
HP Operations Orchestration Studio: Advanced Authoring

Lab Guide

Welcome to HP Operations Orchestration Advanced Authoring training. In this course, you will learn many advanced flow development techniques that will prepare you for working on a variety of Operations Orchestration deployments.

The version of HP Operations Orchestration used in training is 9.00.

Prerequisites

To be successful in this course you should have completed the following training or have equivalent experience with HP Operations Orchestration:

1. Getting Started With HP Operations Orchestration
2. HP Operations Orchestration Administration Essentials
3. HP Operations Orchestration Authoring Essentials

Objectives

By the end of this lesson you will be able to:

- Use Operations and Subflows to expand your Operations Orchestration library
- Incorporate Looping and Iteration into flows
- Filter flow data
- Use Parallel Processing to increase the efficiency of your flows
- Use responses, rules, and transitions to control flow execution
- Use Remote Action Service to extend HP OO to remote networks
- Control access to Operations Orchestration objects
- Persist data across flow runs
- Use XML Processing content and filters to extract information from XML documents
- Work With File Systems
- Work With Email

Exercises

Exercise 1: Working With Operations and Subflows

Exercise 2: Looping and Iteration

Exercise 3: Filtering Flow Data

Exercise 4: Parallel Processing

Exercise 5: Using Responses, Rules, and Transitions Control Flow Execution

Exercise 6: Remote Action Service

Exercise 7: Controlling Access to OO Objects

Optional/Self-study Exercises

Exercise 8: Persisting Data Across Flow Runs

Exercise 9: XML Processing

Exercise 10: Working With File Systems

Exercise 11: Working With Email

Exercise 1

Working With Operations and Subflows

In this exercise you will create new operations and subflows and incorporate them into flows.

Objectives

By the end of this exercise you will be able to:

- Create an operation and use it in a flow
- Set the primary output of the operation
- Create two subflows and use them in a parent flow
- Create results to assign subflow step results to Flow Output Fields
- Access the Flow Output Fields in the parent flow

Operations

The first part of this exercises focuses on operations. In Studio, solutions to this part of the exercise are in My Ops Flows/Exercise Solutions/Essentials/Operations.

Task 1: Create a ping operation

In this task you will create a new operation that pings a host you specify.

1. In My Ops Flows, create a new folder named **Operations**, then create a subfolder in Operations named **Ping Flow**.
2. Create the Ping operation.

Right-click the Ping Flow folder, select New Operation, and select cmd as the type of operation. Name the operation **Ping**, then set the operation properties:

Name: Ping
 UUID: d1447d00-f009-4af0-89b6-ec3e87d5e64e Version: 1 (03/25/10 21:22 admin)
 Assign Categories:

Inputs Outputs Responses Description Scriptlet

Inputs Summary

cmd fields

Command: ping
 Argument: \${host} -n 3
 Environment:
 Path:
 Working Directory:

Inputs

Add Input Remove Input ↑ ↓

Input	Required	Type	Template
host	<input type="checkbox"/>	Not Assigned	No Assignment

- Click the Add Input button and add an input named **host**. No changes are needed to the host input.
- For Command: enter **ping**
- For Argument enter **\${host} -n 3**.
- Save changes and close the Properties window.

This operation pings the host three times. It would be the same as entering **ping host -n 3** at a command line.

- Set the Primary Output for your Ping operation. Open the Outputs tab and select Output String for the primary output. (Initially the value is Code).

Name: Ping
 UUID: 138ec54f-02c5-4fce-89da-aa8330e684fd Version: 1 (05/20/11 14:20 admin)
 Assign Categories:

Inputs Outputs Responses Description Scriptlet

Outputs Summary

Extract Primary Output From Field: Output String Edit Filters

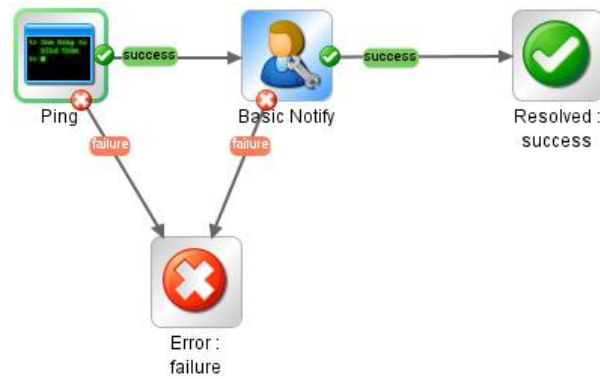
Available Outputs

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

Name

Remove Output

- Create a flow named **Ping Flow** that uses your new Ping operation. Assemble the flow as shown:



Basic Notify is in Utility Operations.

5. Modify Basic Notify inputs.

- a. Remove the destination, notificationServer, from, and serverPort inputs.

Assign To Input	Required	Type	From
notifydata	<input checked="" type="checkbox"/>	Single Value	Value: \${pingOutput}
notifyMethod	<input checked="" type="checkbox"/>	Single Value	Value: Display
subject	<input checked="" type="checkbox"/>	Single Value	Value:
destination	<input type="checkbox"/>	Single Value	Value:
notificationServer	<input type="checkbox"/>	Single Value	Value:
from	<input type="checkbox"/>	Single Value	Value:
serverPort	<input type="checkbox"/>	Single Value	Value:

- b. Set the notifyData input to Use Previous Step Result.

Name: notifyData Input Type: Single Value

Input Data Flow

Assign from Variable: notifyData

Otherwise: Use Previous Step Result

Assign to Variable: notifyData

- c. Set notifyMethod to a Constant Value of Display.

Name: notifyMethod Input Type: Single Value

Input Data Flow

Assign from Variable: notifyMethod

Otherwise: Use Constant

Assign to Variable: notifyMethod

'Otherwise: Use Constant' Configuration

Constant Value: display

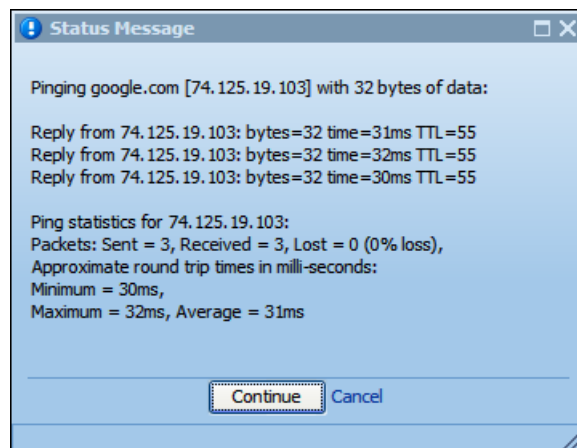
- d. To “disable” the subject, open its Input Editor, select Use Constant, and then leave the Constant Value field empty.

The screenshot shows the 'Input Editor' dialog box for a variable named 'subject'. The 'Input Type' is set to 'Single Value'. Under 'Input Data Flow', 'Assign from Variable' is 'subject', 'Otherwise' is 'Use Constant', and 'Assign to Variable' is 'subject'. Under 'Input Properties', 'Encrypted' is unchecked and 'Required' is checked. 'Validation Format' is '<not validated>' and 'Record Under' is '<run history>'. On the right, the 'Otherwise: Use Constant' Configuration panel shows an empty 'Constant Value' field.

- e. Save your changes.

Task 2: Test the flow

1. To test the flow, click the debug button and run.
2. When prompted to enter a host, enter **localhost** or another accessible host on your network. The ping result is displayed in the window.



Subflows

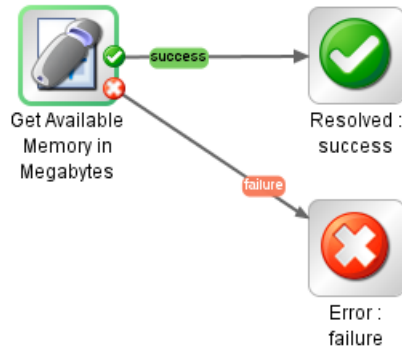
In this part of the exercise you will work with subflows. A subflow is a complete flow that serves as a step in a “parent” flow, much as an operation serves as a step in a flow. In this exercise you will use two subflows in a parent flow.

In Studio, solutions to this part exercise are in My Ops Flows/Exercise Solutions/Essentials/Subflows.

Task 1: Create the Get Available Memory subflow

1. Create a new folder and flow.

In My Ops Flows, create a new folder named **Subflows**. In that folder create a new flow named **Get Available Memory**. Assemble the components as shown:



Get Available Memory in Megabytes is in Operations/Operating Systems/Windows Management/Performance Counters/Memory.

2. Remove unused inputs.

Double-click Get Available Memory in Megabytes to open its Step Inspector and remove the user and password inputs, which are not used in this flow.

Assign To Input	Required	Type	From
host	<input checked="" type="checkbox"/>	Single Value	Prompt User
object	<input checked="" type="checkbox"/>	Single Value	Value: memory
counter	<input checked="" type="checkbox"/>	Single Value	Value: available mbytes
instance	<input type="checkbox"/>	Single Value	Value: null
user	<input type="checkbox"/>	Single Value	Prompt User
password	<input type="checkbox"/>	Single Value	Prompt User

3. Open the Input Editor for host (click the yellow arrow). Set the host input to Assign From and Assign To the host flow variable by using the pull-down menus. Then remove the User Message so that field is empty.

Name: Input Type:

Input Data Flow

Assign from Variable: Otherwise:

Assign to Variable:

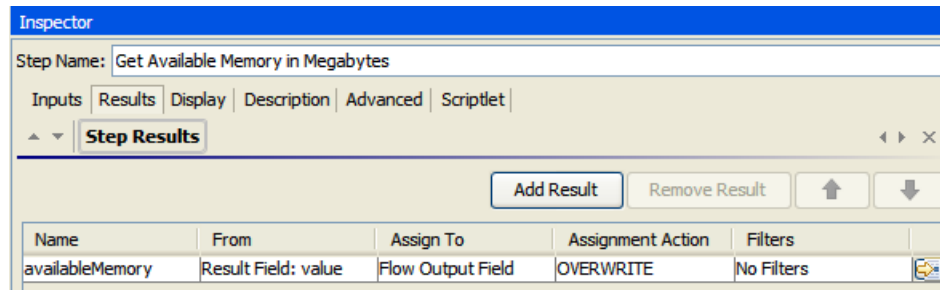
'Otherwise: Prompt User' Configuration

Prompt For: ☒ Text ☐ Selection

User Message:

4. Add an availableMemory result.

Select the Results tab and add a result named **availableMemory** based on the Result Field **value** and assign it to a Flow Output Field (not a Flow Variable). Assigning the result to a Flow Output Field makes it available to the parent flow.



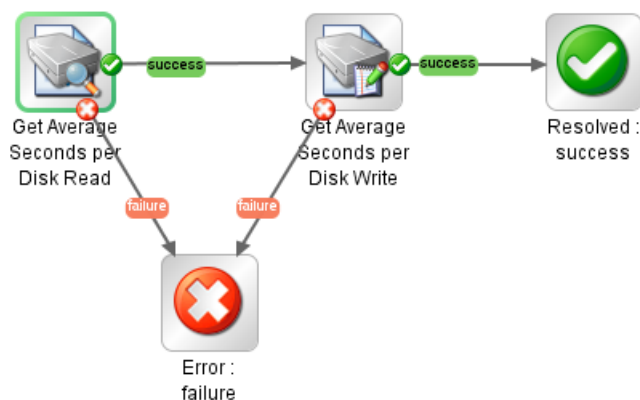
5. Save changes and test the flow.

When done save changes and test the flow in the Studio debugger using **localhost** as the host name, observing the output in the Run Tree and Step Result Inspector.

Task 2: Create the Average Disk Reads-Writes subflow

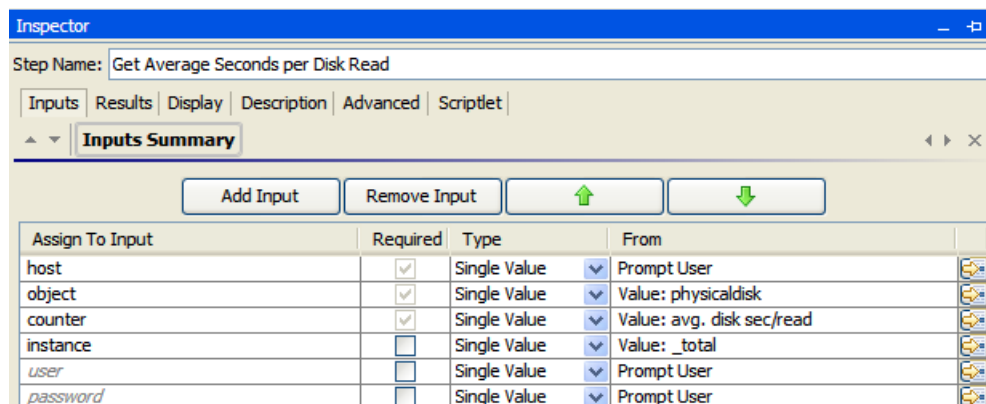
1. In the same folder, create the Average Disk Reads-Writes flow.

