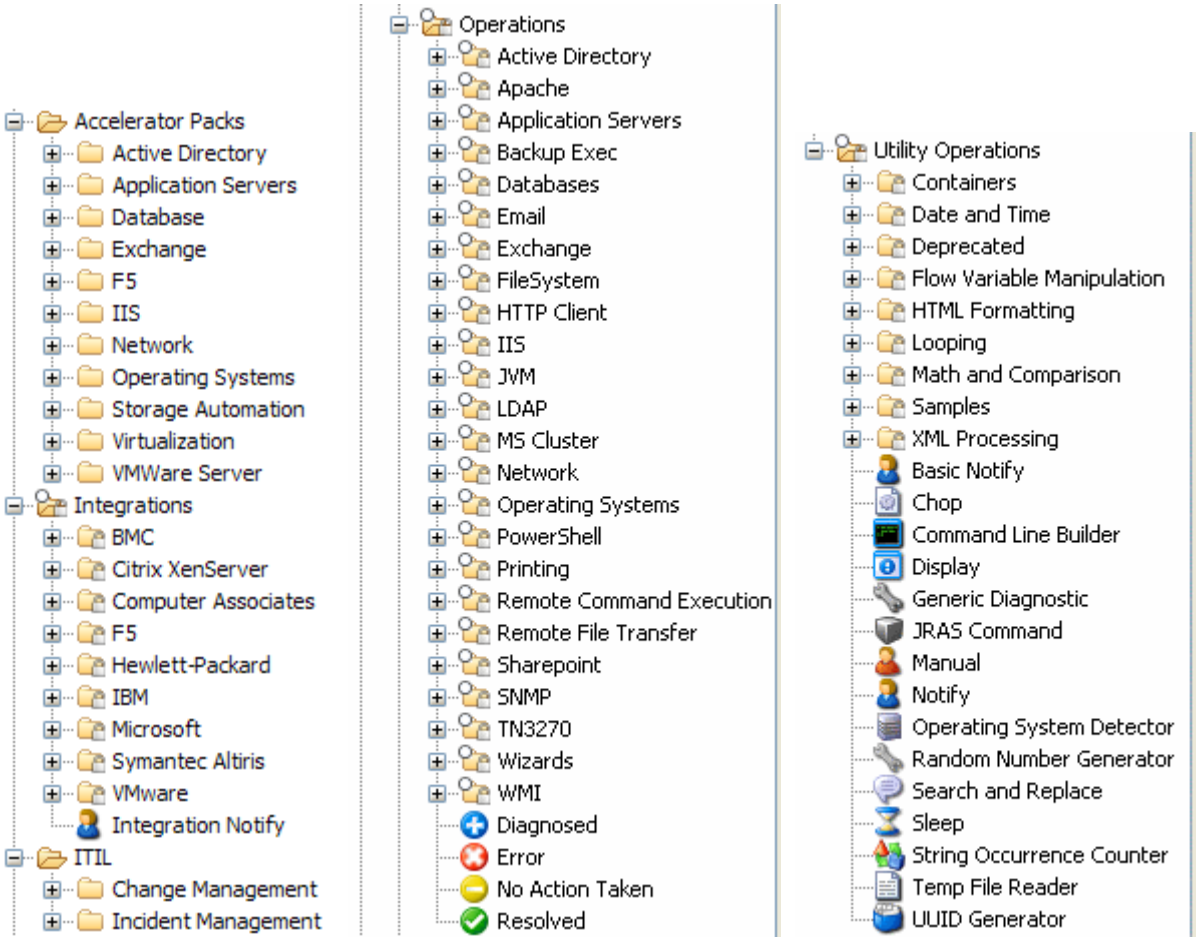


Figure 9 - Folders with content (flows and operations) that is supplied by default

The above illustrations open each of the folders that contain default content to their first level. There's a lot there; other parts of the documentation set and some of the tutorials focus on how to zero in on and use the specific flows and operations that you need. Study the top-level folders and their descriptions below; they can help point you toward the flow or operation that you need:

- **Accelerator Packs**

  These flows are designed to provide the following on most networks:

  - Complex health checks, triage, diagnosis, or remediation flows.
  - Simple flows that gather one or more pieces of data and display it to the user, or simply acknowledge alerts, gather some data, and place it into a ticket.

  The flows at the top level of an Accelerator Pack tend to be full health checks, triage, diagnosis, and remediation.

- **Integrations**

  Operations and flows that can be used from OO to interact with third-party systems-management products.

- **ITIL**

  This folder contains flows that automate integrations to other Enterprise-level software in accordance with ITIL specifications.

- **Operations**

  This folder contains general-purpose operations that work with common technologies. These operations are sealed and cannot be changed once you have installed HP OO Central (Central).

  Because you cannot change operations in the **Operations** folder, they should not have any static values set for inputs. All inputs should either prompt the user or be Not Assigned. There are exceptions to this rule, such as when a very general purpose operation such as a WMI command is used.

  The flows in the **Operations** folder and subfolders are meant to work as subflows or operations. Flows that you would want to run as the parent flow are in the Accelerator Packs.

- **Utility Operations**

  This folder contains operations and subflows that gather and display data, replace simple command-line operations, manipulate and analyze data, provide structure to flows, and other non-technology-specific functions.

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

Icon	Description
	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.

My Changes/Checkouts pane

The **My Changes/Checkouts** pane sits below the **Repository** pane and is where you can see which Library objects you have checked out. Checked out objects appear in the **My Changes/Checkout** pane and are bolded.

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.

Authoring pane toolbar

The **Authoring** pane toolbar buttons provide shortcuts for a number of tasks. The following illustration labels the toolbar. See the list below the diagram for more information on some of the labeled buttons.

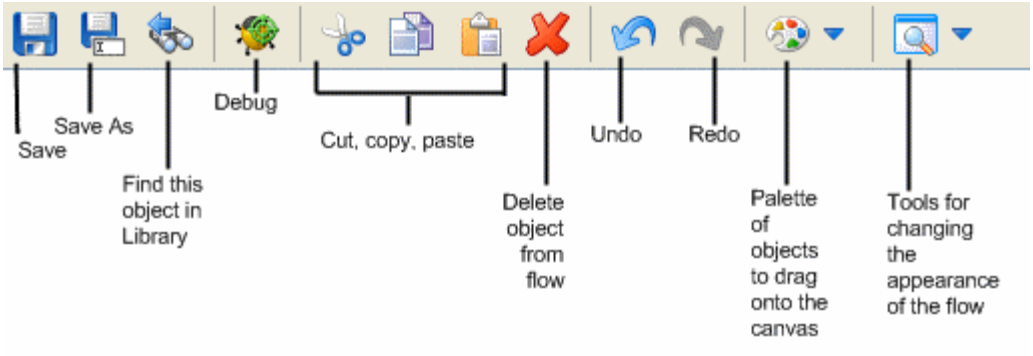


Figure 10 - Flow authoring toolbar

If you have the Central public repository open in Studio, the **checkin/checkout** icon ( ) also shows at the left end of the toolbar, next to the **Save** icon. For more information on the **checkin/checkout** icon, see *Version control of Library objects*.

Some of the toolbar icons are grouped in two palettes, which you open by clicking either one's downward-pointing arrow:

- **Step Palette** icon ( )
- **View Options** icon ( )

The remaining icons on the toolbar should be familiar, with the possible exception of:

- **Select in repository tree** ( ) expands the Library to select the flow or operation that you're working on.
- **Debug Flow** ( ) opens the Debugger and starts a run of the current flow in it.

Step palette

The **Step** palette, shown below, contains icons for dragging the four kinds of return steps (**Success**, **Diagnosed**, **No Action Taken**, and **Failure**), parallel split steps, and callouts onto the flow:

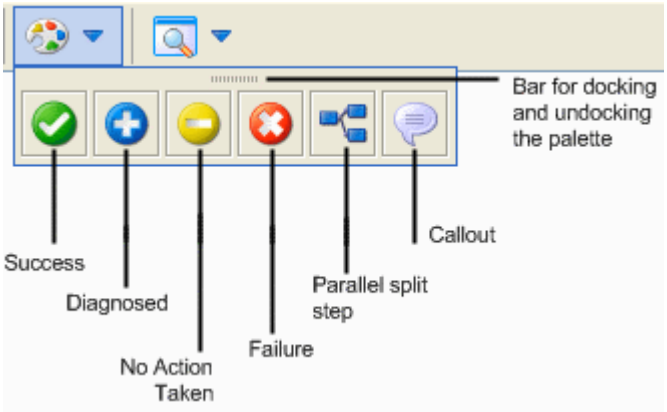


Figure 11 – Step palette

View Options palette

The **View Options** palette, shown below, contains icons for changing the appearance of the flow on the authoring canvas.

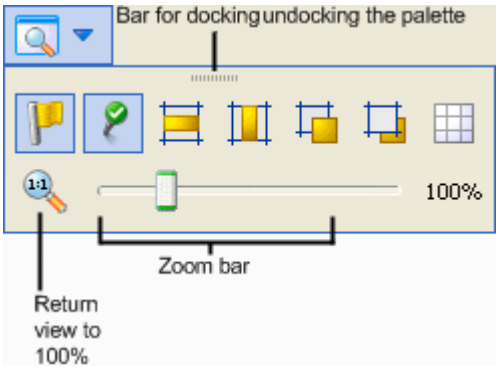


Figure 12 - View Options palette

The icons on the palette do the following:

Show / Hide Labels shows or hides labels of objects such as responses.

Show / Hide Connected Response Icons toggles to represent each of a response's inputs with an icon like those of the flow return steps.

Align selection horizontally aligns the steps that are selected horizontally.

Align selection vertically aligns the steps that are selected vertically.

Bring to Front moves the selected object to the front of the stack.

Send to Back moves the selected object to the back of the stack.

Show / Hide Grid reveals the authoring canvas grid, which you can use for arranging steps. If the grid is showing, then when you stop dragging a step, it snaps to the nearest position on the grid.

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

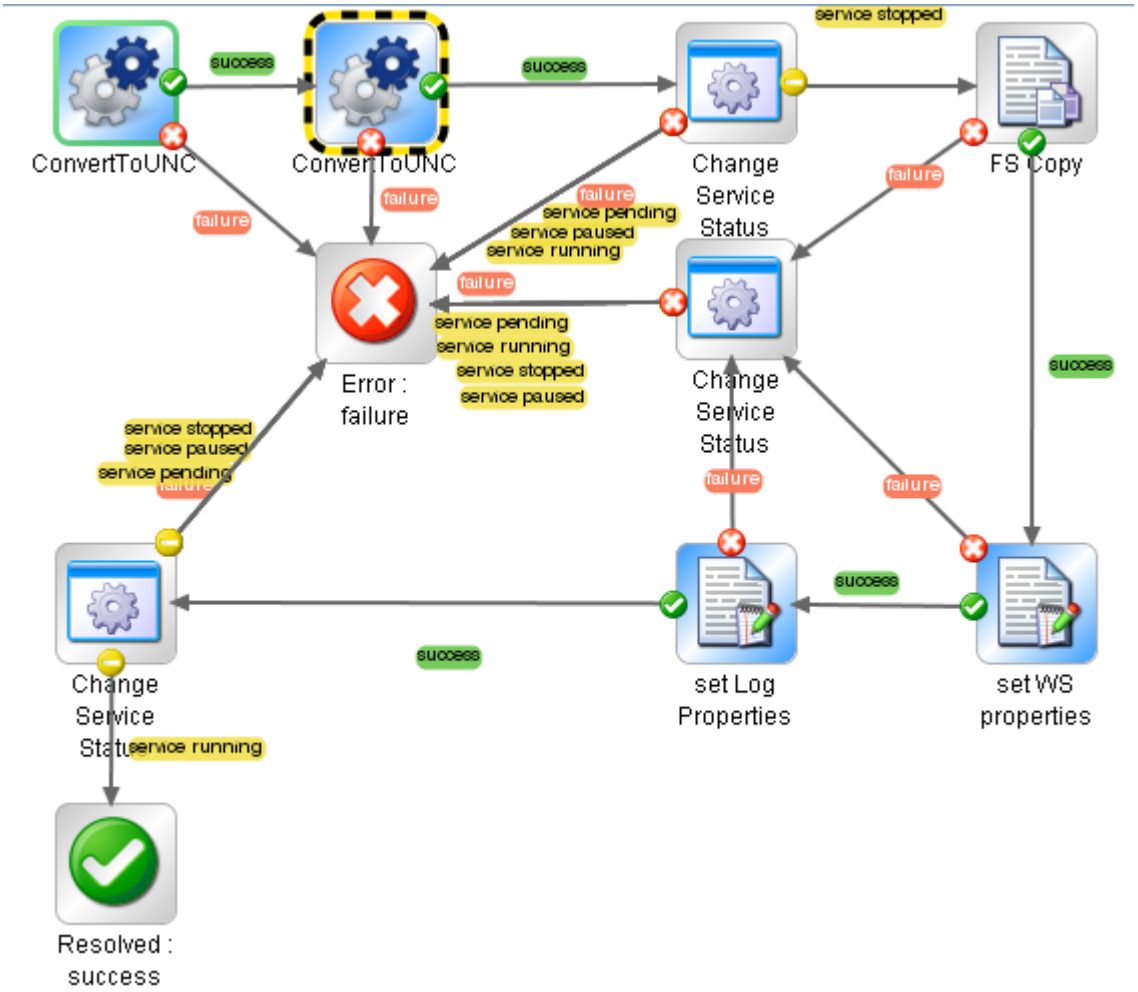


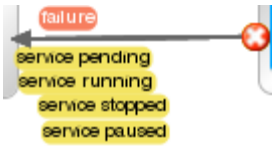
Figure 13 - Flow diagram

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.

- When multiple transitions start from the same source step and go to the same destination step, they are represented visually by a single line, as in the following clip. The names of the responses that are the sources all appear in a stack. 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) makes it easier to find and use the operations and flows that you use frequently. You can add flows and operations that you use a lot to the lists by dragging them from the Library. For more information, see *Bookmarking flows and operations*.

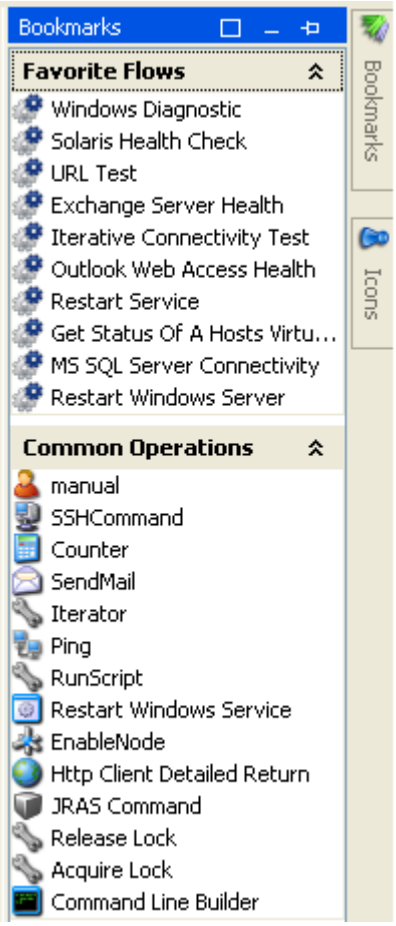


Figure 14 - Bookmarks pane

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. You can use one of these icons to replace the default icon on a flow, step, operation that you drag the icon to.

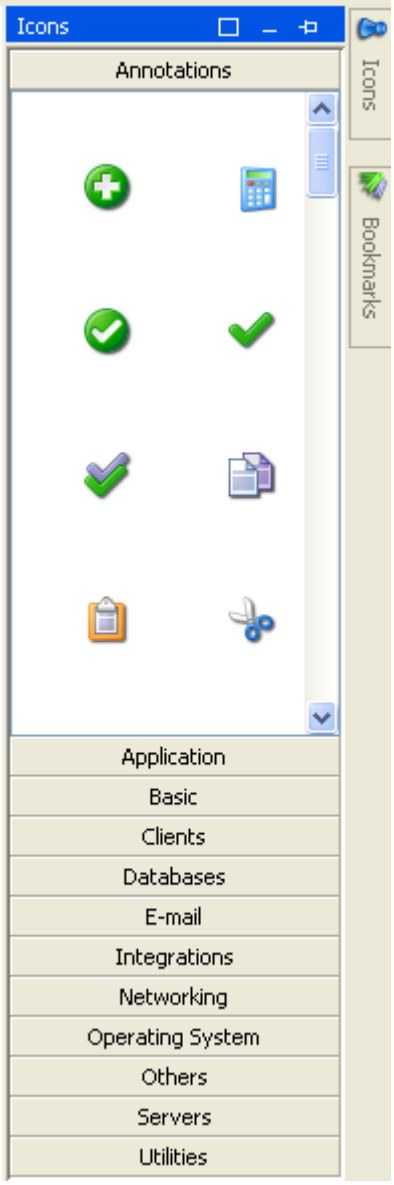


Figure 15 - Icons you can use in flows

Object properties sheets

The properties sheet for flows, operations, and **Configuration** folder objects are the editors in which you add, remove, or change values for the objects. For most objects in the Library, the properties sheet is the interface you use to work with the object. Flows, however, have the **Design** view as well as the **Properties** sheet.

In addition to the fields that you can edit, properties sheets provide the UUID and information about the version of the object, as shown in the following properties sheet header:

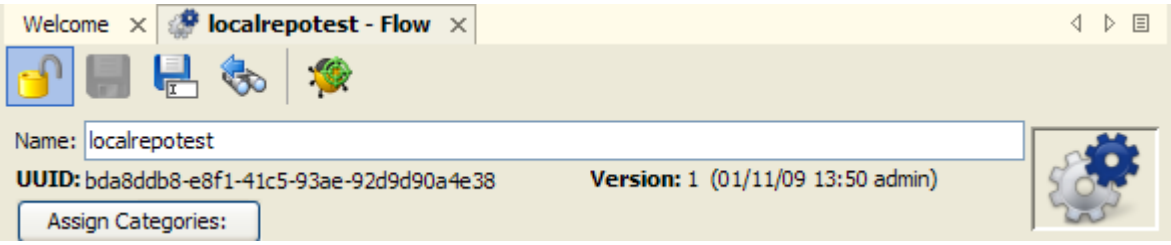


Figure 16 - Properties sheet header

Keyboard shortcuts in Studio

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

Repository pane keyboard shortcuts

Authoring pane keyboard shortcuts

Properties Editors / Inspector keyboard shortcuts

Scriptlet pane keyboard shortcuts

Bookmarks pane keyboard shortcuts

Debugger keyboard shortcuts

Repository 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

Properties Editors / Inspector keyboard shortcuts

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

Scriptlet pane keyboard shortcuts

To do this	Keyboard shortcut
Find	CTRL + f

Bookmarks 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

Version control of Library objects

Version control enables members of a multi-author team to work together in the Central repository without conflicts between various authors' changes. OO enforces version control through the **checkin/checkout** feature, which applies to flows, operations, and other Library objects (such as system accounts and filters).

You must check out an object in order to work on it. When you have the object checked out, you are working on it in your private workspace. All the other authors only see the object as it was the last time it was checked in. Saving your changes saves them in your workspace. After you have saved your changes, checking the object back in to the public repository makes your new version available for other authors to see and check out.

Important: Only the development/testing installation of Central should have multiple authors work in it. (Specifically, multiple authors should never have access to the Central installation in your production environment.)

When you are connected to the Public repository, the **version number** is incremented by .1 each time you save the object. When you check the object in, its version number is set to the next higher whole integer.

For example, suppose that FlowA is at version 2 when Gloria checks it out:

- As soon as Gloria saves changes to FlowA, its version number changes from 2 to 2.1.
- When any other author opens FlowA in Studio, he or she sees version 2 of FlowA.
- After Gloria checks FlowA back in, its version number is set to 3. Others can see the now-current version (version 3) of FlowA, and it is available for checkout.

The object's checkin/checkout status is indicated by the following:

- When an object is checked out by a different author from yourself:
  - In the **Repository** pane's library tree, the object's name is in *italics*.
- When an object is either not checked out or is checked out by a different author:
  - For flows, the canvas in the authoring pane is gray rather than white, and is disabled.
  - For all objects, the Properties sheet is disabled.
- When you have an object checked out:
  - For flows, the canvas in the authoring pane is white, and you can work in it.
  - For all objects, you can work in the **Properties** sheet.
  - In the **Repository** pane's library tree, the object's name is **bold**.

An administrator (a user who is a member of a group with the ADMINISTRATOR capability) can forcibly check in an object that an author has checked out, even if that author is also an administrator. When the administrator forcibly checks in an object, a message alerts the author who is working in it.

More information on these features is in these subsections of this section:

Checking out and checking in a Library object

Managing versions

Checking out and checking in a Library object

When you are authoring in a public repository, you must check out a flow, operation, or Configuration menu object in order to work on it. When you have an object checked out, other authors cannot work on it. As you work on it, you are working in your own workspace, so your work is not visible to other authors.

Key information: Do not connect to the public repository for the same installation of Central when logged in to two different instances of Studio as the same user. If you do, the user's workspace in the repository can become corrupted. This can also occur when Central is clustered.

The difference between checkin/checkout and publishing/updating is whether your Studio is directly connected to the public (Central) repository—that is, whether the public repository has been added to and is open in your Studio. If you are directly connected to the public repository, you check out Library objects and work on them in your workspace of the public repository, then you check them back in to convey your changes to them to the public repository and to make them available to other authors.

In the Studio **Authoring** pane, you can see several visual cues to the checkin/checkout status of an object:

- For the author who has checked out the object:
  - The object's fields are enabled for editing, and if the object is a flow, its canvas is white.
  - The object appears in the **My Changes/Checkouts** pane.
  - The object name in the **Repository** pane and in the **My Changes/Checkouts** pane is **bold**.
  - The **checkin/checkout** icon on the toolbar shows as an open lock (), indicating that the object can be edited.
  - Each time you save changes, the version number, which appears on the object's **Properties** sheet, increments by .1. Each time you check the object in, the version number increments to the next higher whole integer.

    Note: If you are working in a local repository, there is no checkout/checkin. When you save an object, its version number increments to the next higher whole integer.
- For the authors to whom the object is not checked out:
  - The object's fields are disabled, or grayed out, and if the object is a flow, its canvas is gray.
  - The **checkin/checkout** icon on the toolbar shows as a closed lock (), indicating that the object cannot be edited.

Following are a few limitations and considerations for the checkin/checkout feature:

- To check out an object, you must have READ and WRITE permissions for the object.
- You can check out a folder and its contents, including subfolders—as long as no one else has already checked out a subfolder.
- When you check out an object, you are checking out only the object (unless you're checking out a folder and its contents, as described above). You are not checking out objects referenced by the object. For instance, if a flow uses an operation that uses a system account, when you check out the flow, neither the operation nor the system account are checked out by virtue of your having checked out the flow.
- When you do any of the following with an object, it is automatically checked out to you. To convey these changes to the public repository, you must check the affected objects in.
  - Paste
  - Cut
  - Rename
  - Move
  - Delete
  - Save As
  - Import a repository
- To move, rename, or delete a folder, you must be able to check out all the objects within the folder—that is to say, you must have READ and WRITE permissions on all of them.

Subtopics within this section:

Checking out a Library object

Checking in a Library object

Resolving conflicts between two authors' versions of an object

Forcible checkins by an Administrator

Checking out a Library object

Note: When you have an object checked out, the administrator can forcibly check in the object, even if you're a member of a group that has the ADMINISTRATOR capability. If this happens, you will see a message alerting you that your checkout has been reverted.

You can check out an entire folder and its subfolders.

To check out a Library object

- With the object open in the **Authoring** pane, click the **checkin/checkout** icon ( ).

  OR, whether the object is open for editing or not:

  In the **Repository** pane, in the **Library** or **Configuration** folder:

  a. Right-click the object you want to check out.

  c. From the context menu that appears, point to **Repository**, and then click **Check Out**.

To check out a folder and its contents

1. In the **Repository** pane, right-click the folder.
2. From the context menu that appears, point to **Repository**, and then click **Check Out Tree**.

Checking in a Library object

Checking an object in conveys your changes to that object and its new version status from your workspace to the public repository. Other authors on that repository can see the new version of the object, and it becomes available for checkout to other users.

Checking in a folder only checks in the objects within that folder that you have checked out. Suppose a folder contains flow5 and flow6, and you have checked out flow5 and your teammate has checked out flow6. If you check in the folder that contains both flows, only flow5 is checked in. To check in a folder's contents recursively, including its subfolders and their contents, you use the **Check In Tree** command.

When you do any of the following with an object, it is automatically checked out to you. To convey these changes to the public repository, you must check the affected objects in.

- Copy
- Paste
- Cut
- Move
- Delete
- Save As
- Import a repository

Check-in comments help you track what's been done to the object per version. If you ever need to recover an earlier version, comments can help you zero in on the one you want to recover.

To check in a Library object

1. With the object open in the **Authoring** pane for editing, click the **checkin** icon ( ).

   OR, whether the object is open for editing or not:

   In the **Repository** pane, in the **Library** or **Configuration** folder:

   a. Right-click the object you want to check in.

   d. From the context menu that appears, point to **Repository**, and then click **Check In**.

   The **Check In** dialog box appears.

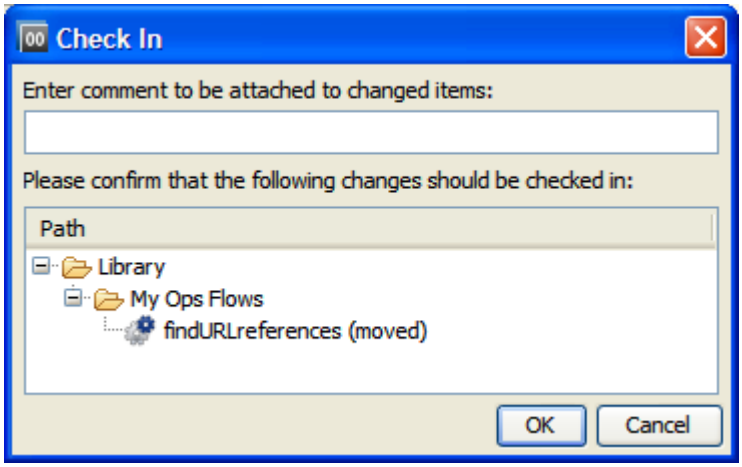


Figure 17 - Check In dialog

2. In the **Enter Check In Comments** box, type a comment that seems suitable to you.
   This is often a description of the changes that you made to the object.
3. Click **OK**.

To check in a folder and its contents

1. In the **Repository** pane, right-click the folder.
2. From the menu that appears, point to **Repository**, and then click **Check In Tree**.

Resolving conflicts between two authors' versions of an object

If two authors check in conflicting versions of an object, either of them can resolve the conflict in the **My Changes/Checkouts** panel of Studio.

Suppose, for example, that Elaine creates an operation named "foo," checks it in, and then renames the operation "safe call" without checking it in. Subsequently, Desmond, working in the same folder, creates his own operation "safe call." When Desmond checks in his "safe call" operation, then in the **My Changes/Checkouts** panel, Eileen will see the conflict icon (a red, double-headed arrow) is superimposed on the operation's icon.

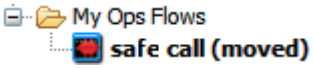


Figure 18 - Object in conflict with another author's object

When this happens you can view the **Properties** sheets for both versions, to resolve the conflict

To resolve the conflict between two versions of an object

1. In the **My Changes/Checkouts** panel, right-click the conflicted object, then click **Resolve Conflict**.
   The two conflicting versions of the object appear in the authoring pane of Studio.

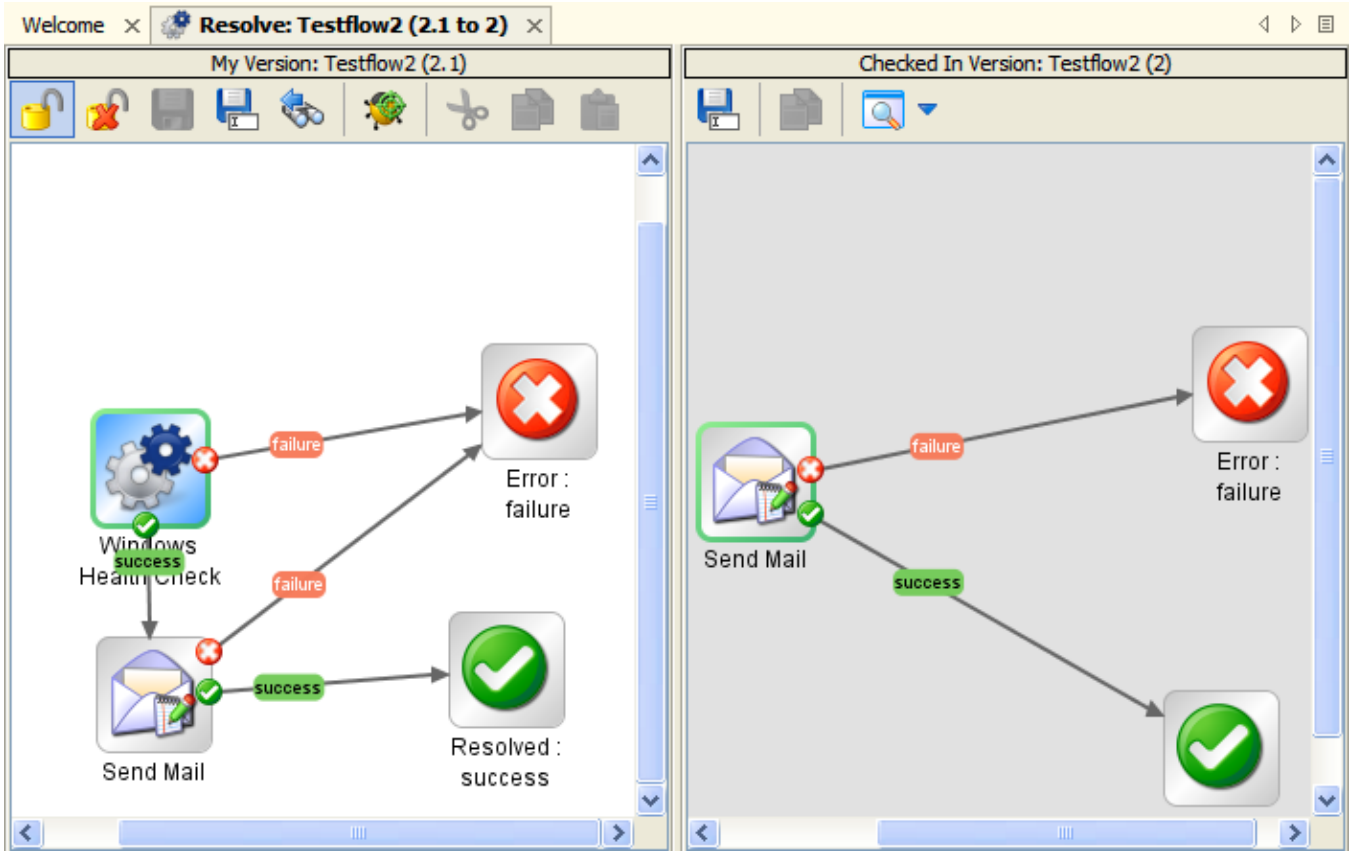


Figure 19 - Resolving conflicting versions of a flow

You can make and save changes in the flow diagram for the source version. In the flow diagram for the target (checked in) version you can do a Save As—save the object with a different name.

2. In your version, you can make changes to the properties of the object.

3. To resolve the conflict by abandoning your changes, click the Abandon Changes icon ( ).

   OR

   To check in your version of the object, click the Checkin icon ( ).

Forcible checkins by an Administrator

The administrator can forcibly check an object in that another author has checked out, even if the other author also has the ADMINISTRATOR capability. When an administrator forcibly checks in an object, any work that has been done on that object since the last time it was saved is lost, unless the author currently working on it saves the object with a different name before closing it.

If an object that you're working on is forcibly checked in and you want to preserve your work, you can use the **Save As** command to save the object with a new name, which gives the object a new UUID.

To forcibly check in an object that is currently checked out by another user

1. Log in to Studio as a user who has the ADMINISTRATOR capability.
2. Right-click the object that you want to force checkin of, point to **Repository**, and then click **Force Check In**.

Managing versions

You can open an earlier version of an object and compare it to the current version (for information on how to do so, see *Opening an earlier version of an object*). Depending on what you want to do with the various versions, you can:

- Abandon any saved changes that you made to the object while it was checked out.

  Abandoning changes does away with changes that have been made since the object was last checked out. Once the object is checked back in, there is no checkout to revert. To recover an earlier state of the object, you do one of the following.
- Open a previous version of the object and then save it with a new name.
- Restore the object to an earlier version of itself.

Procedures for doing the above are in the following:

Abandoning changes

Opening an earlier version of an object

Restoring an object to an earlier version

Abandoning changes

Suppose you've made some changes to an object and saved the changes, and now you don't want the changes to be preserved. You can abandon the changes. Abandoning changes automatically checks the object back in to the public repository. Checking an object in and abandoning the changes you made to it while you had it checked out restores the object's status in the public repository to checked in and restores its state to what it was before checked out. Other authors can check the object out.

To abandon changes

1. In the **Repository** pane, right-click the object whose changes you don't want to preserve.
2. In the context menu that appears, point to **Repository,** and then click **Abandon Change(s)**.

   The **Confirm** dialog box appears.

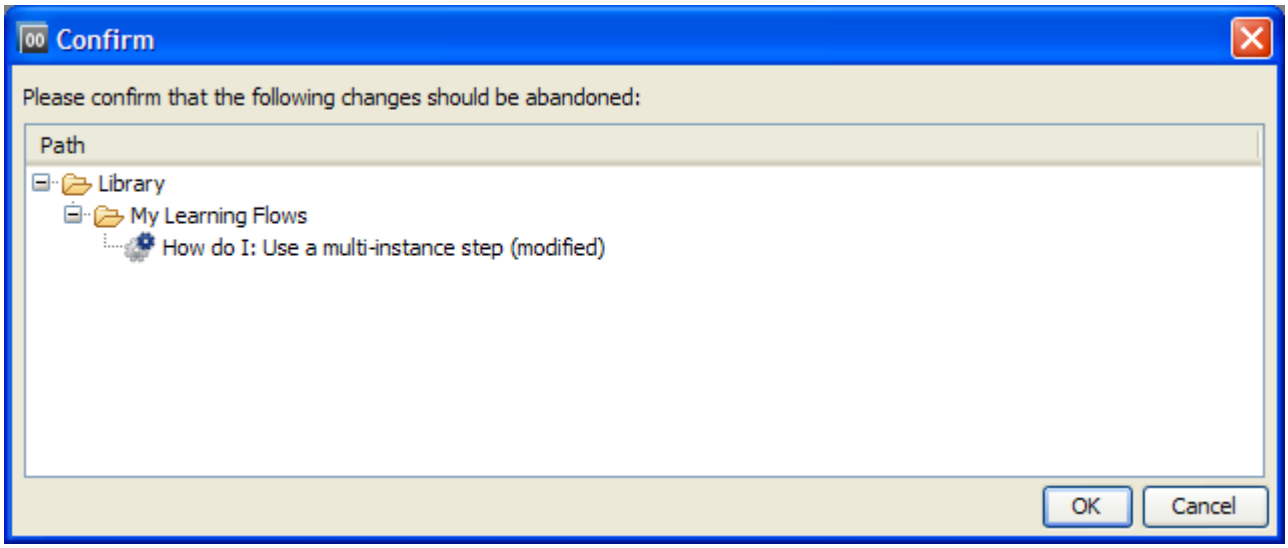


Figure 20 - Confirm dialog box

3. Verify that the object you want to check in without preserving the changes you've made is shown in the dialog box, and then click **OK**.

Opening an earlier version of an object

When you open an earlier version of an object for viewing, you can save the earlier version with a different name from that of the object's current version. To restore an object to its earlier version directly, see *Restoring an object to an earlier version*.

To open an earlier version of an object

1. In the **Repository** pane, right-click the object you want to see an earlier version of.
2. From the context menu that appears, point to **Repository**, and then click **Show History**.

   The **Version History** dialog box appears.

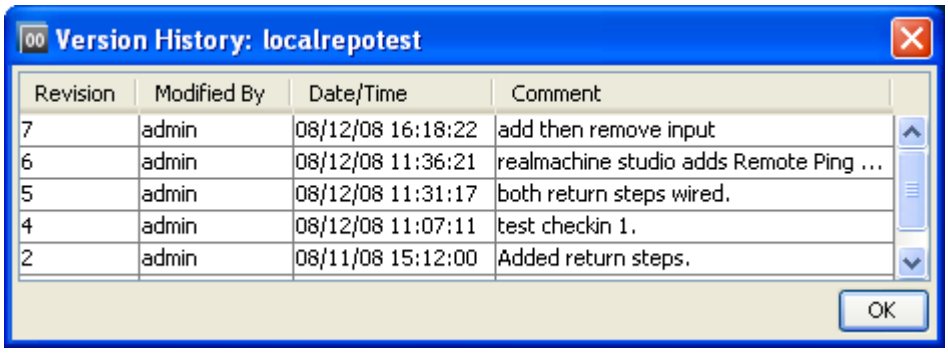


Figure 21 - Selecting a version

3. From the list of versions, right-click the one you need.

   Or, if you don't need any, click **OK**.

   The version that you open appears in your workspace, which is what you see in Studio.

   Using this procedure, you cannot save the version with the name of the object. (To restore an object in a previous version, see *Restoring an object to an earlier version*.)

4. To preserve the version that you have opened, click the Save As icon ( ) and give it a unique name.

   The earlier version of the object is now a new object.

5. Check the new object in.

Restoring an object to an earlier version

This procedure restores an object to an earlier version of itself. To keep both the current and earlier versions of the object, see *Opening an earlier version of an object*.

To restore an object to an earlier version

1. In the **Repository** pane, right-click the object whose earlier version you want to restore.
2. From the context menu that appears, point to **Repository**, and then click **Show History**.

   The **Version History** dialog box appears.

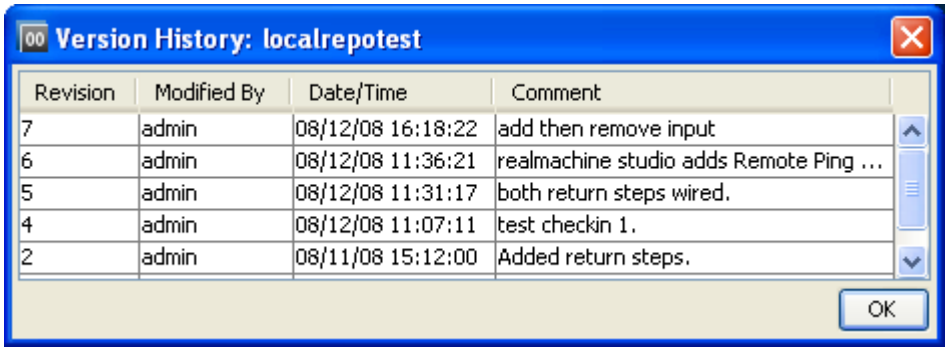


Figure 22 - Selecting a version

3. From the list of versions, right-click the one you need.

   The version that you open appears in your workspace, which is what you see in Studio.

4. Save and check in your work.

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 **Repository** pane, point to **New**, and then click **Folder.**
2. In the dialog box that appears, type the name of the new folder in the text box, and then click **OK.**

   Notes:

   - Naming in Studio is not case-sensitive.
   - Names can be a maximum of 128 characters long.

Creating flows

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. For more information, 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 flow from a template*).
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.

   Best Practice: To help Central users as well as authors who will create flows using the operations you create, add the following information to the operation's **Description** tab. (If you create multiple flows or operations that interact with the same technology, group them into a single folder and provide this information in the folder's **Description** tab. This is the practice for default HP OO content.) Note that putting this information on the **Description** tab makes it available to authors and Central users through the Generate Documentation feature. For more information on Generate Documentation, see *Viewing many operation and flow descriptions*.

   - A description of what the operation does
   - **Inputs** that the operation requires, including where users and authors can find the data that the inputs require and the required format for the data
   - **Responses**, including the meaning of each response
   - **Result fields**, including a description of the data supplied in each result field
   - Any additional implementation **notes**, such as:
     - Supported platforms or applications, including version information
     - Application or Web service APIs that the flow interacts with (this can be particularly important for flows that require an RAS to run, because the RAS operation can hide this information from the author or user of the flow.
     - Other environmental or usage requirements

   You might choose to further document the flow with callouts. For more information, see *Adding callouts to a flow*.
8. Test the flow.
9. To make the flow available to Central users, you publish your repository.

Creating a flow

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

1. Creating a folder for holding the flow and its operations. For instructions, see *Creating a folder*.
2. Creating the flow, as described in this section.
3. Adding operations as steps.
4. Creating connections (known as "transitions") between steps so each response for a step takes the flow to another step.
5. Creating one or more return steps to end the flow and assigning each return step a flow response.
6. Saving the flow.
7. 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**.

Note: To create a blank flow, right-click a folder, point to **New**, and then select **Flow**.

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. In the dialog box that appears, type a name for the flow, using standard characters, and then click **OK**.

   Notes:

   - If two flows reside within the same folder, you cannot give them 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 completing 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 flow from a template*.

In the Library, the new flow's 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.

Creating a flow from a template

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 pings a server and checks 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 **Blank Flow** template provides you with them.

To create a flow from a template

1. In the **Authoring** pane, click the **Welcome** tab to open the Studio **Welcome** page, and then click the **New Flow** icon.

   In the list of templates that appears, when you highlight a flow template, its description appears.

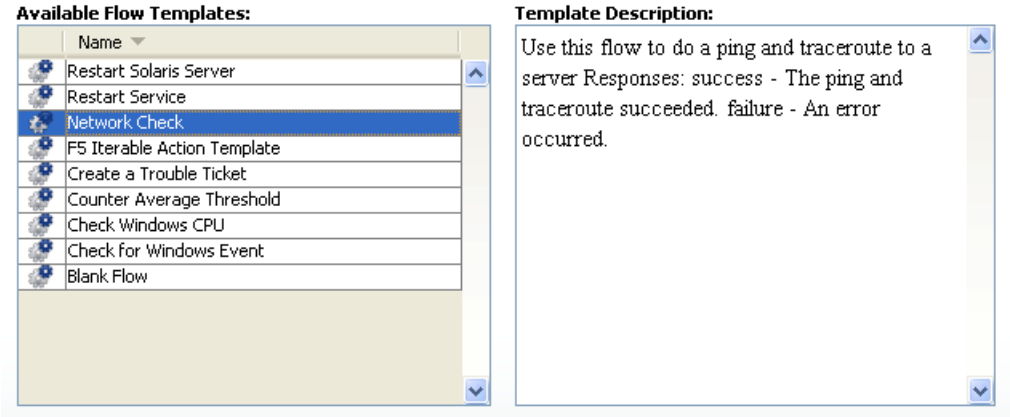


Figure 23 - Templates for commonly used flows

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

Next, you will create operations or create steps in the flow from operations (or flows).

Best Practice: To help Central users who will run your flows and authors who will create other flows using them as subflows, add the following information to the flow's **Description** tab. (If you create multiple flows or operations that interact with the same technology, group them into a single folder and provide this information in the folder's **Description** tab. This is the practice for default HP OO content.) Note that putting this information on the **Description** tab makes it available to authors and Central users through the Generate Documentation feature. For more information on Generate Documentation, see *Viewing many operation and flow descriptions*.

- A description of what the flow does
- **Inputs** that the flow requires, including where authors can find the data that the inputs require and the required format for the data
- **Responses**, including the meaning of each response
- **Result fields**, including a description of the data supplied in each result field
- Any additional implementation **notes**, such as:
  - Supported platforms or applications, including version information
  - Application or Web service APIs that the flow interacts with (this can be particularly important for flows that require an RAS to run, because the RAS operation can hide this information from the author or user of the flow.
  - Other environmental or usage requirements

Callouts can greatly enhance the usability of a flow for other flow authors and for Central users. The information that you provide authors and users in callouts is up to you, but you might add the following, which are some of the uses made of callouts in the How Do I flows in the Studio repository:

- Data movement: how information is passed from one step to another
- Names of flow variables that store data
- Formats required for the data that are required by inputs

Adding callouts to a flow

To add a callout to a flow

1. Do one of the following:
   - On the Steps and Callouts palette, click the Callouts icon and drag onto the flow diagram.

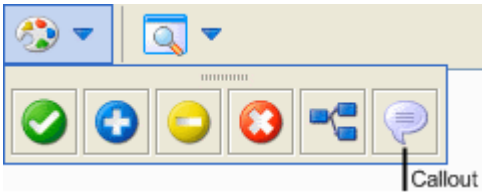


Figure 24 - Steps and callouts palette, with callout icon

   - Right-click in the flow diagram and then click **Insert Callout**.

   The callout appears on the diagram.

Figure 25 - Callout added, before you type text

2. Type the text for the callout.
3. To connect the callout to a step, drag from the gray circle to the step.

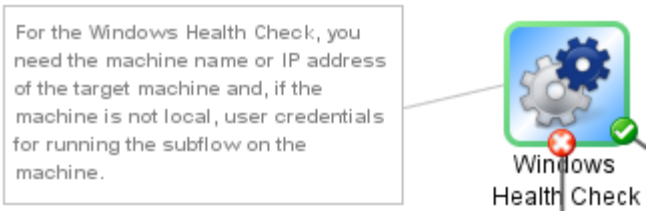


Figure 26 - Callout with text and connection to step

4. Save your work.

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**

  A folder for storing 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. The Studio Search engine uses the Apache Lucene syntax. For more information on using the **Search** tab and the Apache Lucene syntax in Studio, see *Searching for a flow or operation*.

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

Topics within this section

Searching for a flow or operation

Descriptions: Finding the right operation

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 27 - Studio tabs

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

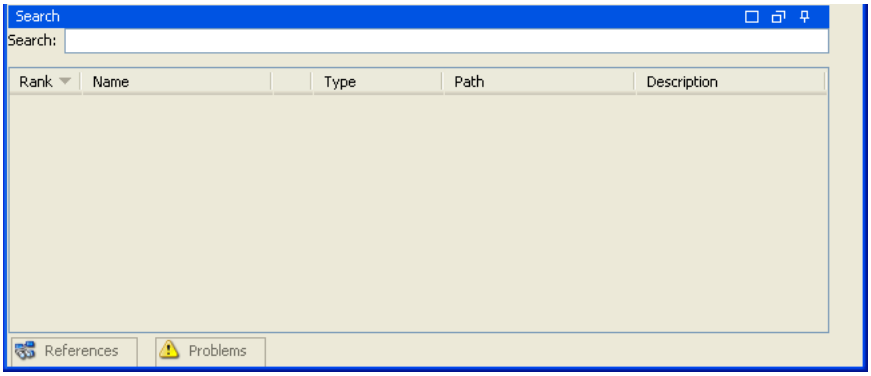


Figure 28 - Search pane

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.

Search

Search: http

80 hits

Rank	Name	Type	Path	Description
*****	Determine Results	other	/Library/iConclude/HTTPClient/Samples	Checks the outcome of the HttpGetDeta...
****	Http Client Get	WebExtension	/Library/iConclude/HTTPClient	<pre>Service to perform an HTTP GET ...
***	Check URL via Http Client Get	flow	/Library/iConclude/HTTPClient/Samples	This sample flow uses the HttpClientDet...
***	Refresh	WebExtension	/Library/Integrations/SiteScope	<pre>Issues a refresh command to a Si...
***	Acknowledge	WebExtension	/Library/Integrations/SiteScope	<pre>Acknowledges a SiteScope monito...
**	Unacknowledge	WebExtension	/Library/Integrations/SiteScope	<pre>Clears an acknowledgement of a ...
**	Total Server Shutdown	WebExtension	/Library/iConclude/Application Servers/B...	<pre>Shuts down a server and waits fo...
**	Suspend Server	WebExtension	/Library/iConclude/Application Servers/B...	<pre>Suspends a server and waits for ...
**	Start Server	WebExtension	/Library/iConclude/Application Servers/B...	<pre>starts a server and waits for it to ...
**	Validate XML Document	WebExtension	/Library/iConclude/XML Processing	<pre>Service to validate an XML docum...

Figure 29 - 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, pointing to **References** 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. (You can also open the step's operation by right-clicking the step, then clicking **Open 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? You can find out by looking at the description for each of the two flows.

Similarly, you can look at an operation's description, or the description for an entire folder of operations and/or flows in the default OO content (the collection of operations and flows that come with Studio, which are included in the **Accelerator Packs**, **Integrations**, **Operations**, and **Utility Operations** folders in the Studio **Repository** panel).

The descriptions for many folders in default OO content contain information, such as the following, that is key to successfully using the flows and operations in the folder:

- A description of what the flow or operation does
- **Inputs** that the flow or operation requires, including where authors can find the data that the inputs require and the required format for the data
- **Responses**, including the meaning of each response
- **Result fields**, including a description of the data supplied in each result field
- Any additional implementation **notes**, such as:
  - Supported platforms or applications, including version information
  - Application or Web service APIs that the flow or operation interacts with (this can be particularly important for flows and operations that require an RAS to run, because the RAS operation can hide this information from the author or user of the flow.
  - Other environmental or usage requirements

If you want to further explore an operation, highlight the step whose operation you're interested in, open its Inspector, 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 may also include notes on restrictions and tips on usage, as in the following description of the **Exchange Server Info** operation.

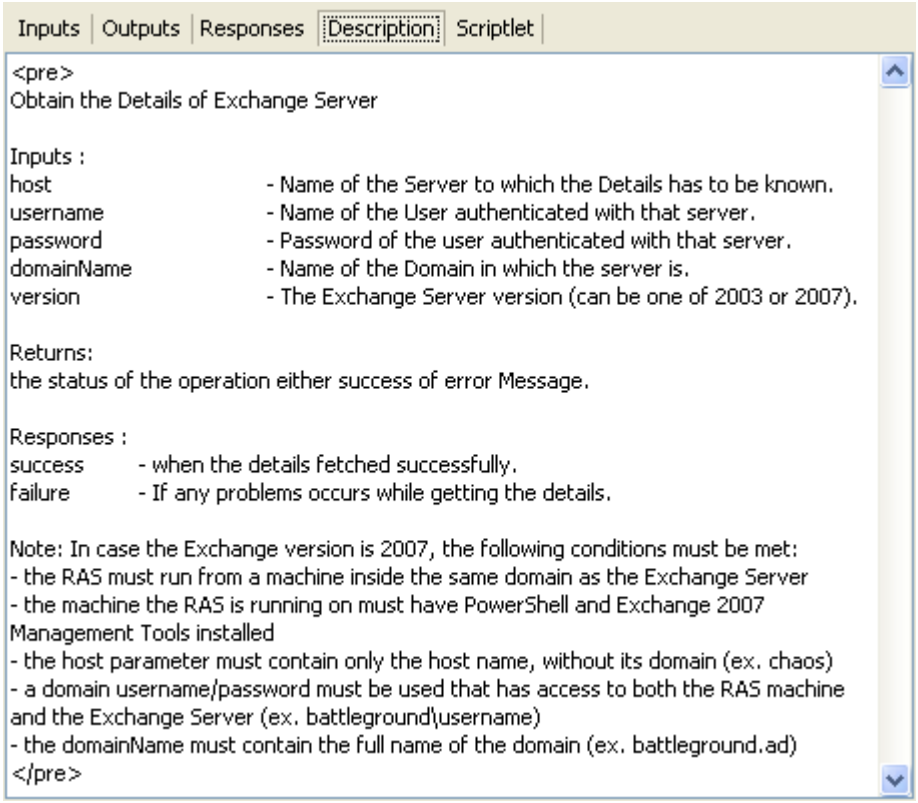


Figure 30 - Operation description

You can also use Studio's Generate Documentation feature to gather this information for many flows and operations into one place. For more information on Generate Documentation, see *Viewing many operation and flow descriptions*.

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*.

To see a folder's description

For more information on what may be included in folder descriptions, see *Descriptions: Finding the right operation*.

To see a folder's description

1. Right-click the folder in the **Repository** pane, and then click **Properties**.
2. On the folder's **Properties** sheet, click the **Description** tab.

To see a flow's description

For more information on what may be included in flow descriptions, see *Descriptions: Finding the right operation*.

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

For more information on what may be included in flow descriptions, see *Descriptions: Finding the right operation*.

To see an operation's description

3. Open the operation in the **Authoring** pane.

   OR

   Right-click a step that was created from the operation, , and then click **Open Operation**.

4. On the operation's **Properties** sheet, click the **Description** tab.

Viewing many operation and flow descriptions

Colleagues who run flows in Central and those who author flows may need detailed descriptions of flows and operations, including:

- What kind of data their inputs need.
- The data that they generate in results.
- What their responses are.
- What steps try to do in a run of the flow.

The Generate Documentation feature pulls this information from what flow authors have provided on the **Description** tab for each flow and operation and presents this self-documentation in a coherent, linked list of HTML files.

There are two ways to generate this information for individual operations or flows, or for the contents of folders:

- Using the (right-click) **Generate Documentation** command and distributing the output HTML files to Central users. For more information on the **Generate Documentation** command, see the following procedure, "To generate documentation of flows with the right-click command."
- Creating a flow using the **Generate Documentation** operation and promoting the flow to your production installation of Central. For more information on the **Generate Documentation** operation, see *Enabling Central users to generate documentation*.

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 (such as subfolders of the **Integrations** and **Operations** folders) that are hidden in Central or on their contents. The **Integrations**, **Operations**, and **Utility Operations** folders are hidden from Central.

Tip: To generate documentation for yourself on the contents of hidden folders, you can copy the folder to another location in the Library and then generate documentation for the copy.

To generate documentation of flows with the right-click command

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 click **Standard Format**.

   OR

   Create a flow that uses the **Generate Documentation** operation. For information on using the **Generate Documentation** operation, see *Enabling Central users to generate documentation*.

3. In the **Choose an output directory** dialog box, specify where OO should put the HTML files, and then click **Save**.

   After the flow and operation self-documentation is generated, the results appear in your Web browser.

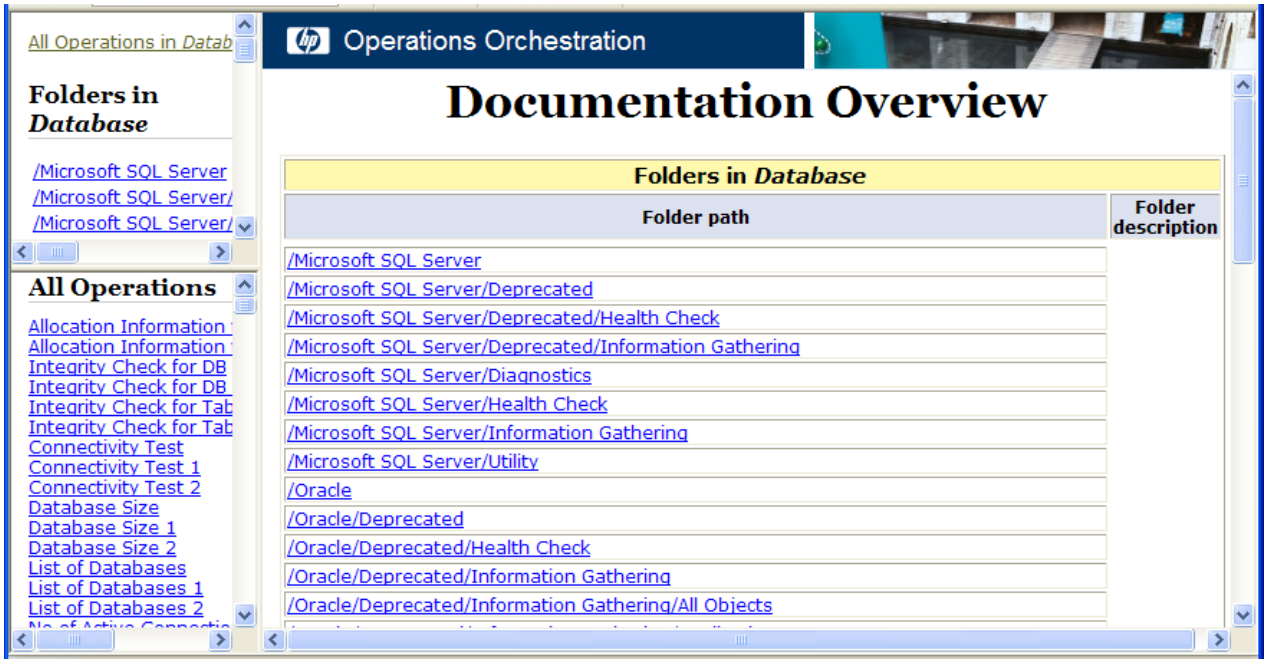


Figure 31 - Generated documentation for the Database Accelerator Pack

4. To view the operations in a subfolder, click the subfolder in either of the **Folders in Database** panes. The operations in the selected subfolder appear in the lower-left pane.

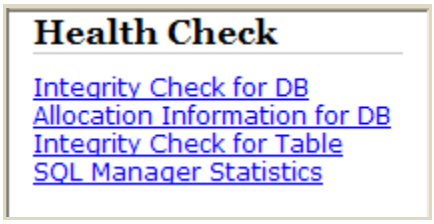


Figure 32 - Operations in the Accelerator Packs / Database / Microsoft SQL Server / Health Check / subfolder

5. To view the documentation for a single operation or flow, do one of the following:
   - Click **All Operations in <foldername>**, and then, in the list in the lower-left pane, click the operation or flow you're interested in.
   - After drilling down to the folder that contains the operation of interest, in the list in the lower-left pane, click the operation or flow.

   The following screen shots show the documentation and diagram for the flow **Integrity Check for DB**.

Ops Flow:
Exchange Server Health

Description

```
Checks the health of an Exchange Server

Inputs:
host - the host to check for events on.
altuser - the username to use when authenticating to host.
altpass - the password for altuser.
mbuser - The user to check the mailbox of.
mbpassword - The password for mbuser.

Responses:
success - The operation succeeded, no problems found.
failure - The operation failed or problems were found.
```

Diagram

Figure 33 – Descriptions of the Exchange Server Health flow, and inputs and responses

If you scroll down, you find the flow diagram, then a repeated listing of the inputs and responses, then explanations of the steps in the flow:

Diagram

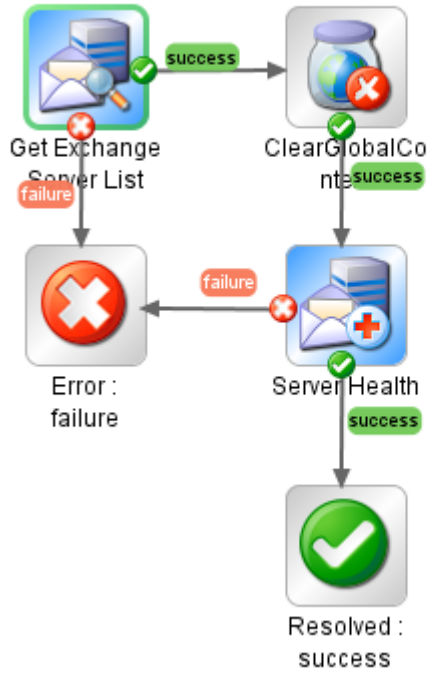


Figure 34 - Flow diagram of Health Check

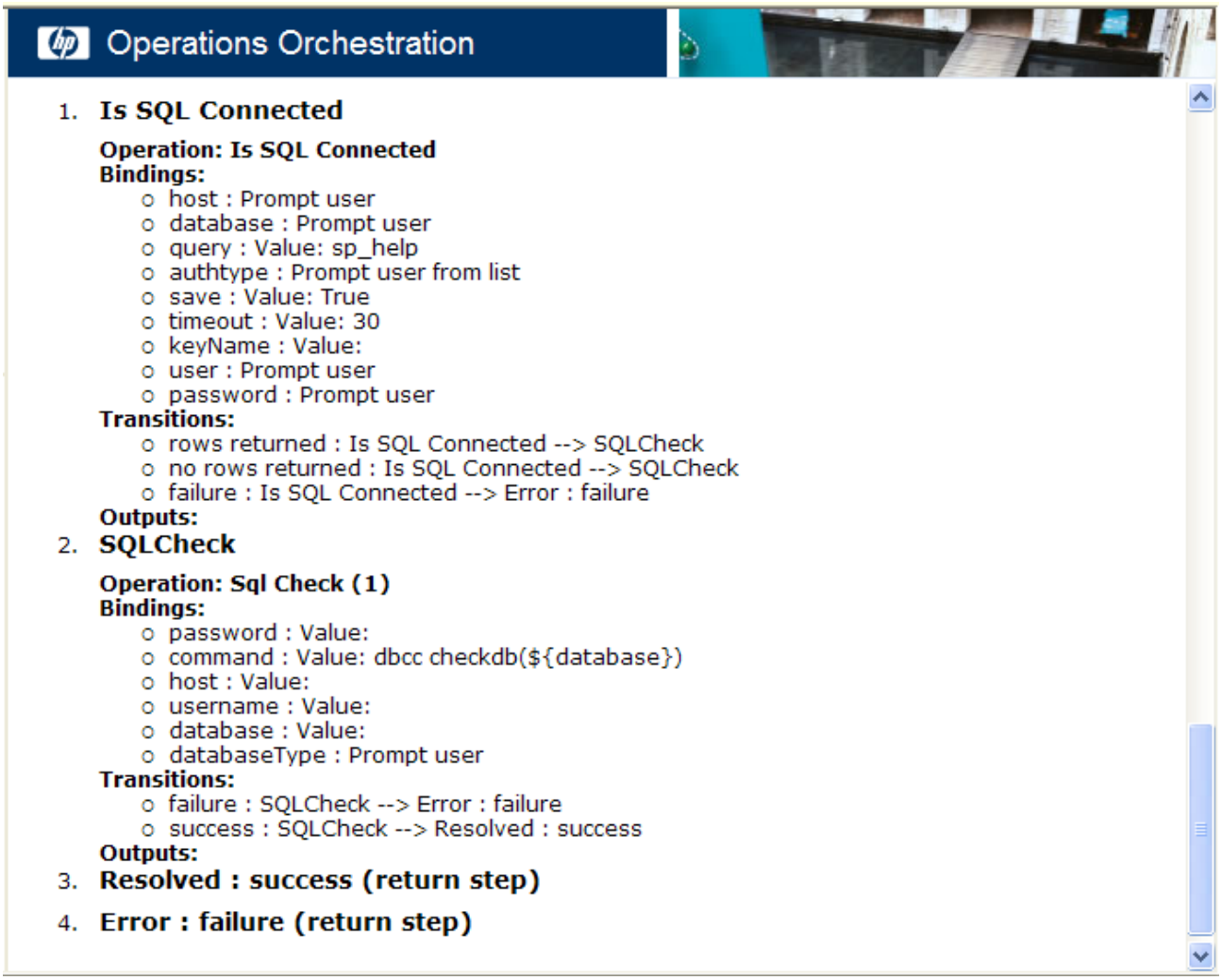


Figure 35 - Detailed information about a flow's steps

Topics in this section

Enabling Central users to generate documentation

Generating documentation in a custom format

Enabling Central users to generate documentation

By creating a flow that uses the **Generate Documentation** operation, you can provide Central users with the benefits of auto-generated flow documentation:

- Guidance to their choice of flows
- What to provide as values for inputs in response to user prompts
- What they can expect to find in a step's raw results and responses

In addition, Central users can schedule the flow, which can take a while to generate its output if there are a large number of flows in the folder selected, at times of low Central usage.

Further, assuming that the Central repository in the production environment is periodically updated with flows or flow changes promoted from the staging environment, regular scheduling of a **Generate Documentation** flow keeps the Central users' flow documentation up to date.

To create a flow with a Generate Documentation step

1. Create the flow.

For more information on creating flows, see *Creating a flow*.

2. In the Library, open the **Integrations** folder, then the **Hewlett-Packard** folder, then the **Operations Orchestration** folder, and then drag the **Generate Documentation** operation to the flow in the **Authoring** pane.
3. Double-click the **Generate Documentation** operation to open the operation's Inspector.
4. On the **Inputs** tab, on the **doc_TargetDir** input line, click the right-pointing arrow ( ).

   The input editor opens.

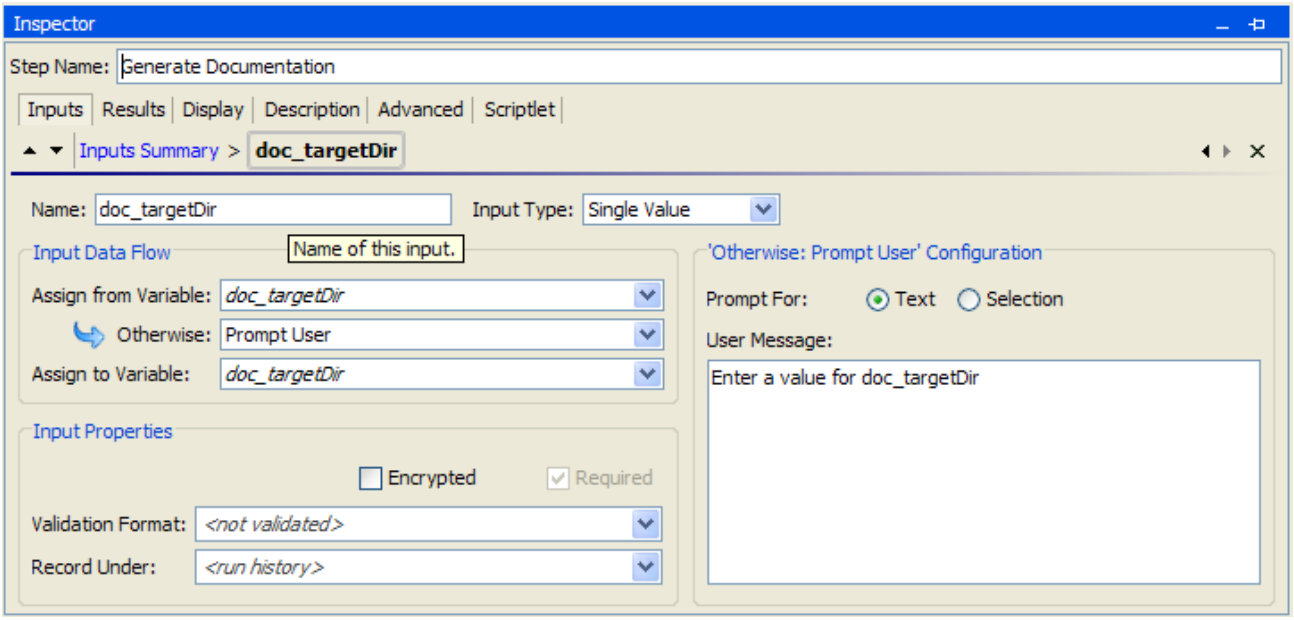


Figure 36 - Generate Documentation operation input editor

The **doc_targetDir** input specifies the location where the documentation is to be generated (the target directory).

5. In the **Assign from Variable** list box, you can select:
   - ***doc_targetDir***

     The user can then assign a constant value to the **doc_TargetDir** input when scheduling the flow.

     OR
   - ***<not assigned>***

     If you choose ***<not assigned>***, in the **Otherwise** list box select:
     - **Prompt User** to have the flow prompt the user for the target directory.

       This allows the user to choose the folder where the generated documentation is output.

       OR
     - **Constant**, and then type the target directory in the **User Message** box.
6. In the input editor, click the down arrow ( ) next to **Inputs Summary** to open the **libraryPath** input.

   This input defines the path to the Library folder for which the documentation will be generated.
7. In the **Assign from Variable** list box, you can select:
   - ***libraryPath***

     The user can then assign a constant value to the **libraryPath** input when scheduling the flow.

OR

- ***<not assigned>***

  If you choose ***<not assigned>***, in the **Otherwise** list box select:

  - **Prompt User** to have the flow prompt the user for the path to the Library folder.

    This allows the user to choose the folder for which documentation is to be generated.

    OR

  - **Constant**, and then type the path to the Library folder in the **User Message** box.

    Note: If you specify a Library path, the path may need to be updated in the operation if the Library folder structure changes.

Notes

- The Library path must always begin with /Library/.
- If you specify a folder that is hidden from Central, you will be providing Central users with documentation on flows and operations that they cannot see in Central.

8. Make the **Generate Documentation** step the start step of the flow.
9. Complete the flow with the **Success** and **Failure** return steps and steps providing any other functionality you desire.

Generating documentation in a custom format

Studio ships with a set of .vm templates in the <HP OO Home> directory, in **\Studio\extra\template** that define the appearance of the HTML files and frames that present the generated documentation.

The default template is DescribeFlows.vm, which produces the default documentation format. To customize the format of the documentation generated for flows and operations, you can modify copies of .vm templates that determine the appearance and behavior of the HTML files and frames. The .vm templates are written in the Apache Velocity template language.

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

- **overview-frame** – This is the upper-left frame, the **Overview** frame.
- **folderFrame** – This is the lower-left frame, the **Folder** frame.
- **headerFrame** – This is the upper-right frame, the **Header** frame.
- **opFrame** – This is the lower-right frame, the **Operation** frame.

The following shows the basic structure (frameset) of the HTML page, with the frame definitions in DescribeFlows.vm.

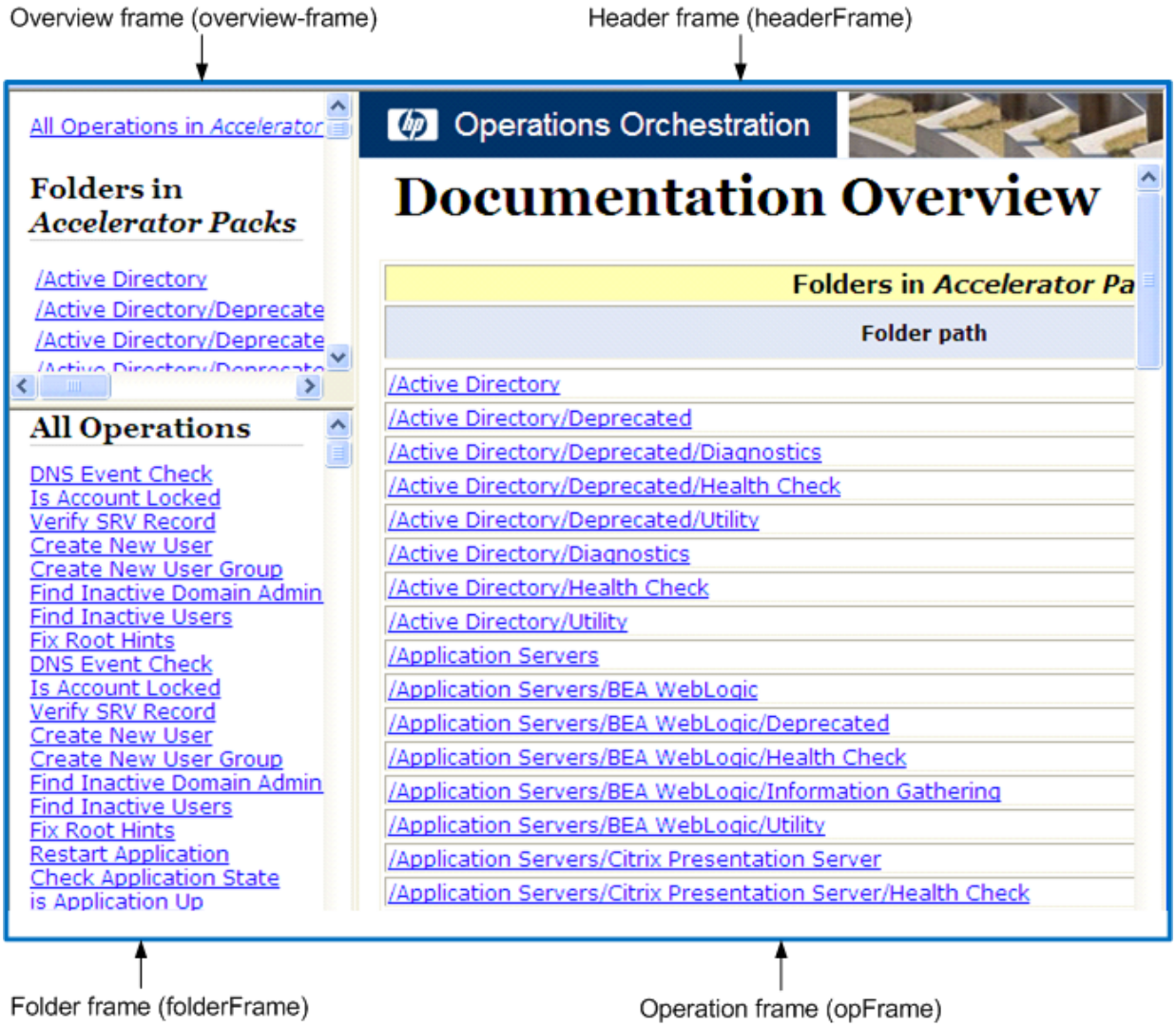


Figure 37 - The Generate Documentation frameset

Topics within this section

.vm template file descriptions

Structure of Generate Documentation output

Modifying Generate Documentation templates

.vm template file descriptions

The .vm templates are found in the <HP OO Home> directory, in \Studio\extra\template\. These templates do not appear in the **Select Documentation Generation Template** dialog when you right-click a folder and click **Generate Documentation** and then **Custom Format.** See *Modifying Generate Documentation templates* to learn how to modify the templates.

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 and places it in **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 selected 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 a 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 a 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 a link to display it in same frame (up to parent folder).

- Folder_contents.vm

  Displays the 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 a 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 a 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 a link to display it in **opFrame** (lower right).

Header.html

The Hewlett-Packard banner.

Header.css

Style sheet used for general fonts, colors, etc.

Hp_rockwell.css

Style sheet 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.

Structure of Generate Documentation output

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
        - ...

Modifying Generate Documentation templates

The high-level procedure for creating an alternative custom presentation of your Generate Documentation output is to:

1. Make a copy of each relevant .vm template, and rename the copy.
2. Make any desired modifications to the renamed copy.

   Each of the modified copies will appear in the **Select Documentation Generation Template** dialog when you use the right-click **Generate Documentation** command and choose **Custom Format**.

   Note that the **Select Documentation Generation Template** drop-down list only allows a single selection.
3. If you want a custom presentation that uses more than one .vm template, make a copy of Folder_template.vm, rename it, and modify it so that it refers to the .vm templates that you want included in your custom presentation.

   The renamed copy of Folder_template.vm will appear in the **Select Documentation Generation Template** drop-down list. To obtain the custom presentation that uses the .vm templates that you have modified from the drop-down list, pick the renamed and modified copy of Folder_template.vm.

For example, suppose that you want to use a custom presentation in which you have changed how the list of all operations and the description of a folder's contents are presented. You could do it as follows:

1. To change how the list of all operations is presented:
   - Copy All_ops_template.vm.
   - Rename the copy to Better_all_ops_template.vm.
   - Make your customizations to Better_all_ops_template.vm.
2. To change how the description of a folder's contents is presented:
   - Copy Folder_overview_template.vm.
   - Rename the copy to Better_folder_overview_template.vm.
   - Make your customizations to Better_folder_overview_template.vm.
3. To change which .vm files are used in the presentation:
   - Copy Folder_template.vm.
   - Rename the copy to Better_folder_template.vm.
   - In Better_folder_template.vm:
     - Change the All_ops_template.vm reference to refer to Better_all_ops_template.vm.
     - Change the Folder_overview_template.vm reference to refer to Better_folder_overview_template.vm.