Assemble the components as shown. Get Average Seconds per Disk Read and Get Average Seconds Per Disk Write are in Operations/Operating Systems/Windows Management/Performance Counters/Disk.



2. Remove unused inputs.

For both of the Disk Read and Disk Write steps, remove the user and password inputs which are not used.



- For both of the Disk Read and Disk Write steps, set the host input to Assign From and Assign To host and remove the User Message.

The screenshot shows the 'Inputs Summary' configuration for a step named 'host'. The 'Name' is 'host' and the 'Input Type' is 'Single Value'. Under 'Input Data Flow', 'Assign from Variable' is set to 'host' and 'Assign to Variable' is also set to 'host'. The 'Otherwise: Prompt User' configuration is visible on the right, with 'Prompt For' set to 'Text' and 'User Message' empty.

- Add diskRead and diskWrite results.

For both of the Disk Read and Disk Write steps, add a Result based on the Result Field **value** and assign it to a Flow Output Field:

Get Average Seconds Per Disk Read: Add a result named **diskRead**:

The screenshot shows the 'Step Results' configuration for the step 'Get Average Seconds per Disk Read'. A table lists the results:

Name	From	Assign To	Assignment Action	Filters
diskRead	Result Field: VALUE	Flow Output Field	OVERWRITE	No Filters

Get Average Seconds Per Disk Writes: Add a result named **diskWrite**:

The screenshot shows the 'Step Results' configuration for the step 'Get Average Seconds per Disk Write'. A table lists the results:

Name	From	Assign To	Assignment Action	Filters
diskWrite	Result Field: VALUE	Flow Output Field	OVERWRITE	No Filters

- When done, save your changes and test the flow in the Studio debugger – you will be prompted twice to enter the hostname, which is localhost. You will correct that minor issue later. Observe the output in the Run Tree and Step Result Inspector. Close the flow window when done.

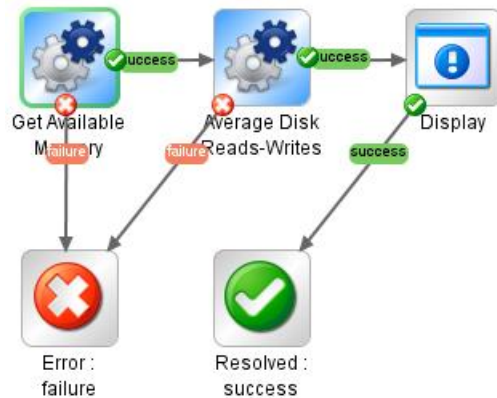
Task 3: Create the Get System Report flow

In this task you will create the Get System Report flow that uses the two subflows.

- Create a new flow in your Subflows folder named **Get System Report**. Assemble the components as shown.

Get Available Memory and Average Disk Reads-Writes are the two subflows you created in tasks 1 and 2. Simply drag them from the Library onto your

design panel and wire the components together as shown. The Display operation is in Utility Operations.



2. Add a **host** flow input.

Select the Properties tab at the bottom of the Studio window and add an input named **host**. Then open the Input Editor by clicking the yellow arrow to the right of the input and set the Input Type to Single Value. Enter a User Message: **Please enter the host name:.** Save changes and select the Design tab at the bottom of your screen when done.

3. Add host inputs for Get Available Memory and Average Disk Reads-Writes.

For each step, open the Step Inspector by double-clicking and add an input named **host**. Make sure to do this for both the Get Available Memory and Average Disk Reads-Writes steps. You don't need to make any other modifications – the host inputs for both steps should look like this:

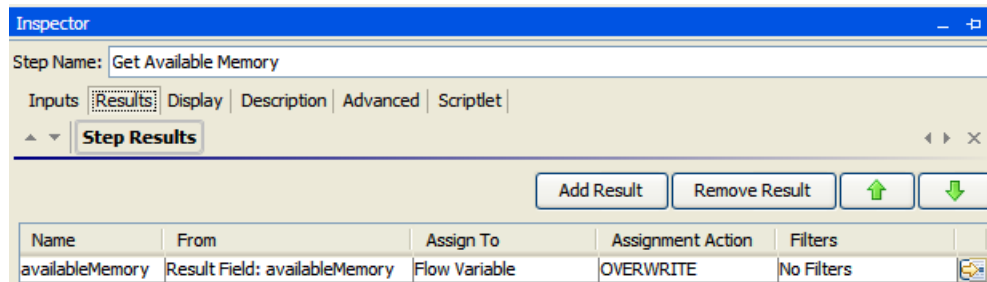
Assign To Input	Required	Type	From
host	<input type="checkbox"/>	Single Value	Prompt User

4. Add Results to Get Available Memory and Average Disk Reads-Writes.

For Get Available Memory and Average Disk Reads-Writes, double-click the step to open its Step Inspector and select the Results tab. Then click Add Result. Add the results as shown below.

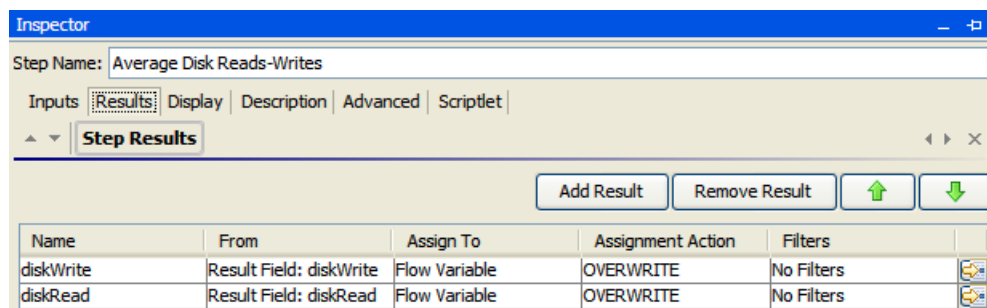
Note that when you add the result, the correct values are pre-populated. That is because they are based on the flow output fields that you set up in your subflows.

Get Available Memory: Add an **availableMemory** result based on the Result Field: availableMemory.



Get Average Disk Reads-Writes: Add two results. You will need to manually modify one of the results, depending on the order in which they were set up in the subflow. Examine the results carefully and make sure they match the values shown below before proceeding.

- a. diskRead, based on the Result Field diskRead
- b. diskWrite, based on the Result Field: diskWrite



5. Display the report in the Display step.

Open the Display Step Inspector and select the Display tab. Enter the following into the Prompt Text field to display the system report:

```
<b>Available Memory</b>: ${availableMemory} Megabytes
<b>Disk Performance</b>:
  Average Reads: ${diskRead} ms
  Average Writes: ${diskWrite} ms
```

Inspector

Step Name: Display

Inputs | Results | **Display** | Description | Advanced | Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Report

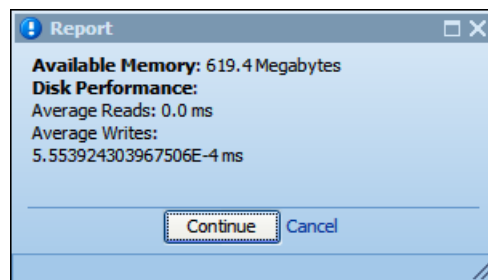
Prompt Width: 0 Height: 0

Prompt Text:

```
<b>Available Memory</b></b>: ${availableMemory} Megabytes
<b>Disk Performance</b></b>:
Average Reads: ${diskRead} ms
Average Writes: ${diskWrite} ms
```

6. Save your changes and test the flow.

The flow takes a few seconds to complete as it gathers system information. Use localhost for your host name. Your report should look like this:



Review

Subflows are a powerful way to simplify flow design and to build your own library of custom content. A subflow is treated as any other step when incorporated into a parent flow. The key is to remember that flow variables defined in a subflow are local to the subflow and not immediately available in the parent flow. You need to use results to create a flow output field that can then be picked up in the parent flow.

Exercise 2

Looping and Iteration

In this exercise you will build a flow that demonstrates looping. The flow generates a list of random numbers and then iteratively compiles the numbers to a list.

In Studio, solutions to this exercise are in My Ops Flows/Exercise Solutions/Advanced Authoring/Looping.

Objectives

By the end of this exercise you will be able to:

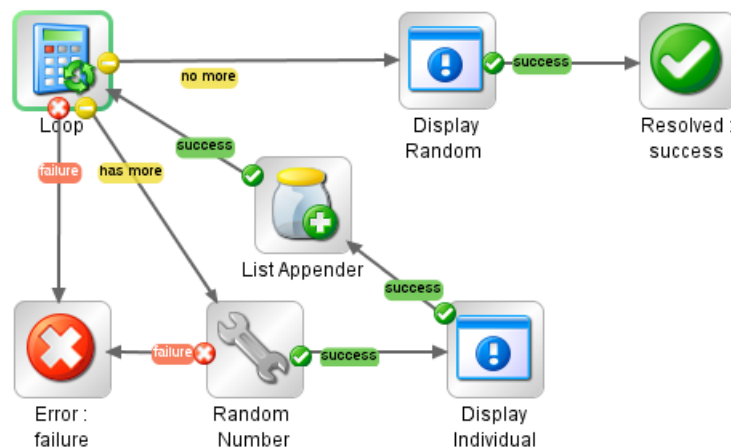
- Use looping to generate a list of random numbers
- Compile a list based on an iterative process
- Create a selection list based on an iterative process

Detailed Instructions

Task 1: Use Looping to generate random numbers

In this task you will build a flow that generates a specified number of random numbers. The random numbers are displayed individually and compiled into a list, which is finally displayed to the user.

1. Create a flow folder and flow.
 - a. Create a folder named Looping in My Ops Flows, and in that folder create a new flow named Generate Random Numbers.
 - b. Assemble the components of your flow as shown below. Loop is in Utility Operations/Looping, List Appender is in Utility Operations/Containers/Lists, and Random Number Generator is in Utility Operations. Display Random and Display Individual are renamed Display operations located in Utility Operations.



- c. Notice that the **has more** result on Loop defines a looping path for the flow. The flow takes the looping path as long as there are more numbers to loop through. Once looping is done, the flow continues down path defined by the **no more** response.
2. Configure inputs for each step.

Double click each step to configure inputs and results:

- **Loop**

Loops the number of times specified by count input.

- ♦ count: Constant Value of 10.
- ♦ reset: **false**

- **Random Number Generator**

Generates a random number between min and max.

Inputs tab:

- ♦ max: Constant value of 100
- ♦ min: Constant value of 1

Results tab:

- ♦ Add a randomNumber result based on the Result Field returnResult.

- **Display Individual Random Numbers**

Modify the Display tab to display `${randomNumber}`.

- **List Appender**

Compiles the list of random numbers in a flow variable named randomNumberList.

- ♦ keyName: Constant value **randomNumberList**
- ♦ Assign From and Assign To should be **Not Assigned**

Inspector

Step Name: List Appender

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > keyName

Name: keyName Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Use Constant

Assign to Variable: <not assigned>

Input Properties

Encrypted Required

Validation Format: <not validated>

Record Under: <run history>

'Otherwise: Use Constant' Configuration

Constant Value: randomNumberList

- ◆ resultText: Assign from and to variable **randomNumber**:

The screenshot shows the 'Inspector' window for the 'List Appender' step. The 'Inputs Summary' tab is selected, and the 'resultText' input is highlighted. The configuration for 'resultText' is as follows:

- Name: resultText
- Input Type: Single Value
- Input Data Flow:
 - Assign from Variable: randomNumber
 - Otherwise: Prompt User
 - Assign to Variable: randomNumber
- 'Otherwise: Prompt User' Configuration:
 - Prompt For: ☒ Text ☐ Selection
 - User Message:
 - String to Append:

- ◆ delimiter: Use a Constant Value of "
" which displays each number on a separate line:

The screenshot shows the 'Inspector' window for the 'List Appender' step. The 'Inputs Summary' tab is selected, and the 'delimiter' input is highlighted. The configuration for 'delimiter' is as follows:

- Name: delimiter
- Input Type: Single Value
- Input Data Flow:
 - Assign from Variable: <not assigned>
 - Otherwise: Use Constant
 - Assign to Variable: <not assigned>
- 'Otherwise: Use Constant' Configuration:
 - Constant Value:

- Display Random Number List
 - ◆ Modify the Display tab to display **Random number list:**
\${randomNumberList}.

3. Test the flow.

Save your changes and run the flow in the Studio debugger.

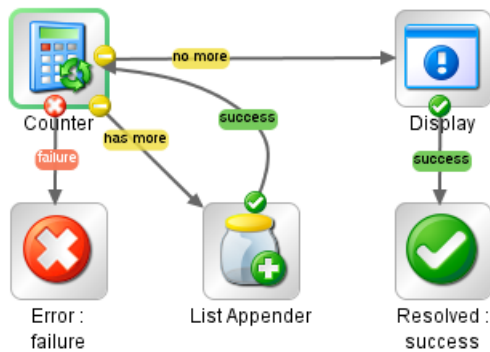
When you run the flow you should see pop-up windows showing you the individual random numbers that are generated, followed by a pop-up window showing you the compiled list of random numbers.

This example in this task simply compiles a numbered list according to the min, max, and increment values you provide.

Task 2: Use Counter to compile a list

The Counter operation is similar to loop but is more flexible. It allows you to specify a starting and ending number (**from** and **to** respectively), as well as an increment value. For example you can increment the count by 1, 2, 5, 10, etc. Incrementing by 0 or a negative number produces unwanted results.

1. In your Looping folder create a new flow named **Counter**.
2. Assemble the components for your flow.



Counter is in Utility Operations/Looping, List Appender is in Utility Operations/Containers/Lists, and Display is in Utility Operations.

3. Configure inputs for each step.

Double-click the step to configure inputs.

- **Counter:**
Set incrementBy to **Prompt User**. No other changes are needed.
- **List Appender:**
 - ◆ keyName: Constant value of **numberList**. This is the flow variable that stores the compiled list.
 - ◆ resultText: **Result of previous step**.
 - ◆ delimiter: **
**. Use an HTML break so the list will be nicely formatted when displayed in the Display step.
- **Display**
Modify the Display tab to display **\${numberList}**.

4. Test the flow.

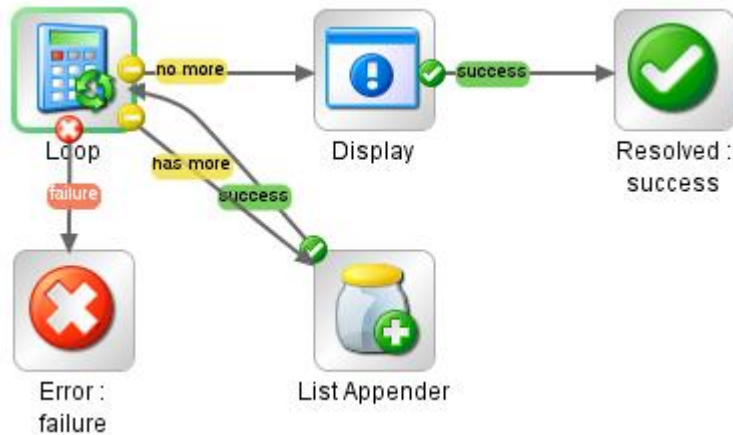
Save your work and run the flow in the debugger. Enter the starting and ending numbers, along with an increment value of 1 or more. You will see the incremented list of numbers.

Task 3: Use a Selection List

In this task you will author a flow that presents the user with a selection list based on a compiled list.

1. Build the Selection List flow as shown.

There are no changes to the Loop step – it will simply prompt the user to enter a number to count to.



2. Modify List Appender inputs:

- keyName: Constant value of **compiledList**
- resultText: Result of previous step
- delimiter: , (comma)

Assign To Input	Required	Type	From
keyName	<input checked="" type="checkbox"/>	Single Value	Value: compiledList
resultText	<input checked="" type="checkbox"/>	Single Value	Result of previous step
delimiter	<input type="checkbox"/>	Single Value	Value: ,

3. Modify Display input to present the selection list to the user

Step Name: Display

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > field1

Name: field1 Input Type: Single Value

Input Data Flow

Assign from Variable: field1

Otherwise: Prompt User

Assign to Variable: field1

Input Properties

☐ Encrypted ☐ Required

Validation Format: <not validated>

Record Under: <run history>

'Otherwise: Prompt User' Configuration

Prompt For: ☐ Text ☒ Selection

List Source: Flow Variable

Named: compiledList

Source Delimiter: ,

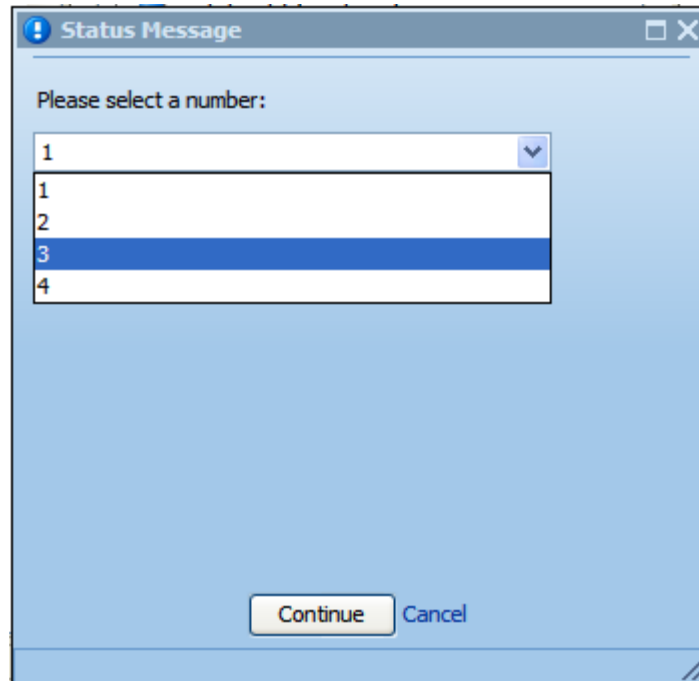
User Message:

Please select a number:

- Modify the field1 input:
 - ◆ Prompt For: **Selection**

- ♦ List Source: **Flow Variable**
 - ♦ Named: **compiledList**
 - ♦ delimiter: ,
 - ♦ User message: **Please select a number**
4. Save changes and run the flow in the Studio debugger.

You are prompted to enter a number to count to. Then you will see a selection list based on the number you entered:



Exercise 3

Filtering Flow Data

In this exercise you will learn how to filter data contained in a flow variable to extract only the information you need. The filtered data becomes the new value for the flow variable. You will use pre-defined Operations Orchestration filters, which are available in the Filter Editor. You will also use a Regular Expression to filter complex data.

In Studio, solutions to this exercise are in My Ops Flows/Exercise Solutions/Advanced Authoring/Filtering Data.

Objectives

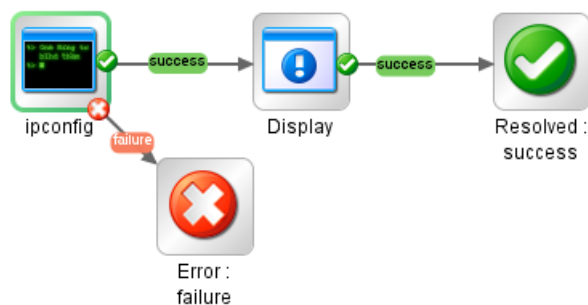
By the end of this exercise you will be able to:

- Use a number of filters to extract data from a flow variable
- Use a regular expression to filter data
- Work with Scriptlets

Detailed Instructions

Task 1: Build a flow that displays your IP address

In this task you will create a flow that determines your local IP address and displays it to you. The ipconfig step uses three filters that extract the IP address from ipconfig output.




1. Create a flow folder.

To begin, create a new flow folder in My Ops Flows and Name it **Filtering Data**.

2. Create the ipconfig operation.

In your new folder, create a new cmd operation that executes an ipconfig command on your system (right-click the folder, select New Operation, select cmd as the type):

Name: 

UUID: 4f76cf45-55e1-4d2a-b3e2-394fcc6d8265 Version: 2 (0)

Inputs Summary

cmd fields

Command:

Argument:

Environment:

Path:

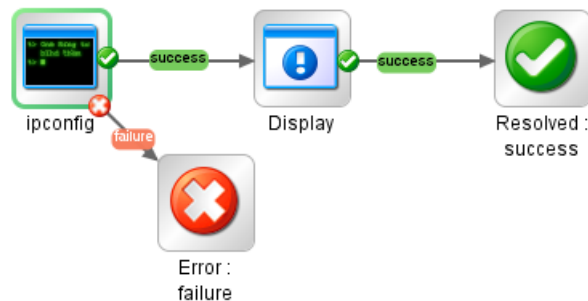
Working Directory:

Inputs

Input	Required	Type	Template
-------	----------	------	----------

3. Build your flow.

Create a new flow named Get Local IP Address, then assemble the components as shown. The ipconfig step is based on the ipconfig operation you created in the previous step. Use the Display step in Utility Operations/Display.



4. Add an ipAddress result to the ipconfig step.

In the ipconfig step, add a result named ipAddress based on the Output String.

Inspector

Step Name:

Step Results > ipAddress

Name	From	Assign To	Assignment Action	Filters
ipAddress	Result Field: Output String	Flow Variable	OVERWRITE	3 filters

5. Add filters to obtain the IP address.

Use the Filter Editor (yellow arrow at the right) to add the three filters in this order:

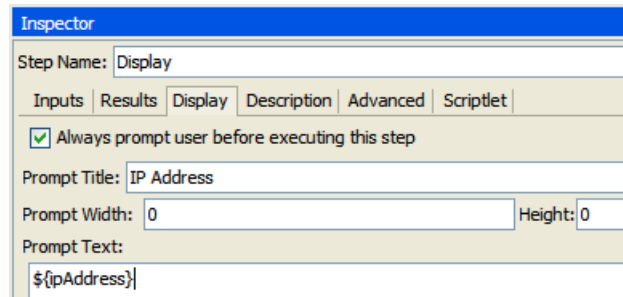
- Select Line (First Line Containing IP Address)

- Strip All Characters Up To And Including ‘.’
- Strip Whitespace

Test each filter as you go to see how they work. Use Quick Command to execute an ipconfig command for testing.

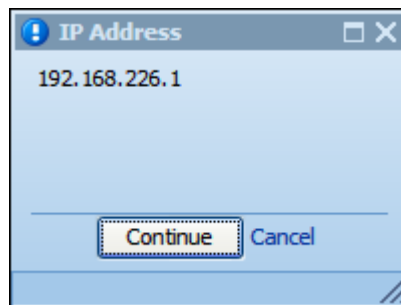
6. Display the ipAddress in the Display Step.

Display the ipAddress flow variable in the Display step:



7. Test the flow.

Close all editors, save your work, and test the flow. You should see a pop-up window with your IP address.



Task 2: Use a Regular Expression to filter out the IP address

In this task you will use a Regular Expression to replace the three filters you used to extract the IP address.

You can duplicate your existing flow if you wish and make changes to that – right-click the flow and select Edit/Duplicate. Then rename the duplicated flow Get Local IP Address – Regex.