   Now, when you use the Generate Documentation command, the following will appear in the **Select Documentation Generation Template** drop-down list:

   - Better_all_ops_template.vm
   - Better_folder_overview_template.vm
   - Better_folder_template.vm

4. From the drop-down list, pick **Better_folder_template.vm**.

For information on the structure of the .vm files and interaction of the frames that they define, see *.vm Template file descriptions* and *Structure of Generate Documentation output*.

To generate documentation in a custom format

1. Navigate to the <HP OO Home> directory, \Studio\extra\template.
2. For any template that you want to customize:
   - Make a backup copy of the template.
   - Make a copy of the template.
   - Rename the copy.
   - In a text editor, make your changes to the copy.
   - Back up the custom version of the template.

   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.
3. In Studio, right-click the folder whose contents you want to see information about.
4. From the context menu that appears, point to **Generate Documentation**, and then to **Custom Format.**
5. In the **Select Documentation Generation Template** dialog that appears, from the drop-down list, pick the custom template that you want to use.

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

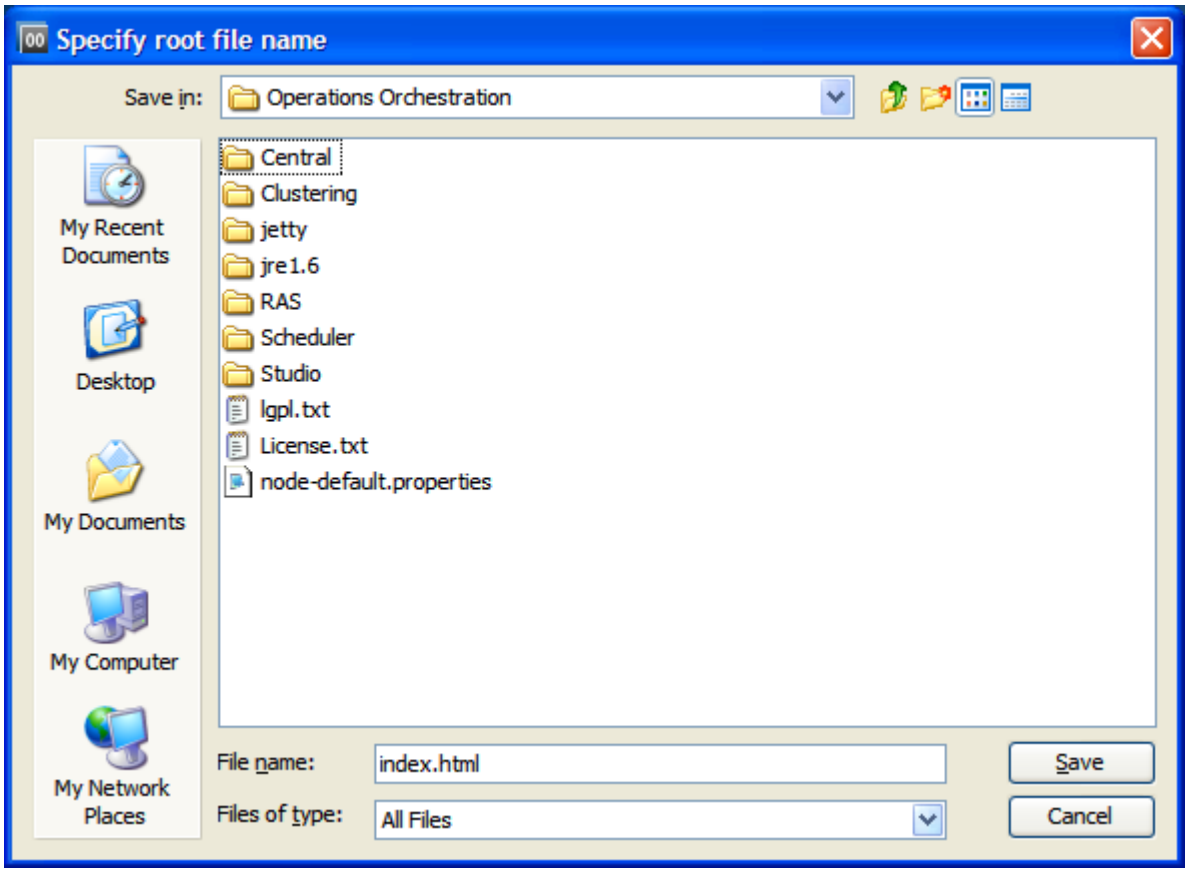


Figure 38 - Picking a target location for output

6. Navigate to an existing folder in which you want the generated HTML files to be placed.
7. 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.

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.

Studio has two kinds of references for determining which flows use an operation:

- **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 **References** and then click **What uses this?**

   OR

   To view the references from the operation or flow, click **References** and then click **What does this use?**

The **References** pane opens, displaying the references to or from the operation or flow.

Tip: To bring the last obtained references back up after you close the **References** pane, 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*

Using the context (right-click) menu, you can copy or duplicate flows and operations. The options available in the context menu depend on whether the flows and operations are in sealed (such as **Integrations**) or unsealed (such as **My Ops Flows**) folders.

Sealed folders	Unsealed folders
Copy operations	Copy and duplicate flows and operations
Copy and duplicate flows	

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 CTL+V.

   You can rename the copy.

Now you're ready to work with the flow.

Renaming flows or operations

You can rename flows and operations using the context menu. The options available in the context menu depend on whether the flows and operations are in sealed (such as **Integrations**) or unsealed (such as **My Ops Flows**) folders.

Sealed folders	Unsealed folders
Rename flows	Rename flows and operations

To rename flows or operations

1. In the Library, find and right-click the flow or operation.
2. Click **Rename**.
3. Type the new name in the highlighted field, 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** pane. The flow or operation is still available at its normal location in the Library. Adding a flow or operation to the **Bookmarks** pane makes it available also from the right-click menu in the flow canvas.

The **Bookmarks** pane has two sections, or *shelves,* one for flows and the other for operations.

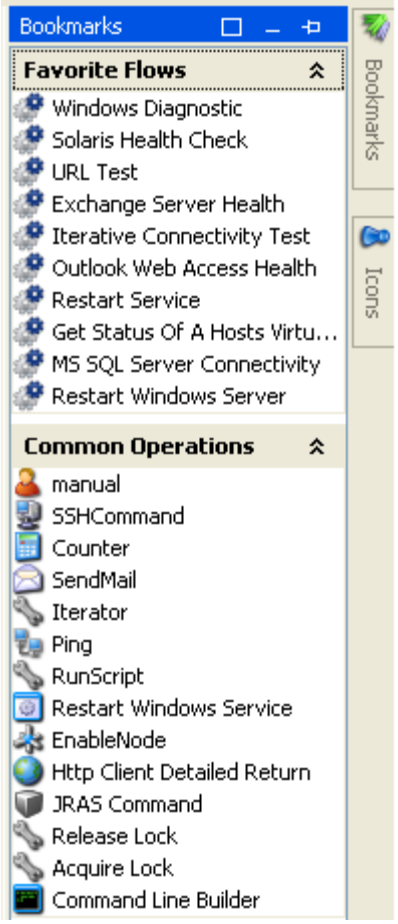


Figure 39 - Bookmarks pane — flow and operation sections

Bookmarks: Adding and removing flows and operations

To add a flow or operation to the Bookmarks pane

1. Click the **Bookmarks** tab to open the **Bookmarks** pane, and then click the **Pin** icon (⊹) in the upper-right-hand corner of the pane to keep it open.
2. Drag the flow or operation from the Library or **Search** box to the **Bookmarks** pane—flows to the **Favorite Flows** section and operations to the **Common Operations** section.

Bookmarks pane shelves: Adding, removing, and renaming

To remove, add, or rename a shelf

1. Right-click in the title bar of one of the **Bookmarks** pane shelves, and then select the action that you want to carry out.

2. Where appropriate, supply the information in the dialog box that appears.

Bookmarks pane shelves: Showing, hiding, and moving

To hide, show, or collapse a shelf

1. To hide a shelf, right-click in the shelf's title bar, then click **Hide**.
2. To show a hidden shelf, right-click in a blank area of the Bookmarks pane, and then click **Show All**.

   OR

   Click **Show** and then, from the list of hidden shelves, select the shelf that you want to show.
3. To collapse a shelf, in the shelf's title bar, click the double chevrons (⌃⌃).

   To re-expand the shelf, double-click the double chevrons again.

To move a shelf

- Right-click in the shelf's title bar, then click **Move Up** or **Move Down**.

Restoring deleted Library objects

In Studio, besides the standard "undo" feature for many actions, you can restore Library objects that you have deleted.

When you delete a Library object (a flow, operation, or an object in the **Configuration** folder of the Library), it is removed from the Library. However, you can view a list of the objects that you deleted, and then restore any of them that you want back. The record of deletions starts from the first deletion you made after installing Studio.

Note: You must have Read and Write permissions for a deleted object that you want to restore.

To restore a deletion

1. On the **Repository** menu, click **View Delete History**.

   Your history of deletions appears in the following dialog.

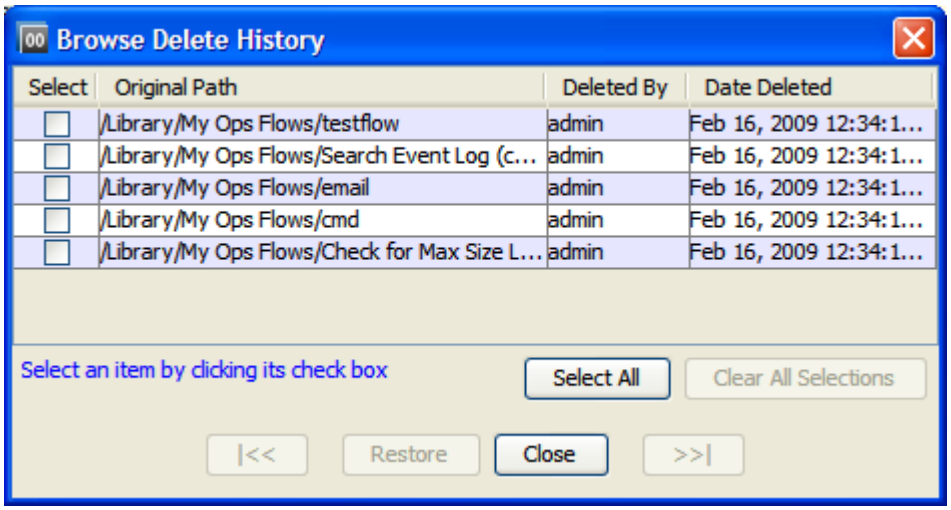


Figure 40 – The history of deletions

When the list of deletions is longer than the dialog box can show at once, you can move through the history a page at a time with either the forward (>>|) or the back (|<<) browse buttons.

2. Click the **Select** box by the deletions that you want to restore, and then click **Restore**.

   Note that you can:

   - **Select All** of the deletions for restoring.
   - **Clear All Selections** that you have made.
   - **Cancel** the restore action.

3. If the object that you're restoring was in a folder that has been deleted, the following dialog appears:

Figure 41 - Error when parent folder has been deleted

   - To stop trying to restore the object, click **Skip**.

     OR

     To specify a new folder to place the restored object in, click **Browse**, navigate to and select the new folder, and then click **OK**.

4. If the object that you're restoring resided in a folder that was moved since you deleted the object, the following dialog appears:

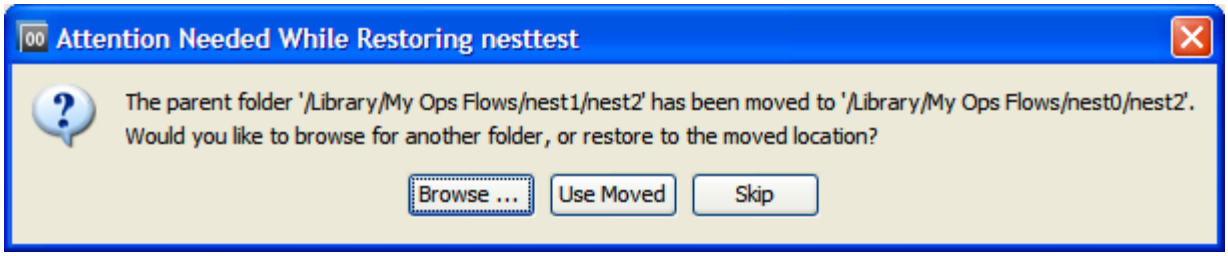


Figure 42 – Error when parent folder has been moved

   - To stop trying to restore the object, click **Skip**.

     OR

     To specify a new folder for the object, click **Browse**, navigate to and select the new folder, and then click **OK**.

     OR

     Click **Use Moved** to restore the object to the parent folder at the parent folder's new location.

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, and other characteristics. You can change those elements of a step without affecting the operation from which the step was created.

Important! Changing inputs or results in 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) that was created before you changed the inputs or results 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

Creating a step from commonly used flows and operations

Changing the Start Step

Copying steps

Modifying a step

Re-arranging steps in the flow diagram

Prompting the user before running the step

Adding steps to a flow

The first place to look for operations or flows that you can use to make a step is the Library folders that contain flows and operations in the *default content* (that is, the flows and operations that were provided with the installation). 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*.

The following folders contain default content:

- **Accelerator Packs**

  This folder contains flows designed to solve common IT problems by providing the following on most networks:

  - Complex health checks, triage, diagnosis, or remediation flows.
  - Simple flows that gather one or more pieces of data and display it to the user, or simply acknowledge alerts, gather some data, and place it into a ticket.

  The flows at the top level of an Accelerator Pack tend to be full health checks, triage, diagnosis, and remediation.

- **Integrations**

  This folder contains 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.

- **ITIL**

  This folder contains flows that automate integrations to other Enterprise-level software in accordance with ITIL specifications.

- **Operations**

This folder contains general-purpose operations that work with common protocols and software systems, such as LDAP and Linux.

These operations are sealed and cannot be changed. Because you cannot change them, they do not have any static values set for inputs. All inputs either have no assigned value (No Assignment) or prompt the user for a value. There are exceptions to this rule, such as when a very general purpose operation such as a WMI command is used.

The flows in the **Operations** folder and its subfolders are meant to work as subflows. Flows that you would want to run as the parent flow are in the **Accelerator Packs** folder.

- **Utility Operations**

  This folder contains flows and operations that provide low-level functions that can be used 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 want to make changes to an operation to create effects that are specific to your flow's current needs, you should make these changes to 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 depended on the operation's running against a different machine.

To add a step to a flow

1. From any one of the following, drag an operation or flow onto the authoring canvas:
   - **Repository** pane
   - **Search** results
   - **Bookmarks** pane

   You 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 **Rename**, then type the new name in the highlighted field, and then 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

The right-click menu facilitates creating steps from some of the most-often-used operations and flows. 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 Hosts Virtual Machines
  - MS SQL Server Connectivity
  - Restart Windows Server
- Common 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 Common 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 **Common 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 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 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:

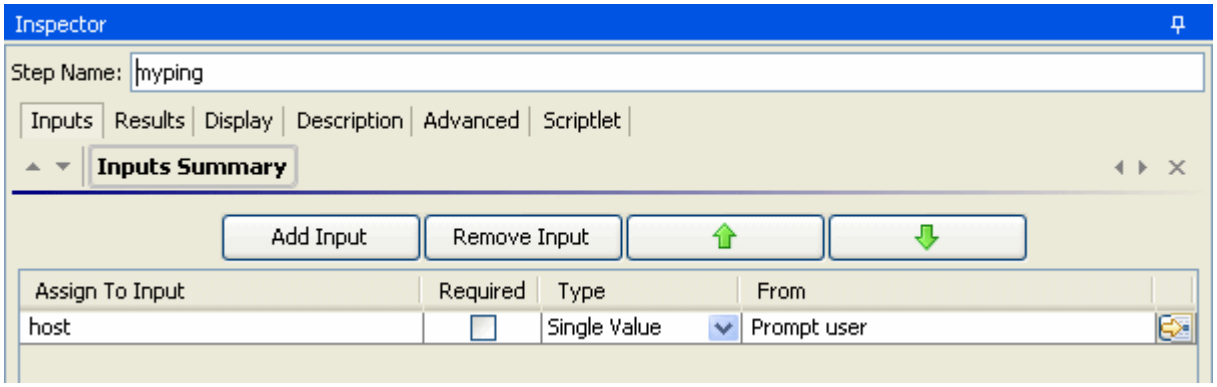


Figure 43 - The step inspector

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 holding down the SHIFT or CTRL key and then clicking the steps you want to move. You can then drag the steps as a group.

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** check box.
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.
4. 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 (represented by a response icon) 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.

The response icons are:

- **success**
- **failure**
- **diagnosed**
- **no action taken**

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. By default, the name of a response becomes the name of the transition that connects it to a succeeding step. For instance, the name of a transition from a **success** response becomes **success**.

Figure 44 - Transitions in a flow

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

- Controlling 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 OO role for the account that executes the flow.

  Gated transitions are colored red in the flow diagram in both Studio and Central.

- Handing off the flow to another person. This might be necessary if the next step requires information from someone else or if the transition is gated, requiring permissions that your account doesn't have.

  During a run of the flow, a hand-off opens a new e-mail message with the URL of the flow included in the body of the message. The person running the flow can address the e-mail message to the person taking over the flow and then send the message.

- Providing a basis for calculating the value of a run of the flow.

  When you assign a value to a transition, 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 description 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 45 - 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. The reference has the format ${*flow variable name*}.

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

Re-arranging transitions

Adding a transition

To add a transition between two steps

1. With the flow open on the authoring canvas in Studio, on the step that you want to connect to the next step, click the icon that represents one of the responses, and drag to the step that that response should lead to.

   The transition that you create is labeled with the response from which it originated.

2. To open the Inspector for the transition, select the transition by single-clicking it or its name, and then click the **Inspector** tab at the bottom of the flow canvas (or double-click the transition).
3. 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.

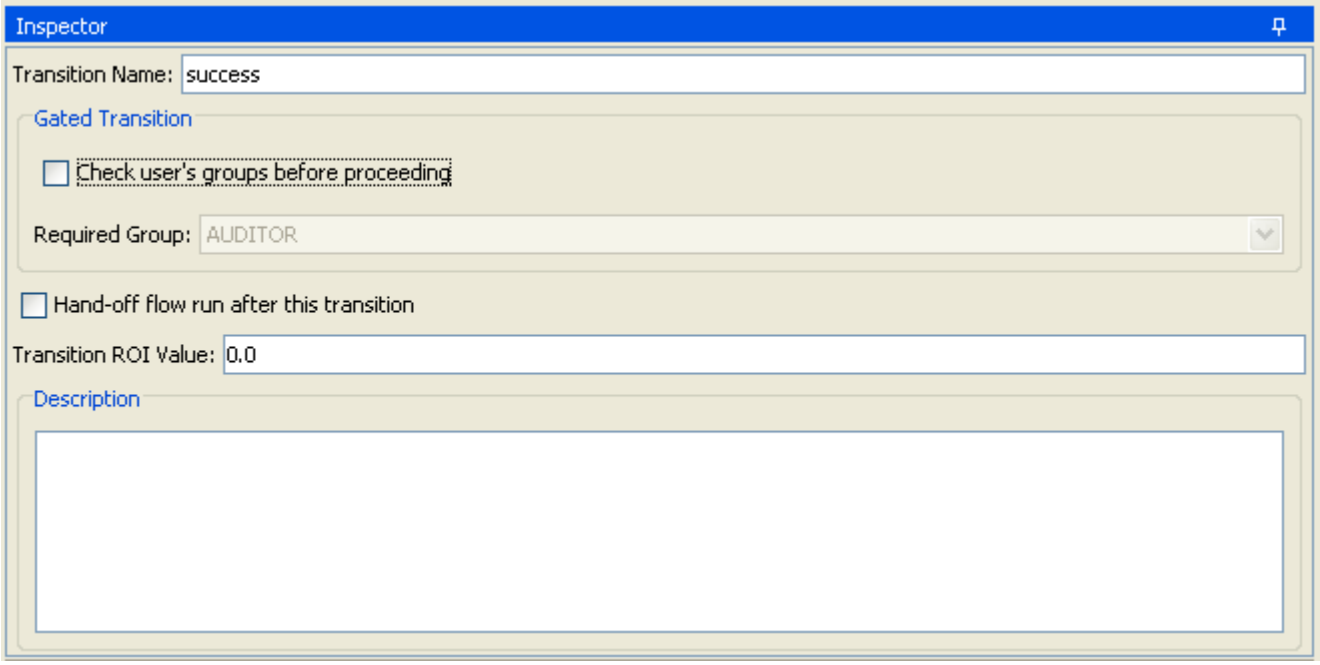


Figure 46 - Transition Inspector

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

4. To change the transition's name, in the **Name** text box, type the new name.
5. To limit who can run the step beyond the transition, in the **Gated Transition** area:
   - Select the **Check user's groups before proceeding** check box.
   - In the **Required Group** drop-down list, select the group that the user should be a member of in order to continue executing the flow.
6. To require that the run be handed off following the transition, select the **Hand-off flow run after this transition** check box.
7. To count the completion of the transition in the flow's value, type a value beside **Transition ROI Value.**
8. In the **Description** text box, type a description.

   Note: This is where you type a description of what happened in the preceding step that caused this transition to be followed. It appears in the **Results Summary** area of the **Flow Library** tab in Central.

   You can use flow variables in the description to store changeable information. For example, to identify a server whose name is stored in the **servername** flow variable, you could type, "Server ${servername} is available for connection."
9. Click **OK** and 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 stacked. 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

1. Position your mouse over the transition where you want to place the curve-defining point.
2. To create the point, hold down SHIFT and click the mouse.

3. Drag the point until the transition curves the way you want it to.

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

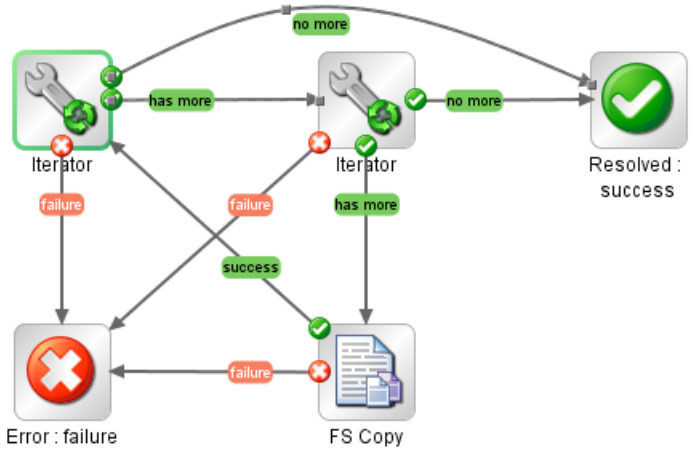


Figure 47 - Before SHIFT+dragging a transition

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

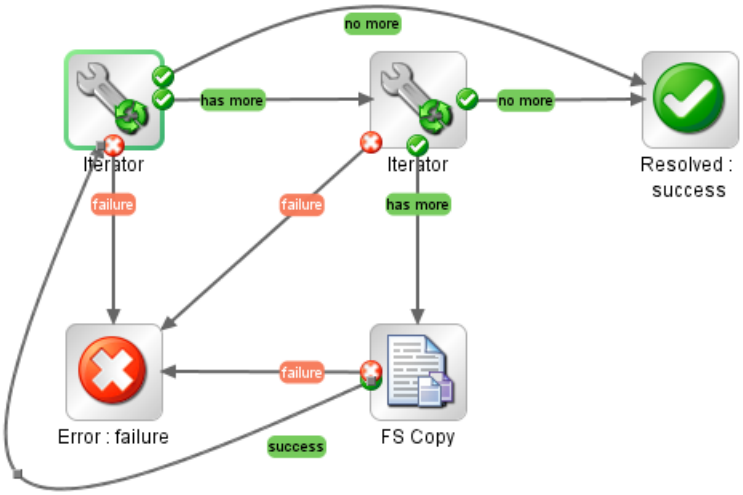


Figure 48 - After SHIFT+dragging the transition

To remove a curve-defining point

- Position the cursor over the point, hold down SHIFT, and click the mouse.

To move a transition name

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

Return steps

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

- Resolved: success
- Diagnosed: 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 step icon to the flow canvas.

   Figure 49 - Return step icons

   In each return step name, the response of the return step is shown after the colon (:). 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, such as **Error: threshold not met**.

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 steps 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 it 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).

  You can assign the value of an input to a flow variable, which you can then reference throughout the flow. This is recommended for data that is provided from outside the flow.

  Flow inputs are very useful for providing data to steps that report data for recording in Central charts. (Such steps are typically return steps, which usually get their data from flow variables. You can assign the flow variables their values from flow inputs or step outputs.)

- 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.

Operations are the base from which steps are made, and each step executes the operation from which it is made. Thus, step inputs are created from the inputs of the operation from which the step was created. Therefore, data assigned to an operation input can be used in the steps created from the operation. However, in a flow, a step's input provides the value that the operation input uses in that particular instance. You can change which source a step input gets its value from. For instance, an operation input may get its value from **flowvariable1**. When you create a flow step from that operation, you can specify that the step input get its value not from **flowvariable1**, but from **flowvariable2**. Thus, you can tailor the step's inputs to the needs of a particular flow.

For an example, 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.

You can assign a flow input's value to a flow variable and configure a step input to get its value from that flow variable. You can also configure alternative data sources (such as a user prompt, a constant, user credentials) for a step input if the flow variable that is its data source has not been defined.

Note: If you don't specify a valid data source for a step input, the input by default prompts the user. This requires, of course, that the user have the information necessary for the flow to do its work.

Creating an input

Specifying how an input gets its value

Inputs and flow run scheduling

Removing an input

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

1. To open the flow or operation's **Properties** sheet, in the **Repository** 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. For more information, see the HP OO Central Users' Guide (Central_UsersGuide.pdf) or the HP OO SDK Guide (SDKGuide.pdf).

   The new input appears in its own row.

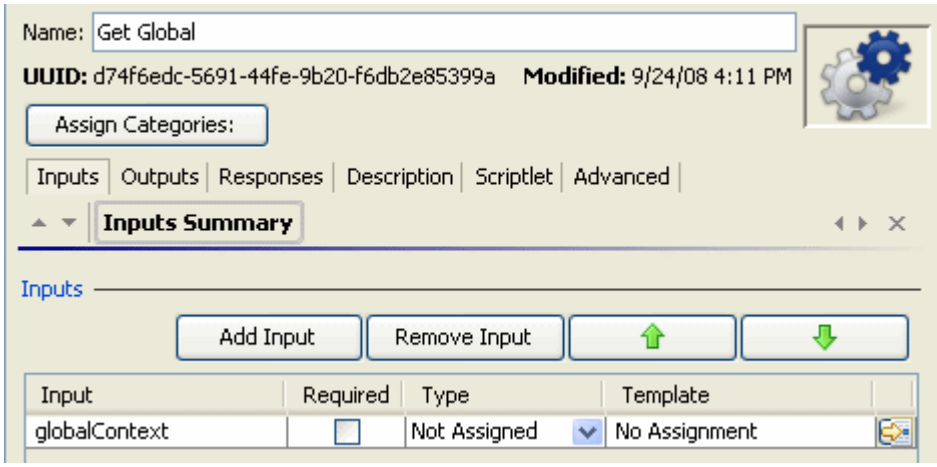


Figure 50 - Flow Properties sheet with a new input

   The column labels for the last column are slightly different in a step Inspector, but they specify the same components of an input. In both operations and steps, the last column shows the source of the value for the input if it cannot be assigned from a flow variable.

3. To change the input's name, double-click in the row's **Input** column and type the new name.

4. If the input's data is required for the step (or later steps in the flow) to function, select the **Required** check box.

   One of the first considerations in creating an input is the **Type** of its value assignment—that is what kind of value—is assigned to it. You can change the type of value assignment either here or in the input editor.

5. To change the type of value assignment, click in the row's **Type** column and select one of the assignment types from the drop-down list:
   - **Single Value**
   - **List of Values**
   - **Credentials**
   - **Not Assigned**

     The **Not Assigned** type is available in this column only for operations and flows.

   To change the source of the input's value, you open the input editor.

6. 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 ( ).

For more information on how to specify the data source of an input, see *Specifying how an input gets its value*.

Specifying how an input gets its value

On the **Inputs** tab of a flow's or operation's **Properties** sheet or a step's Inspector, you created a minimal definition of the type of input. If you don't do anything more to define how the input gets its value, the operation or step gets its value either from a flow variable of the same name as the input (if one is defined) or from the flow user's response to a user prompt. However, there are a number of ways to use the flow that require that its inputs (and its steps' inputs) **not** get their values from user prompts. The following are just a few:

- Running a flow automatically
- Getting credentials from a system account or from the credentials of the user who is logged in on the system that is running the flow
- Getting the input value from an earlier step's result or from a flow variable
- Scheduling a flow
- Running a flow from outside Central

The input editor provides you with more ways to specify how the input gets its data.

To define the data source for an input

1. To open the input editor for defining how the input gets its data and how it behaves otherwise, on the **Inputs** tab, on the line of the input of interest, click the right-pointing arrow ( ).

   The input editor looks like the following:

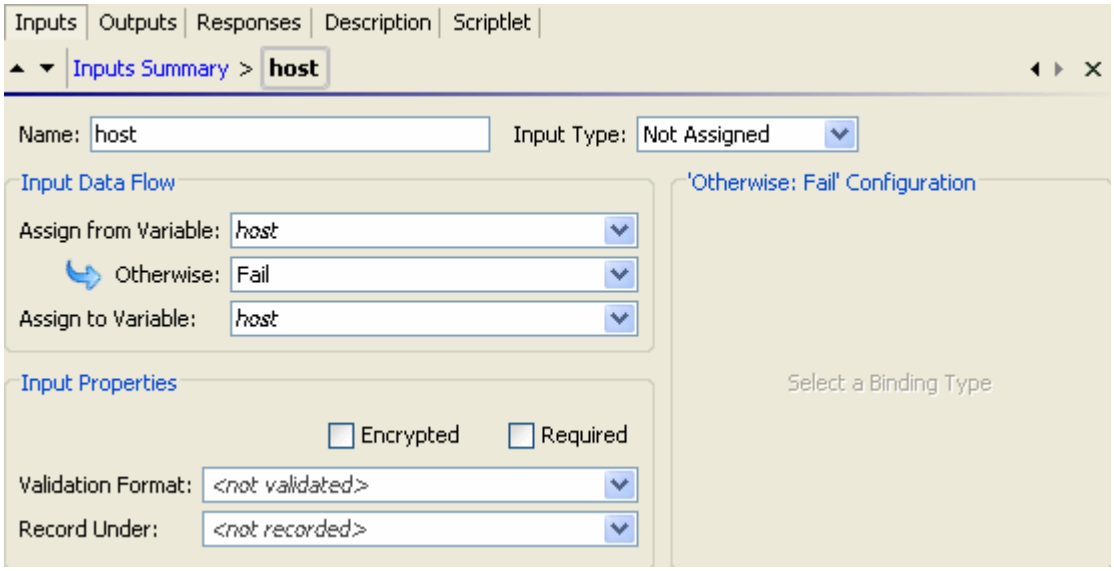


Figure 51 - Input editor

Note that here as well as on the **Inputs** tab, you can:

- Change the input's name and type.
- Make the input required (or not).

There are basically two kinds of values for inputs:

- Values (either single values or lists of values)
- Credentials

After selecting an input type, you will specify where the input gets its value. The input type affects the input's possible data sources and thus the options that you see in the input editor.

2. To select how the input gets its value, from the **Input Type** list, select one of the following:
   - **Single Value**
   - **List of Values**

     Select this input type for an input that provides multiple values to a multi-instance step or an iterator step.
   - **Credentials**
   - **Not Assigned**

   Next you specify where the input gets its value.
3. To assign the input its value from a flow variable, in the **Assign from Variable** list box, select or type the name of the flow variable.

   By default, the box contains the name of the input as the name of the flow variable.

   OR

   To specify explicitly that the input not get its value from a flow variable, select **<not assigned>**.

   Notes:

   - 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 on them, see *Reserved flow variables*.
   - If the **Input Type** is **Not Assigned** and you choose **Not Assigned** in the **Assign from Variable** box, the operation will fail.

Tip: If you assign an input a specific value but leave the input's name in the **Assign from Variable** box, the input tries to obtain its value from a flow variable of the same name as the input. As a result, 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 for the input's name when scheduling the flow, the scheduled flow run(s) will still use the specific value that you have assigned the input in Studio**.**

If you selected **<not assigned>** in the **Assign from Variable** box, or if the flow variable that you specify there doesn't exist or has no value stored in it, the input gets its value by the means you select in the **Otherwise** list box.

4. To encrypt the input's value, under **Input Properties**, select the **Encrypted** check box.

   When you encrypt the input's value, it appears as a row of asterisks when the flow is run in Central.

   Important: When the input's assignment is the **Logged-in User Credentials**, its value is always encrypted.

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

   For constant values, user prompts, and the **Use Previous Step Result**, 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.

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

6. To validate the input's value with a format, in the **Validation Format** drop-down list, select a format.
7. To record the input's value in the flow's run histories in Central, from the **Record Under** drop-down list, select the domain term under which you want to record the input's value.

   The **Input Data Flow** and **Configuration** sections of the input editor change according to the input type that you selected. To specify the data source for the input, use one of the following procedures:

   - *Creating a single constant input value*
   - *Creating a single input value from what the flow user types*
   - *Creating a single input value from a flow user selection*
   - *Creating a constant list of input values*
   - *Creating an input value list from text the flow user types*
   - *Creating an input value list from flow user selections*
   - *Creating the input's value from the previous step's result*
   - *Assigning credentials as the input's value*
8. Save your work.
9. 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**.

Creating a single constant input value

To specify a single constant input value

1. After completing the procedure *To define the data source for an input*, choose **Use Constant** in the **Otherwise** list box.

2. In the **'Otherwise: Use Constant' Configuration** section, in the **Constant Value** box, type the value for the input.

   OR

   Type the flow variable reference, using the following format:

   `${flowvariablename}`

   Tip: To specify a null value, after selecting **Use Constant** in the **Otherwise** list box, click in the **Constant Value** box, but do not type anything.

Creating a single input value from what the flow user types

To specify a constant input value obtained from user-entered text

1. When completing the procedure *To define the data source for an input*, choose **Single Value** in the **Input Type** list box.
2. In the **Otherwise** list box, select **Prompt User**.
3. In the **'Otherwise: Prompt User' Configuration** section, beside **Prompt For**, select **Text**.
4. In the **User Message** box, type the prompt text that lets the flow user know what kind of data you need from him/her.

Creating a single input value from a flow user selection

To specify a single input value from the user's selection from a list

1. When completing the procedure *To define the data source for an input*, choose **Single Value** in the **Input Type** list box.
2. In the **Otherwise** list box, select **Prompt User**.
3. In the **'Otherwise: Prompt User' Configuration** section, beside **Prompt For**, select **Selection**.
4. In the **List Source** list box, select one of the following:
   - **Selection List**

     This enables you to select from a set of pre-defined lists to present to the user. You can add to the set of pre-defined lists by creating a list. For information on creating a list, see *Selection lists for user prompts*.

     - In the **Named** list box, select the list you want to present to the user.
   - **Domain Term**

     Domain terms are specialized selection lists that have certain advantages over manually created selection lists, such as facilitating Dashboard reporting and more rigorous flow auditing.

     For instance, to specify that a flow run against certain classes of servers and not others, you could add domain terms for the various kinds of servers in your system and provide a user prompt in which the user could select the classes of servers that you want to run a given flow against.

     For more information on domain terms, see *Domain terms for Dashboard charting*.

     - In the **Named** list box, select the domain term list you want to present to the user.
   - **Flow Variable**

     The list presented to the user is populated by the contents of a flow variable.

     1. Still in the **'Otherwise: Prompt User' Configuration** area, in the **List Source** list, select **Flow Variable**.

2. In the **Named** list, type or select the flow variable that contains the list.
3. In the **Source Delimiter** box, type the character (or characters) that divide the elements of the list from each other.

5. In the **User Message** box, type the prompt text that lets the flow user know what kind of data you need from him/her.

Creating a constant list of input values

A list of values for a single input enables you to run an operation against multiple targets. For instance, you can use this method with a step configured as a multi-instance step to run an operating system Health Check or install a software update on multiple machines.

To specify a constant list of input values

1. When completing the procedure *To define the data source for an input*, choose **List of Values** in the **Input Type** list box.
2. Under **Input Properties**, in the **Input Delimiter** box, type the character or character sequence that separates the elements in the list.
3. In the **Otherwise** list box, select **Use Constant**.
4. In the **'Otherwise: Use Constant' Configuration** section, in the **Constant Value** box:
   - Separating each of the values with the character or character sequence you typed in the **Input Delimiter** box, type the values for the input to use.

     For the operation that uses this input to succeed as you intend it to, the character(s) that separate the values in the list must be exactly as you typed it (or them) in the **Input Delimiter** box, including spaces or the lack thereof.

     OR

     Type one or more flow variable references, using the following format:

     ```
     ${flowvariablename1}<delimiter>${flowvariablename2}<delimiter>${flowvariablename3}
     ```

     You can provide a list from a list of flow variable references that each contains lists, or a list of a combination of values and flow variables that contain single values or lists of values. The only requirement is that the delimiter used between the list items, whether inside or outside the flow variables, be the delimiter you defined in the **Input Delimiter** box.

     For example, assume that:

     - For the input **host**, the pipe character (|) is the delimiter
     - **flowvariableA** contains: 10.51.0.5|12.225.8.71
     - **flowvariableB** contains: 220.220.3.9|10.51.110.12

     You could successfully type the following in the **Constant Value** box:

     ```
     10.2.0.200|18.35.100.7|${flowvariableA}|${flowvariableB}
     ```

     In this case, the operation that uses this **host** input would run on the machines with the following IP addresses:

     - 10.2.0.200
     - 18.35.100.7
     - 10.51.0.5
     - 12.225.8.71
     - 220.220.3.9
     - 10.51.110.12

   For more information on assigning the value of a multi-instance step's list input to a flow variable, see *Moving data into and out of a multi-instance step*.