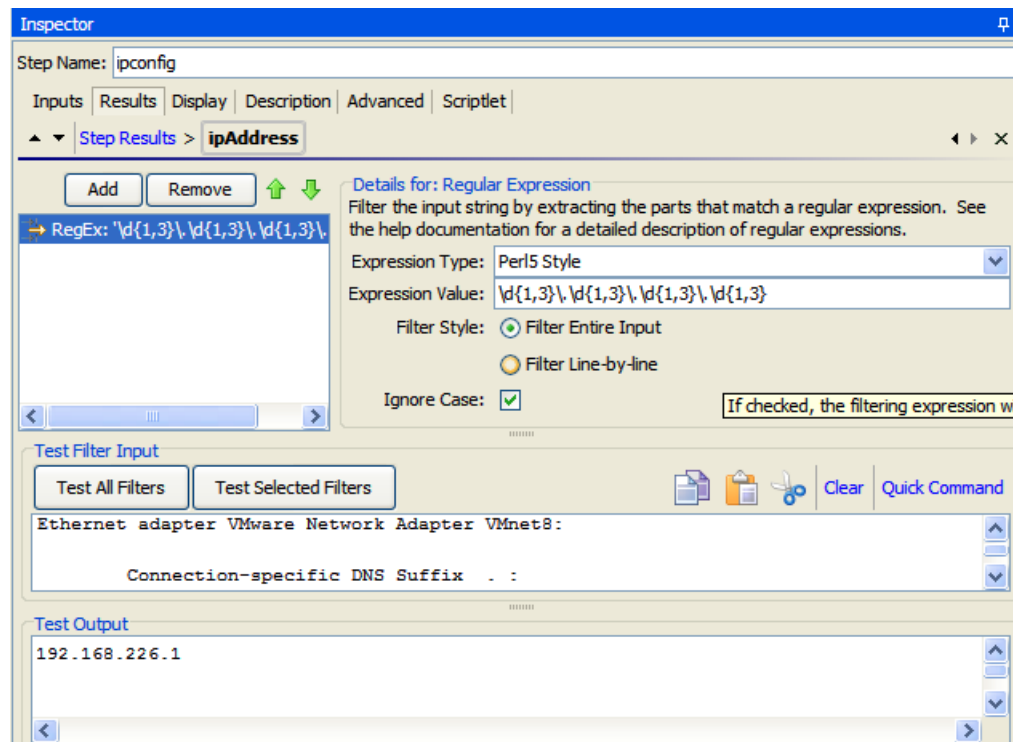
1. In the ipconfig step, replace the three filters with one Regular Expression.

Open the ipconfig step and select the Results tab, then open the filter editor for the ipAddress result.

Remove all three of the existing filters by selecting the filter and clicking Remove. Then add a new Regular Expression filter and add this expression:

```
\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}
```

Use Quick Command to execute an ipconfig command and use that for testing your expression.



2. Test the flow.

Close all editors, save your work and test the flow – you should see your IP address in a pop-up window.

Task 3: Work with Scriptlets

In this exercise you will create a simple flow that uses a scriptlet to convert a number of bytes into Kbytes and presents the result to the user.

1. Create a new folder in your my Ops Flows folder named Scriptlet, then create a new flow in that folder named Convert Flow.
2. Locate the Display operation from the Utility Operations folder and build the flow as shown. Rename the Display step Convert:



3. Open the Inspector window for Convert by double-clicking and set the field1 input to prompt the user. For the Prompt label, enter something like "Enter the number of bytes to convert:"

4. Click the Scriptlet tab and enter the following JavaScript code. The scriptlet takes the number entered into the field1 flow variable and divides by 1024 to return a value in Kbytes in the scriptletResult flow variable.


```
if (field1.length == 0) scriptletResult = 0
else scriptletResult =
    java.lang.Math.round(java.lang.Double.parseDouble(field1)/1024)
scriptletContext.put("scriptletResult",scriptletResult);
```

5. When done entering the scriptlet click Check Script. If no errors are found, close the Inspector window. If you see errors, correct them before proceeding.
6. Open the Step Inspector for the Resolved: success step, then open the display tab. Enter the following into the Display tab. Make sure to select Always prompt user and enter a Prompt Title:

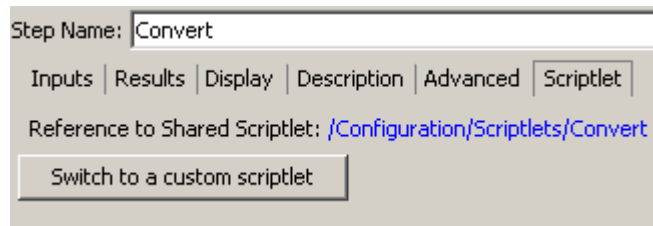
`${field1}` bytes equals `${scriptletResult}` Kbytes.

7. Save your flow and run it in the debugger. Enter a number and observe the output.

Save and re-use the scriptlet

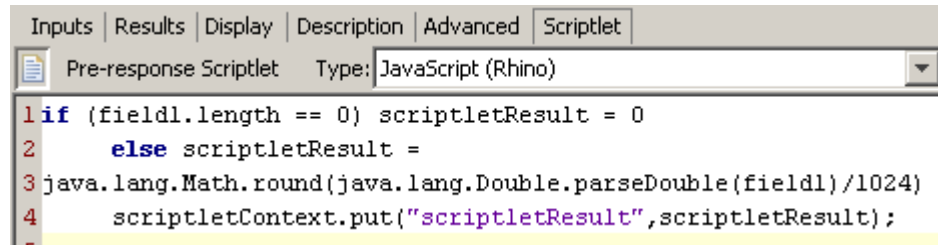
8. Expand the Scriptlets folder in Configuration.
9. Go back to the Scriptlet tab of your Convert step. Copy the scriptlet by clicking and dragging the icon next to Pre Response Scriptlet into your Scriptlets folder.
 
10. Once copied into your Scriptlets folder, the new scriptlet is named with its UUID and other information. Rename it Convert.
11. To re-use the saved scriptlet, simply drag it from the Scriptlets folder onto the Pre-response Scriptlet icon.

When the saved scriptlet is used in the step, the Scriptlet tab will look like this:



The screenshot shows a dialog box with the title "Step Name: Convert". Below the title, there are several tabs: "Inputs", "Results", "Display", "Description", "Advanced", and "Scriptlet". The "Scriptlet" tab is currently selected. Under the "Scriptlet" tab, there is a text field labeled "Reference to Shared Scriptlet:" containing the path [/Configuration/Scriptlets/Convert](#). Below this text field is a button labeled "Switch to a custom scriptlet".

Clicking **Switch to a custom scriptlet** opens your scriptlet editor, and that scriptlet will be used instead of the saved scriptlet:



The screenshot shows a scriptlet editor window. At the top, there are tabs: "Inputs", "Results", "Display", "Description", "Advanced", and "Scriptlet". The "Scriptlet" tab is selected. Below the tabs, there is a section labeled "Pre-response Scriptlet" with a "Type:" dropdown menu set to "JavaScript (Rhino)". Below the dropdown, there is a text area containing the following JavaScript code:

```
1 if (field1.length == 0) scriptletResult = 0
2   else scriptletResult =
3 java.lang.Math.round(java.lang.Double.parseDouble(field1)/1024)
4   scriptletContext.put("scriptletResult",scriptletResult);
```


Exercise 4

Parallel Processing

In this exercise you will work with two types of parallel processing, multi-instance and parallel-split steps. In Task 1 you modify a flow to use a multi-instance step, and in Task 2 you build a flow that executes along two parallel paths.

In Studio, solutions to this exercise are in My Ops Flows/Exercise Solutions/Advanced Authoring/Parallel Processing.

Objectives

By the end of this exercise you will be able to:

- Use a multi-instance step
- Use a parallel-split step

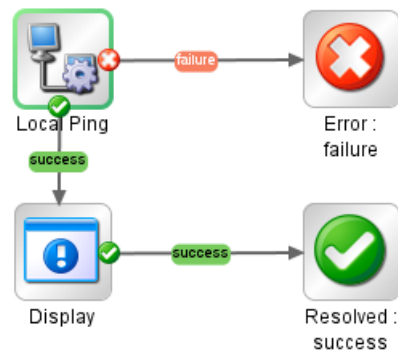
Detailed Instructions

Task 1: Work with a Multi-Instance step.

In this task you will build a small flow that pings a single host and returns the ping output. In the next task you will toggle the Local Ping step to the multi-instance state.

1. Build a Ping flow that uses a single instance of Local Ping.

In My Ops Flows, create a folder named **Parallel Processing**. In that folder, create a new flow called **Multi-Instance**. Assemble the components of the flow:



Local Ping is in Operations/Network, and Display is in Utility Operations.

2. Configure inputs for Local Ping.

In the Step Inspector for Local Ping, remove the inputs for packetCount and packetSize – they are not used.

Then set the value of targetHost to a constant value of **localhost**.

Inspector

Step Name: Local Ping

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Inputs Summary > targetHost

Name: targetHost Input Type: List of Values

Input Data Flow

Assign from Variable: targetHost

Otherwise: Use Constant

Assign to Variable: targetHost

'Otherwise: Use Constant' Configuration

Constant Value: localhost

3. Add a result to Local Ping.

Add a returnResult based on Result Field: returnResult.

Step Name: Local Ping

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Step Results

Add Result Remove Result

Name	From	Assign To	Assignment Action	Filters
returnResult	Result Field: returnResult	Flow Variable	OVERWRITE	No Filters

When done, save changes and close the Local Ping Step Inspector.

4. Display the Local Ping returnResult in the Display step.

Open the Display tab and add the following Prompt Text to display the ping output. Note the use of HTML tags to format the Display output:

**\${targetHost};
\${returnResult}**

Step Name: Display

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

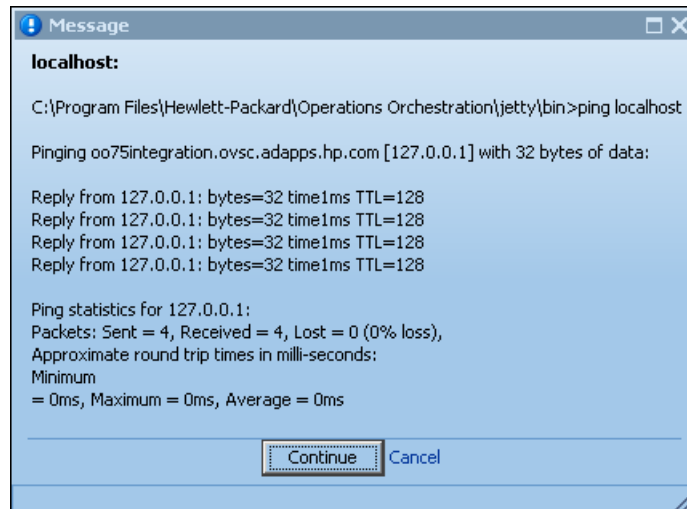
Prompt Title: Message

Prompt Width: 0 Height: 0

Prompt Text: \${targetHost};
\${returnResult}

5. Test the flow.

You should see the ping output in a pop-up window.



Task 2: Create a Multi-Instance Step

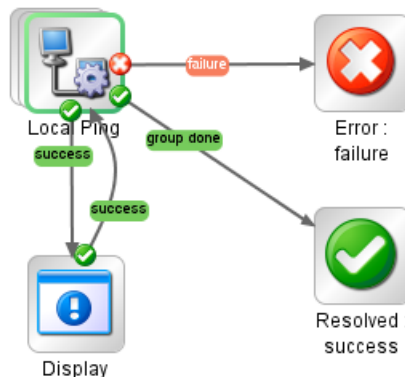
NOTE: Because of the security configuration on the online learning environment, you can only use the hosts and IPs as shown in the steps below. If the IP addresses provided do not work, substitute **localhost** for the IP address.

1. Toggle Local Ping to Multi-Instance.

Right-click Local Ping and select Toggle Multi-Instance.

2. Re-wire the flow to handle the Group Done response.

Wire Display back to Local Ping, then wire the new Group Done response on Local Ping to Resolved: Success.



3. Add multiple ping targets to Local Ping.

In the Local Ping Step Inspector, open the input editor for targetHost and add a comma-separated list of hosts to ping as an Input Type of List of Values, using a comma (,) as the delimiter. For example:

localhost,156.152.46.88,156.152.46.87

Step Name: Local Ping

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > targetHost

Name: targetHost Input Type: List of Values

Input Data Flow

Assign from Variable: targetHost

Otherwise: Use Constant

Assign to Variable: targetHost

'Otherwise: Use Constant' Configuration

Constant Value: localhost,156.152.46.88,156.152.46.87

Input Properties

Input Delimiter: , ☐ Encrypted ☒ Required

Save changes and close the Local Ping Step Inspector when done.

4. Test the flow in Studio and in Central.

Test the flow in Studio and observe the output.

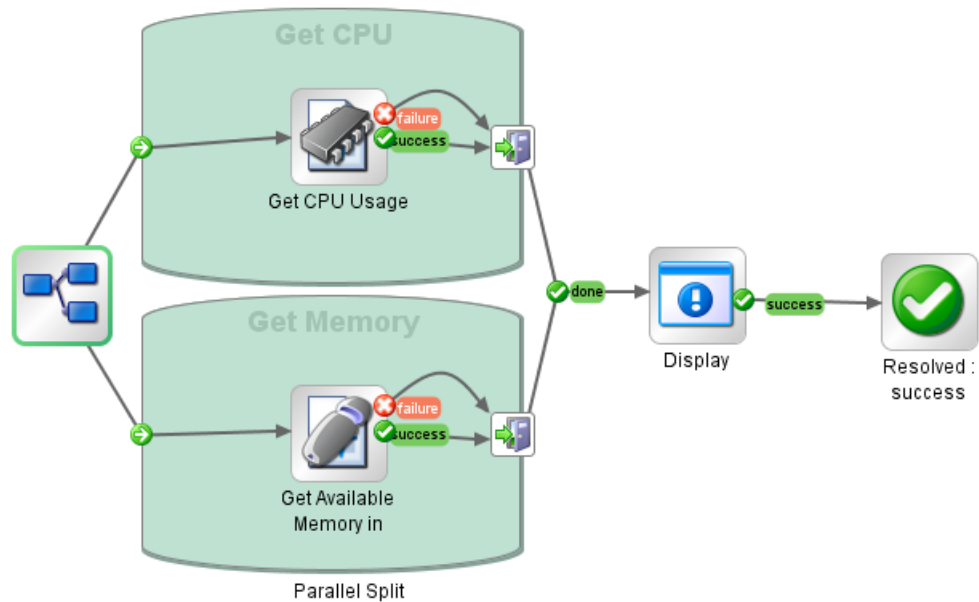
Then log into Central and test the flow. Observe any differences between running the flow in Studio. Also look at the report for the flow run to see how multi-instance steps are represented in the run report.

Task 3: Work With Parallel-Split Steps

In this task you will build a flow that uses a two-lane Parallel-Split step.

1. In your Parallel Processing folder, create a new flow named **Parallel-Split**.

Assemble the components of your flow as shown. The Parallel Split is available in the Step Palette.



Get CPU Usage and Get Available Memory are in Operations/Operating Systems/Windows Management/Performance Counters/CPU and Memory respectively.

Rename the lanes as shown. To rename, right-click the lane and select Rename:

- Lane 1: Get CPU
- Lane 2: Get Memory

2. Configure Inputs and Results for Get CPU Usage.

Inputs:

Set the host input to a Constant Value of **localhost**, and remove the user and password inputs.

Step Name: Get CPU Usage

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
host	<input checked="" type="checkbox"/>	Single Value	Value: localhost
object	<input checked="" type="checkbox"/>	Single Value	Value: processor
counter	<input checked="" type="checkbox"/>	Single Value	Value: % processor time
instance	<input type="checkbox"/>	Single Value	Value: _total
user	<input type="checkbox"/>	Single Value	Prompt User
password	<input type="checkbox"/>	Single Value	Prompt User

Results:

Add a result named **CPU** based on the Result Field **value**.

Step Name: Get CPU Usage

Inputs Results Display Description Advanced Scriptlet

Step Results

Add Result Remove Result ↑ ↓

Name	From	Assign To	Assignment Action	Filters
CPU	Result Field: value	Flow Variable	OVERWRITE	No Filters

3. Configure Inputs and Results for Get Available Memory in Megabytes.

Inputs:

Set the host input to a Constant Value of **localhost**, and remove the user and password inputs:

Step Name: Get Available Memory in Megabytes

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
host	<input checked="" type="checkbox"/>	Single Value	Value: localhost
object	<input checked="" type="checkbox"/>	Single Value	Value: memory
counter	<input checked="" type="checkbox"/>	Single Value	Value: available mbytes
instance	<input type="checkbox"/>	Single Value	Value:
user	<input type="checkbox"/>	Single Value	Prompt User
password	<input type="checkbox"/>	Single Value	Prompt User

Results:

Add a result named **availableMemory** based on the Result Field **value**:

Step Name: Get Available Memory in Megabytes

Inputs Results Display Description Advanced Scriptlet

Step Results

Add Result Remove Result ↑ ↓

Name	From	Assign To	Assignment Action	Filters
availableMemory	Result Field: value	Flow Variable	OVERWRITE	No Filters

- Configure the Display step.

Open the Display tab to display the ping output for the three hosts. Note the use of HTML tags to format the display output.

CPU Usage: ` ${CPU}`

Available Memory: ` ${availableMemory}`

Step Name: Display

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Message

Prompt Width: 0 Height: 0

Prompt Text:

```
<b>CPU Usage:</b> ${CPU}
<b>Available Memory:</b> ${availableMemory}
```

- Test the flow in Studio and Central.

You should see a CPU and Available Memory report in a pop-up window.

Observe any differences in running the flow in Studio and Central.

Review

In this exercise you learned how to use two forms of parallel processing, Multi-Instance and Parallel-Split. In the next exercise you will learn how to use Responses, Rules, and Transitions to control flow execution.

Exercise 5

Using Responses, Rules, and Transitions Control Flow Execution

This exercise demonstrates various ways of manually and automatically controlling flow execution. You will also write a flow that hands-off execution at different steps.

In Studio, solutions to this exercise are in My Ops Flows/Exercise Solutions/Advanced Authoring/Responses.

Objectives

By the end of this exercise you will be able to:

- Develop a flow that allows you to manually select an execution path
- Use Responses to select an execution path based on a Selection List stored in the Configuration folder
- Use transitions to control which groups can execute steps in a flow
- Use a system evaluator to control flow execution based on filtered data

Detailed Instructions

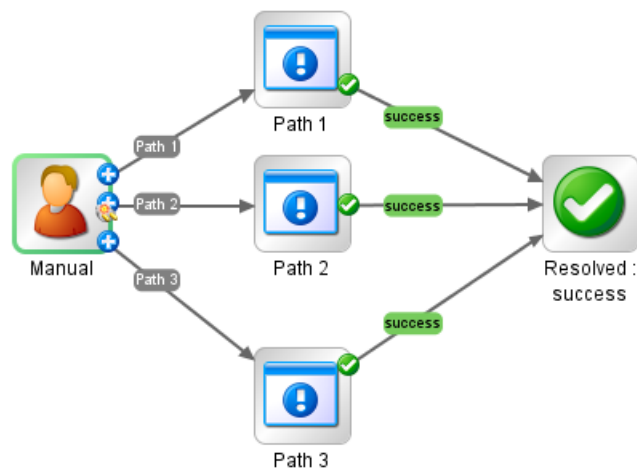
Task 1: Manually direct flow execution

In this task you will build the flow shown in step 1 below. Carefully follow the steps – the Manual operation used in this flow has a unique user interface.

1. In My Ops Flows, create a new folder named **Responses** and in that folder create a new flow named **Manual Flow Direction**.
2. Assemble the components of the Manual Flow Direction flow as shown but do not wire the steps together yet. The three Path steps are renamed Display operations. The Manual and Display operations are in Utility Operations. To rename the Display steps, right-click and select Rename, then enter your new name – Path 1, Path 2, and Path 3 as shown.



- To wire the steps together, click the "Plug" icon on Manual and drag it to each of the Path steps. Notice that the transition takes the name of the step it connects to. Do this for each of the three Path steps, then wire the Path steps to the Resolved: Success step. Your flow should look like this:



- For each Path step, modify the Display tab with a message: **You selected Path 1**. Repeat for Path 2 and Path 3. The example below shows Path 1:

Inspector	
Step Name: Path 1	
Inputs Results Display Description Advanced Scriptlet	
<input checked="" type="checkbox"/> Always prompt user before executing this step	
Prompt Title: Path 1 Message	
Prompt Width: 0	Height: 0
Prompt Text:	
You selected Path 1!	

- Run the flow in the Studio debugger. You will be prompted for Path 1, Path 2, and Path 3. The correct message will appear on your screen depending on which menu item you select.

Task 2: Use Responses and Rules to direct flow execution

In this task you will add responses to an operation and then direct flow execution down a specified path based on the response the user selects. This flow also introduces the concept of modifying the properties of an operation and creating a Selection List stored in your Configuration folder.

1. Create a selection list for use in your flow.

In the Library, expand Configuration and right-click Selection Lists, then select New and name the new selection **Responses Exercise**.

2. Add three selection items and name them **Case 1** through **Case 3**, then add one item named **Other** for a total of six selection items. Click the Add button and double-click in the highlighted field to enter the values item names. Save the list and check it in when you are done.


The screenshot shows a configuration window for a Selection List named 'Responses Exercise'. It includes fields for Name, UUID, Version, and Description. Below these is a table with a header 'Value' containing three rows: 'Case 1', 'Case 2', and 'Case 3'. 'Add' and 'Remove' buttons are located to the right of the table.

Value
Case 1
Case 2
Case 3

3. In your Responses folder, create the Switch operation.

Copy the Display operation (Utility Operations/Display) into your Responses folder and rename it **Switch**. To copy, right-click the operation and select Edit/Copy, then right-click your target folder and select Edit/Paste.

Note: To change the icon, double-click the operation in the Library, then drag an icon from the Icons tab at the right of your Studio window onto the icon displayed in the Operation Properties. (Right-click the icon to reset).

The icon used in this exercise is Others/cube_molecule .

4. Add Responses.

To edit the properties of the Switch operation, double-click it in the Library.

In the Properties window, open the Responses tab and add the responses named **Case 1** through **Case 5**, as shown below. Rename the success response **Default**, and set it as the default. Then set the icon for each of the Case responses to Diagnosed (white plus on blue). You will define rules for the responses in the next step.

Name: Switch

UUID: af7486ee-c9fc-442a-afa0-bdeaf6006bc3 Version: 5.4 (05/23/11 13:00 admin)

Assign Categories:

Inputs | Outputs | Responses | Description | Scriptlet

Responses Summary

Add Response Remove Response

Response	Default	On-Fail	Type	Rules
Default	<input checked="" type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	No Rules Defined
Case 1	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	1 Rule [Source: Field 1, No Filters, Exact Match: Case 1]
Case 2	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	1 Rule [Source: Field 1, No Filters, Exact Match: Case 2]
Case 3	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	1 Rule [Source: Field 1, No Filters, Exact Match: Case 3]
Failure	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	No Rules Defined

5. Define rules for responses.

For each Case response, open the filter editor (yellow arrow at the right) and define a rule that applies to Field 1, with an exact match of the name of the response – Case 1 through Case 3. Here's an example using Case 1:

Inputs | Outputs | Responses | Description | Scriptlet

Responses Summary > Case 1

The 'Case 1' response will be selected when all of the following rules are true

Add Remove

Apply Rule To Field	Rule Type	Rule Text
Field 1	Exact Match	Case 1

When completed your Responses tab should look like this. When done, save changes and close the Properties window.

Name: Switch

UUID: af7486ee-c9fc-442a-afa0-bdeaf6006bc3 Version: 5.4 (05/23/11 13:00 admin)

Assign Categories:

Inputs | Outputs | Responses | Description | Scriptlet

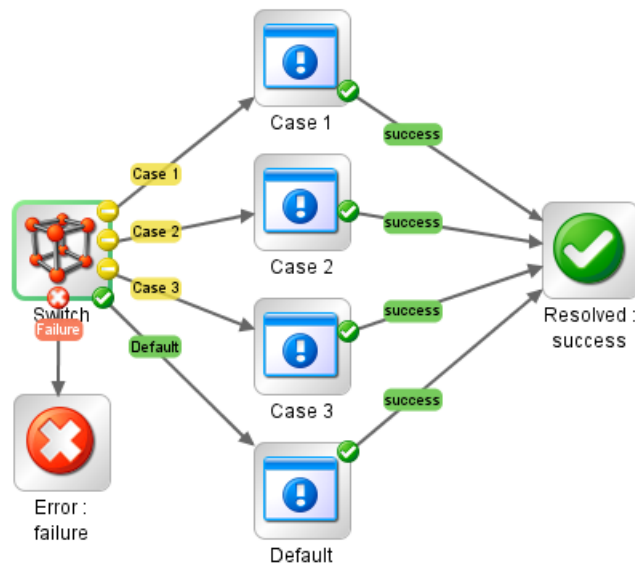
Responses Summary

Add Response Remove Response

Response	Default	On-Fail	Type	Rules
Default	<input checked="" type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	No Rules Defined
Case 1	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	1 Rule [Source: Field 1, No Filters, Exact Match: Case 1]
Case 2	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	1 Rule [Source: Field 1, No Filters, Exact Match: Case 2]
Case 3	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	1 Rule [Source: Field 1, No Filters, Exact Match: Case 3]
Failure	<input type="checkbox"/>	<input type="checkbox"/>	✓ + - ✗	No Rules Defined

6. Assemble your flow.

Create a new flow named **OO Switch** and assemble the components as shown below. The Switch step is Switch operation you copied and modified. Each of the Case steps are renamed Display operations (located in Utility Operations).