Creating an input value list from text the flow user types

When you prompt the user to type a list of values for the input to use, the user's list must use the delimiter specified in the input for separating values.

To specify a constant list of input values obtained from user-typed text

1. When completing the procedure *To define the data source for an input*, choose **List of Values** in the **Input Type** list box.
2. Under **Input Properties**, in the **Input Delimiter** box, type the character or character sequence that separates the elements in the list.
3. In the **Otherwise** list box, select **Prompt User**.
4. In the **'Otherwise: Prompt User' Configuration** section, beside **Prompt For**, select **Text**.
5. In the **User Message** box, type the prompt text that lets the flow user know what kind of data you need from him/her.

   Make sure that the prompt text tells the user how to type a successful list, particularly with attention to the delimiter character (or character sequence) that is required. Note particularly that inclusion of a space when the delimiter character sequence doesn't specify one is an easy way to fail.

Creating an input value list from flow user selections

When you specify that the input is a list value, the list that you present to a user in the user prompt is a multi-selection list. This could be handy for, say, enabling a flow user to pick a list of machines for the flow to target.

To specify a list of input values from user selections from a list

1. When completing the procedure *To define the data source for an input*, choose **List of Values** in the **Input Type** list box.
2. Under **Input Properties**, in the **Input Delimiter** box, type the character or character sequence that separates the elements in the list.
3. In the **Otherwise** list box, select **Prompt User**.
4. In the **'Otherwise: Prompt User' Configuration** section, beside **Prompt For**, select **Selection**.
5. In the **List Source** list box, select one of the following:
   - **Selection List**

     This enables you to select from a set of pre-defined lists to present to the user. You can add to the set of pre-defined lists by creating a list. For information on creating a list, see *Selection lists for user prompts*.

     In the **Named** list box, select the list you want to present to the user.
   - **Domain Term**

     Domain terms are specialized selection lists that have certain advantages over manually created selection lists, such as facilitating Dashboard reporting and more rigorous flow auditing.

     For instance, to specify that a flow run against certain classes of servers and not others, you could add domain terms for the various kinds of servers in your system and provide a user prompt in which the user could select the classes of servers that you want to run a given flow against.

     For more information on domain terms, see *Domain terms for Dashboard charting*.

     In the **Named** list box, select the domain term list you want to present to the user.

- **Flow Variable**

  The list presented to the user is populated by the contents of a flow variable.

  1. Still in the **'Otherwise: Prompt User' Configuration** area, in the **List Source** list, select **Flow Variable**.
  2. In the **Named** list, type or select the flow variable that contains the list.
  3. In the **Source Delimiter** box, type the character (or characters) that divide the elements of the list from each other.

6. In the **User Message** box, type the prompt text that lets the flow user know what kind of data you need from him/her or, in the case of list prompts, the fact that he or she can make multiple selections, if that is possible.

Creating the input's value from 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.

To specify the previous step's result as the source of the input value(s)

1. When completing the procedure *To define the data source for an input*, in the **Input Type** list box, choose **Single Value** or **List of Values**, depending on whether you want multiple values for the input.
2. In the **Otherwise** list box, select **Use Previous Step Result**.
3. If the input value is multiple values, then under **Input Properties**, in the **Input Delimiter** box, type the character or character sequence that separates the elements in the list.

   If the input delimiter that you specify does not match the delimiter in the previous step's result, the operation will fail.

Assigning credentials as the input's value

Besides a user prompt, you can use the logged-in user's credentials or a system account to assign a set of credentials to an input's value.

Logged-in user credentials provide credentials to the operation when the log-in account on which the flow is being run has the required access permissions to perform the flow's tasks.

A system account is a reference to a set of 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. For information on creating system accounts, see *System accounts: secure credentials*.

To specify user credentials as the source of the input value

1. When completing the procedure *To define the data source for an input*, in the **Input Type** list box, choose **Credentials**.
2. In the **Otherwise** list box, select one of the following:
   - **Prompt User**

     If you select **Prompt User**, then in the **'Otherwise: Prompt User' Configuration** section, in the **User Name and Password Message** box, type the prompt message for the user.
   - **System Account**

     If you select **System Account**, then in the **'Otherwise: System Account' Configuration** section, in the **Account Named** list box, select the system account to use for the operation's credentials.
   - **Logged-in User Credentials**

     The logged-in user is considered to be the user account under which the flow is started.

Inputs and flow run scheduling

To create a schedule that automatically starts runs of a flow, the flow must be able to run without needing user initiation or input. Any inputs that are user prompts must either be changed to specific values or be able to get their value from a flow variable that has been assigned a value.

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 **Use Constant** as the data source for the input and, in the **Constant Value** box, reference the flow variable with the syntax
  `${flowvariablename}`
  For information on assigning a data source to an input, see *Specifying how an input gets its value*.

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 or prior to the parallel split step.

  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.

  To make a scriptlet result available outside of the step, the scriptlet must create a flow variable, and assign the result to it.

- 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 by default 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 ( ).

   The input editor opens.
4. In the editor, in the **Assign to Variable** box, name the variable that you want the value assigned to.
5. Save your work.

To create a flow variable from a result

1. Open the operation's **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 local flow variables

In Scriptlet-type operations, you use a command with the following syntax in the scriptlet to create a local flow variable:

```
scriptletContext.putLocal("<localflowvariablename>", <value>);
```

where `<value>` can be a variable or an object created within the scriptlet.

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 or False, the standard Boolean inputs
Boolean – False Default	True or False
CA Unicenter Log Type	Types of logs used by CA Unicenter: CB, RS, or LOG
DCDiag Tests	Names of DCDiag tests: All, Intersite, Connectivity, Replications, NCSecDesc, CutoffServers, Advertising, KnowsOfRoleHolders, FsmoCheck, RidManager, MachineAccount, OutboundSecureChannels, ObjectsReplicated, frssysvol, systemlog, Kccevent
Delete or Zip	Delete or Zip
Event Log Searcher	*, System, Security, Application
Event Severity	Error, AttentionRequired, Warning, Information, All
Exchange Mailbox Permissions	Exchange mailbox permissions: Delete, Read permissions, Change permissions, Take ownership, Full access
FTP Type	Binary or Ascii
Host Type	Used in Windows Log Rotator (Local or Remote): Local Host or Remote Host
Host Type (Deprecated)	local or remote
HP Service Manager Resolution Fix Types	Permanent or temporary
HP Service Manager Ticket Types	Change, Problem, Incident
Hyper-V Boot Order	The devices from where a Hyper-V machine can boot: Floppy, CD, IDE, PXE
Hyper-V Controller Position	Position of the IDE Controller for the VHD of a Hyper-V VM: 0 or 1
Hyper-V Controllers	The IDE Controllers for a Hyper-V Virtual Machine: IDE Controller 0 or IDE Controller 1
IIS Site Status	Running or Stopped

Reserved flow variable	Possible values, initial values
Job History Status	Used for Backup Exec Log Analyzer: All, Successful, Failed, Completed With Exceptions
Linux Log Severity	DEBUG, INFO, NOTICE, WARNING, ERR, CRIT, ALERT, EMERG
LinuxLogRotatorOption	Archive or Delete
List Type	Used for BackupExec Log Analyzer (CAL or AAL): ClearedAlertList or ActiveAlertList
Local Or Remote	Local Host or Remote Machine
Mail Body Type	html or text
Microsoft Exchange Version	3003 or 2007
NAS Diagnostics	Used for BackupExec Log Analyzer (CAL or AAL): , 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
NetDiag Tests	All, Autonet, Bindings, Browser, DcList, DefGw, DNS, DsGetDc, IpConfig, IpLoopBk, IPSec, IPX, Kerberos, Ldap, Member, Modem, NbtNm, Ndis, NetBTTransports, Netstat, NetWare, Route, Trust, WAN, WINS, Winsock
NNM - Lifecycle State	The Lifecycle States supported by Network Node Manager: Registered, In Progress, Completed, Closed
NNM - Node Management Modes	Management Modes a node can have in HP Network Node Manager: Out of Service, Managed, Not Managed
NNM – Priority	The priorities supported by HP Network Node Manager: None, Low, Medium, High, Top
Notification Option	Health flow notification options: Email, Display, Write to File, None
Protocols	Protocols Supported by Remote command execution op(s): ssh, ROSH, Telnet, GlobalShell, RSH, local, WMI
Radia Priorities	Must, Should, May, MayNot, ShouldNot, MustNot
Remedy Case Statuses	New, Assigned, Work In Progress, Pending, Resolved, Closed
Remedy Urgency	High, Medium, Low, Urgent
Remote Or Local	remote or local
RemoteCopyProtocols	Protocols used to copy files between two remote hosts: local, scp, ftp, sftp
SAS Version	sas7 or sas6

Reserved flow variable	Possible values, initial values
Scp copy Action	To or From
Script Languages	vbscript or jscript
SE-AddOrRemove	Add or Remove
SE-VisibilityOptions	Array/Volume visibility to the Host HBA port: All, Visible, NonVisible
Selection List Delete Or Zip	Used to Select for the flow 'Windows Log Rotator': LocalDelete, LocalZip, RemoteDelete, RemoteZip
Service Desk Object	Servicecall, Incident, Change, Workorder, Problem, Person, Workgroup, Organization, Service, ServiceLevelAgreement, Project, Configuration, MainCon
Service Status	Running, Stopped, Paused
SFTP Operations	cd, ls, pwd, lpwd, lls, chgrp, chmod, chown, compression, symlink, get, lcd, lstat, mkdir, put, rm, stat, exit
Site Scope Acknowledgement Type	Enable or Disable
SMTP Greetings	HELO or EHLO
SNMP Varbind Type	TIMETICKS, COUNTER, GAUGE, INTEGER, OID, STRING, ADDRESS, HEX
Sql Authentication	Windows or Sql
SQL Database Type	Oracle, MSSQL, Sybase, Netcool, DB2
SQL Reporting Server Format	HTML 3.2, HTML 4.0, HTMLOWC, MHTML, CSV, XML
Storage Redundancy Options	Redundant Parity Based Storage or Redundant Mirrored Storage
String Comparator Match Type	Exact Match, Contains, Contains Once, Does Not Contain, Match All Words, Match No Words, RegEx
Tomcat Actions	Start, Stop, reload, undeploy
Transport Layer Protocols	TCP or UDP
User Session Control Levels	Active Directory User Session Control Levels: EnableInputNotify, EnableInputNoNotify, EnableNoInputNotify, EnableNoInputNoNotify
VHD Type	Type of virtual hard disk: Fixed or Dynamic
Virtualization Technologies	This selection list allows users to select which virtualization technology their servers run, when running flows from the Virtualization Accelerator Pack: VMWare Virtual Infrastructure, Microsoft Hyper-V, Citrix XenServer
VMReference Type	Ways to reference a virtual machine in VMWare ESX/VC

Reserved flow variable	Possible values, initial values
	integration: INVENTORYPATH, NAME, IP, HOSTNAME, UUID, VMID
VMWare Actions	Start, Stop, Enumerate, Status
VMWare Managed Object Types	VirtualMachine, Network, ComputeResource, ResourcePool, HostSystem, DataStore
VMWare Virtual Machine Power States	poweredOn, poweredOff, suspended
Windows Access Control	Allow or Deny
Windows Clustering States	Online or Offline
Windows File System Inheritance Propagation	This folder, subfolders and files, This folder and subfolders, This folder and files, This folder only, Subfolders only, Files only, This object only
Windows File System Permissions	Append Data, Change Permissions, Create Directories, CreateFiles, Delete, Delete Subdirectories and Files, Execute File, Full Control, List Directory, Modify, Read, Read and Execute, Read Attributes, Read Data, Read Extended Attributes, Read Permissions, Take Ownership, Traverse, Write, Write Attributes, Write Data, Write Extended Attributes
Windows Registry Value Type	REG_SZ, REG_EXPAND_SZ, REG_BINARY, REG_DWORD, REG_MULTI_SZ
Windows Script Languages	VBScript or Jscript
Windows Service Startup Mode	Boot, System, Auto, Manual, Disabled
Windows Service Status	Running, Stopped, Paused
WMI Query Format	csv or text
Yes No	Yes or 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 e-mail. It would speed up resolution if the flow could keep running with subsequent steps while the one step is creating the ticket or e-mail.

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 can use a step that has concurrent processing.

  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 is 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 e-mail 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 e-mail.

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 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, suppose the following events take place in the following order:

  a. In Lane A, step A1 passes “42” to **threshold**.

  b. In Lane B, step B2 passes “34” to **threshold**.

  c. Lane B completes.

  d. Lane A completes.

  The value in **threshold** would be “42”, even though step B2 passed its value (“34”) to **threshold** after step A1 passed its value (“42”) to **threshold**.
- For parallel split steps, permissions for execution of flows and operations and for use of other elements (such as system accounts) 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. 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 it 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 do not contain return steps. Rather, 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.
   - 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.

3. 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 ( ).

4. 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:

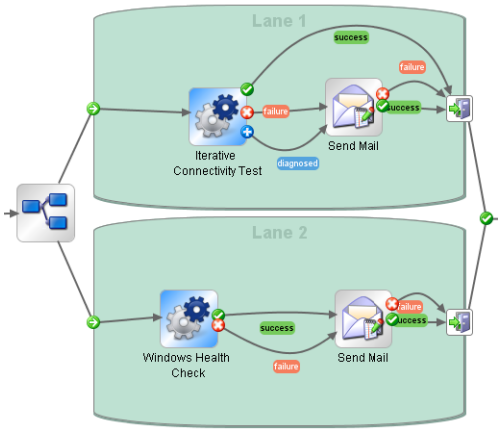


Figure 52 - Parallel split steps

Moving or copying parallel split steps and their components

To move a parallel split step or its components

- With the flow open in the **Design** view, do one of the following:
  - To move a parallel split step, click the **Parallel Split step** icon ( ), which represents the entire step, and drag.
  - To move a lane step, select the step and drag, either within the lane or to another lane.

There is another way to move lanes up and down in the flow canvas. 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 change the visual order of lanes

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

To copy a parallel split step or its components

- With the flow open in the **Design** view, do one of the following:
  - Right-click the lane or step you want to copy, or right-click the **Parallel Split step** icon ( ) of a parallel split step to copy the entire step, and then click **Copy** from the drop-down menu. Move the cursor where you want to paste the lane, step, or entire parallel split step, right-click and then click **Paste** from the drop-down menu.
  - Click the lane or step you want to copy or the **Parallel Split step** icon ( ) of a parallel split step to copy the whole step, and then do one of the following:
    - Press CTRL+C then move the cursor where you want to paste the lane, step, or whole parallel split step and press CTRL+V.
    - Click the **Copy** icon ( ) in the **Authoring** pane toolbar. Move the cursor where you want to paste the lane, step, or whole parallel split step, and then click the **Paste** icon ( ) in the toolbar.
    - Click the **Edit** menu, and then click **Copy**. Move the cursor where you want to paste the lane, step, or whole parallel split step, click the **Edit** menu, and then click **Paste**.

Adding a lane

To add a new lane

- Right-click in an existing lane and then click **Add Lane** from the drop-down menu.

The new, empty lane is added below the currently selected lane.

Duplicating a lane

To duplicate a lane

- Right-click in a lane and then click **Duplicate Lane** in the drop-down menu.

A new, empty lane with the same title as the one you copied appears directly below it.

Deleting a lane

To delete a lane

- Do one of the following:
  - Right-click the lane you want to delete and then click **Remove Lane** from the drop-down menu.
  - Click the lane you want to delete, and then do one of the following:
    - Click the **Remove** icon ( ) or the **Cut** icon ( ) in the **Authoring** pane toolbar.
    - Click the **Edit** menu and then click **Cut**.

Resizing a lane

To resize a lane

1. Select the lane by clicking in a blank part of it.
   Handles appear at the sides and corners.
2. Drag the side or corner handles.

Renaming a lane

To rename a lane

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

Changing a lane's start step

To change the start step of a lane

- Do one of the following:
  - Right-click the lane step that you want to be the lane's start step, and then click **Set Start Step** from the drop-down menu**.**
  - Drag the lane-start icon connector from the lane step that is its current target to the step that you want to be the lane's start step.

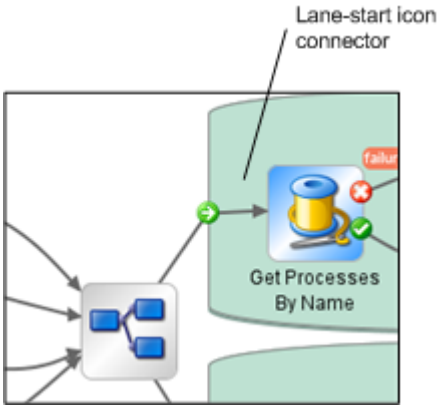


Figure 53 - Lane-start icon connector

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 flows*.

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 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 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

A multi-instance step has the additional response **group done**. 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:

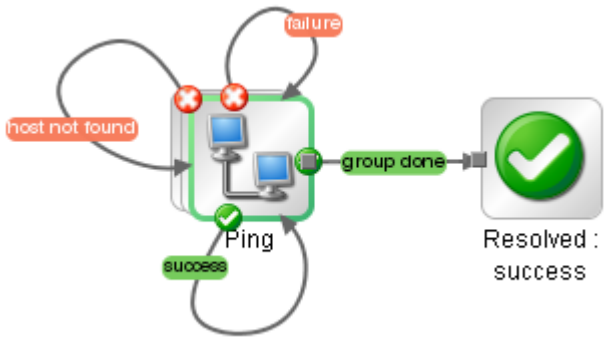


Figure 54 - The essentials of using a multi-instance step in flow design

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 e-mail 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.

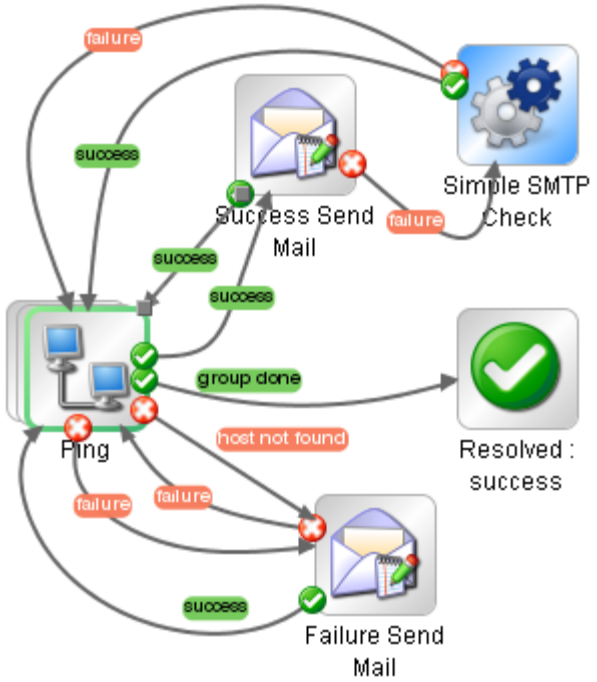


Figure 55 - Good design: Connecting response destinations back to multi-instance step

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.

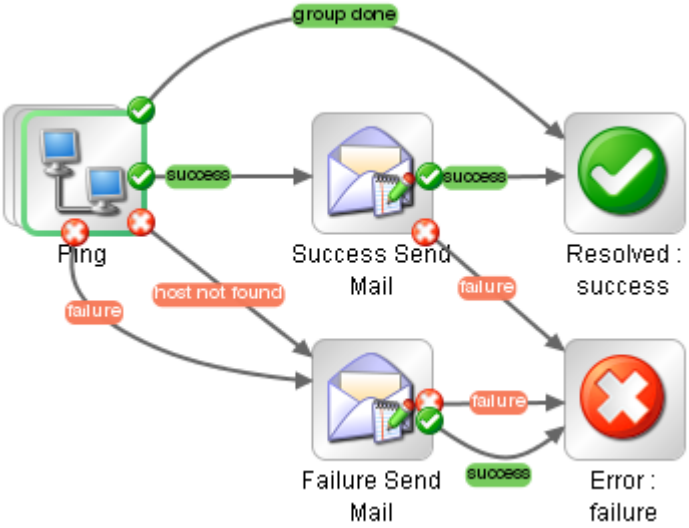


Figure 56 - Bad design: Response destinations NOT connected back to multi-instance step

- 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.

This is how processing works in a multi-instance 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 nonblocking 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):

   Figure 57 - 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, in the **Input Type** drop-down list select **List of Values**, and then, in the **Input 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 on the **Description** tab for the step.
   - When you make a step into a multi-instance step, you cannot 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 **Constant Value** text box (when you select **Use Constant** 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.

List inputs, flow variables, and multi-instance steps

A list input for a multi-instance step takes different values depending on whether the list input is assigned to a flow variable with the same name as the input or a flow variable with a different name.

- If the list input is assigned to **a flow variable with the same name**: After each evaluation of the multi-instance step, the value of the variable is the element of the list that corresponds to the result being processed.
- If the list input is assigned to **a flow variable with a different name from the list input**: The list input's value is stored in two flow variables:
  - The flow variable with the list input's name has as its value the element of the list that corresponds to the result being processed
  - The variable that has a different name from the list input is assigned the whole list value.

Therefore, if you want to reference each element in the list input's list, you need to reference the flow variable that has the same name as the list input.

For example, suppose a flow like this one.

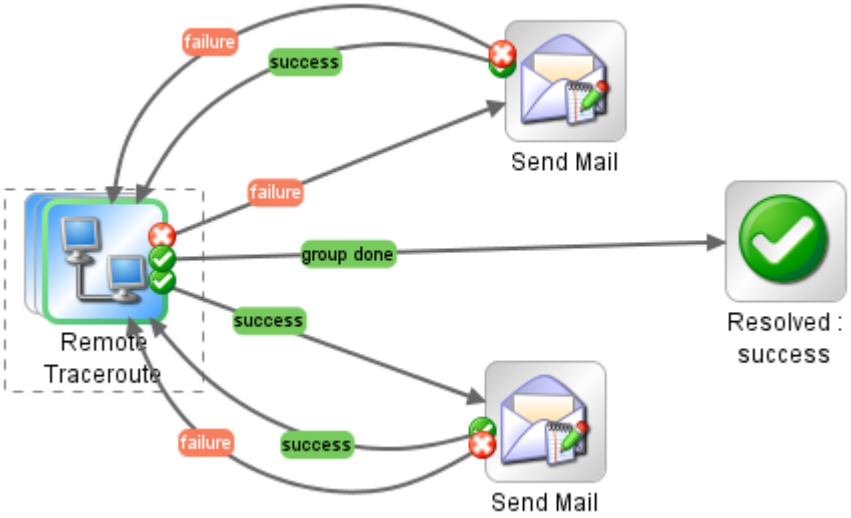


Figure 58 - Multi-instance step in flow

This flow has the following features:

- The **Remote Traceroute** step is a multi-instance step.
- The **Remote Traceroute** step has an input flow variable named **target** which is a list input and is assigned to a flow variable named **machine**.
- The values in the list of the input **target** are the names of the machines that **Remote Traceroute** will target.
- The **Send Mail** steps should send a success or failure e-mail for each target machine in **target**.

To include the name of each target machine in the body of the e-mail that one of the **Send Mail** steps sends for each result, the body of the e-mail must reference the flow variable **target**, not the

flow variable **machine**. If the e-mail body referenced **machine**, it would print the entire list, rather than the particular target machine for which **Remote Traceroute** just succeeded or failed.

You might specify, for instance, that the e-mail body for the **Send Mail** steps on the success path and failure path include the following, respectively:

- Remote Traceroute succeeded for ${target}.
- Remote Traceroute could not connect to ${target}.

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 flows*.

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 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:

- Removing checkpoints doesn't change how much data is sent to the database in the course of a flow run, but only at what points data is sent.
- 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, in the **Step Persistence** area, 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 **HTTP Client 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 responses, for evaluating output, determining results and responses, or adding variables to the flow.

You can use JavaScript or Perl scriptlets:

- To filter a flow, step, or operation's results.

  One of the uses for filtering an operation's results is as part of a rule that determines an operation's response.

- In the body of an operation, step, or flow (on the **Scriptlet** tab of each).

  This is not a generally recommended use of a scriptlet. Scriptlets are best used for filtering results.

Tip: If you know how to script in either of the two scripting languages used in OO—JavaScript (Rhino) or Sleep—the OO scriptlet templates for those languages define the syntax and objects that 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 a 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, select the language in which you will write your script in the **Type** drop-down list and then click **Insert Template**.

Tip: For examples of existing scriptlets, look in operations in the content that ships with OO (such as the operations under the **Operations\Operating Systems\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*.

Best Practice: When creating a scriptlet operation, in the scriptlet, specify the scriptlet response as **success**. Then, on the **Responses** tab of the operation, select **failure** as the default response.

To create a scriptlet rule for an operation response

1. On the **Responses** tab of the operation's **Properties** sheet, click **Add Response** and then enter a name for the response in the box that appears.

   OR

   Right-click the right-pointing arrow on the row of the response.
2. In the **Rule Type** drop-down list, select **Scriptlet**.
3. Click the right-pointing arrow on the row of the response.
4. 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.

5. Save your work.

Debugging a scriptlet

You can only debug scriptlets on flows that reside in local repositories. To debug a scriptlet in a public repository, update a local repository from the public repository. Then, with Studio connected to the local repository, run the JavaScript debugger on the flow containing the scriptlet, using the following procedure.

To debug a scriptlet

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

   The JavaScript Debugger starts.

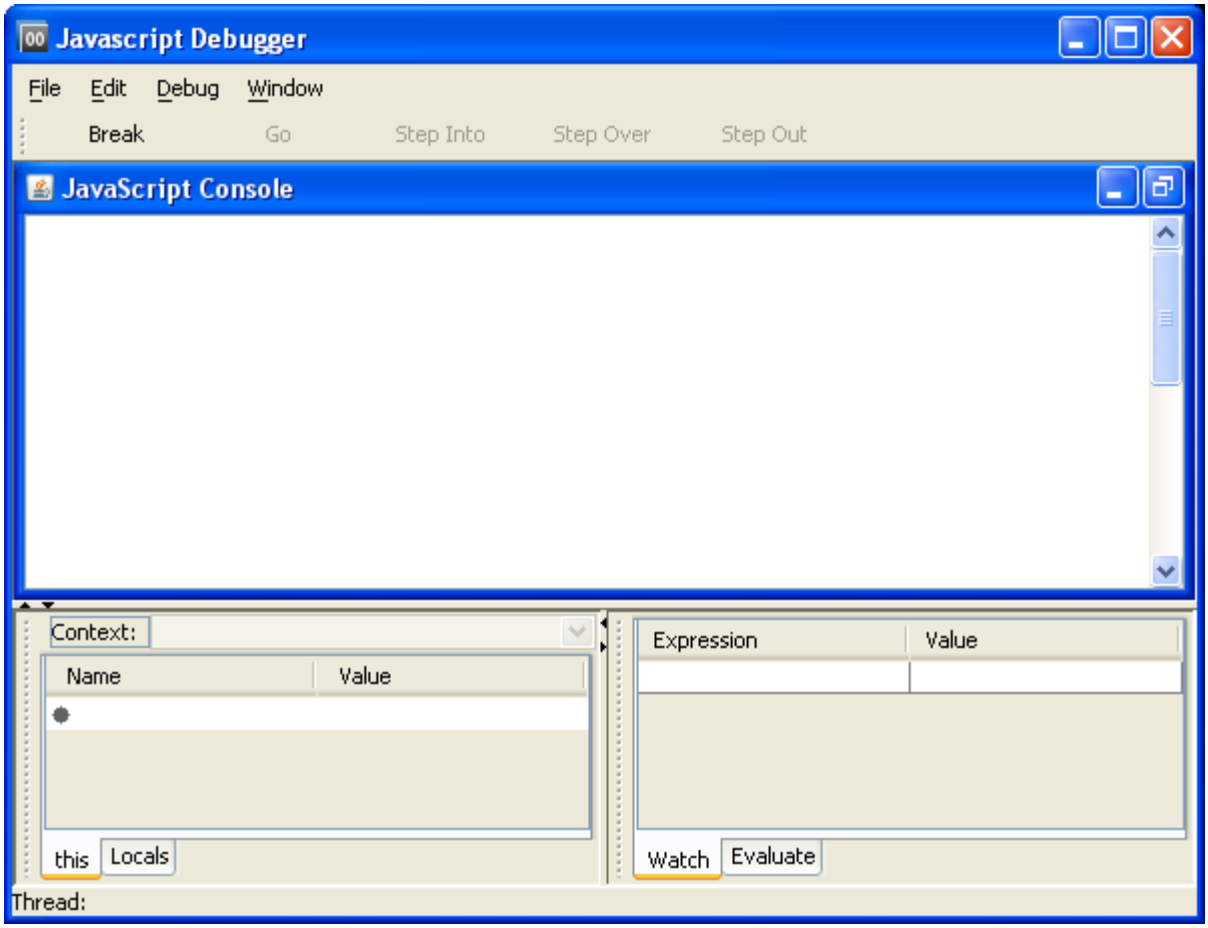


Figure 59 - Javascript Debugger

2. From the **Debug** menu, select **Break on Function Enter**.

   All the scriptlets that the flow, its steps, and its operations contain load into the 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.

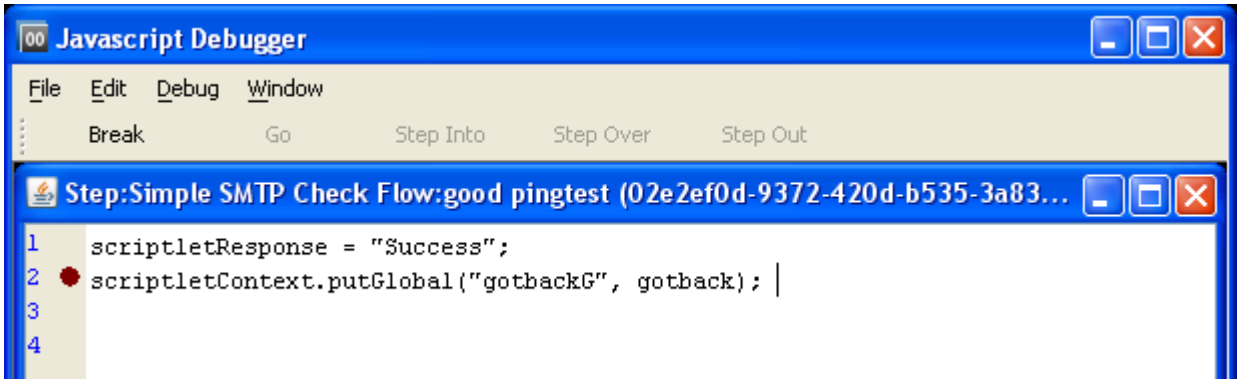


Figure 60 – JavaScript Debugger breakpoint set for scriptlet line

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 **Debug Flow** button ( ).

   The debugger starts, debugging the flow either in Central (if the flow is in the public repository) or in Studio (if the flow is in a private repository).

   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** pane.

   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 it 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 **Repository** pane, 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

There are two kinds of operation outputs.

- 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.

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.

Specifying an operation's primary output

When you create an operation, you can specify its primary output. For more information on outputs and results, see *Outputs and results*.

To specify an operation's primary output

1. To open the operation for editing, in the Library, right-click the operation, and then select **Open**.
2. In the operation's editor, click the **Outputs** tab.
3. From the **Extract Primary Output from Field** drop-down list, select a source field.

For information on the data that are provided in each output field, click the operation's **Description** tab.

You might choose to create a filter for the output in order to extract the particular data you want. For more information on creating filters, see *Filtering outputs and results*.

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 designate a field as the primary output of an operation

1. To specify the source of a primary output, in the **Outputs** tab of the operation, from the **Extract Primary Output 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. 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 add a secondary output for an operation

1. 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**.

2. 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*

   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.

3. Save your work.

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. After creating the filter or filters you need, close the filter editor.

   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.
     - **APPEND** - Place this value at the end of the current value of the flow variable or flow output field.
     - **PREPEND** – Place this value in front 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.

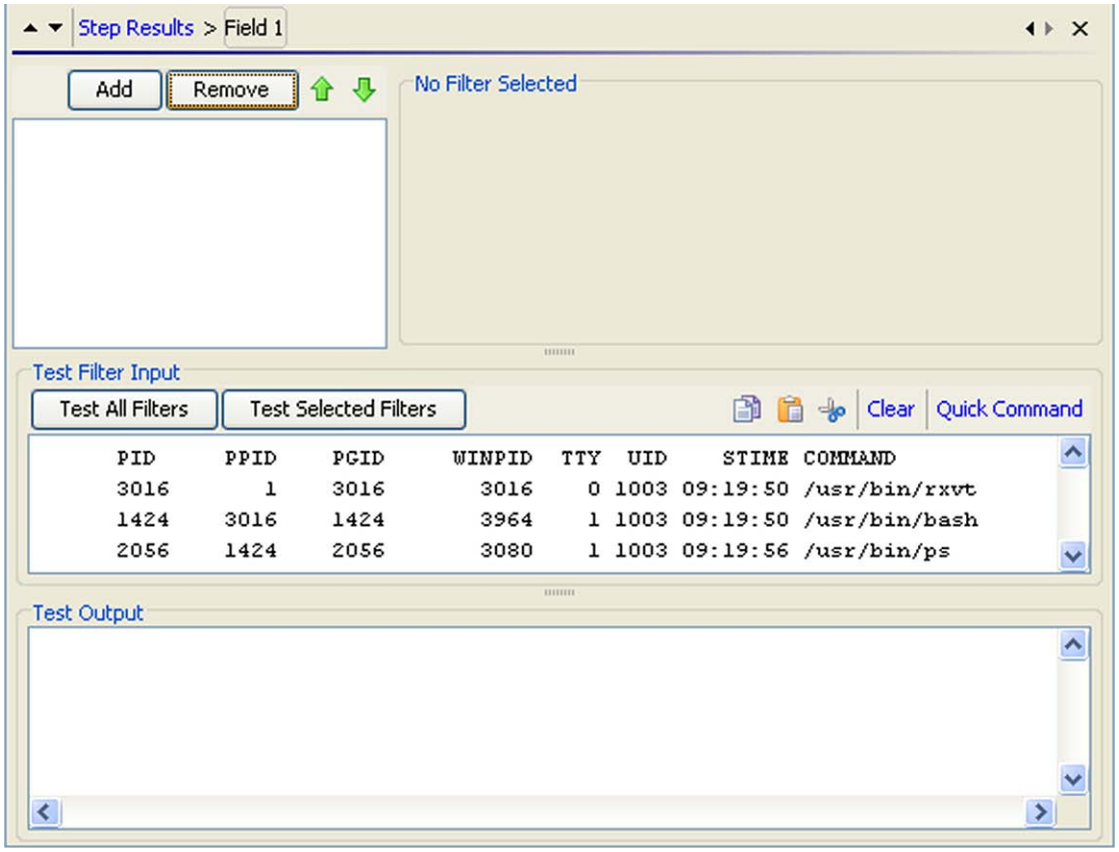


Figure 61 - Filter editor

- 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 (for information on the types of filters, see *Filter details*).
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

See the following subtopics for defining particulars of each kind filter (for information on creating filters, see *Creating a filter*):

- *Diff Case*
- *Extract Number*
- *Format*
- *Line Count*
- *Regular Expression*
- *Remove Duplicate Lines*
- *Replace*
- *Round Number*
- *Scriptlet*
- *Select Line*
- *Select Range*
- *Sort*
- *Strip*
- *Strip Whitespace*
- *Table*

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**.

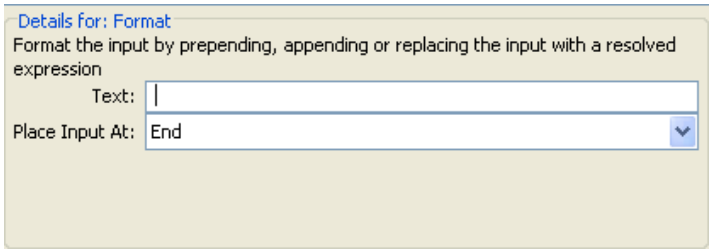


Figure 62 – Format filter definition

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*.

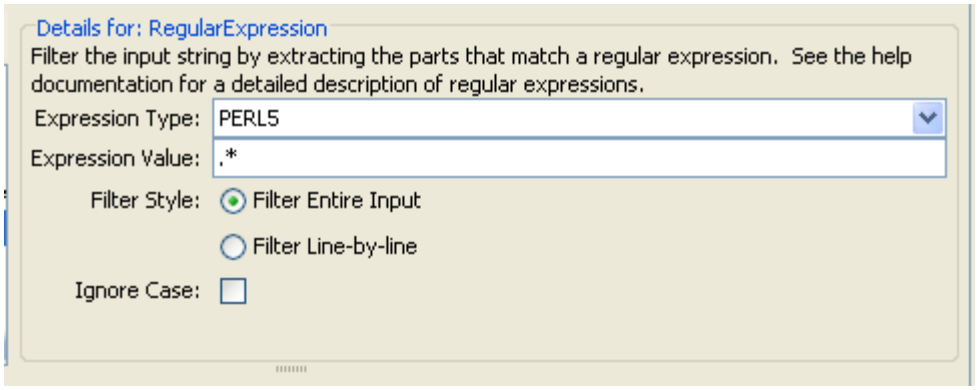


Figure 63 - Regular Expression filter definition

- 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.