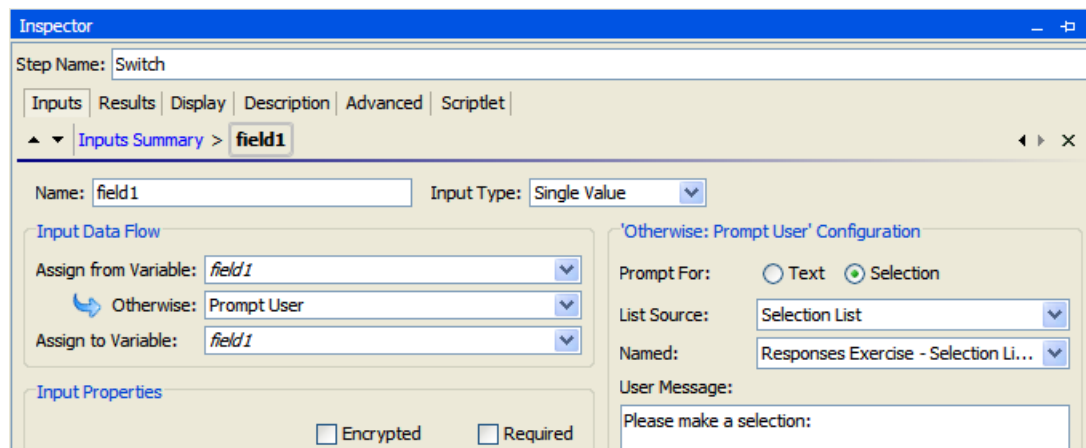
Hints – There are a lot of responses on the Switch step. To pick one in particular, just hover the mouse over it for a moment – the response name will highlight in red. Make sure the correct response goes to the correct Case step or you will get unexpected results.

Also, if you run out of room on your design panel, open the View Options and shrink the display so the entire flow is visible.

7. Use your selection list as the input for Switch.

Open the Step Inspector for the Switch step by double-clicking and then open the input editor (yellow arrow) for field1. Make the following changes:

- Input Type: **Single Value**
- Prompt For: **Selection**
- List Source: **Selection List**
- Named: **Responses Exercise**
- User Message: **Please make a selection:**



Save your changes and close the Step Inspector.

8. Add a user messages to each of the Case steps.

Open the step inspector for each of the Case 1 through Case 3 steps and display a message indicating which selection was made – for example, “You selected Case 1”.

Case 1 step:

Step Name: Case 1

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Message

Prompt Width: 0 Height: 0

Prompt Text:

You selected Case 1.

Case 2 Step:

Step Name: Case 2

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Message

Prompt Width: 0 Height: 0

Prompt Text:

You selected Case 2.

Case 3 Step:

Step Name: Case 3

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Message

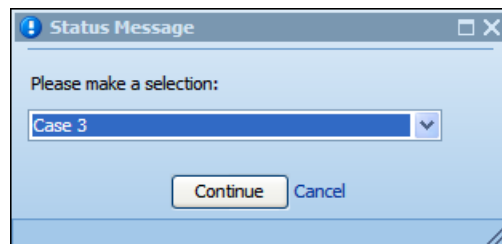
Prompt Width: 0 Height: 0

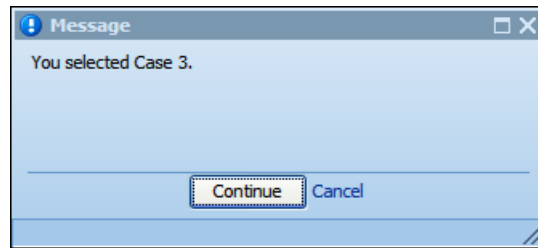
Prompt Text:

You selected Case 3.

9. Test the flow.

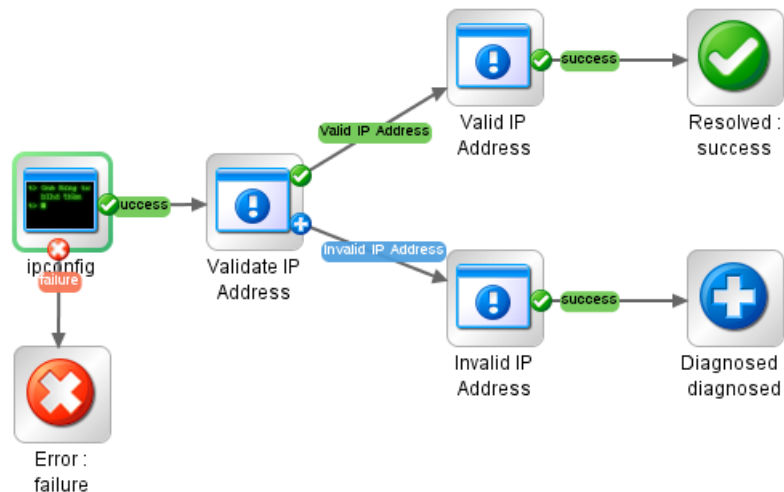
Save your changes, then run the flow in the debugger and make your selection from the pull-down menu. You should see the appropriate display message. In the example below, Case 3 was selected:



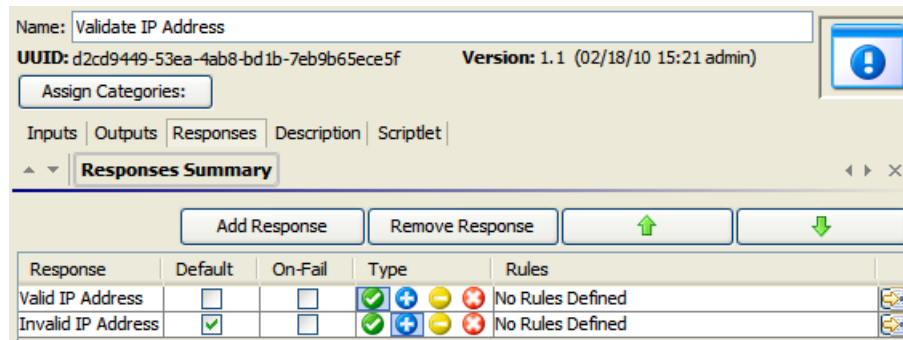


Task 3: Use Responses and a System Evaluator to Direct Flow Execution

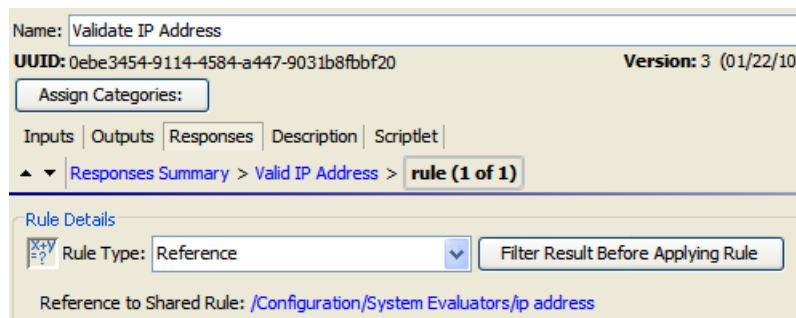
In this task you will use an IP Address System Evaluator, which is available in the Configuration folder, to validate an IP address. The System Evaluator is in the Validate IP Address step, which is based on a Display operation that has been modified with an additional response that uses the IP Address System Evaluator to determine the execution path of the flow.



1. Create the Validate IP Address operation.
Copy Utility Operations/Display, paste it into your folder. Rename it Validate IP Address.
2. Set up Responses for Validate IP Address.
 - a. Open the Properties window for Validate IP Address by double-clicking it in the Library, then select the Responses tab.
 - b. Rename the success response Valid IP Address, and add a new response named Invalid IP Address which will be the default response.
 - c. Change the icon on Invalid IP Address to Diagnosed (white plus).

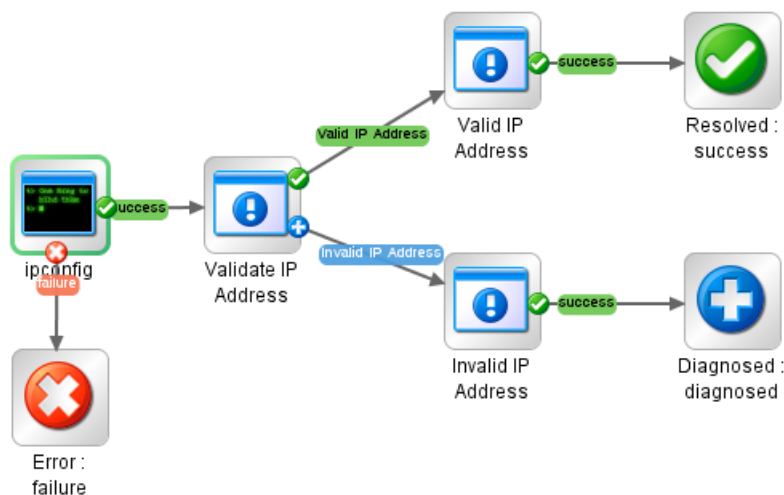


3. Define the rule for the Valid IP Address response.
 - a. Open the Rule Editor for Valid IP Address, then click Add to add a rule for Field 1.
 - b. Then click the Rule Editor for Field 1.
 - c. Expand the Configuration/System Evaluators folder and locate the ip address evaluator. Then click and drag the evaluator over the icon for Rule Details. The type changes to Reference. The editor will look like this:



Save your changes.

4. Add components to your flow as shown:



- **ipconfig:** Use the ipconfig operation you created earlier.
- **Validate IP Address:** use the operation you just modified.

- **Valid IP Address and Invalid IP Address:** These are simply Display operations (in Utility Operations/Display) that have been renamed.
5. In Validate IP Address, assign the IP address to validate.
- In the Step Inspector for Validate IP Address, open the input editor for field1 and assign the value for field1 from the ipAddress flow variable.

Inputs Summary > field1

Name: field1 Input Type: Single Value

Input Data Flow

Assign from Variable: ipAddress

Otherwise: Prompt User

Assign to Variable: field1

Input Properties

☐ Encrypted ☐ Required

6. Display the results in Validate IP Address and Invalid IP Address.
- Set up the Display tabs in the Valid IP Address and Invalid IP Address steps so they display a message telling you whether the IP address is valid or not.

Valid IP Address:

Inspector

Step Name: Valid IP Address

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Valid

Prompt Width: 0

Prompt Text: \${field1} is a valid IP address.

Invalid IP Address:

Inspector

Step Name: Invalid IP Address

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Invalid

Prompt Width: 0

Prompt Text: \${field1} is not a valid IP address.

7. Test the flow.

Save your changes and test the flow. It should get your IP address, validate it, and present the appropriate message.

You can force an invalid response to test whether the validation step is really working. In the Validate IP Address step, use the Input Editor for field 1 and change the value for Assign From Variable to the default value, field1. This will

force the step to prompt for a value. Save your changes and run the flow. When you are prompted to enter a value for field1, enter an invalid IP address or any random string of characters. You should see a message that it's an invalid IP address.

Exercise 6

Remote Action Service

In this exercise you will:

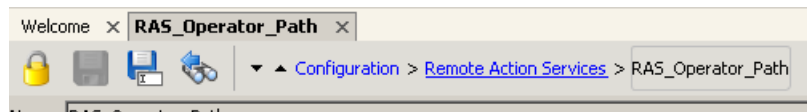
- View and monitor RASes in Studio and Central
- Add a RAS Reference in Studio
- Set up a RAS Override in Central
- Change the default RAS for an operation
- Install an iAction
- Create and use OO content from the RAS

Detailed Instructions

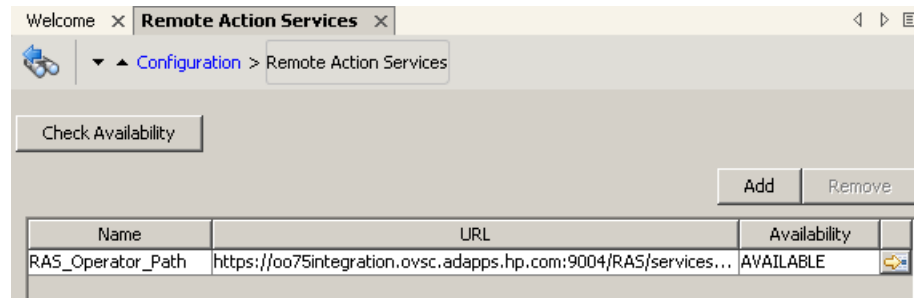
Task 1: View and monitor RASes in Studio and Central

Central and Studio allow you to monitor the available Remote Action Services on your system. In Studio you can create new RAS References. Both provide various ways of providing override values for defined RASes

1. View available RASes in Central.
 - a. Start Central and log in.
 - b. Go to Administration/Node Administration
 - c. The Remote Action Service (RAS) Infrastructure View shows the installed RASes on your network. Initially you have only one RAS, the default service that installs with Central with a reference of RAS_Operator_Path.
2. View RASes in Studio.
 - a. Start Studio and log in.
 - b. Expand Configuration/Remote Action Services
 - c. Double-click RAS_Operator_Path
 - d. To view a RAS summary screen, click Remote Action Services in the toolbar:



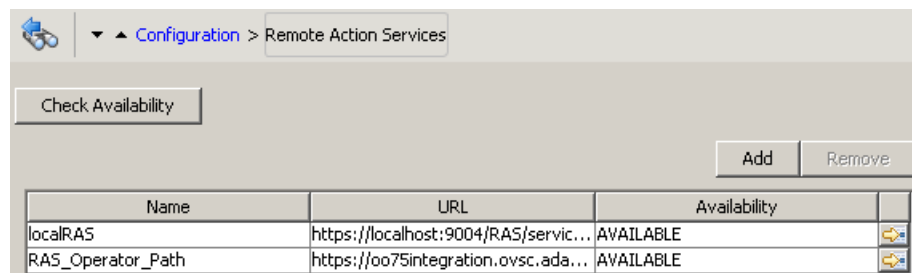
- e. The Remote Action Services tab shows you the RASes and their availability.



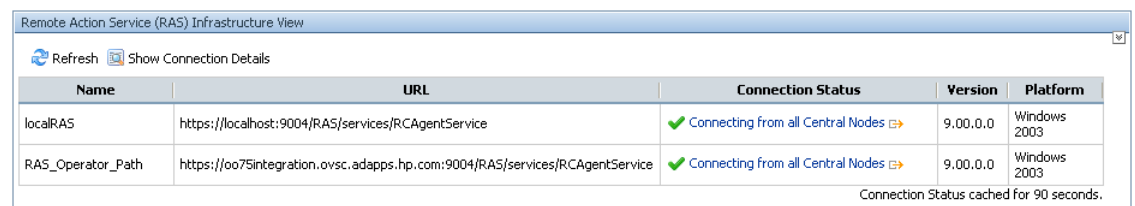
Task 2: Add a RAS Reference in Studio

You can add as many RAS References as you need. In this task you will add a new reference that points to your default Remote Action Service to illustrate the process of adding and monitoring RASes. If you have a second RAS available on your network, you can add a reference to that as well.

1. In the Remote Action Services tab, click Add or right-click your Remote Action Services folder and select New.
2. Name your RAS Reference.
3. Add the RAS URL and an optional description:
<https://localhost:9004/RAS/services/RCAgentService>
4. Save and check in the RAS Reference
5. Click the Remote Action Services link in the toolbar to view the status of your RASes:



6. Log into Central and open the Administration/Node Administration tab to view your deployed Remote Action Services:



Task 3: Set up a RAS Override in Central

Using a RAS Override means temporarily supplying a new RAS URL to an existing named RAS reference. Once you set up the override RAS in Central, operations

that refer to the named RAS References will be directed instead to the override URL. The override remains in effect until you remove it.

1. In Central, go to Administration/Runtime Environment/Remote Action Service (RAS) Overrides.
2. Click the Add button.
3. From the pull-down menu select RAS Operator Path, then enter the override URL:

<https://localhost:9004/RAS/services/RCAgentService>

4. To test, run the Windows Health Check flow which makes extensive use of RAS operations. In the Flow Library, navigate to Accelerator Packs/Operating Systems/Windows and run the flow. For host, enter **localhost**, and for Notification Option select **None**. No other inputs are needed. The flow should run normally.
5. When done, go back to Administration/Runtime Environment/Remote Action Service (RAS) Overrides and remove the override.

Task 4: Change the default RAS for an operation


In this task you will modify the properties of an operation to use a different RAS Reference.

1. Create a working folder in My Ops Flows.
2. Copy the Ping operation in Operations/Network and paste it into your working folder.
3. Double-click the copied Ping operation to view it's Properties editor.
4. In Configuration, expand Remote Action Services. Select the localRAS reference you created earlier and drag it onto the globe icon in the Properties editor. The RAS reference will change.

RAS Operation fields

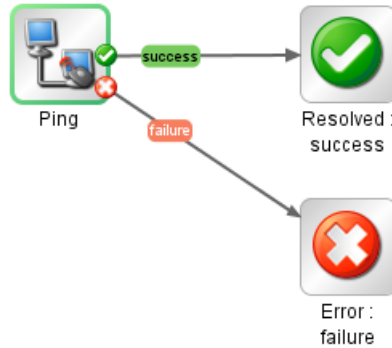
Action Class:

Archive:

RAS: 

Override RAS:

- Test the modified operation in a flow. In the Step Inspector for the Ping step, remove all the optional inputs, then run the flow using localhost as the targetHost and local as the protocol.



Ping step inputs:

Step Name:

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input

Assign To Input	Required	Type	From
targetHost	<input checked="" type="checkbox"/>	Single Value	Prompt User
protocol	<input checked="" type="checkbox"/>	Single Value	Prompt User
packetCount	<input type="checkbox"/>	Not Assigned	No Assignment
packetSize	<input type="checkbox"/>	Not Assigned	No Assignment
hostname	<input type="checkbox"/>	Not Assigned	No Assignment
username	<input type="checkbox"/>	Not Assigned	No Assignment
password	<input type="checkbox"/>	Not Assigned	No Assignment
keyFile	<input type="checkbox"/>	Not Assigned	No Assignment
usernamePrompt	<input type="checkbox"/>	Not Assigned	No Assignment
passwordPrompt	<input type="checkbox"/>	Not Assigned	No Assignment
remoteOS	<input type="checkbox"/>	Not Assigned	No Assignment

Task 5: Specify an override RAS in a step

Provide a RAS Reference to the `${overrideJRAS}` input in the Ping flow you created in the previous step.

- In the flow you created in the previous task, open the Step Inspector for the Ping step and add an input named `overrideJRAS`.

Note – to determine the input to use, view the operation properties and use the override RAS as shown.

- Open the input editor for `overrideJRAS` and provide the name of the RAS reference to use as a Constant Value. Use `RAS_Operator_Path`

Step Name: Ping

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
targetHost	<input checked="" type="checkbox"/>	Single Value	Prompt User
protocol	<input checked="" type="checkbox"/>	Single Value	Prompt User
overrideJRAS	<input type="checkbox"/>	Single Value	Value: RAS_Operator_Path
overrideJRAS	<input type="checkbox"/>	Not Assigned	No Assignment

3. Save changes and test the flow – it should work normally.

Task 6: Install an iAction

Your instructor will provide an iAction JAR file to use in this step.

1. Retrieve the EchoAction.jar file provided by your instructor and copy it to:
C:\Program Files\Hewlett-Packard\Operations
Orchestration\RAS\Java\Default\repository
2. Restart the RSJRAS service. In the Services control panel, locate the RSJRAS service and click the Restart link.

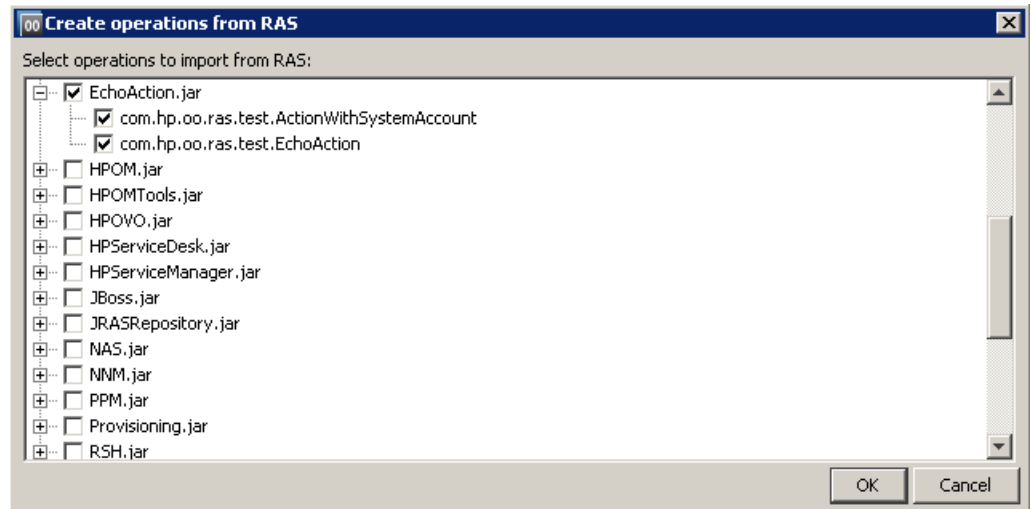
To open the Services control panel, open a command window and enter services.msc, or click the Services quick launch button at the bottom of your Surgient VM window.

Once the service restarts, you can create operations from the imported iAction and use them in a flow.

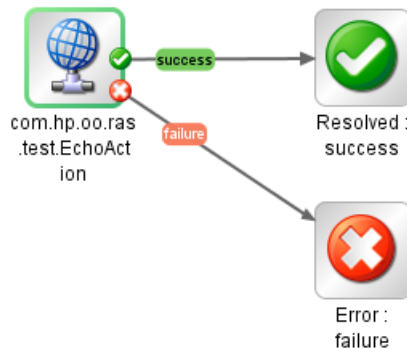
Task 7: Import operations from the RAS

In Studio you can create operations from iActions deployed to the RAS.

1. In Studio, create a working folder in My Ops Flows named Echo Action.
2. Select the Echo Action folder, then open the File menu and select Create Operations from RAS... In the pull-down menu select RAS_Operator_Path.
3. In the import window, locate EchoAction.jar and select it. Then click OK.



4. Open your Echo Action folder, and there you will find the imported operations in a subfolder named EchoAction.jar. Expand that folder.
5. Create a flow as shown and run it in the Studio debugger – it should run successfully. You don't need to configure any inputs for the EchoAction operation.



Exercise 7

Controlling Access to OO Objects

You can control access to objects in the OO library and flow runs. In this exercise, you will set permissions to a flow in the library and see how it subsequently appears to different users in Central. You will also author a flow that hands off flow execution as it progresses.

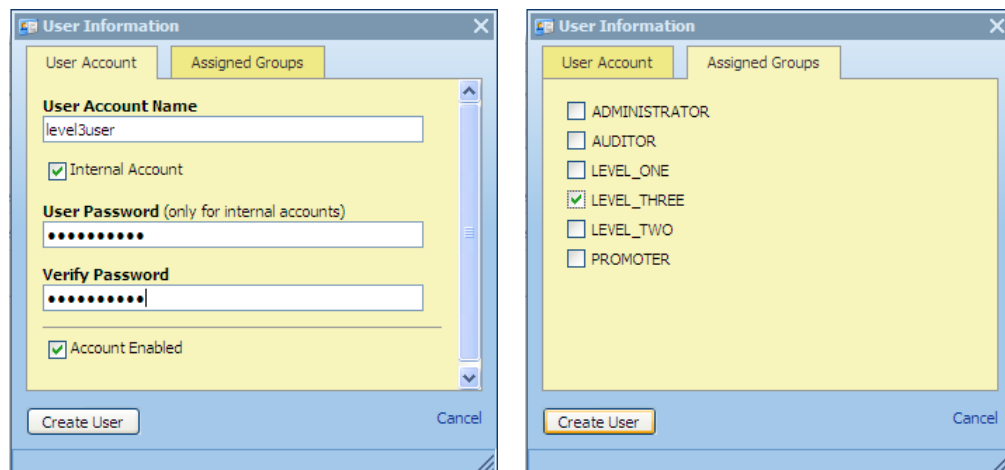
Important – Before You Start

This exercise requires the three user accounts created in the Administration module:

- level1user, assigned to the LEVEL_ONE group
- level2user, assigned to the LEVEL_TWO group
- level3user, assigned to the LEVEL_THREE group

If you need to create the three users, log in to Central as admin/admin, go to the Administration tab, and use the Manage User's tab to create the users. The passwords are the same as the user names. Make sure to do the group assignment before you create the user. Use the user name as the user password.