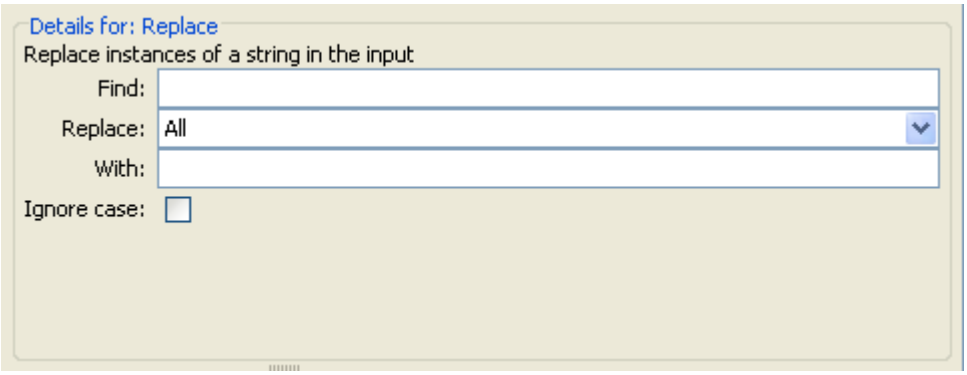


Figure 64 - Replace filter definition

- 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** check box.

Round Number

Rounds numbers as you specify in the filter editor.

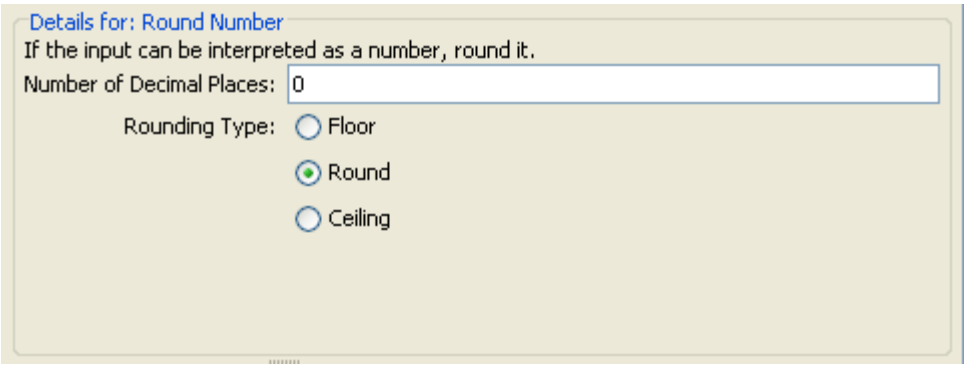


Figure 65 - Round Number filter definition

- 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** box.
- 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.

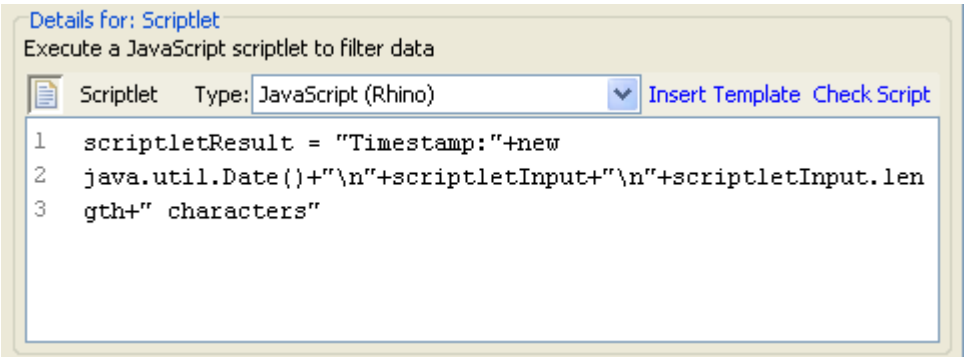


Figure 66 - Scriptlet filter definition

- From the **Type** drop-down list, select the **JavaScript (Rhino)** or the **Sleep** scriptlet language.

  The default is **JavaScript (Rhino**), which places an initial line of a scriptlet in the text box.
- 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 for 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.

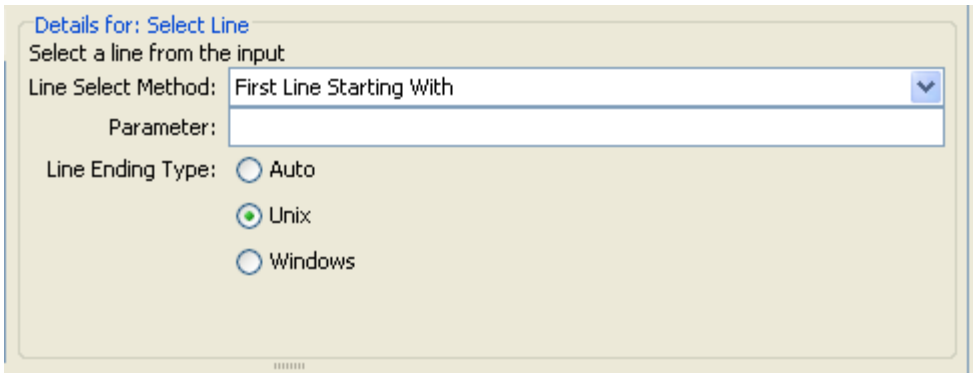


Figure 67 - Select Line filter definition

- 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: its length in characters and the position of its first character from the start of the input data.

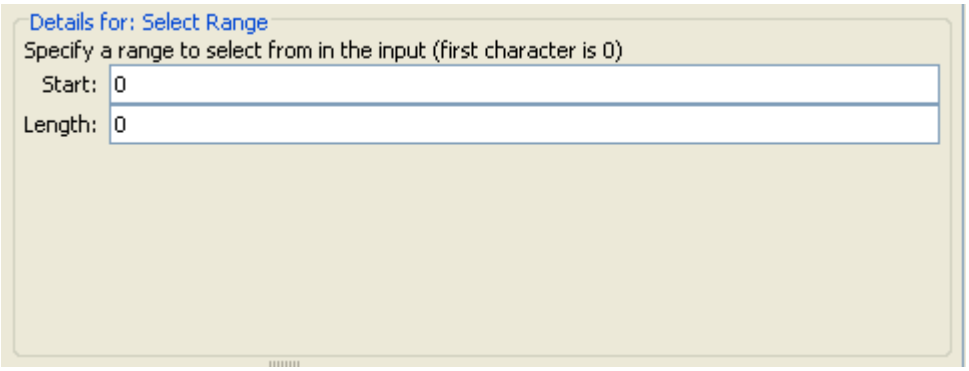


Figure 68 - Select Range filter definition

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.

Figure 69 - Sort filter definition

Specify the direction of the sort: for ascending order, leave the **Ascending** check box selected; for descending order, deselect the **Ascending** check box.

If you choose to treat the data as numbers by selecting the **Treat as Numbers** check box, the lines that do not begin with numbers are sorted to the bottom.

Strip

Strips characters from the raw results.

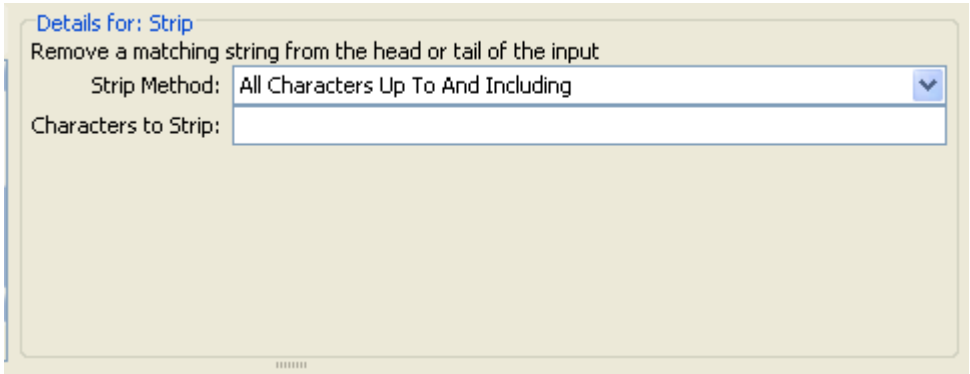


Figure 60 - Strip filter definition

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.

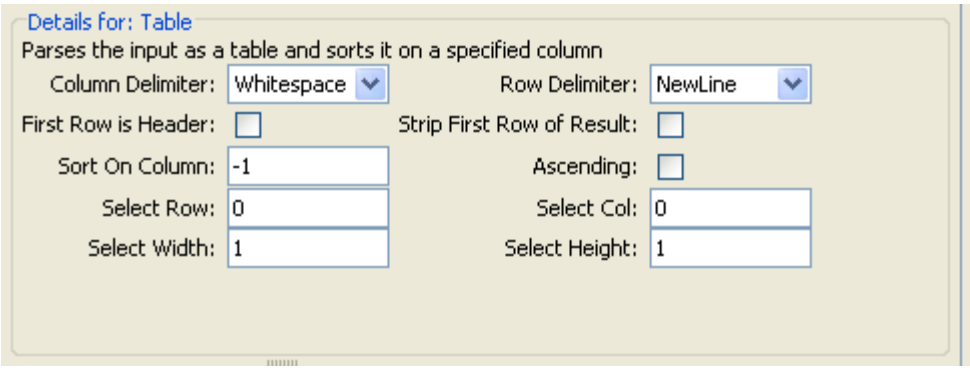


Figure 70 - Table filter definition

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.

  The value **-1** means do not sort on any column.
- 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**

The value -1 has specific effects for the **Select Row**, **Select Col**, **Select Width**, and **Select Height** boxes:

- In **Select Row**, -1 selects all the rows in the data.
- In **Select Height**, -1 selects all the remaining rows in the data below the row specified in **Select Row**.
- In **Select Col**, -1 selects all the columns in the data.
- In **Select Width**, -1 selects all the remaining columns in the data to the right of the column specified in **Select Col**.

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 a 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	Type	Rules
port open	☐	☐		1 Rule [Source: returnCode, No Filters, Exact Ma...
port closed	☐	☐		1 Rule [Source: returnCode, No Filters, Exact Ma...
host not found	☑	☐		1 Rule [Source: returnCode, No Filters, Exact Ma...

Figure 71 - Step responses and response types

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.
- A default response that you specify 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** and then type a name for the new 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.

The 'Port listening' response will be selected when all of the following rules are true Add Remove

Apply Rule To Field	Rule Type	Rule Text
code	Match At Least One Word	

Figure 72 - 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 details editor, at the right end of the rule's row, click the right-pointing arrow ( ).

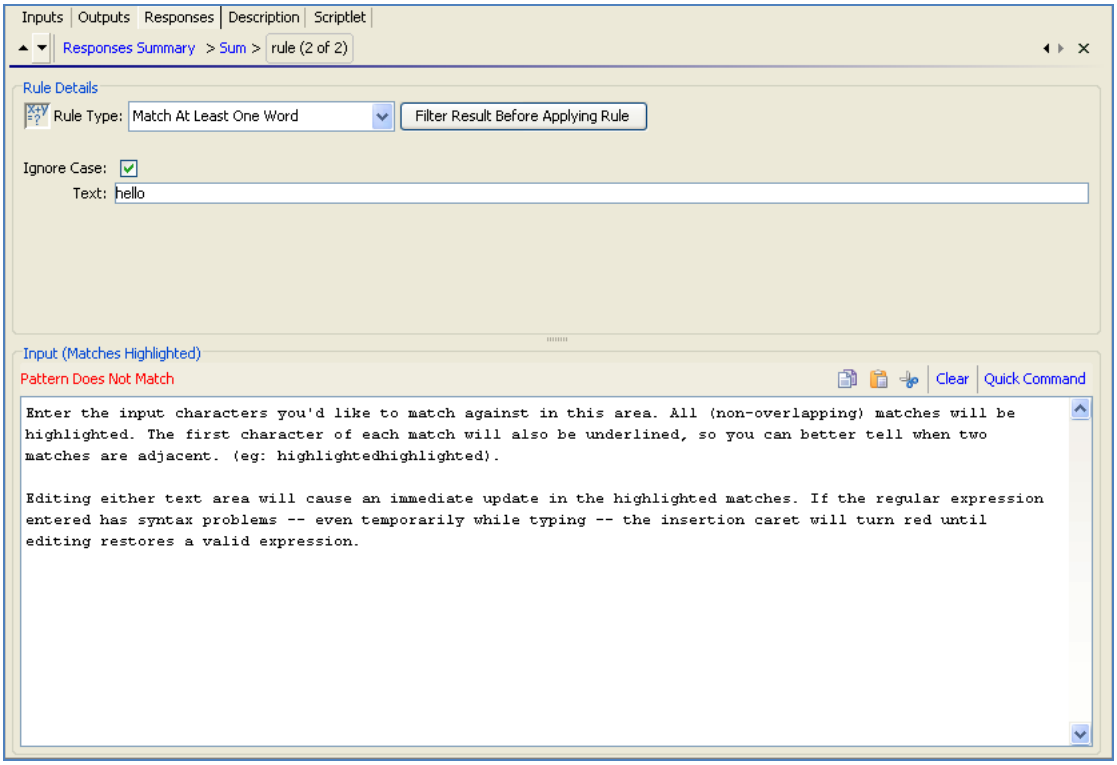


Figure 73 - Rule details editor

Note: If you choose **Scriptlet** as the rule type, the rule details 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 outputs and results*.

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

The rule type that you chose in the response rule editor appears in rule details editor, already selected in the **Rule Type** drop-down list.

9. To choose another rule type, select a different one in the **Rule Type** drop-down list.
10. 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*.

11. To work on another rule for the operation's response, 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 **Find Inactive Domain Administrators** flow has three responses. To see how the following responses are obtained, in Studio, find and double-click the **Find Inactive Domain Administrators** flow in the Library.

- Failure—The flow was unable to get a list of inactive domain administrators for some reason.
- Success—The flow did not find any inactive domain administrators.
- Diagnosed—The flow found inactive domain administrators.

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 the **Responses** tab.
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 **Responses** tab, click the response you want to delete and then click **Remove Response**.

3. Save your work.

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** pane, click the **Icons** tab on the right side of the flow canvas.

Figure 74 - Icons pane

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

Creating system properties

System properties are both powerful and potentially dangerous, both for the same reasons: Because of what they are and how they work in flows.

System properties are flow variables that are stored in the OO global context. Because their scope is global across the repository any reference to the system property obtains the system property's value. This can create desirable efficiencies of authoring and running flows.

On the other hand, you must be cautious about creating system properties because of their global scope and because flow authors cannot change the values of system properties either when he or she creates or changes an operation, a flow input, or a flow step's input. Nor can the Central user assign a different value to the system property when the flow is run in Central. If an input can get its value from a flow variable, then it does: Its value is not assigned from a user response, a specific value, or any other source. Because system properties are global flow variables, they are available to any run of any flow in the repository.

Further, a system property and its value are visible only is in the Studio Debugger. The only other way to detect the existence of a system property is if a flow uses the "wrong" value for an input no matter how you change the relevant step in Studio. For instance, suppose there is a system property ${overrideJRAS} defined with the value:

https://htudor.north.mycorp.net:9004/JRAS/services/RCAgentService

If you created an input that tried to get its input from ${overrideJRAS} and alternatively obtained its value from a user prompt in which the user could specify a RAS reference that pointed to a different RAS, the user prompt would never appear, because the input would have already gotten its value from the system property. Further, defining the system property could break any flows that also use ${overrideJRAS}, or cause them to have unexpected results.

To create a system property

1. In the **Repository** pane of Studio, open the **Configuration** folder, right-click the **System Properties** folder, and then click **New**.
2. In the text dialog box that appears, type a name for the new system property, and then click **OK**.

   The system property's **Properties** sheet appears.
3. In the **Description** box, type a description of the system property.
4. In the **Property Value** box, type a value for the system property.

   For example, to create a system property that provides a JRAS override throughout the OO repository, you would:

   - Name the system property:

     overrideJRAS
   - Give it a value such as:

     https://htudor.north.mycorp.net:9004/JRAS/services/RCAgentService

   The system property's reference in any flows would be:

   ${overrideJRAS}

Flow design

When creating flows, keep the following 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 subflows to simplify flow design*.

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 some list inputs such as the following (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 *Modifying a step*.

2. Click the **Results** tab and 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. Click the **Results** tab and 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

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.

To switch the operation that the step is based on

1. In the flow's **Design** view, open the step's Inspector, and then click the **Advanced** tab.

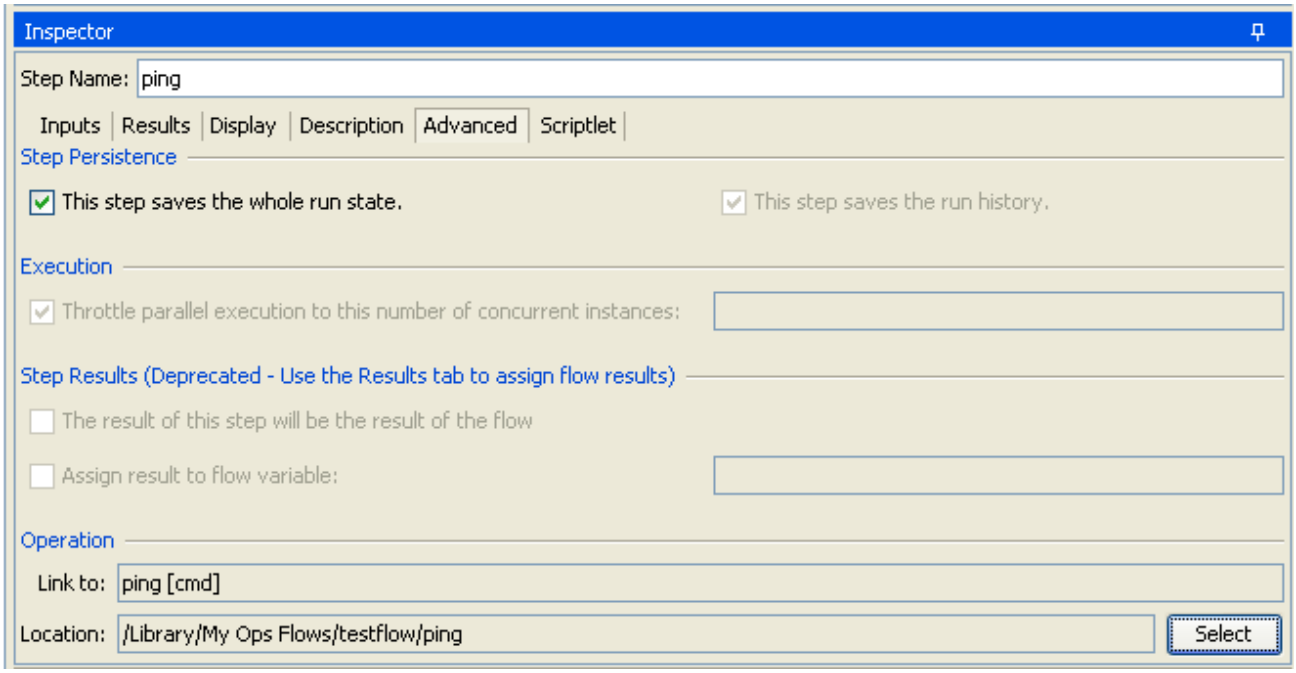


Figure 75 - Step Inspector Advanced tab

2. On the **Advanced** tab, under **Operation**, click **Select**.

   The **Select Operation** dialog box appears.

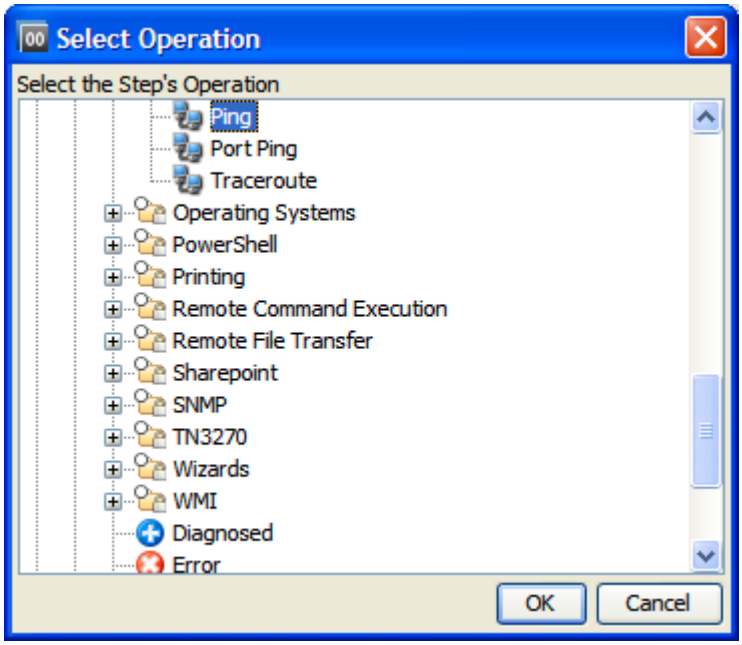


Figure 76 - Select Operation box

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. Review and make any changes necessary to the value assignments for inputs, to reflect differences between the old operation's inputs and those of the new one.
6. Save your work.

Step descriptions

A description that includes likely search words helps you or others find your flow. Suppose, for example, that you want to find an operation that had the description “Performs a ping and a traceroute to the server”. You might specify the following search criteria on the Search tab:

```
description:ping traceroute
```

The results produced by such a search would include 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 **Use Constant**.
- 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 **Use Constant** data source assignment.

  For information on changing data sources for inputs, see *Inputs: Providing data to operations*.

Debugging flows

The debugger in Studio helps you track down the 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.

The Studio debugger debugs a flow in Central if the flow is in the public repository or in Studio if the flow is in a private repository. This is especially important to be aware of when a flow contains parallel processing, as it behaves differently in Studio than it does in Central.

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

When you debug a flow using the Studio debugger:

- If you are connected to the public repository, the flow is debugged in Central.
- If you are connected to a private repository, the flow is debugged in Studio.

Best Practice: It is best practice to debug subflows before debugging their parent flows.

When debugging a flow you might find the keyboard shortcuts useful. For the debugger's keyboard shortcuts, see *Debugger keyboard shortcuts*.

The following procedure applies both to debugging a flow in Central (connected to the public repository) and to debugging one in Studio (connected to a private repository).

To debug a flow

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. The toolbar for the debugger contains the controls you'll use to run the flow, whether to completion or step by step. When you run the flow from start to finish, the run is interrupted only by any breakpoints that you may have set.

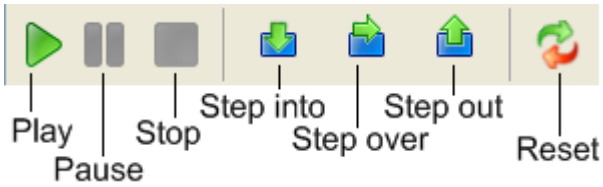
   

   Figure 77 - Studio debugger toolbar

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 the **Windows Health Check** flow.

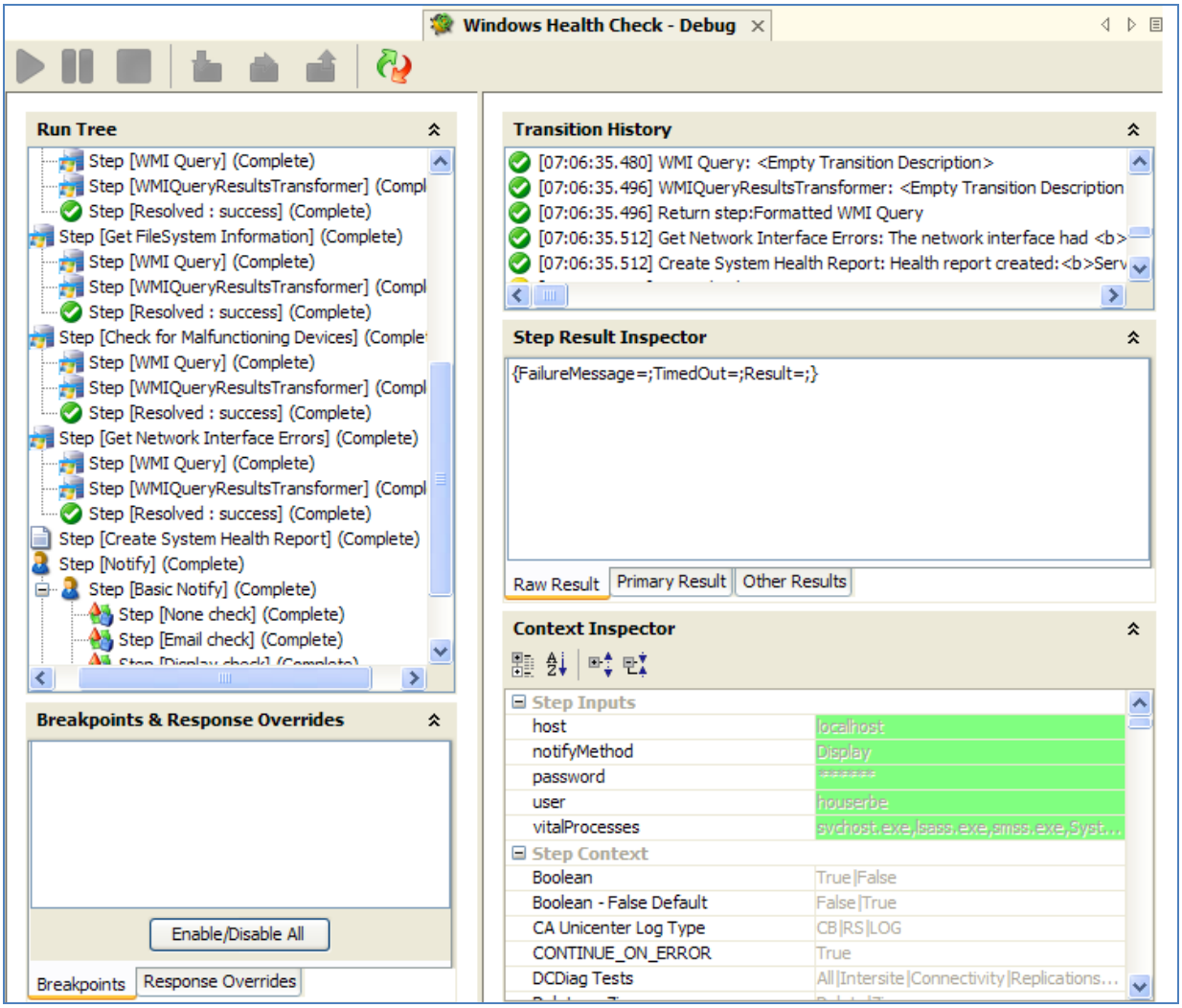
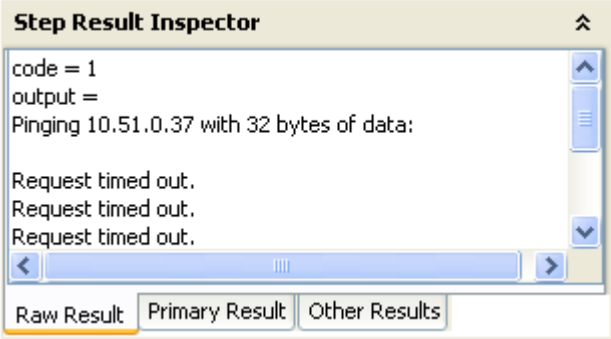


Figure 78 - Flow debugger mid-flow

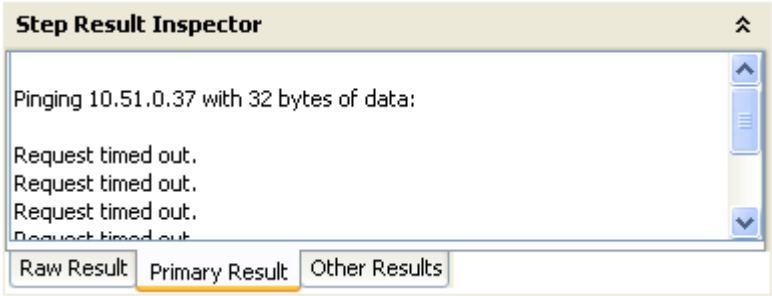
- **Run Tree** shows each step that runs, including steps in subflows of the flow. Steps that will run simultaneously in Central are run in a serial sequence in the debugger.
- **Breakpoints & Response Overrides** Breakpoints are flags that enable you to automatically pause a run at a certain step in 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 pane 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 current values of flow variables (global as well as local) for each step.

3. To see the information in each of these panes for a completed step, click the step in **Run Tree**.

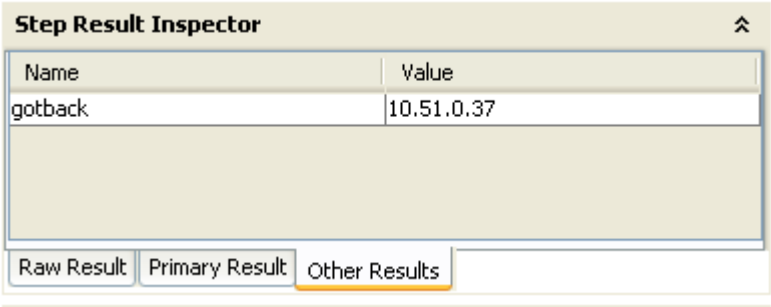
4. In the **Step Result Inspector**:
   - To see the raw result of the step, click **Raw Result**:



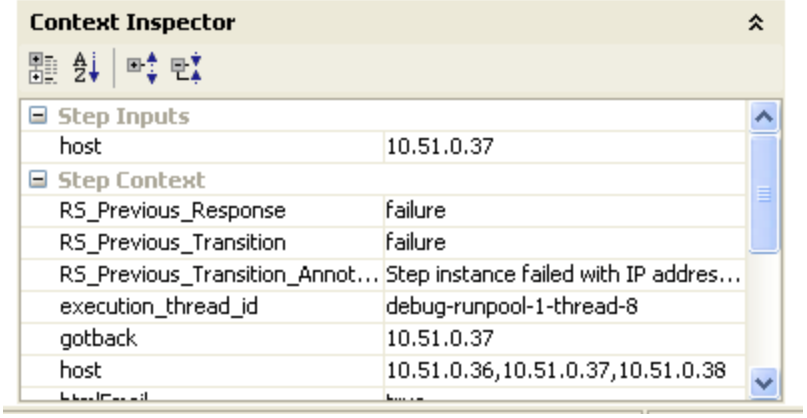
   - To see the primary result, click **Primary Result**:



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



5. 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 flow and any parent flows. (If a flow is a step in another flow, the relationship between the two flows is subflow to 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.

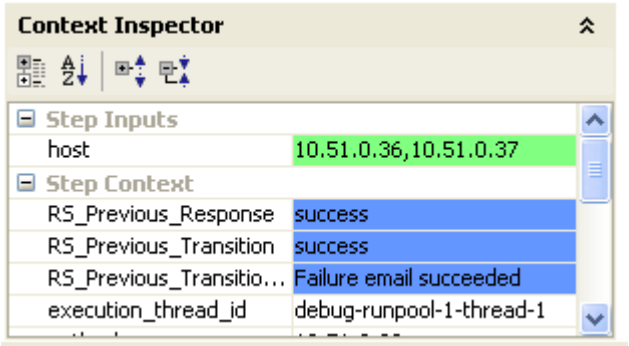


Figure 79 - Color coding of flow variable values

- 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** 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.

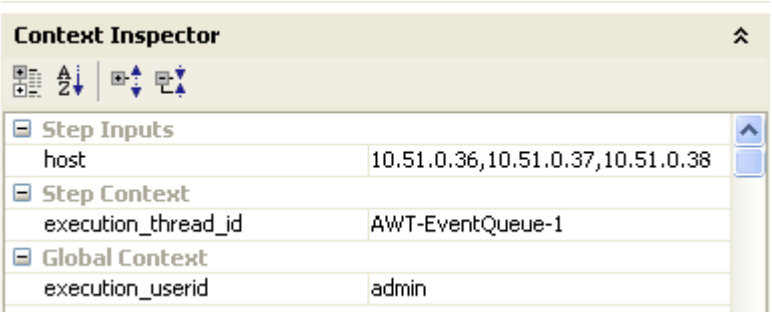


Figure 70 - The debugger Context Inspector

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** icon ( ).

Working with breakpoints

Breakpoints provide automatic pauses in the running of a flow in the Studio debugger. 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 Flow** icon ( ) in the toolbar.

2. In the debugger **Breakpoints & Response Overrides** pane, the **Breakpoints** tab shows the existing breakpoints.

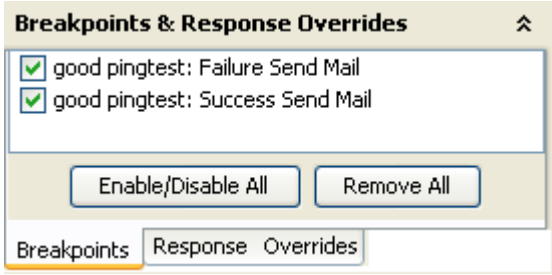


Figure 80 - Enabling and disabling breakpoints

3. Select or de-select the breakpoint's check box.

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.

   The choices are **None**, **success**, **failure**, and **Prompt**.

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 Flow** icon () in the toolbar.
2. In the debugger **Breakpoints & Response Overrides** pane, the **Response Overrides** tab shows the existing response overrides.

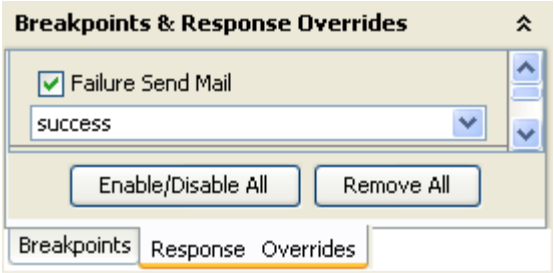


Figure 81 – Working with response overrides

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

Logging on as another user

You can log on 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 on as a different user

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

  You can then log on as a different user.

Repositories: Libraries for flows and their objects

Flows, operations, and any OO objects (such as domain terms, remote action services, scriptlets, and selection lists) 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. This means that there are two ways you can develop flows:

- Work in the public repository.

  This assumes that you have one Central server for developing flows without affecting your IT infrastructure, and another Central server in the production environment, where flows are run in the real world.

  If you work this way and there are multiple authors creating flows, you check out flows, operations, and other repository objects such as system accounts, system filters, and domain terms to work on them, then check them in to commit your changes to the public repository. The checkout/checkin requirement for working in the public directory prevents change conflicts.

  Key information: Any one user account must not connect to the public repository for the same installation of Central from two different instances of Studio. If this happens, the user's workspace in the repository can become corrupted. This can also occur when Central is clustered.

  When a flow that you have perfected on the development server is ready for the real world, you then publish the flow from the development Central server to the Central server in the production environment.

- You develop flows offline, in your local repository, then publish them to the public repository on the Central server.

  When you work offline (that is, you are not working in a public Central repository), you do not need to check out an object to work on it. Publishing to the Central repository will, however, involve checking affected objects in to the Central repository.

  When you are offline, you can also export and import selected portions of a repository.

Only the author who creates a local repository has access to it. No one can either update or publish to a local 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 box appears:

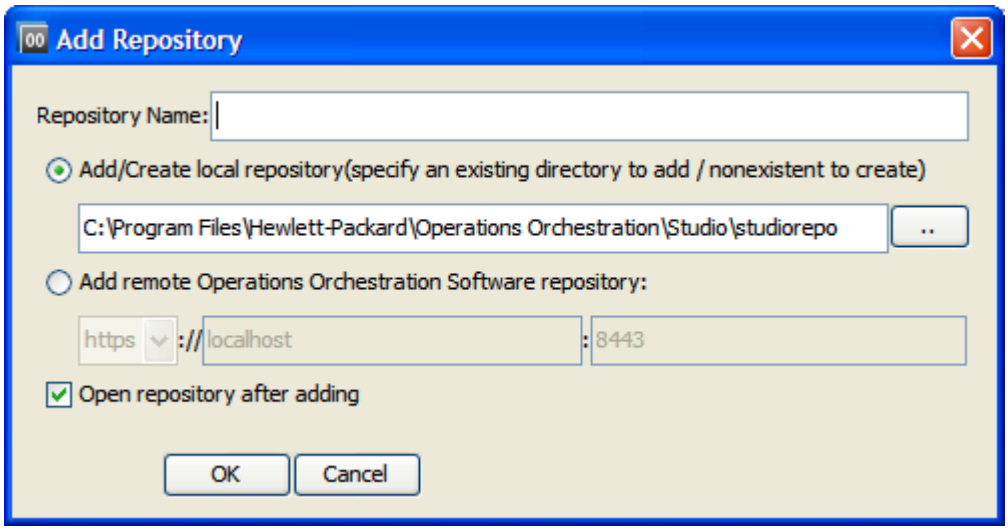


   Figure 82 - Adding a repository

2. In the **Repository Name** text box, type a name for the repository.
   The name can be anything you'd like it to be.
3. To add a local repository, select **Add/Create local repository** and specify a location for the repository folder.
   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: how to*
- *Updating from a repository: how to*
- *Rolling back a publish or update*
- *Exporting a repository*
- *Importing a repository*

To publish or update flows, operations, and other 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

Publishing and updating are how you exchange flows with a target repository that you are not working in directly—that is, a repository that is not open in your Studio. (By contrast, when your Studio is connected to a Central public repository, you are working in your workspace of the public

repository, and using checkin/checkout to make your work available to other authors working in the repository.)

If you are publishing to a public repository in which other authors are working, then when you click the **Apply** icon in the course of publishing, OO tries to check out the objects that have changes that you're publishing to the repository. If another author has checked out the objects that you're trying to publish, the publishing attempt fails. For more information on the checkin/checkout feature, see *Version control of Library objects*.

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 on 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 serves the production installation of Central.

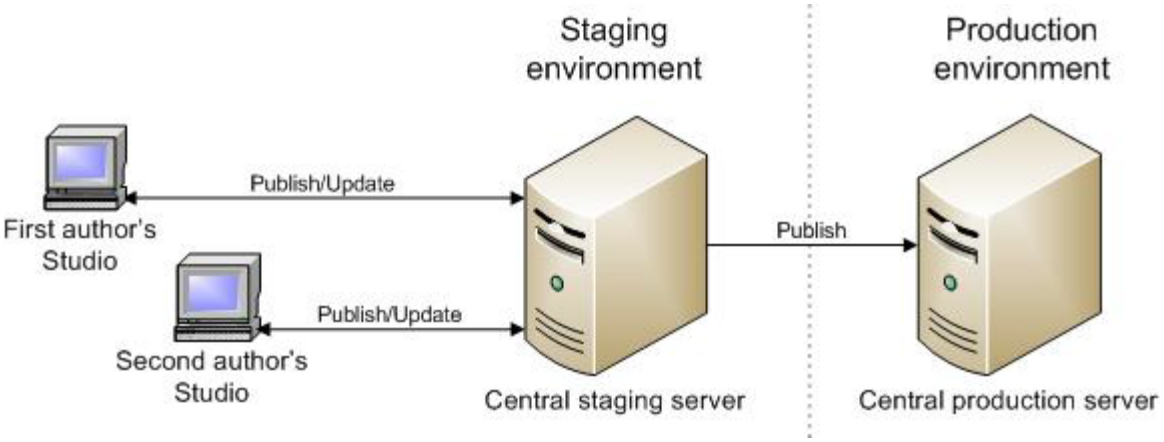


Figure 83 - Recommended arrangement for or more two authors creating flows

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 to which Studio is connected.

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 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** pane, 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*.
- If you are updating a local repository from a public repository, make sure you have checked in the flows that you want to get onto the local repository.
- When you publish, you are publishing **from the source** to the **target**; when you update, you are updating **from the target** to the **source**.
- 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. Click the **Publish / Update** tab at the bottom of the Studio window, and then click the **Pin** icon ( ) to keep the **Publish / Update** pane 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** pane's toolbar, click either the **Update Preview** icon ( ) or the **Publish & Update Preview** icon ( ).

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

3. To completely expand each folder in the pane, click the plus icon ( ).

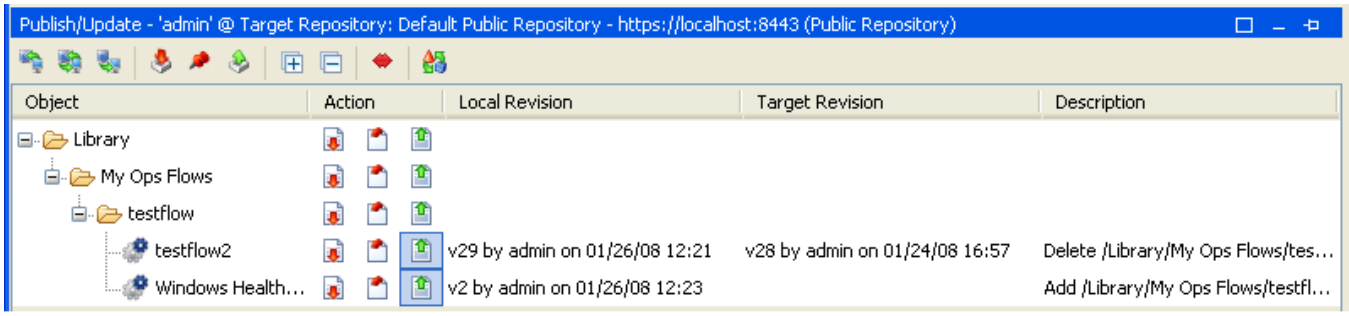


Figure 84 - Publish / Update pane

Note the following information in the pane:

- 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 are different but 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.

4. To look at either of the versions by itself, right-click in the row for the object (Testflow1, in our example), 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** pane.

   OR

   To look at both versions side by side, right-click in the row for the object (Testflow1), and then click **Compare**.

5. 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 in local**
   - To publish the object to the target, **Modify in target**
   - To take no action, **No Change**

6. To apply the changes that you've specified, click the **Apply** icon ( ).
7. In the **Enter Publish Comment** dialog box that opens, enter a comment about the published changes and then click **OK**.
8. When a message appears saying that all changes have been successfully applied, click **OK**.
9. 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 to which Studio is connected. 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 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*.

Tip: Before you update from a public repository, make sure that the flows that you're interested in have been checked in to the public repository.

To update from a repository

1. 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*.

2. From the **Repository** menu, select **Update Source from 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** pane open.

   The **Publish / Update** pane 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.

4. 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, which was published to the public repository from another local repository.

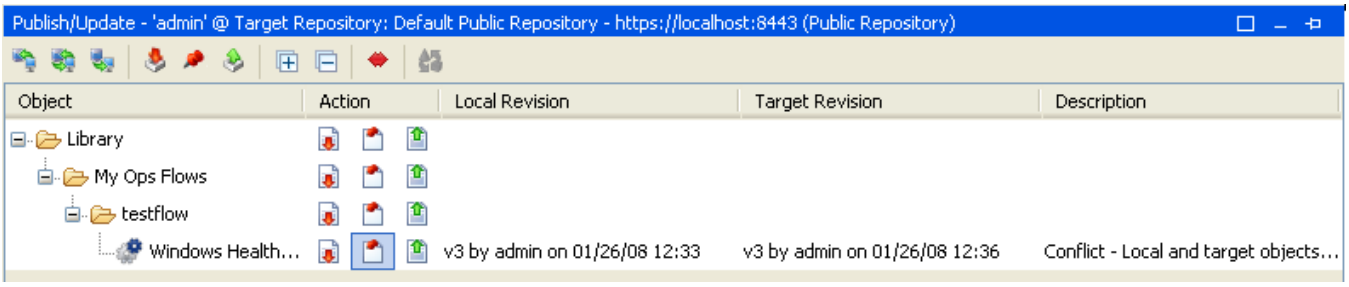


Figure 85 – Publish\Update pane with conflict

5. 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** pane.

   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).

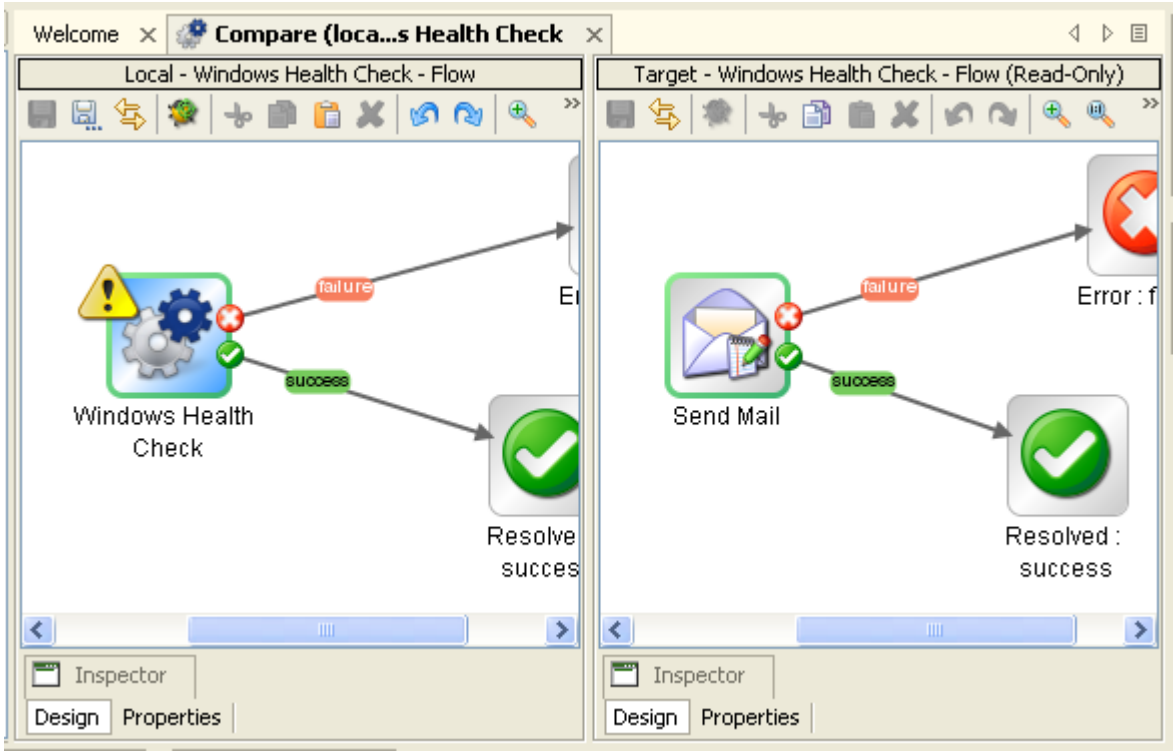


Figure 86 - Comparing two versions of a flow

You can make and save changes in the local repository's flow diagram. (Note that the target [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 in local**
- To publish the object to the target, **Modify 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

- Import the .jar, and then save your work.
  For more information, see *Importing a repository*.

Exporting a repository

To make flows and OO objects available to an 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.

Exporting repositories to a safe backup location is also how you create backups of your flows and their dependent objects.

To export a repository

1. In the Library, right-click the folder that contains the OO objects you want to export, point to **Repository**, and then click **Export as New Repository**.
2. In the **Select Repository Directory** dialog, navigate to the location where you want a new folder to reside.
3. In the **File name** box, type the folder name.
   The folder should not be an existing folder.
4. Click **Save**.
   The **Export Options** dialog appears.

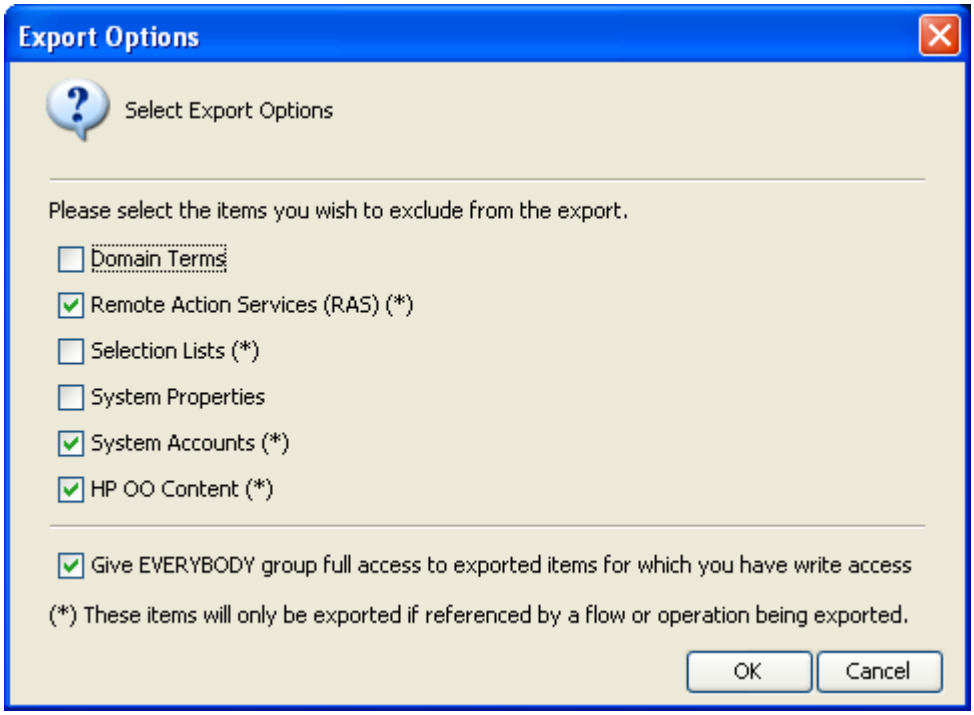


Figure 87 - Options for export

5. 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 a flow that uses an operation that runs a Windows or Linux shell 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. To learn about using the Shell Wizard, see *Shell Wizard: creating a flow that uses a shell operation*.

   If you export a subfolder of the Library and are going to point the Shell Wizard or Web Service Wizard (see *Creating operations from Web services* for more information about the 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 operations 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 (RAS)

     A RAS is a reference to a 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 a 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, they are those that you do **not** want to export.

6. To give everyone Read, Write, Execute, and Link To 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**.
7. 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 a repository folder, a Java archive (.jar) file, or a .zip file, if the repository has been stored in one of those formats.

When you import a flow, you also import the flow's configurable 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. 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 a .jar/.zip that contains a repository)** dialog box that opens, navigate to the directory that contains the repository (which may be an Accelerator Pack), .jar file, or .zip 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.

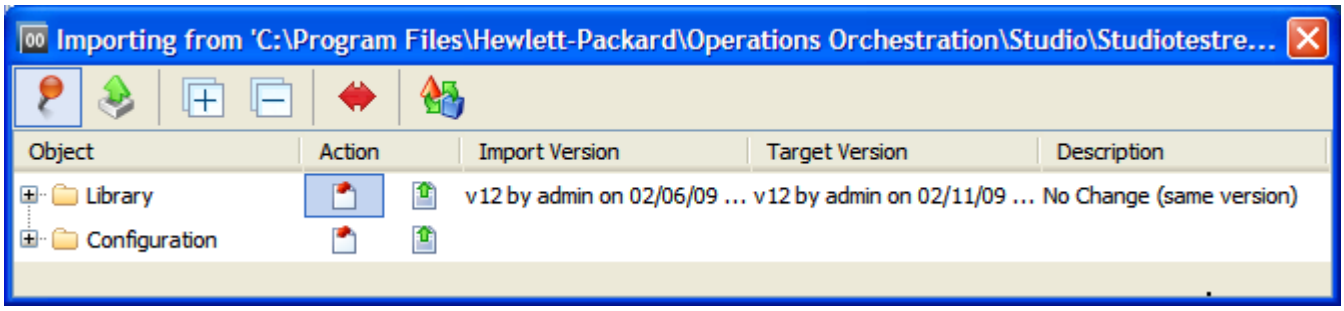


Figure 88 – Importing from dialog box

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 OO objects.

When you expand a folder, the dialog shows this information for each object to be imported within that folder. Note that even if the action shown for a folder is **No Action** (), there may be changes within that folder for you to examine.

5. To completely expand each folder in the dialog box, click the plus button () in the toolbar.

   When you expand the folders, the dialog shows similar information for each OO object.

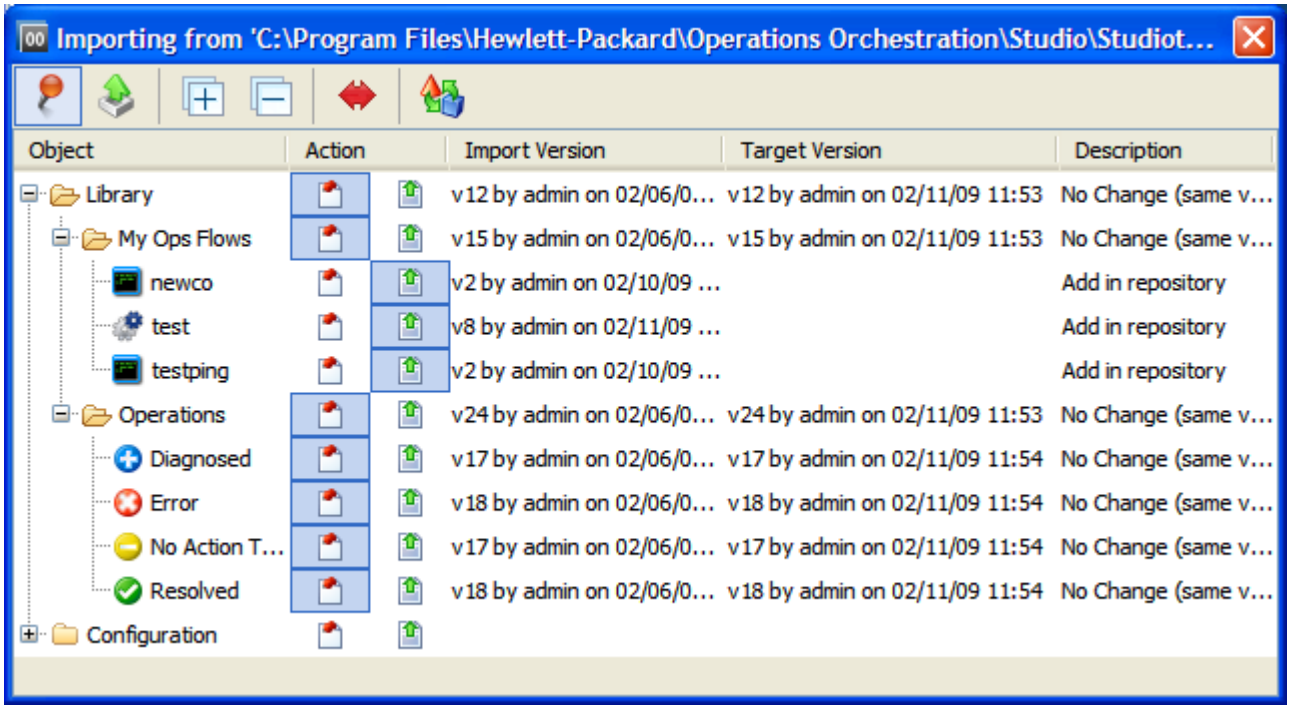


Figure 89 - Selecting objects to import

Note particularly 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 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** button ( ), 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 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** button ( ).

When repositories are exported, they reside outside of Studio, so to import them you do not need to set a target repository.

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 *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 Flows and Operations**.

  A list of any problems, with their locations and descriptions, 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 an encrypted repository in any way, 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

When you encrypt a repository, you are exporting it and encrypting the export of the 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:

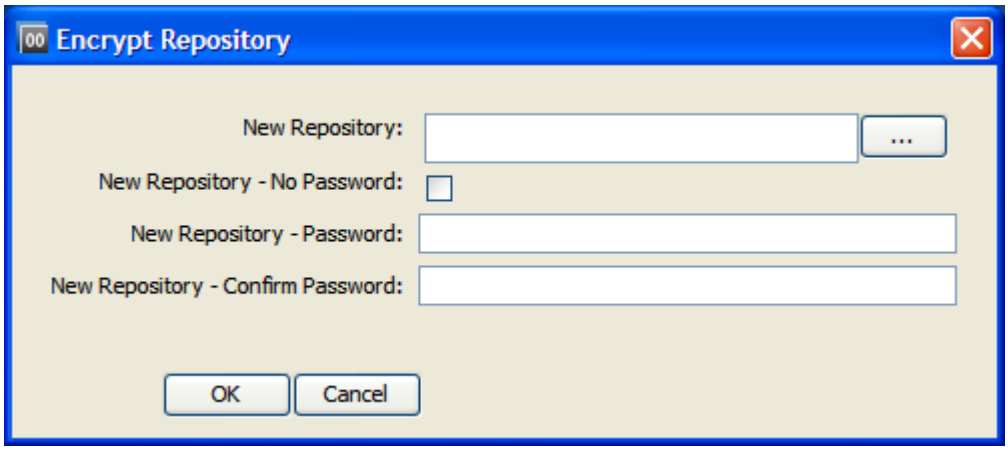


Figure 80 - Encrypting a repository

2. 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.

3. Type and confirm a password for the encrypted copy, and then click **OK**.

Opening an encrypted repository

To open an encrypted repository

1. To open the encrypted copy of a repository, from 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.

2. When prompted for the password, supply the password that you specified when you encrypted the repository.

Decrypting a repository

To decrypt a repository

1. Open the encrypted repository.
2. 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

1. Add (if necessary) and open the encrypted repository.
2. From the **Repository** menu, click **Re-encrypt Repository**.

   The **Re-encrypt Repository** dialog box appears.

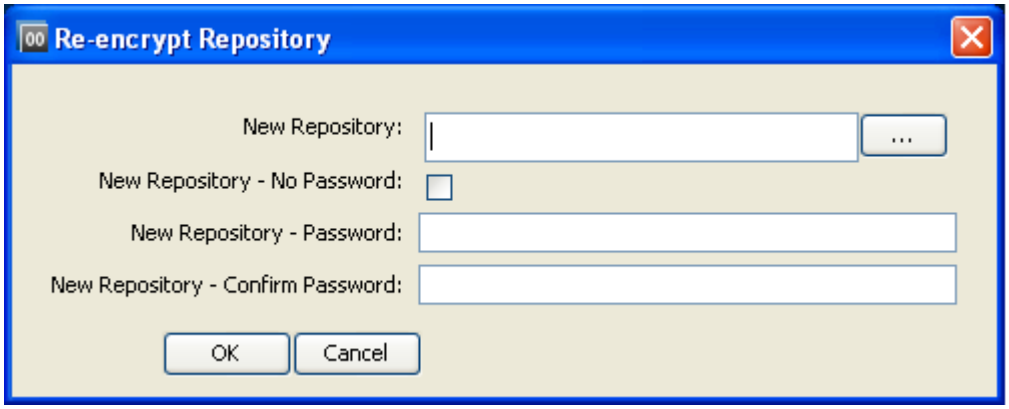


Figure 90 - Re-encrypting a repository

3. 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. From the **Repository** menu, choose **Import Repository**.
2. In the **Select file for backup** dialog box, navigate to the folder that contains the .jar file that defines the repository.

   The folder that contains the .jar file contains two subfolders, **backups** and **data**.
3. Click **Open**.

   After checking the Central public repository for version conflicts with the objects that you want to import, OO imports the repository.
4. If there are conflicts, resolve them as you do when you import repositories.

   For more information on importing repositories, see *Importing a repository*.

Creating operations from Web services

The Web Services Wizard creates 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) reference that points to the Web service.

To create operations for a Web service with the Web Services Wizard

1. In the OO home directory, in the **\Studio\tools** folder, double-click **wswizard.exe**.

   The Web Services Wizard starts.

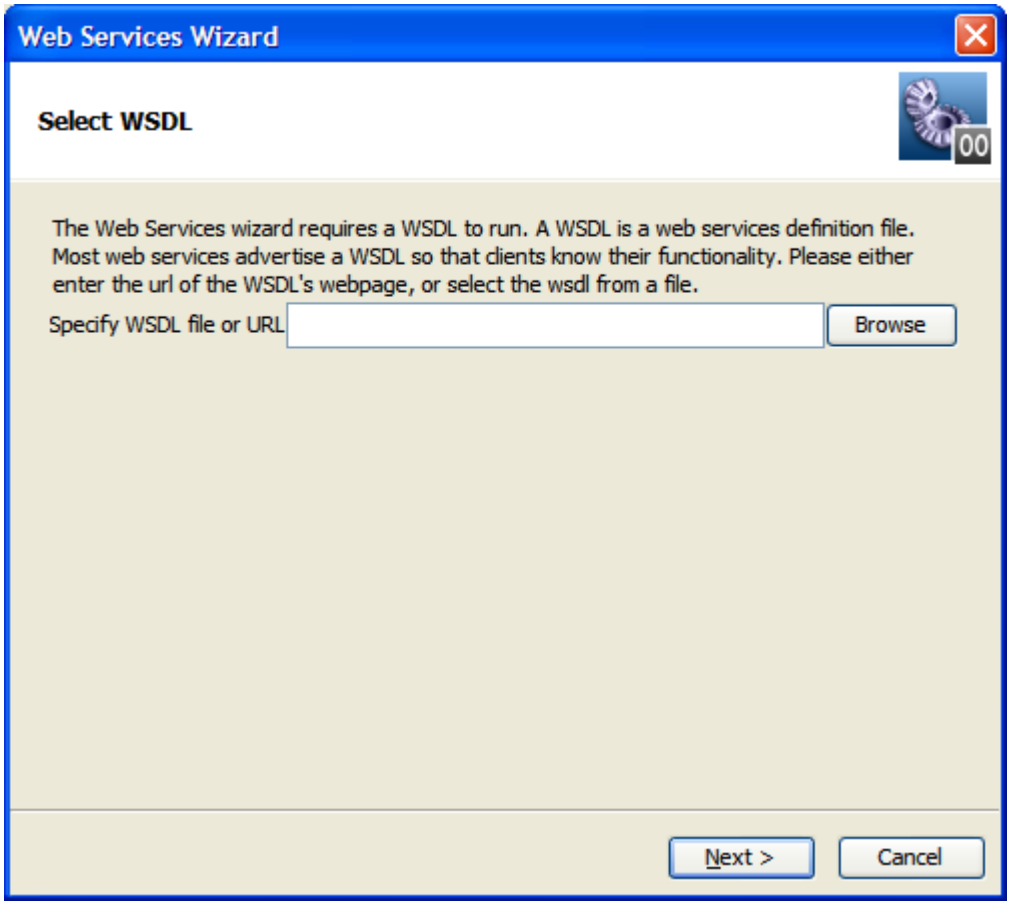


Figure 91 – Web Services Wizard

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 wizard loads the WSDLs.

   The **Choose Operations to generate** pane appears.

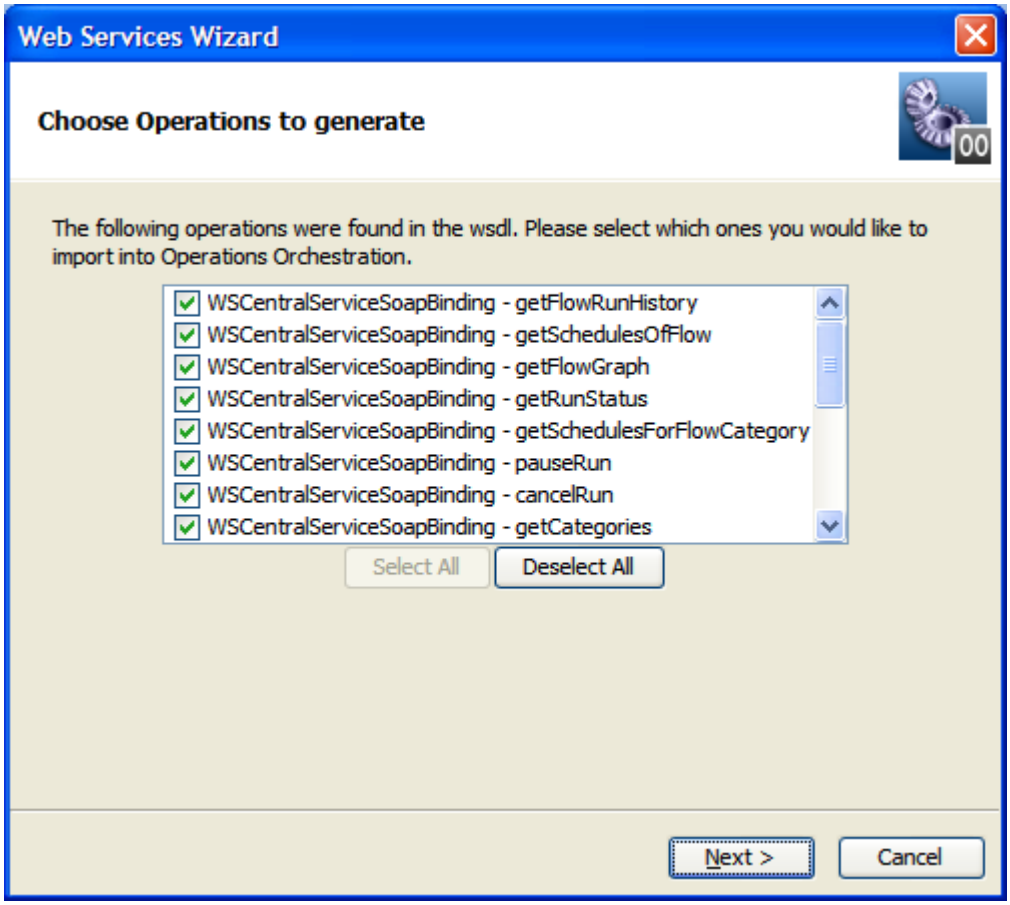


Figure 92 - Choosing operations to create

The page lists all the WSDL's methods, which can be turned into operations.

4. Deselect (remove the check from the check box 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 on machines, using commands, or against APIs that would be otherwise inaccessible to OO flows.

For example, you could use a RAS to accomplish the following:

- Integrating with the Exchange API.
- Restarting a server in a different domain from the one in which Central is installed.
- Carrying out an operation on a Windows server when Central was installed on a Linux server.

Depending on what you need a RAS for, you can use the RAS that is by default included when you install Central, or you may need to install a standalone RAS remotely.

If the RAS provides Central with access to a different domain, it only needs to be installed in that domain.

If the RAS provides Central with access to a different API, it does so either by accessing a Web service that provides access to the API or by having operations that use the API added to the RAS.

For the RAS to give Central access to commands and machines that it would not otherwise have access to the following are required:

- The RAS must be provided with or have access to the code that it in turn enables OO flows to execute.

  A RAS can obtain code or access to it in one of two ways:

  - If the RAS interacts with a Web service: The RAS already has the code (an IAction) required to interact with any Web service.

    Although the RAS can interact with a Web service, Central needs the commands and inputs that the Web service requires so that the flow author can create flows that use them. Therefore, OO provides authors with the Web Service wizard for creating operations from the target Web service. For information on using the Web Service wizard, see *Creating operations from Web services*.

  - If the RAS interacts with an API without the mediation of a Web service, the operation author must create an IAction that contains the code required for interaction with the commands of the API.

    You use either Java or a programming language that is supported by .NET to create an IAction. This level of authoring requires the ability to program with .NET or Java as well as Windows Scripting Host or Perl scripting.

    After creating the IAction, the developer compiles it into a Java archive (.jar) file or dynamic-link library (.dll) and adds it to the RAS's Repository folder. Detailed information on creating IActions and adding them to a RAS is in the *HP OO Software Development Kit Guide* (SDKGuide.pdf).

    As with a Web service, Central needs the commands and inputs in the IAction(s) in order for flows to be created that use the IAction. To provide Central with the needed commands, OO enables authors to import operations from a RAS. For information on importing operations from a RAS, see *Creating operations from a RAS*.

- Central must be directed to the RAS.

  You direct Central to the RAS by creating a *RAS reference* in Studio, then dragging the RAS reference onto a RAS-enabled operation. The RAS reference has a name and a URL that accesses the RAS. The default RAS reference in Studio is **RAS_Operator_Path**.

  Note: Not all operations can have a RAS reference added to them. RAS-enabled operations, which have a field on their **Inputs** tab for adding a RAS reference, include the following:

- Operations that are created from a RAS or a Web service.
- A new operation that you create in Studio and that is of the RAS operation type.
- The operations in default OO content that must run on a RAS.

  Many operations in default OO content have RAS-enabled counterparts, such as those in the **Operations\Remote Command Execution** folder in the Studio **Repository** pane.

Topics in this section

How Central runs RAS-dependent operations

Checking the availability of a RAS

Adding an existing RAS

Adding a RAS reference

Reconfiguring an existing RAS reference

Changing RAS references

Creating operations that access Web services

Troubleshooting the running of RAS-dependent operations

Creating operations from RASes

Removing a RAS reference

Importing RAS content

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 or .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 reference 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.

For information on specifying values for the **RAS Operation fields** on a RAS operation's **Properties** sheet, see *RAS operation: IAction programming for a RAS*.

Checking the availability of a RAS

For an overview of Remote Action Services, see *Operating outside Central with Remote Action Services*.

To check the availability of an existing RAS

1. In the **Library**, right-click the **Configuration\Remote Action Services** folder and click **Properties**.
2. In the **Remote Action Services** sheet, highlight the RAS whose availability you want to check, and then click **Check Availability**.

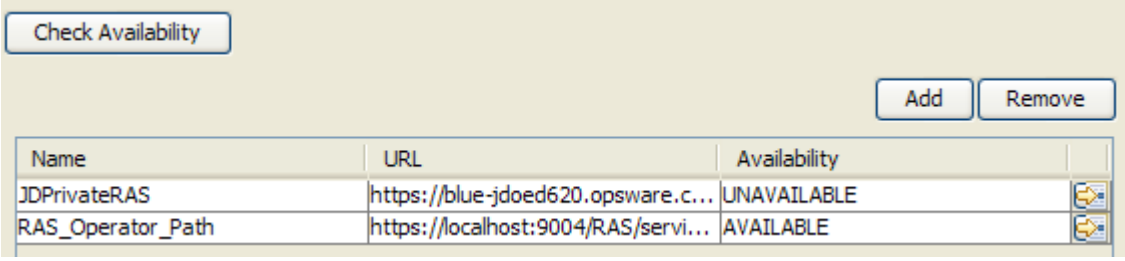


Figure 84- Remote Action Services sheet

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** sheet, 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 OO.

For an overview of Remote Action Services, see *Operating outside Central with Remote Action Services*.

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 in the **Configuration\Remote Action Services** sheet.

Adding a RAS reference

For an overview of Remote Action Services, see *Operating outside Central with Remote Action Services*.

To add a new RAS reference

1. In the **Library**, right-click the **Configuration\Remote Action Services** folder and click **Properties**.

   When the **Remote Action Services** sheet opens in the **Authoring** pane, click the upper **Add** button.

   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**.

   The RAS editor for the new RAS opens.

3. In the **URL** text box, 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.

4. 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:

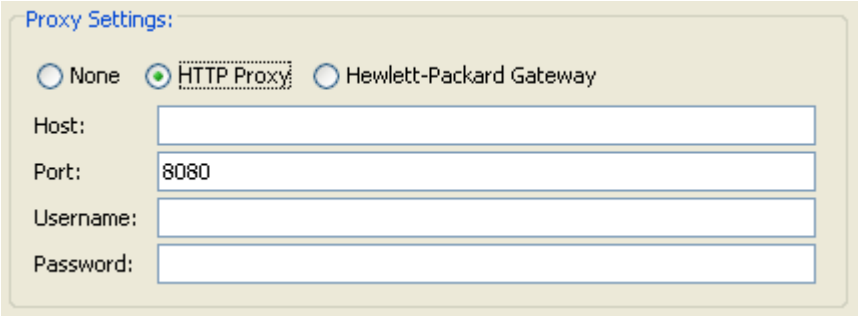


Figure 93 - Settings for HTTP Proxy

   - 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:

Figure 94 - 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 of which the gateway is a member.

5. Save the new RAS, and then click **Remote Action Services** beside **Configuration**.

6. In the **Remote Action Services** sheet, 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.

Reconfiguring an existing RAS reference

For an overview of Remote Action Services, see *Operating outside Central with Remote Action Services*.

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 RAS editor for the selected RAS 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:

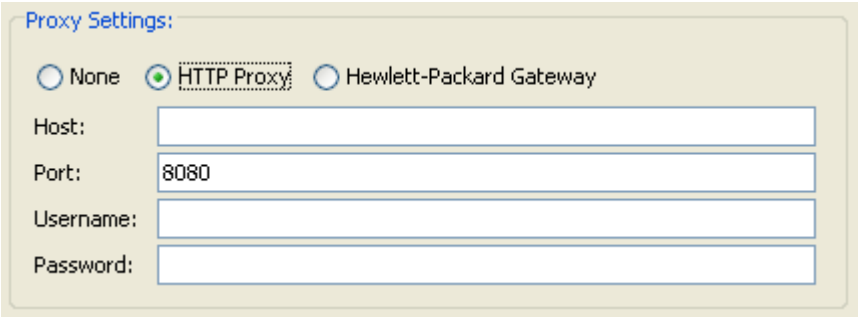
     

     Figure 95 - Settings for HTTP Proxy

     - 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:

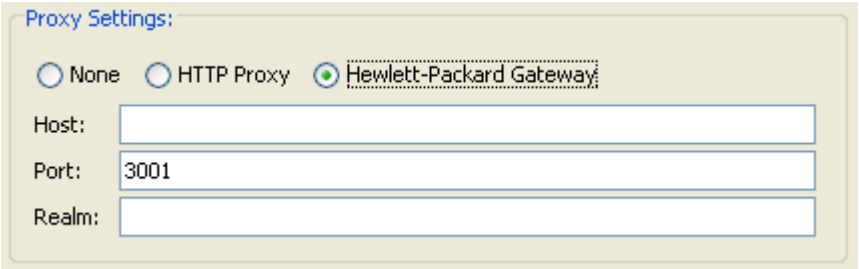


Figure 96 - HP Gateway proxy settings

- 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. Save the RAS, and then click **Remote Action Services** beside **Configuration**.

5. In the **Remote Action Services** sheet, 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.

Changing RAS references

Operations that are built from IActions have RAS references, because creating the RAS reference is part of creating the IAction. The part of an operation's **Properties** sheet that specifies the RAS reference looks like the following:

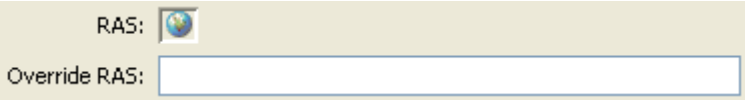


Figure 97 - RAS reference fields

For more information, see *RAS operation: IAction programming for a RAS*.

All the IAction-based operations provided in the OO content folders (the **Integrations**, **Operations**, and **Utility Operations** folders) include, in the **Override RAS** field, a reference to the **overrideNRAS** or **overrideJRAS** flow variable. It is strongly recommended that when you create a RAS operation or create an operation from an IAction, you include the flow variable reference **${overrideJRAS}** (for Java IActions) or **${overrideNRAS}** (for .NET IActions) in the **Override RAS** field.

- When you create a RAS operation, you can change the RAS reference by dragging a new RAS reference from the **Configuration\Remote Action Services** folder to the **Globe** icon ( ) in the **RAS** field of the **Inputs** tab on the operation's **Properties** sheet. You can only change the RAS reference this way on the operation itself.
- When you are working in flows with existing RAS operations, you can override the RAS reference. How you do so depends on the scope for which you want to change it:
  - If all the RAS operations in OO of a given type (that is, that use an NRAS override or a JRAS override) use the same RAS, you can create a system property that references that RAS.

Key information: Note, however, that a system property is a flow variable that is added to the global context, so it is not possible to change the value for any input that has its value assigned from that system property. Setting **${overrideJRAS}** and/or **${overrideNRAS}** as a system property means that the RAS reference assigned to either system property will be used in any run of any flow that uses that global flow variable, with no chance of changing that assignment. System properties have the potential for breaking flows or creating unexpected results in flow runs, without any way of correcting the situation at run time.

For information on creating a system property, see *Creating system properties*.

- When in a single flow run you want to change the RAS reference for all RAS operations of a given type, set the RAS override as a flow variable, assigned from either a flow or a step input.
- When in a single flow run you want to change the RAS reference for all RAS operations of a given type (that is, that use an NRAS override or a JRAS override) after a certain point in the run, set the RAS override as a flow variable whose value is:
  - The name of a RAS reference that you have created in the **Configuration\Remote Action Services** folder
  - Assigned as a step input.
- When in a single flow run you want to change the RAS reference for a RAS operation that is the basis only for a given step, assign the RAS override to a step input, but make sure that it is not assigned to a flow variable.

To change the RAS reference on a RAS operation

- On the **Inputs** tab of the RAS operation's **Properties** sheet, drag a RAS reference from the **Configuration\Remote Action Services** folder to the **Globe** icon ( ) beside the **RAS** field label.

To override the default RAS reference with another RAS reference

1. In a flow, create either a flow input or, on the step created from the operation containing the RAS reference, a step input.
2. Specify that the input assign its value to the flow variable that is referenced in the relevant operation's **Properties** sheet **Inputs** tab, in the **Override RAS** field.

   In operations provided in OO content, the flow variable referenced in the **Override RAS** field is **overrideJRAS** or **overrideNRAS**.
3. Make sure that the **Override RAS** field references the flow variable that gets its value from the flow input you created in the preceding step.

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 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 OO for extended functionality. For information on doing so, see *Administering Operations Orchestration Software* (AdminGuide.pdf).

For information on using the Web Service 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 RAS reference 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 or that is installed in the domain of the target machine).

  For more information on adding, configuring, and replacing RAS references for RAS operations, see the relevant topics in this section, *Operating outside Central with Remote Action Services*.
- 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 reference in the flow variable, and create a flow whose logic determines which RAS reference that is.

Suppose you want to restart either ServerA in DomainA or ServerB in DomainB. 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 step with an input that prompts the user for the domain\server name, including the domain name.

   Tip: The **Display** operation (in the **Utility Operations** folder) is handy when you don't want a step to do anything except filter some data that is provided by a user prompt.

2. In the Display step, create a step result, providing its data from the input that you created.
3. For the result, create a filter that extracts the domain name, which you then store in a flow variable named, say, domainName.

   Note: You will also need to assign the server name to a flow variable for later use, but our focus here is selecting the correct RAS service based on which domain the user wants the flow to run in.

4. Create a structure that maps the value of domainName to the name of the RAS reference that runs the flow through the RAS service for that domain and stores the name of the RAS reference in a flow variable, say, rasName.

   The easiest way to do this is to use the List Map subflow, which takes two lists as inputs. In one list you supply the possible domain names, and in the other list, in the same order as the first,

you supply the names of the RAS references that should be chosen according to the domain name that the user specified.

5. Provide the flow variable **rasname** (and the flow variable containing the server name) as an input to the RAS operation that restarts the server.

Creating operations from RASes

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 OO content that uses the Remote Action Service (pointed to by the RAS reference named "RAS_Operator_Path," which is provided by default in OO).

To import custom Web service content, you create a RAS reference and assign it the URL of a RAS that contains the Web service content you want. Identification of the RAS primarily by name rather than by URL means that changing the URL in 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 **Properties**.

   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 reference, and then click the lower **Add** button.
3. In the text box that appears, type a name for the RAS.

   The RAS editor for the new RAS opens.

4. Click **Remote Action Services** beside **Configuration**.

5. In the **Remote Action Services** sheet, 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.

Topics in this section

Creating operations from a RAS

Adding an operation that is not in the default RAS

Creating operations from a RAS

For an overview of Remote Action Services, see *Operating outside Central with Remote Action Services*.

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 **RAS Import** 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.

Adding an operation that is not in the default RAS

You can create the basis for RAS-dependent operations by adding IActions contained in a Java archive (.jar) or dynamic-link library (.dll) file to the RAS. Then, to create OO operations that you can use in OO flows, you create operations from the RAS.

For an overview of Remote Action Services and their relationship with actions that are not included in or accessible from the Central repository, see *Operating outside Central with Remote Action Services*.

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 OO home directory:
   - A .dll (added to RAS), whether it is your own or comes from a third party to the following location, or 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 a 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 RASes*.

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.

To learn how to create IActions, see the material on authoring IActions in the *SDK Guide* (SDKGuide.pdf).

Removing a RAS reference

For an overview of Remote Action Services, see *Operating outside Central with Remote Action Services*.

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

For an overview of Remote Action Services, see *Operating outside Central with Remote Action Services*.

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, select the folder into which you want to import it and then, from the **File** menu, choose **Create Operations from RAS** 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 default Library content.

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
- Constant

  If the constant value contains a flow variable (such as ${domain}), the evaluator is applied to the constant value with the variable's value (rather than the variable reference) included in the constant 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

  The default data evaluator for phone numbers supports only the North American telephone-number format (1-nnn-nnn-nnnn) for calling from within North America. Validating other regional phone-number formats requires different evaluators. For information on creating a new system evaluator, see *Creating an evaluator*.

- UUID

Evaluators can use standard evaluators such as =, !=, Begins with, Contains, Match All Words, and Match At Least One Word, 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

Note: The default **Telephone Number** system evaluator validates against the North American telephone-number format (1-nnn-nnn-nnnn) for calling from within North America. To create an evaluator that validates telephone numbers against regional telephone-number formats, you will probably want to use a regular expression. For more on regular expressions, see *Working with regular expressions*.

To create an evaluator

1. In the Library, right-click the **Configuration\System Evaluators** folder and click **New**.

   OR

   Click **Properties**, 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**.
   - If you clicked **New**, the system evaluator editor for the new evaluator opens.
   - If you clicked **Properties**, the new evaluator appears in the list of evaluators in the **System Evaluators** sheet. Click the right-pointing arrow ( ) at the right end of the row to open the system evaluator editor.

   The appearance of the system evaluator editor depends on the kind of evaluator you selected.

4. Type the text, string, expression value, or scriptlet that you want to use the evaluator to test the input.
5. To clear the contents of the **Test Filter Input** box, click **Clear**.
6. Test the evaluator by either:
   - Typing a sample of the text you want to validate in the **Test Filter Input** box.
   - 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**.
7. 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\System 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 the up or down arrow (▲ ▼) beside **Configuration**.

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. A 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** flow 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 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 Under** 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—such as 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, or the action that the flow took. 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 (such as servers and applications) that have been affected.

Some domain terms have values by default; others obtain their values from the flow's inputs; yet others have the values that you specify for them.

The domain terms that you can use for reporting data to Dashboard charts include the following. (For information on how to add domain terms or add specified values to them, see *Adding domain terms or domain term values*.)

- **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," or "gateway.". 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 or domain term values

The domain terms are listed in Studio's **Repository** panel, in **the Configuration\Domain Terms** folder. You create values for a domain term or otherwise change the domain term in its editor.

To add a domain-term

1. In the **Repository** pane, expand the **Configuration** folder, right-click the **Domain Terms** folder, and click **New**.
2. Type the new domain-term's name in the **Name** text box.
3. Type its description in the **Description** text box.
4. Save your work.
5. If you are working in the shared Central repository, check the new domain term in.

To add or remove a domain term value

1. In the **Repository** pane, expand the **Configuration** and **Domain Terms** folders.
2. If you are working in the shared Central repository and have not already checked out the domain term for editing, select the domain term, and click the Checkout icon ( ).
   For more information, see *Checking out and checking in a Library object*.
3. Double-click the domain term that you want to add the value to.
4. In the domain term editor that appears in the **Authoring** pane, click **Add**, then type the value and its description in the **Name** and **Description** columns, respectively.
5. To remove a domain term value, highlight the value and click Remove.

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.**

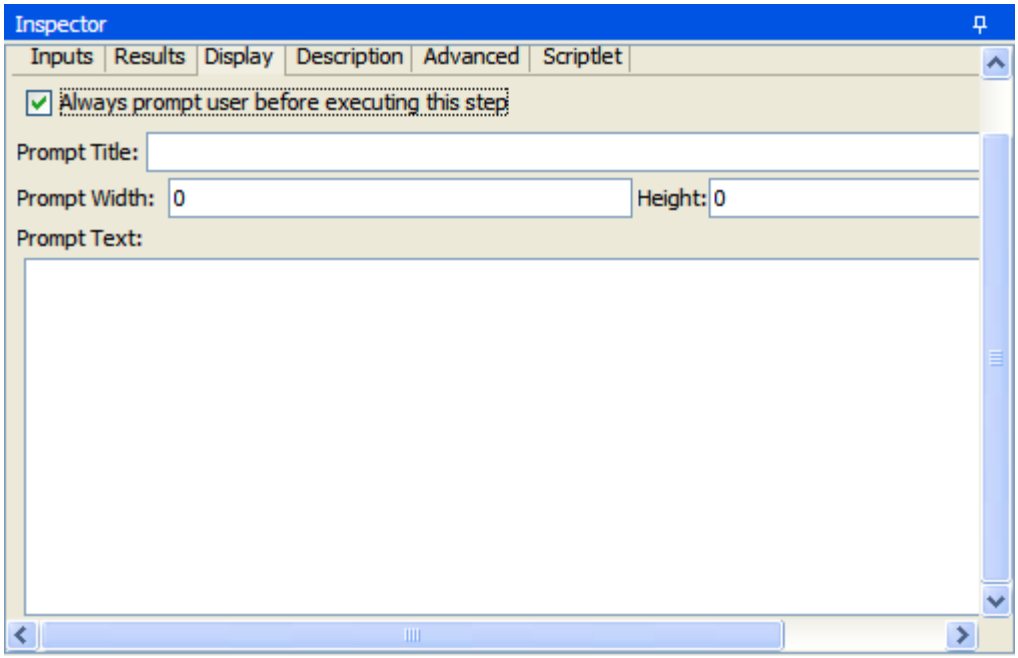


Figure 90 - Specifying a display for users

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 (using CTRL+click) categories, and then click **OK**.

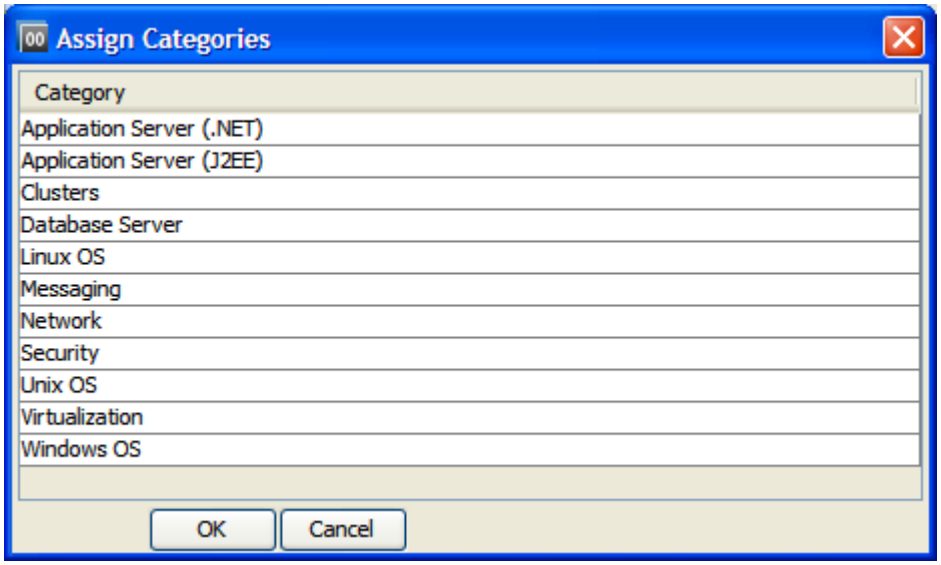


Figure 98 - Assigning a flow to a category

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 provided by default, many of them provided specifically 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 **Service Status** 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 **Repository** 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 99 - 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 selection list's tab.

To edit a selection list

1. In the **Repository** pane, right-click the **Configuration\Selection Lists** folder and click **Properties**.

   The list of selection lists appears.

   return here

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 **Selection Lists** tab.

To remove a selection list

1. In the **Repository** pane, click the plus sign to the **Configuration\Selection Lists** folder.
2. Right-click the selection list that you want to remove, then click **Delete.**
3. In the message box that appears, 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 **Repository** 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.

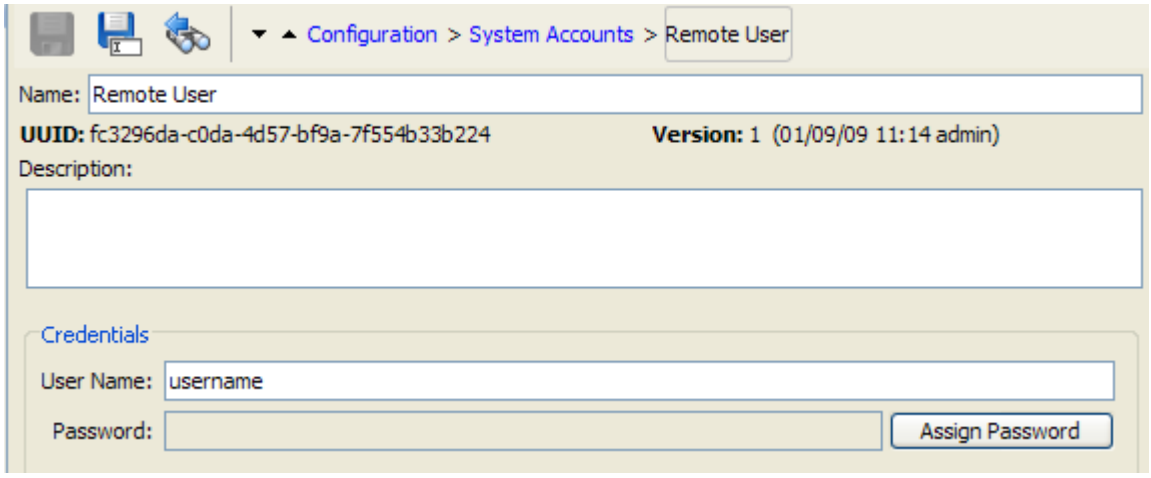


Figure 100 - System account editor

3. Give the system account a description, if you wish.
4. Type the user name of the account that the system account represents, using the following syntax:
   `<domain>\<username>`
5. Click **Assign Password** and in the box that appears, type the password, then type it again when prompted.
6. Click the **Save** icon and then, to close the system account editor, click the **X** that is on the selection list's tab.

Editing a system account

To edit a system account

1. In the **Repository** pane, open the **Configuration\System Accounts** folder, highlight the system account that you want to edit, and click the right-pointing arrow ( ) at the right end of the account's row.
2. Make any changes necessary and save your work.

Deleting a system account

To delete a system account

1. In the **Repository** 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 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, EVERYBODY, PROMOTER, Service Desk, 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 To** 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 a 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. However, to execute a flow a user or author also needs **Read** and **Write** permissions for the objects that the flow refers to.

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 To** 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 tables relates the Library objects and the actions you can perform on them to the permissions required to do so.

HP OO objects and the permissions needed to work with them

Object	Action	Necessary permission(s)
Folder		
	View contents	Read
	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	Read, 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	Read, Link To
	Use at runtime	Read, 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.

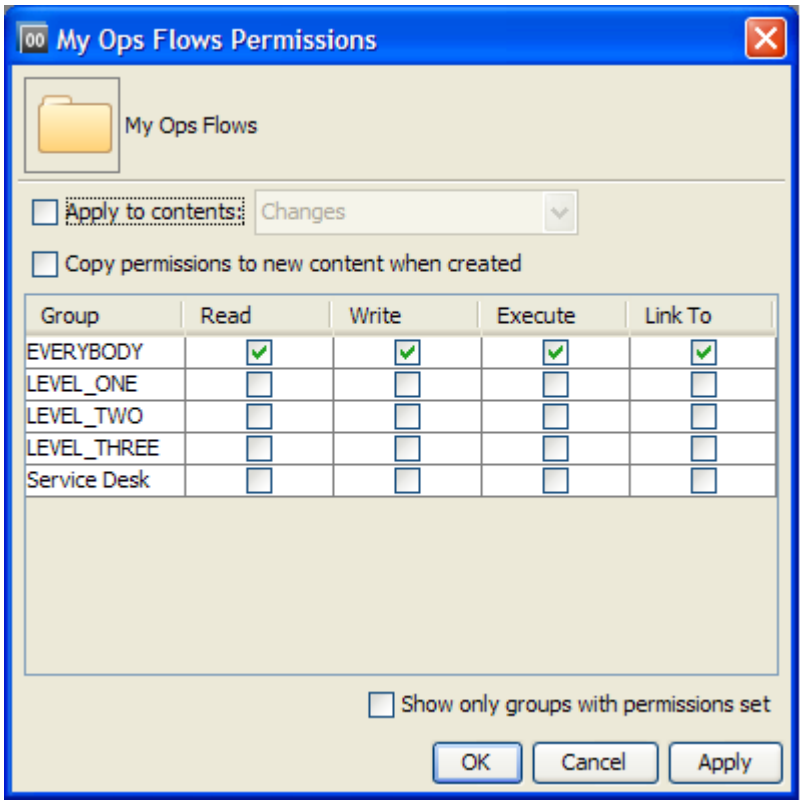


   Figure 101 – Setting permissions

   Tip: You can hide the groups for which there are no permissions set by selecting the **Show only groups with permissions set** check 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 check box, 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**.

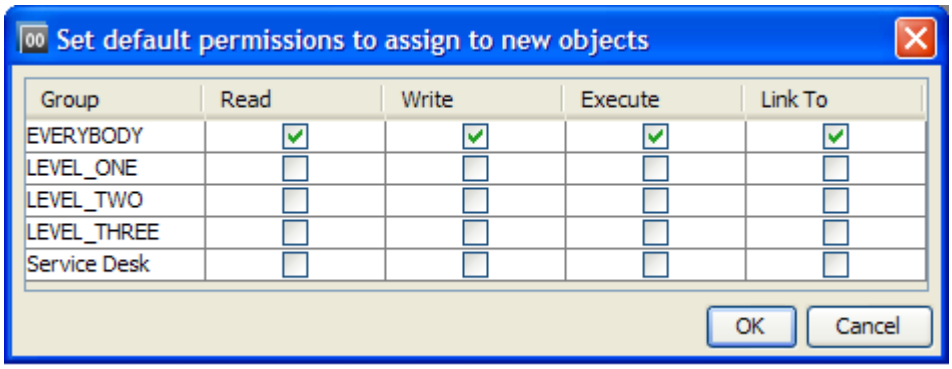


Figure 102 - Setting default permissions for new objects

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* and *outputs* for steps and operations.
- Referencing a Remote Action Service (RAS) where the operation will run (if it runs outside Central—that is, if the operation's commands are not defined in the Central repository or the operation runs on machines that are remote from the Central server).
- Writing a *scriptlet*.
- Specifying *responses* that determine transitions and mapping results to the responses.
- Assigning the operation's *access permissions* to 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.
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**.

If the new operation is invalid or incomplete, its name is displayed in the **Repository** pane in red type. Moving the cursor over the name 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 cmd, 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) operation

Use for executing an existing command, script, batch, or executable file that, outside of a flow, you would run from the command line. Programming that you carry out with cmd operations

sometimes can also be carried out by creating an IAction or using a scriptlet. However, using scriptlets is not recommended.

A cmd operation can run one command (including script, batch, or executable). To run a series of commands, you need to use a shell operation. For information on shell operations, see *Shell operation*.

- For more information on using a scriptlet, see *Scriptlet operation*.
- For information on using IAction programming, see *Creating IActions for operations* and the OO *SDK Guide* (SDKGuide.pdf).

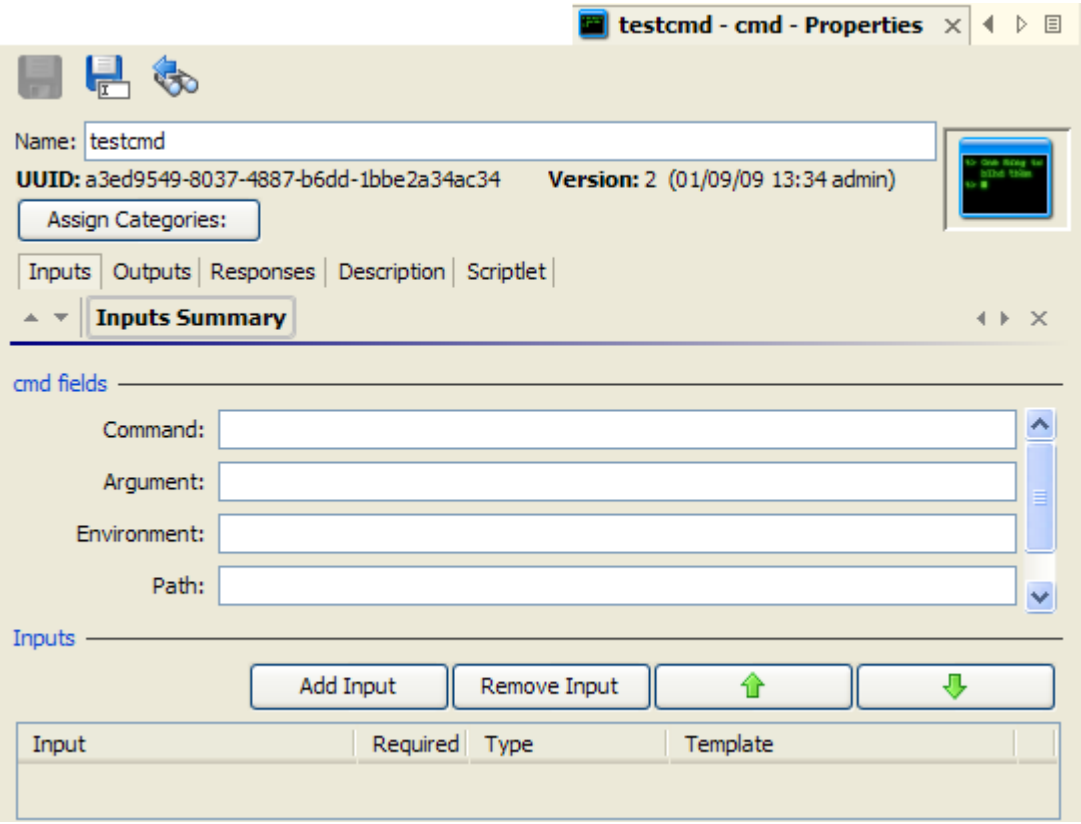


Figure 103 – Cmd operation properties

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 **Result Fields**.

- Code
- Output String
- Error String
- FailureMessage
- TimedOut

Flow Run Summary Report operations

Produces a summary report of the steps executed up to but not including the step that contains this command.

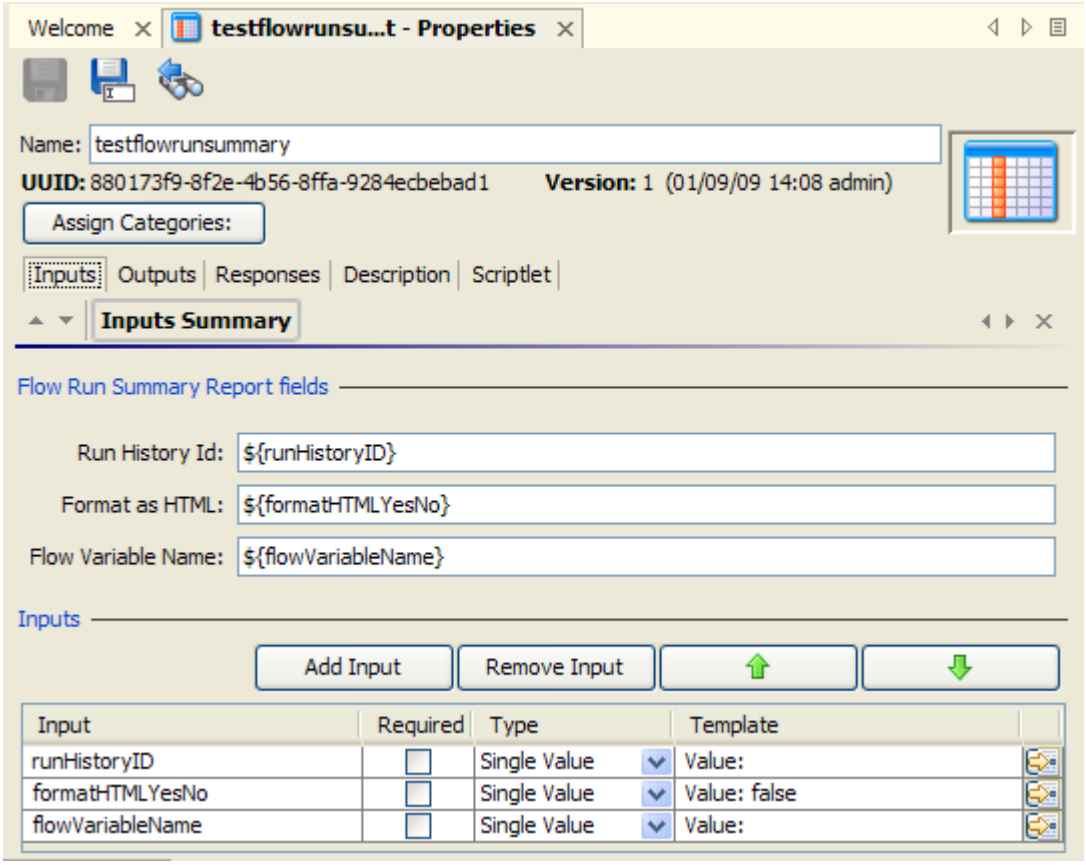


Figure 104 - Flow Run Summary Report operation properties

Following are the inputs to Http operations, and what they need for values:

runHistoryID – The numerical ID for the run history in the OO database. If left blank, the ID of the current run history is used.

formatHTMLYesNo - If you choose a constant value of **true**, the operation formats the results as an HTML table. The default is **false**, which formats the results as plain text.

flowVariableName – If defined, this input places a bidimensional array [N][4] in a flow variable with this name, where:

- N is the number of steps in the report.
- The four columns contain: at index [0] the path to the step in the flow, at index [1] the step name, at index [2] the response, and at index [3] the description.

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.

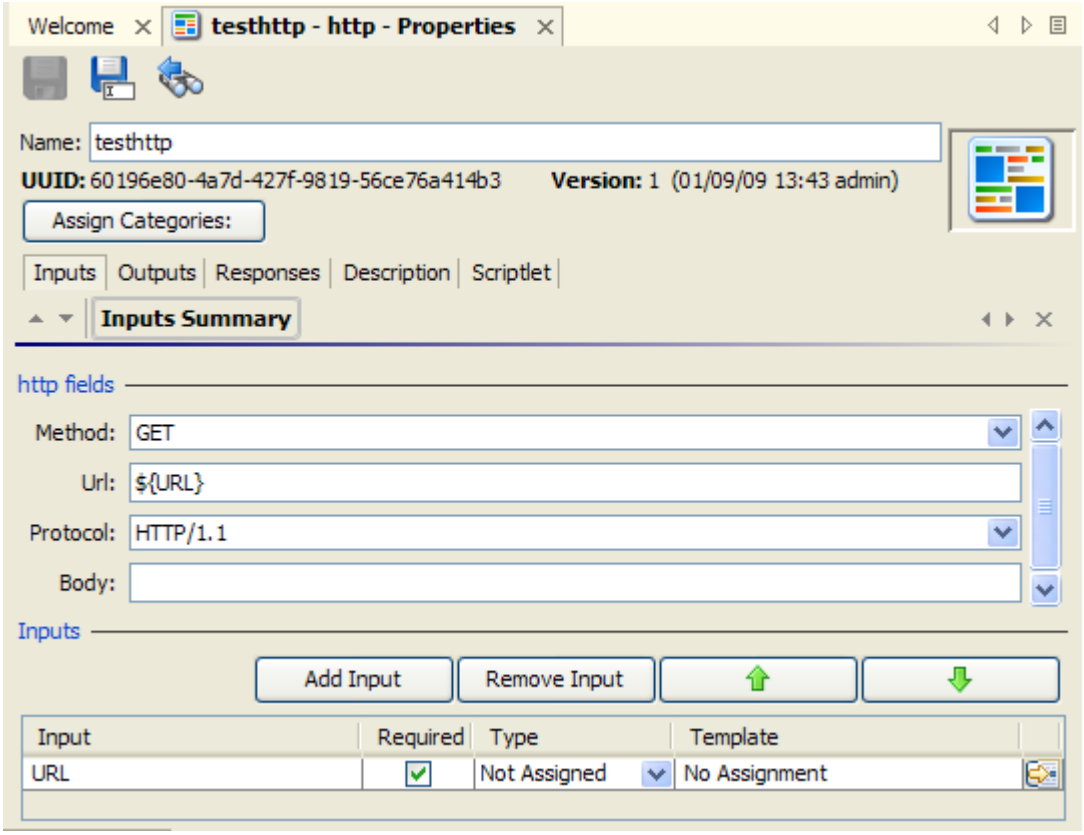


Figure 105 - HTTP operation properties

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.

- **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).

- **Result**
  The result of the operation.

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.

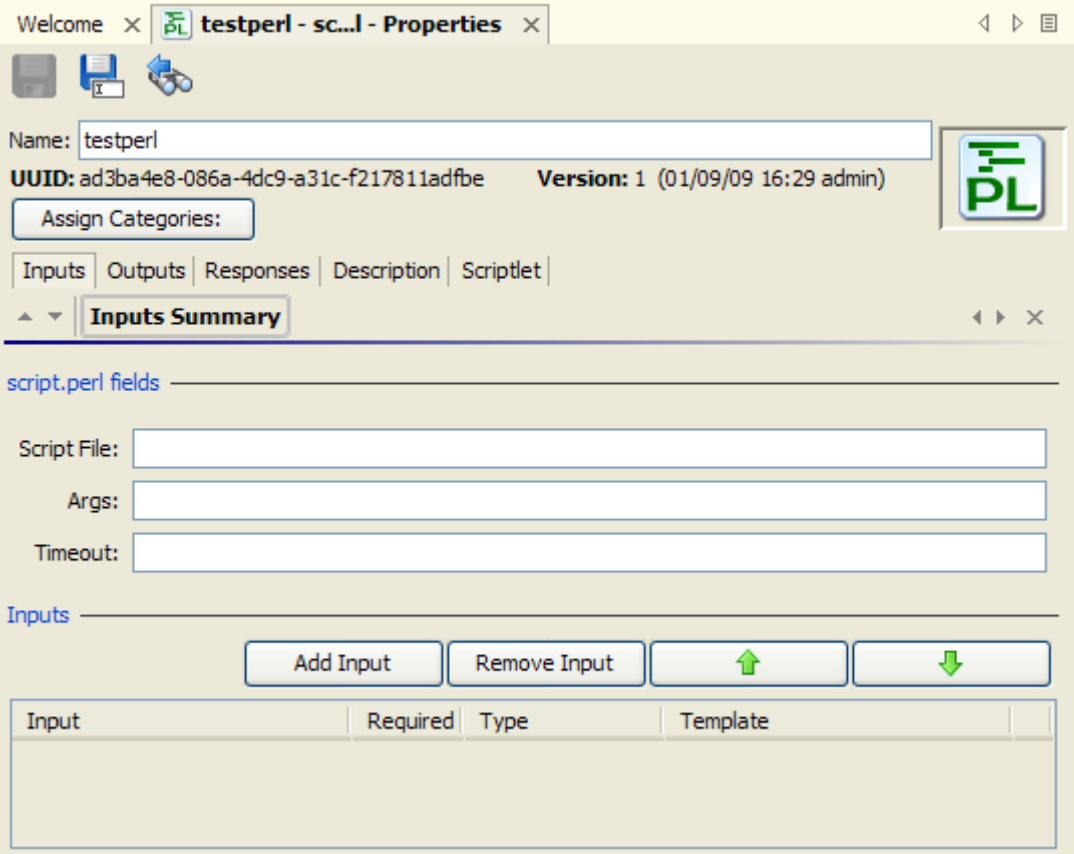


Figure 106 - Perl script operation properties

Following are the inputs to Perl script operations, and what they need for values:

Script File – The name of 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 a 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.

- **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).
- **Result**
  The result of the operation.

RAS operation: IAction programming for a 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.

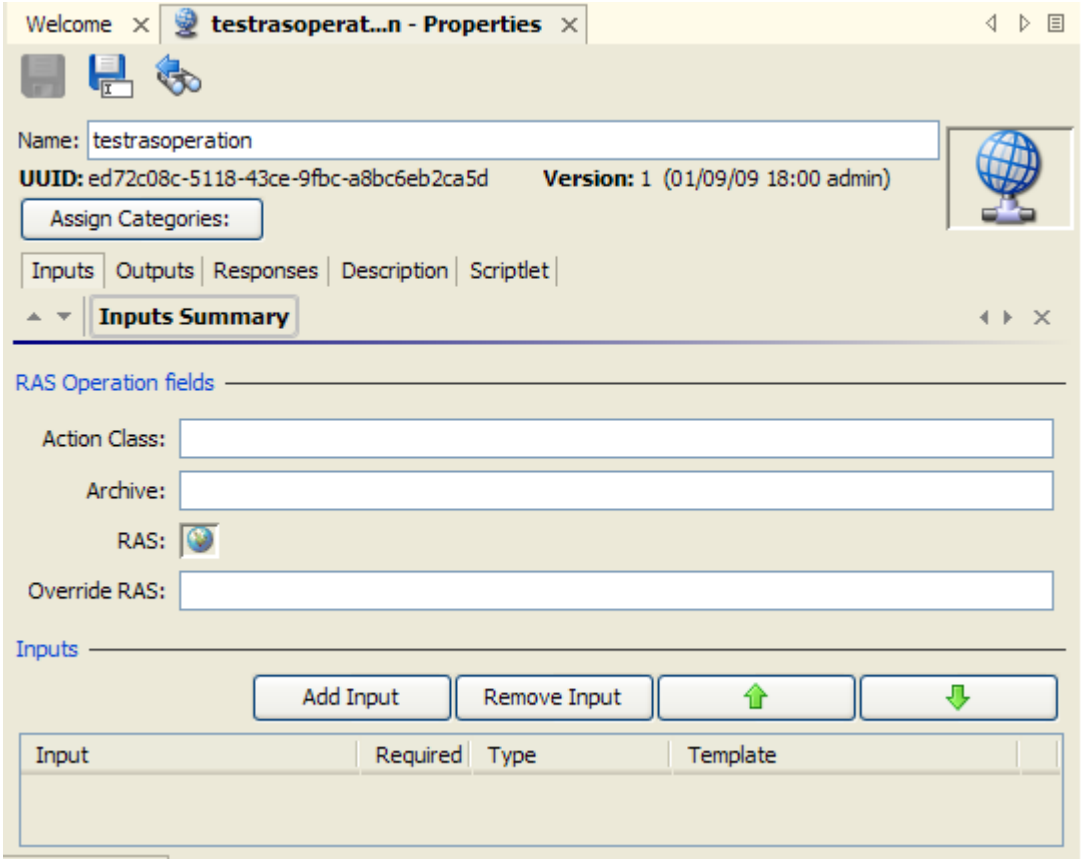


Figure 107 – RAS operation properties

Following are the inputs to RAS 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.

RAS – 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. To change the RAS reference, drag a RAS reference from **the Configuration\Remote Action Services** folder to the **Globe** icon ( ) beside the **RAS** field label.

Override RAS - The name of a RAS to use in place of the one associated with the operation (the one specified in the **RAS** field, described above).

Tip: You can create and use a flow variable, such as ${rasOverride} to store the RAS that the override specifies.

For information on changing the RAS reference in the operation or overriding it in the step, see *Changing RAS references*.

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 RASes, .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 operations*.

Scriptlet operation

Use for creating a scriptlet that you want to have available for reuse in multiple flows.

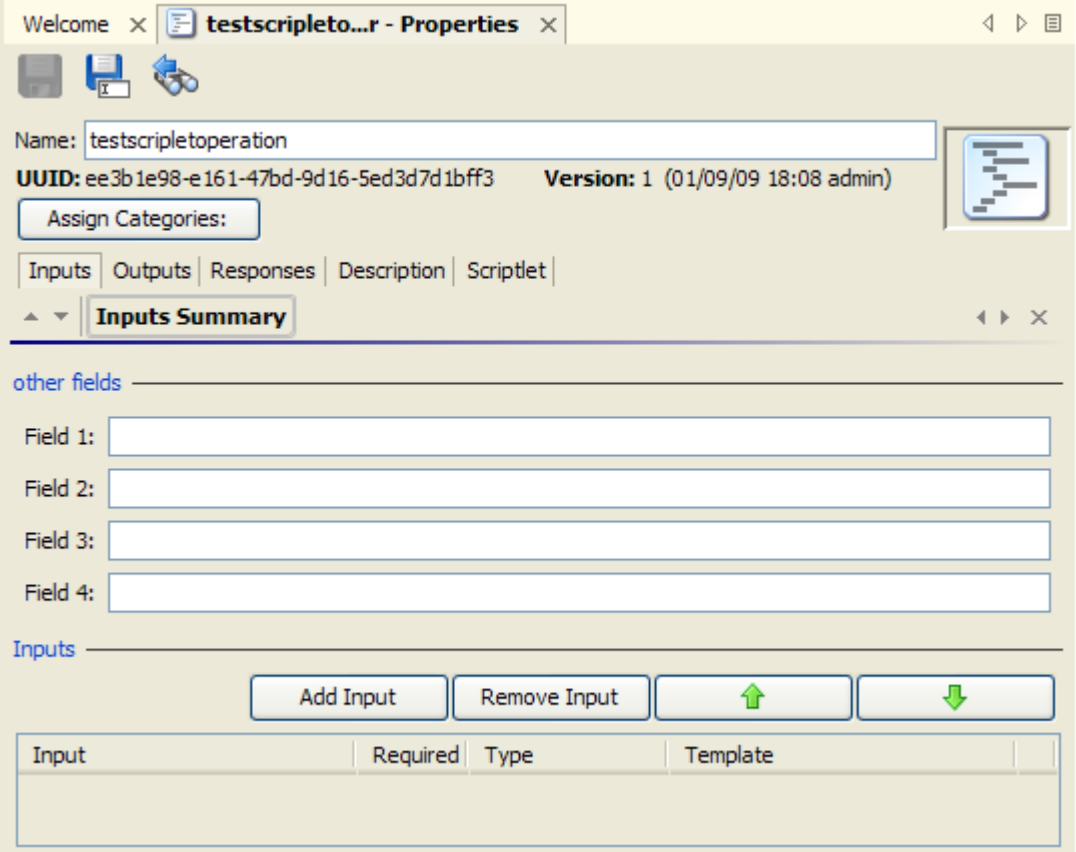


Figure 108 - Scriptlet operation properties

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 **Scriptlet** 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.

- **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\Operating Systems\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.

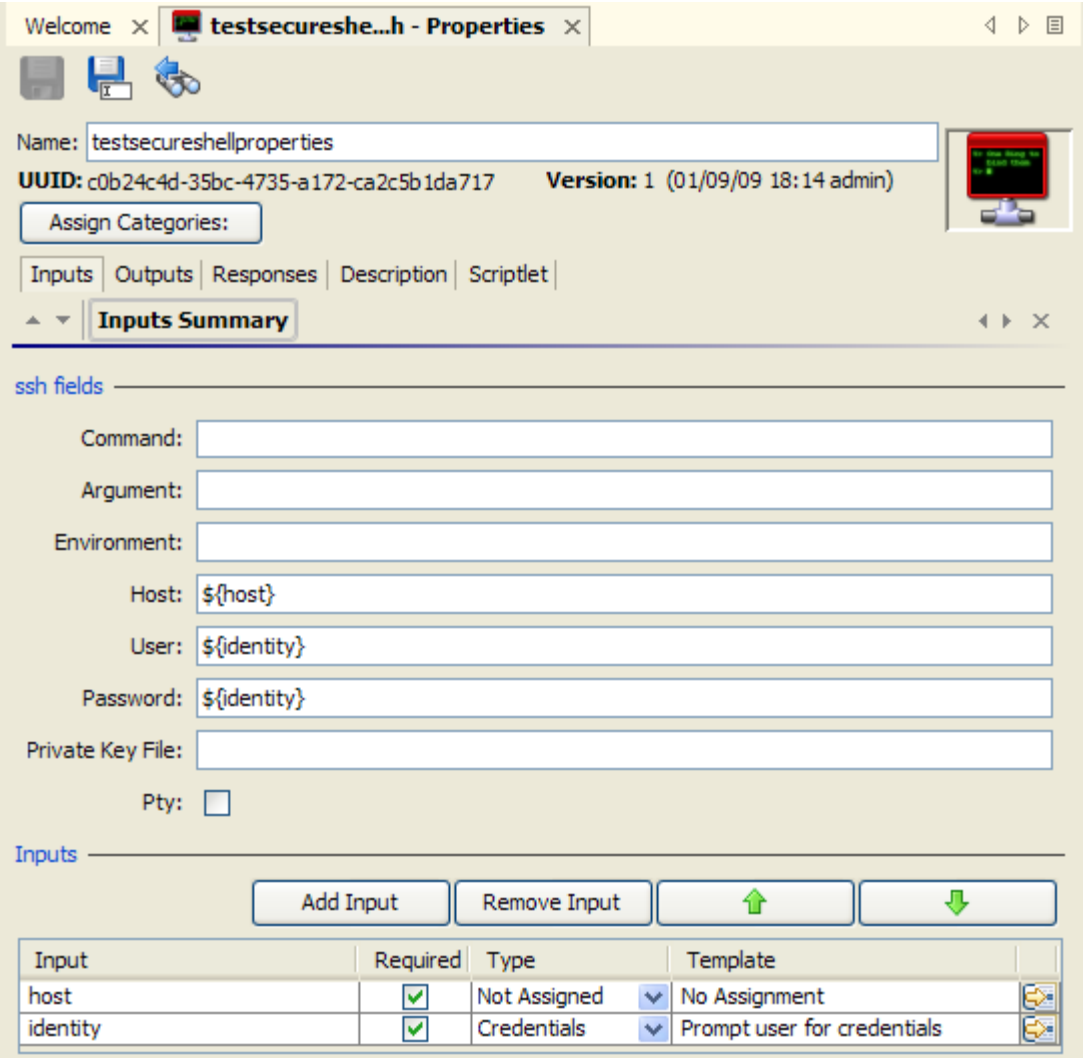


Figure 109 - Secure shell operation properties

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** check box 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 a 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
- Result

Telnet operation

Use for sending messages or commands to a server using the Telnet protocol.

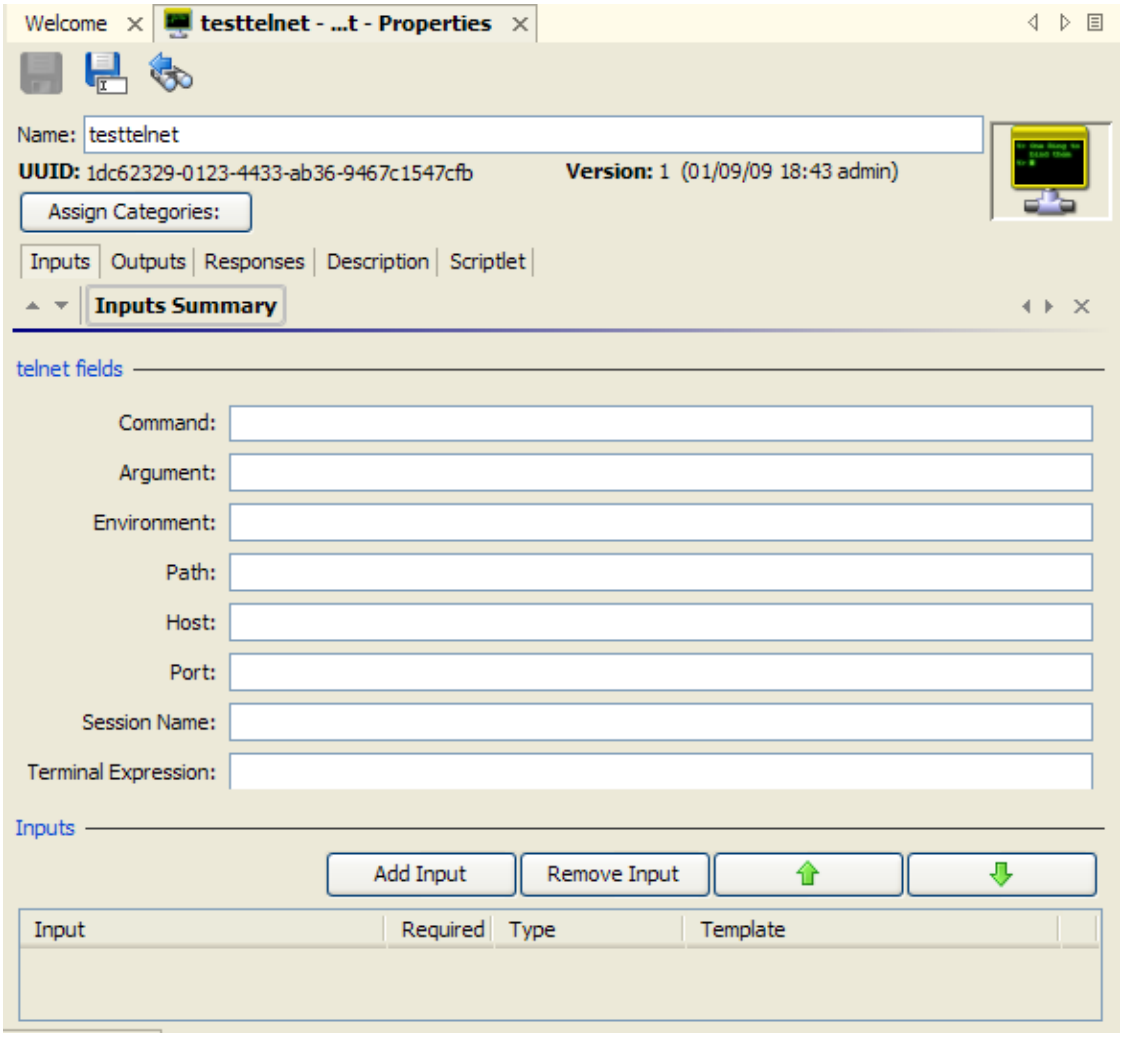


Figure 110 - Telnet operation properties

Following are the inputs to Telnet operations, and what they need for values:

Command – The command that the operation executes.

Argument – The command-line argument (switch).

Environment – Additions to the existing environment variables.

Path – The path to the command.

Tip: To specify a particular port for the host, specify the host and port with the syntax:

```
<hostname> <portname>
```

Note that for Telnet operations, the separator between <hostname> and <portname> is a space.

Host – The target computer for the 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.

Terminal Expression – A regular expression that you can use to search the operation's response for a sequence that indicates that the operation has completed. This can vary from command to command, so using this field requires knowledge of the returns of telnet commands. If you do not

use this field, the telnet operation allows the command to run until the telnet timeout value is reached.

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.

- **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).
- **Result**

  The result of the operation.

Shell operation

In contrast to a command-line operation, which can run only one command, a shell operation can run a series of commands, which are specified in an input to a step created from the shell operation. There are two means of using shell operations:

- Use one of the three shell operations that are supplied in default OO content (the collection of flows and operations that are installed when Studio is installed). Of these three shell operations, one uses the SSH protocol, one uses the telnet protocol, and one uses either protocol.
- Use the Shell Wizard to create a flow that uses a shell operation.

  For information on using the Shell Wizard, see the following topic, *Shell Wizard: creating a flow that uses a shell operation*.

  For information on using a shell operation, see *Using a shell operation*.

Shell Wizard: creating a flow that uses a shell operation

The Shell Wizard guides you through creating a flow that contains a shell operation based on either the Secure Sockets (SSH) or the telnet protocol. The wizard includes recorder-like technology that creates steps in a flow from the commands that you carry out in the wizard's shell window.

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):

- Host name
- Name and password of the user account under which the command(s) will execute on the host

Note: The Shell Wizard cannot communicate with a server that does not have either an SSH or a Telnet server installed.

To create a flow that uses a shell operation

1. In the HP OO home directory, in **\Studio\tools**, click or double-click **shellwizard.exe**.

The Shell Wizard starts.

2. In the **Select repository** page, 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.
   - 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. Having typed the repository to open or located it by navigating, click **Next**.

   The **Specify flow information** page

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 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.

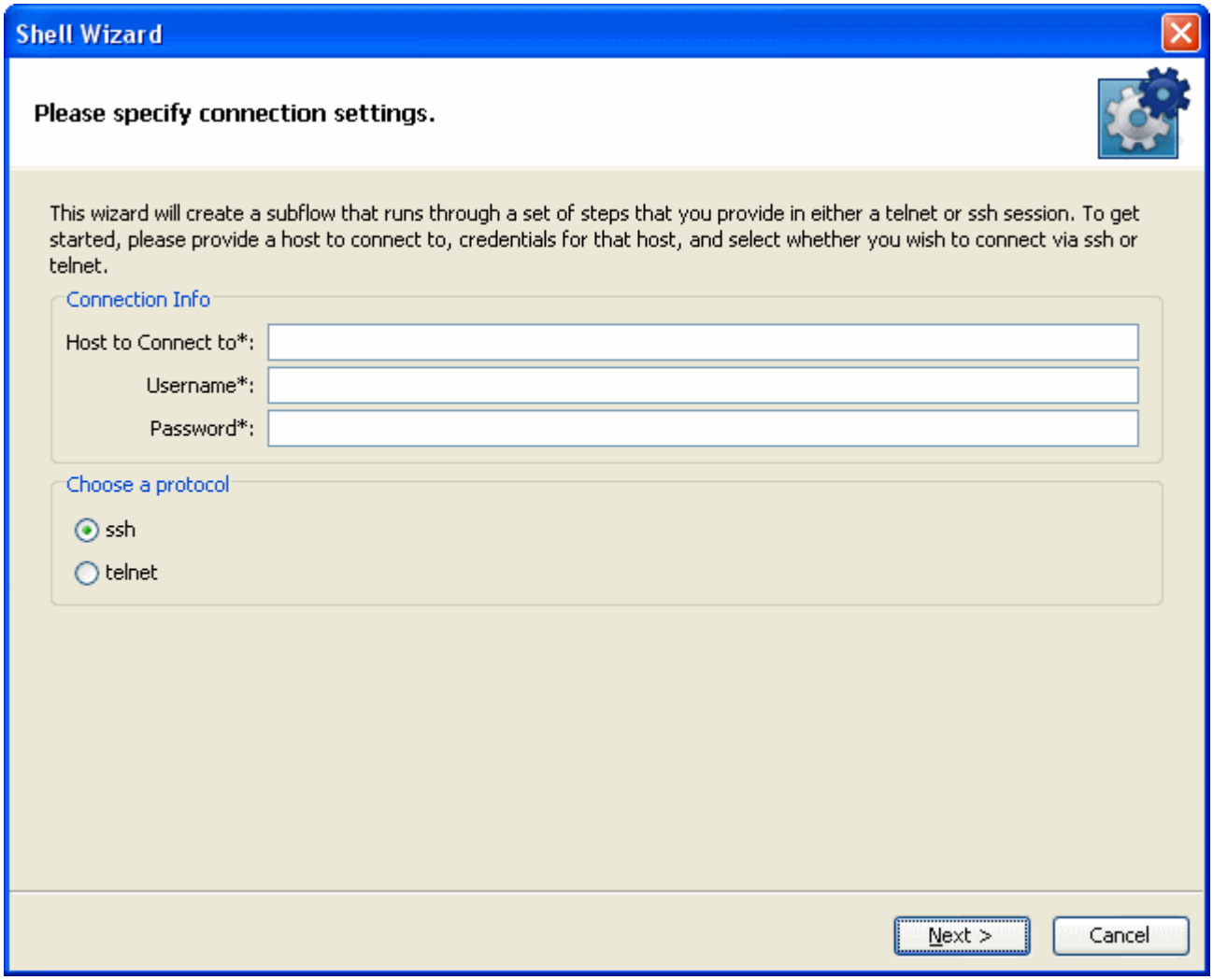


Figure 111 - Shell Wizard, specifying connection settings

6. Under **Connection Info**, in the **Host to Connect to** box, type the name or IP address of the computer 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**.

   If you specify the telnet protocol for sending the command, the following appears.

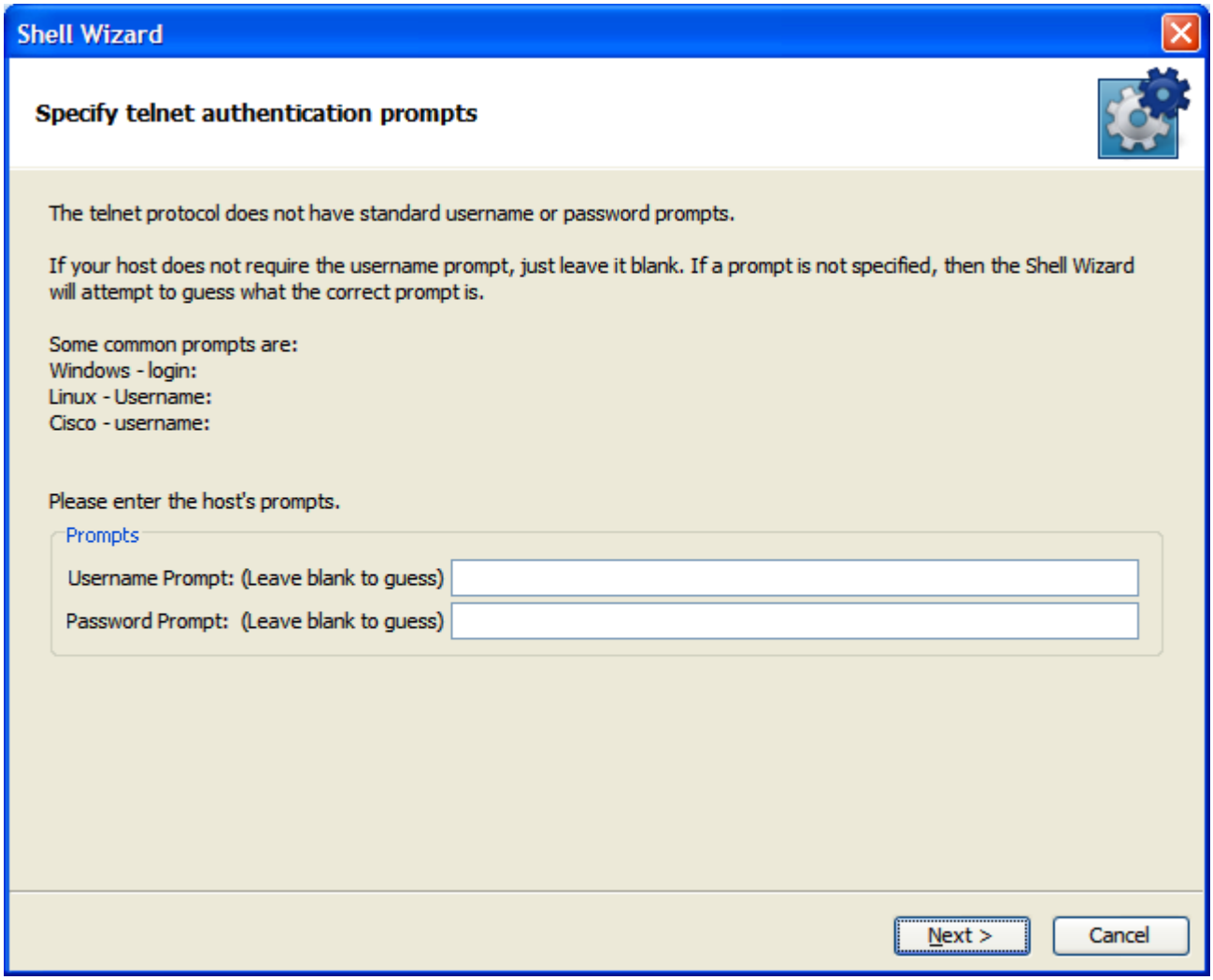


Figure 112 - Specifying the host prompts for credentials

9. If you know what the username and password prompts are on the host machine, enter them. The next page of the wizard contains a command window on the host machine.

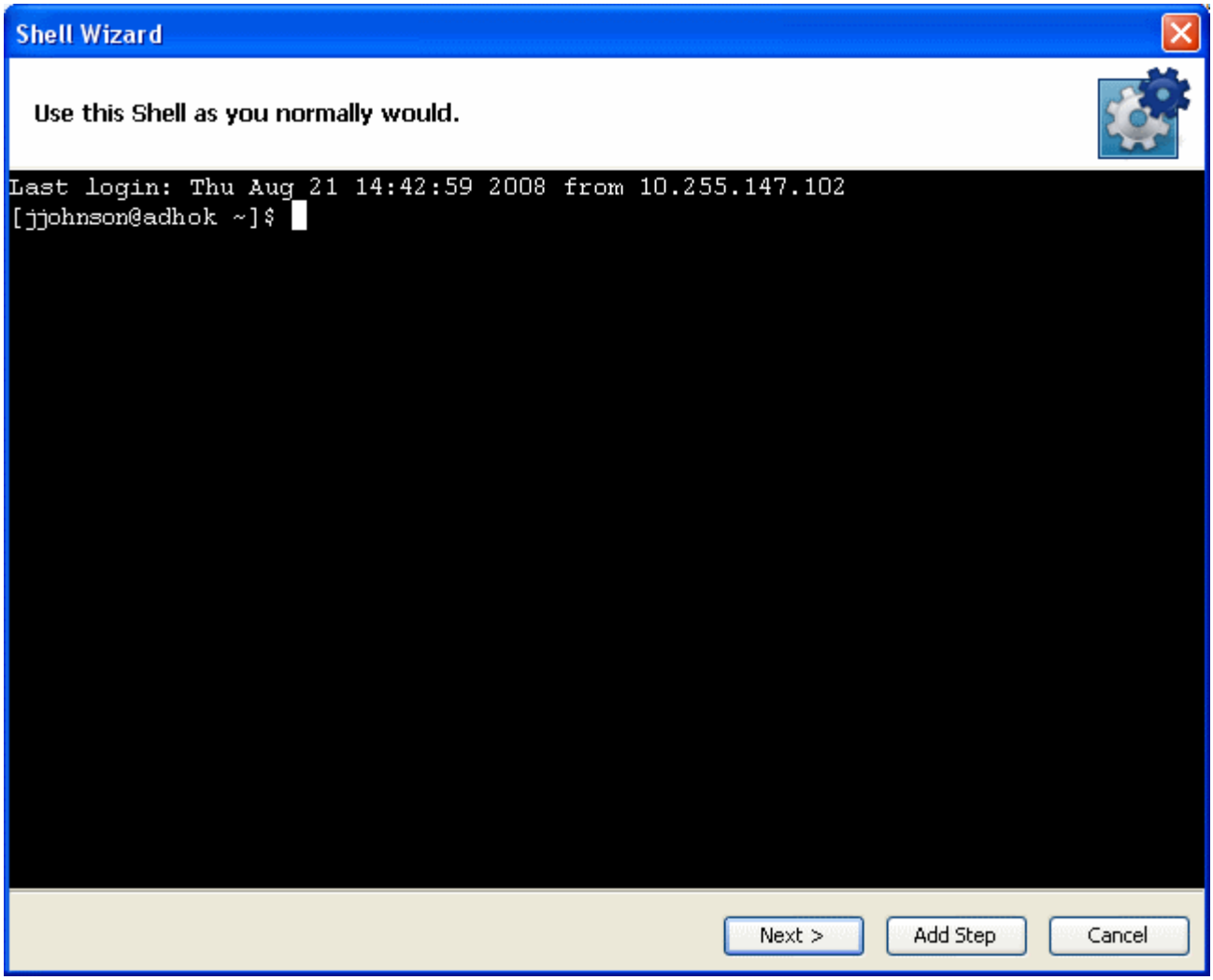


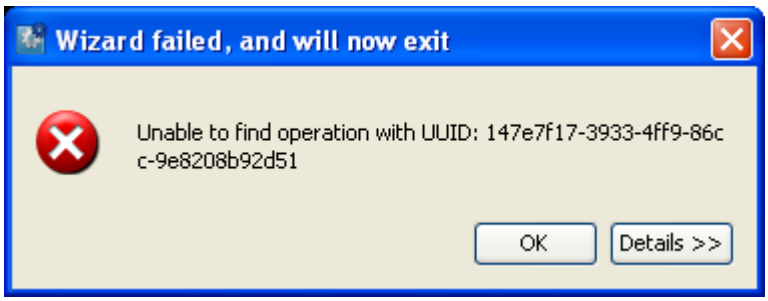
Figure 113 - Command-line window on host computer

10. Type the command or command sequence that you want to execute.
11. 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**.
12. To create another step in the flow that the Shell Wizard creates, type another command or sequence of commands, and then click **Add Step**.

    You can add as many steps as you wish.
13. 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.

14. In the **Wizard is Finished** page, click **Finish**.

Best Practice: To help authors who will create flows using the operations you create, add the following information to the operation's **Description** tab. (If you create multiple flows or operations that interact with the same technology, group them into a single folder and provide this information in the folder's **Description** tab. This is the practice for default HP OO content.) Note that putting this information on the **Description** tab makes it available to authors and Central users through the Generate Documentation feature. For more information on Generate Documentation, see *Viewing many operation and flow descriptions*.

- A description of what the operation does
- **Inputs** that the operation requires, including where authors can find the data that the inputs require and the required format for the data
- **Responses**, including the meaning of each response
- **Result fields**, including a description of the data supplied in each result field
- Any additional implementation **notes**, such as:
  - Supported platforms or applications, including version information
  - Application or Web service APIs that the flow interacts with (this can be particularly important for flows that require an RAS to run, because the RAS operation can hide this information from the author or user of the flow.
  - Other environmental or usage requirements

Using a shell operation

To use a shell operation, you either:

- Create a step in a flow from one of the shell operations in OO default content.
- Use the flow you created with the Shell Wizard.

The primary result for a shell operation is the raw output (character stream) of its SSH or telnet session. If the commands that you are executing with the shell operation are menu-driven, are visually formatted (for instance, in tables), include nonprintable characters, or present more than screen, obtaining visualized results will help you read and filter on the results.

Visualized results are the representation of the results in XML format, which preserves the formatting such as tables and presents the results in the screens in which they appear.

The following procedure assumes that you have created a flow that contains a step that is based on a shell operation.

To obtain visualized results of a shell operation

1. In the flow or subflow, open the Inspector for the step that is based on the shell op.
2. On the **Results** tab, add a new result and, in the **From** column, select **Field:visualized**.

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 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
^	Matches the beginning of a string
$	Matches the end of a string
.	Any character except newline
\b	Word boundary
\B	Any except a word boundary
\d	Any digit 0-9
\D	Any non-digit
\n	Newline
\r	Carriage return
\s	Any white space character

Wildcard	Uses
\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 Filters.**

   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 Filters.**

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 OO objects, see *Controlling access to OO objects*.

For information on defining OO roles or mapping external roles or groups to the OO groups described here, see the *HP OO Administration Guide* (AdminGuide.pdf) and Help for Central.

- Hiding the flow from 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 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_ONE** OO role but a subsequent part of the flow needs to be escalated because it requires user credentials that a **LEVEL_ONE** user doesn't have. When you create the flow, you could make a *gated transition* of the transition that leads to the step where the credentials are required. A gated transition is 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 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.

  The **Acquire Lock** and **Release Lock** operations are part of the default OO content. For more information using them, see their **Description** tabs.

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 RAS. The RAS 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 OO SDK to actions of the target system's or application's SDK. The IAction interface thus mediates between OO and systems external to OO.

Standard operations interact with the 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. Thus you can create operations that:

- Interact with systems throughout the network or Internet.
- Integrate OO with other entities that RAS can interact with through IActions.

Programming operations with Web extensions requires .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.
2. Copy the .dll or .jar into your Web service (your Web server's \bin\Actions directory).
3. Import the Web service into Studio.

To learn how to author and implement IActions, see the material on authoring IActions in the OO *SDK Guide* (SDKGuide,pdf).

Troubleshooting

I am logged into two instances of Studio connected to the same public repository, and my changes in my workspace in one instance of Studio are not reflected in the other Studio.

Key information: Any one user account must not connect to the public repository for the same installation of Central from two different instances of Studio. If this happens, the user's workspace in the repository can become corrupted. This can also occur when Central is clustered.

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 **Repository** 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:
   `dharma.app.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:

   ```
   dharma.app.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.

Index

.vm template files, 44

Accelerator Packs
  importing, 130

Access permissions
  creating group masks for, 160
  setting defaults for groups, 160

Administrator
  and forcible checkins, 26

Authoring canvas
  keyboard shortcuts, 20

Authoring objects
  controlling access to, 156

Authoring pane
  keyboard shortcuts, 20

Authoring pane, overview, 15

Authors, multiple. *See* Multiple authors

Bookmarking, **51**

Bookmarks
  flows and operations, adding and removing, 51
  shelves, adding and removing, 51
  shelves, hiding, 52
  shelves, moving, 52
  shelves, renaming, 51
  shelves, showing, 52

Bookmarks pane
  keyboard shortcuts, 21

Bookmarks pane, overview, 18

Breakpoints
  disabling, 119
  enabling, 119

Callouts
    adding to flows, 33
  adding from palette, 15

Categories, 153

Central
  and RAS-dependent operations, 139

Changes
  abandoning, 27

Checkin, 11, 24
  abandoning changes, 27
  forcible, 26

Checkins
  resolving conflicts between, 25

Checkout, 11
  reverting, 27

Checkpoints, 89

Cmd operation, 162

Command-line operation, 162

Concepts
  advanced, 8

Concurrent execution, 77

Concurrent processing
  multi-instance steps, 83

Configuration folder objects
  recovering deleted, 52

Configuration Item
  defined, 151

Configuration objects
  checking in, 21, 22
  checking out, 21, 22

Conflicts
  resolving, 25

context
  defined, 7
  global, defined, 7
  local, defined, 7

Controlling access
  to authoring objects, 156

copyright notices, ii

Dashboard Reporting
  Quick View, 150

Data
  passing, 112

Data movement, 112

Debugger
  changing flow variable values, 118
  keyboard shortcuts, 21
  overview, 115
  resetting flows within, 115
  restarting flows within, 115

Deletions
  restoring, 52

DescribeFlows.vm
  template files structure, 46

Description, flow
  to see, 37

Description, operation
  to see, 37, 38

Descriptions, 114

Domain terms

adding, 152
creating, 151
defined, 151
editing, 152
values, adding, 152

Evaluators
creating, 149
deleting, 150
editing, 150

Filters, 98
creating, 98
re-using, 106
saving, 100, 106

Flow
checking in, 21, 22
checking out, 21, 22
creating, 3
debugging, 114
description, 36
return step, adding, 63
searching, 34
searching, 34
step versus operation, 55
step versus subflow, 55

Flow description
to see, 37

Flow design
considerations, 111
simplifying, 111

Flow operations
advanced authoring, 6

Flow output fields, 112

Flow Run Summary Report operation, 163

Flow state
defined, 89

Flow variables, 72
changing values within Debugger, 118
creating, 72
creating, 73
global, 9
local, 9
local, creating, 73
reserved, 74
scope, 9

Flow, creating
overview, 29

Flow, parts, 2

flows
multiple domains, running in, 145

Flows. *See also* Operations, *See* Flows
adding callouts, 33
adding steps, 54
advanced concepts, 8
and subflows, 112
bookmarking, **51**
categories, 153
checkpoints, 89
copying, 49
creating, 29, 30
creating from templates, 31
creating steps, 54
deploying changes, 181
finding, 33
gated transitions, 181
hiding from Central, 181
importing, 130
inputs, 9, 63
introduction to, 1
limiting use, 181
moving between repositories, 123
outputs, 93
providing with data, 63
recovering deleted, 52
requirements, 3
responses, adding, 109
running automatically, 114
sealed, 34
sealed, defined, 8
self-documentation, 38, 41
steps. *See* Steps
testing, 181
transitions. *See* Transitions

Folders
creating, 29

Generate documentation
modifying custom templates, 47

Generate Documentation
command, 38, 43
custom, 43
operation, 38, 41

Global context, 7

Good first place to go, 1

Group masks
creating, 160

Group permissions, 156

Groups
setting default access permissions for, 160

How do I... folder, 1

HP OO objects
permissions, setting, 158

Http operation
properties, response definitions, 164

IActions
creating for operations, 182

Icons pane, overview, 18

Inputs, 9, 63

and flow run scheduling, 72
assigning data sources to, 65
creating, 63, 64
deleting, 72
encrypting, 65
flow variables, 74
recording for Dashboard Reporting, Quick View, 150
removing, 72
user prompts, supplying lists to, 70

Inspector
keyboard shortcuts, 21
opening, 57

Keyboard
shortcuts, 20

Keyboard shortcuts
Authoring canvas, 20
Authoring pane, 20
Bookmarks pane, 21
Debugger, 21
Inspector, 21
Properties editors, 21
Repository pane, 20
Scriptlet panel, 21

Lane steps
copying, 81
moving, 81

Lanes
adding, 82
changing start step, 82
copying, 81
deleting, 82
duplicating, 82
moving, 81
renaming, 82
resizing, 82

legal notices, ii
copyright, ii
restricted rights, ii
trademark, ii
warranty, ii

Library object
checking in, 24
checking out, 24
earlier version, opening, 28
earlier version, restoring, 28

Library objects
recovering, 52

Library, overview, 12

Lists
for user prompts, 70

Local context, 7

Local flow variables
creating, 73

Login
switching, 121

Messages
displaying to users, 153

Multi-instance steps, 77, 83
creating, 85
debugging, 88
in flow design, 83
moving data, 86
throttling, 88

Multiple authors
and version control, 11

multiple domains
running flows in, 145

My Changes/Checkouts pane
overview, 15

Nonblocking steps, 77, 88
creating, 88

OO objects
controlling access to, 156
permissions, 156

Operation
checking in, 21, 22
checking out, 21, 22
description, 36
output data, 7
raw results, 7
response, 7
scriptlet, 7
searching, 34

Operation description
to see, 37, 38

Operation outputs
filtering, 98

Operation responses
rules, 106

Operation types, 162
Cmd, or command line, 162
Flow Run Summary Report, 163
shell, 173

Operations
advanced authoring, 6
advanced concepts, 8
and Web services, 144
architecture, 7
bookmarking, **51**
compared with steps, 9
copying, 49
core functionality, 7
creating, 161
creating from a RAS, 146
creating from RAS, 146
creating from Web services, 135

creating IActions for, 182
creating with IActions, overview, 183
defined, 7
finding, 33
finding flows that use them, 49
information flow, 7
inputs, 9
optional inputs, 162
outputs, 93, 94
outputs, adding, 95
outputs, deleting, 95
outputs, filtering, 98
properties, 162
Properties sheet, 161
Quick View, 4
recovering deleted, 52
response definitions, 162
responses, 93, 95
scriptlets, creating, 90
sealed, 34
sealed, defined, 8
viewing many, 38
viewing more information, 49
What does this use, 49
What uses this, 49

Ops flow. *See* Flow

Output data, 7

Outputs
adding, 95
changing the source, 96
deleting, 95
described, 10

Palettes
Steps and Callouts, 15
View Options, 16

Parallel execution, 77

Parallel processing. *See* Concurrent processing

Parallel split step lane steps
moving, 81

Parallel split step lanes
moving, 81

Parallel split steps, 77, 78
adding from palette, 15
adding lanes, 82
and scriptlets, 79
changing start steps, 82
copying, 81
creating, 80
debugging, 83
deleting lanes, 82
duplicating lanes, 82
lane order, 79
moving, 81
moving data, 79
renaming lanes, 82
resizing lanes, 82

Perl script operation
properties, response definitions, 166

Permissions
setting, 156
setting for HP OO objects, 158
setting for system objects, 158

Properties editors
keyboard shortcuts, 21

Quick View
inputs, recording for Dashboard Reporting, 150
Operations, 4

RAS
requirements, 138

RAS operation
properties, response definitions, 167

RAS reference
changing, 143
overriding, 143

RAS references
adding, 140
configuring from existing RAS, 146
reconfiguring, 142
removing, 148

RAS-dependent operations
troubleshooting, 145

RASes
adding an existing, 140
adding operations, 146, 147
availability, checking, 140
creating operations from, 146
creating operations from, 146

RASs
Using, 138

Raw results
defined, 7

References to operations
finding, 49

Regex. *See* Regular expressions

Regular expressions, 179

Remote Action Services. *See* RASes, *See* RAS

Repositories, 121
adding, 121
Backing up, 135
Backup, importing the, 129
decrypting, 134
encrypted copies, 134
encrypting, 133
exporting, 129
importing, 130
opening, 121

previewing the publish, 124
previewing the update, 126
private, 121
public, 121
public, undoing, 129
publishing, 124
restoring, 135
setting a target repository, 123
source, 123
target, 123
update, undoing, 129
updating, 126
validating, 133

Repositories, encrypted
opening, 134

Repository pane
keyboard shortcuts, 20

Repository, public
publishing to, 123
updating from, 123

Response
defined, 7

Responses
defined, 95
Overriding within Debugger, 120

restricted rights legend, ii

Results
adding, 96
changing the source, 97
deleting, 96
described, 10
filtering, 100

Results, step
defined, 95

Return steps
adding, 62
adding from palette, 15
changing responses, 62
defined, 62

Sample flows
How do I... folder, 1

Scheduling
and inputs, 72

Scriptlet
defined, 7

Scriptlet operation
properties, response definitions, 169

Scriptlet panel
keyboard shortcuts, 21

Scriptlets, 90
and parallel split steps, 79
creating, 90, 91
debugging, 92
saving for reuse, 93

Sealed
defined, 34

Secure shell operation
properties, response definitions, 170

Selection lists, 154

Self-documentation
flows, 38, 41

Shell operation, 173

Shell operations
getting visualized raw results, 178
Shell Wizard, 173
using, 178

Shell Wizard
creating flows with, 173

Shortcuts
keyboard, 20

Single-response steps
creating, 89

Start Step
changing, 56

Step icons
changing, 109

Steps
adding, 54
adding to flows, 56
advanced concepts, 8
changing icon, 109
changing operation, 112
compared with operations, 9
connecting, 58
copying, 57
copying between flows, 57
creating, 53, 54, 56
descriptions, 114
execution, 6
inputs, 9
modifying, 57
moving, 58
nonblocking, 88
nonblocking, creating, 88
results, 93, 95
results, adding, 96
results, deleting, 96
single-response, 89
transitions. *See* Transitions
user prompts, 58

Steps, multi-instance. *See* Multi-instance steps

Steps, parallel split. *See* Parallel split steps

String formats. *See* Evaluators

Studio. *See* Studio

good first place to go, 1
keyboard shortcuts, 20
Library, 12
starting, 11
visual overview, 11

Subflows
using, 111

System accounts, 155
creating, 155
deleting, 156
editing, 156

System evaluators, 148

System objects
permissions, setting, 158

System properties
creating, 110
using, 110

Target repository
setting, 123

Telnet operation
properties, response definitions, 171

Template files, 46
structure, 46

templates
creating flows from, 31

Toolbar
overview, 15
palettes, 15
Steps and Callouts palette, 15
View Options palette, 16

trademark notices, ii

Transitions, 58
adding, 60
creating, 58
moving, 61

User messages
displaying, 153

User prompt inputs
supplying lists to, 70

User prompts
selection lists for, 154

Users
switching, 121

Variables. *See* Flow variables

Version control, 11, 21

Version, earlier
opening, 28
restoring, 28

Versions
managing, 27

warranty, ii

Web Service content
using, 148

Web services
creating operations from, 135