Example: level3user, password level3user



Objectives

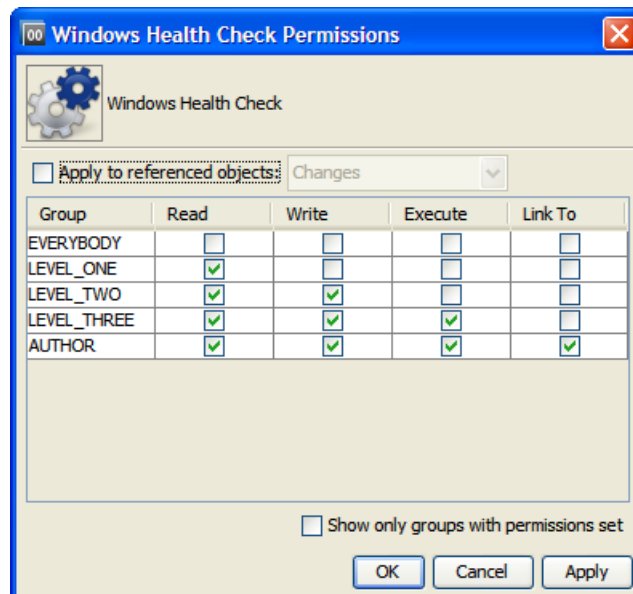
By the end of this exercise you will be able to:

- Control access to OO objects
- Control which groups can execute steps in a flow
- Hand-off a flow run to another user

Detailed Instructions

Task 1: Control Access to OO Objects

1. In Studio, locate the Windows Health Check flow (Library/Accelerator Packs/Operating Systems/Windows/Windows Health Check).
2. Right-click the flow and select Permissions.
3. Disable all permissions for the EVERYBODY group, then set permissions as shown below. When done, check in the flow.
 - LEVEL_ONE: Read
 - LEVEL_TWO: Read-Write
 - LEVEL_THREE: Read-Write-Execute
 - AUTHOR: Read-Write-Execute-Link to



4. Log into Central as level1user or level2user and navigate to the Windows Health Check flow. You will see that you do not have permission to run the flow.
5. Log in as the level3user and navigate to Windows Health Check – you will see that you can run the flow.
6. When done, return the permissions to their default state by giving EVERYBODY Read, Write, Execute, and Link To permissions. Make sure to check in the flow when you are done assigning permissions.

Task 2: Control Access to Steps

In this task you will create a flow that hands off execution to different user groups. This task requires one user each in the LEVEL_ONE, LEVEL_TWO, and LEVEL_THREE

groups that are defined in Central. When you run the flow, you will use transitions to hand off the flow run to one of those groups.

IMPORTANT: Handing off a flow run is accomplished by email. If you can send and receive email on your computer or virtual machine, then follow the steps below. If you are working on an HP virtual machine with restricted networking, simply copy the URLs provided and paste them into a browser window instead of sending emails. This will mimic the behavior of the email links that OO uses to hand off the flow runs.

1. In Studio, create a new folder and flow in My Ops Flows.

In Studio, create a flow folder in My Ops Flows named Hand Off, then create a new flow named Hand Off Steps.

Assemble the components as shown below. The flow is constructed from renamed Display operations (Utility Operations/Display) with modified icons. Name each step Step 1, Step 2, and Step 3.



2. Enter a brief Display message for each step.

Open the Display tab in the Inspector for each step and add a short message, such as **Step 1 Complete**, **Step 2 Complete**, and **Step 3 Complete**. This example shows Step 2:

Inspector

Step Name: Step 2

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

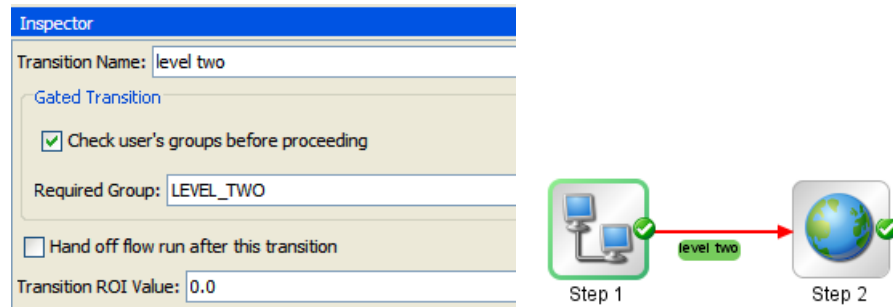
Prompt Title: Message

Prompt Width: 0 Height: 0

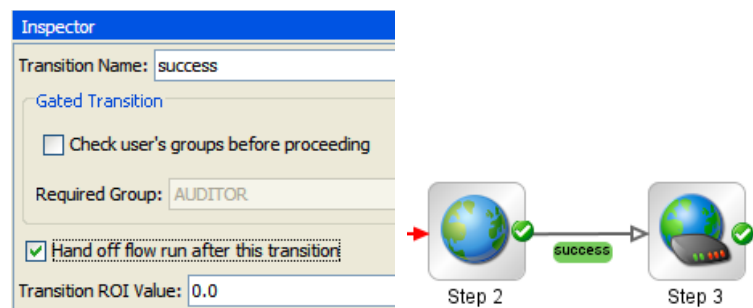
Prompt Text:

This is step 2.

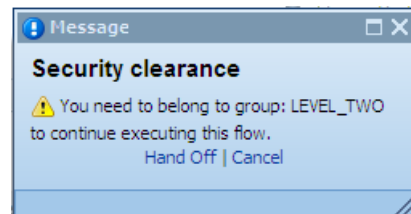
3. Specify hand-offs in each transition.
 - a. Open the first transition by double-clicking, change the name to level two, and select Check user's groups before proceeding. Then use the pull-down menu to select the LEVEL_TWO group. When the flow runs in Central, the user will be prompted to hand off the flow run. The arrow will turn red, indicating that it is a "gated" transition.



- b. Open the second transition, but this time select Hand off flow run after this transition. This causes the flow run to be handed off to another user without specifying a particular group. The transition icon changes to an grey outline.



4. Run the flow in Central.
- Check the flow in, then log into Central as level1user.
 - In the Library, navigate to your flow and run it with the Run All button.
 - When the Security Clearance dialog box appears, select Hand Off. This will launch an email window.



- Enter a valid email address and send the email, then log out as the level1user. If you don't have access to email, copy the URL and paste it into a browser window to mimic the email behavior.
5. Resume the flow run from the email as level2user.

When you receive the email, click the link to resume the flow run. Log into Central as level2user to resume the flow run. You will notice that the flow run resumes from its handed-off step.

When the flow proceeds to the next step, it is handed off again. But this time no group is required to resume the flow run. Complete and send the email, then log out of Central.

6. Resume the flow run from the email as level3user.

This time when you resume the flow run, you can do so as any valid OO user since no group was specified when you set up the hand-off in the transition. Use the email link to resume the flow run, this time logging in as level3user. The flow run resumes from the point where it was handed off.

Log out as level3user after the flow run is complete.

Feel free to experiment with different hand-off scenarios. You can also view the reports for the flow runs.

Review

In this exercise you learned various techniques for controlling flow execution. In the next exercise you will learn how to filter flow data.

Exercise 8

Persisting Data Across Flow Runs

In this exercise you will use System Properties and OO Data Persistence operations to persist data across flow runs.

In Studio, solutions to this exercise are in My Ops Flows/Exercise Solutions/Advanced Authoring/Data Persistence.

Objectives

By the end of this exercise you will be able to:

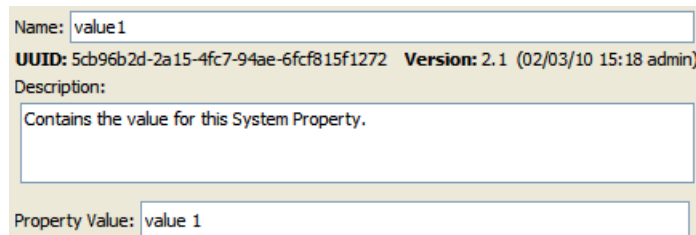
- Use System Properties to store persistent data
- Use OO Integration Library content to create and modify System Properties
- Use data persistence content in the OO Library to persist data across flow runs

Detailed Instructions

Task 1: Use System Properties to store persistent data

You can store data in your Configuration/System Properties folder and the stored data will be available to all flows. Each System Property defines a flow variable with the value you specify.

1. In Studio, expand Configuration and locate System Properties.
2. In System properties create three new properties named value1, value2, and value3. Add a value to each system property. This example shows the value1 System Property:

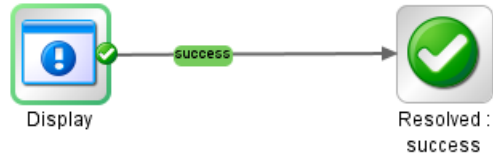


The screenshot shows the configuration for a System Property named 'value1'. It includes a 'Name' field with 'value1', a 'UUID' of '5cb96b2d-2a15-4fc7-94ae-6fcf815f1272', a 'Version' of '2.1 (02/03/10 15:18 admin)', a 'Description' field with the text 'Contains the value for this System Property.', and a 'Property Value' field with 'value 1'.

3. Write a simple flow that retrieves values and lets you set new values.

Create a new folder in My Ops Flows named **Data Persistence**. Then create a new flow named **Data Persistence With System Properties** in your Data Persistence folder.

This flow simply retrieves values stored in the System Properties, lets you change them, then displays the new values. The values are not stored across flow runs – the values stored in System Properties become the default values whenever you start the flow.



Display

Inputs tab:

- ◆ Remove the field4 variable.
- ◆ For field1 through field3 use the Input Editor to define the inputs as shown, using **value1** for field1, **value2** for field2, and **value3** for field 3. **Value1** through **value3** are listed in the pull-down menu under System Properties.

Step Name: Display

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > field1

Name: field1 Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Prompt User

Assign to Variable: value1

Otherwise: Prompt For: User Message:

Display tab:

- ◆ Set the Display tab as shown:

Step Name: Display

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: System Properties

Prompt Width: 0 Height: 0

Prompt Text:

Values for stored System Properties:

value1: \${value1}

value2: \${value2}

value3: \${value3}

Resolved: Success

Display tab:

- ◆ Set the Display tab to show the new values entered for the System Properties.

Step Name: Resolved : success

Inputs Results **Display** Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: values

Prompt Width: 0 Height: 0

Prompt Text:

New values for System Properties:

value1: \${value1}

value2: \${value2}

value3: \${value3}

4. Test the flow.

You will be shown the existing values for value1, value2, and value3 and will be prompted to enter new values. Then you will be shown the new values you entered.

The next time you run the flow, however, the previous values for value1, value2, and value3 are shown.

Task 2: Use OO Integration Library content to create and modify System Properties

In this task you will build a flow that creates a new System Property and stores a value there. Subsequently, the flow will update the value in the System Property.

1. Build the System Property flow:

Check Null is in Utility Operations/Flow Variable Manipulation, System Property is in Integrations/Hewlett-Packard/Operations Orchestration/Repository, and Display is in Utility Operations.

2. Modify CheckNull's keyName input so it has a constant value of testProperty:

Inspector

Step Name: Check Null

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input

Assign To Input	Required	Type	From
keyName	<input checked="" type="checkbox"/>	Single Value	Value: testProperty

3. Modify Set System Property. Give propertyName a constant value of testProperty and propertyValue a constant value of "This is the value for the testProperty System Property" or some other text:

Inspector

Step Name: Set System Property

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
propertyName	<input checked="" type="checkbox"/>	Single Value	Value: testProperty
propertyValue	<input checked="" type="checkbox"/>	Single Value	Value: This is the value for the testPrope...
createNewProperty	<input type="checkbox"/>	Single Value	Value: true
checkinComment	<input type="checkbox"/>	Single Value	Value:

- Modify the Display tab of the Display step to show `${testProperty}`:

Inspector

Step Name: Display

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Message

Prompt Width: 0 Height: 0

Prompt Text:

`${testProperty}`

- Run the flow in the Studio debugger. You will see the value of testProperty displayed, and the new testProperty will be present in the System Properties folder.

Task 3: Use OO Data Persistence operations to persist flow data

In this task you will write a flow that allows you to store values in flow variables across flow runs.

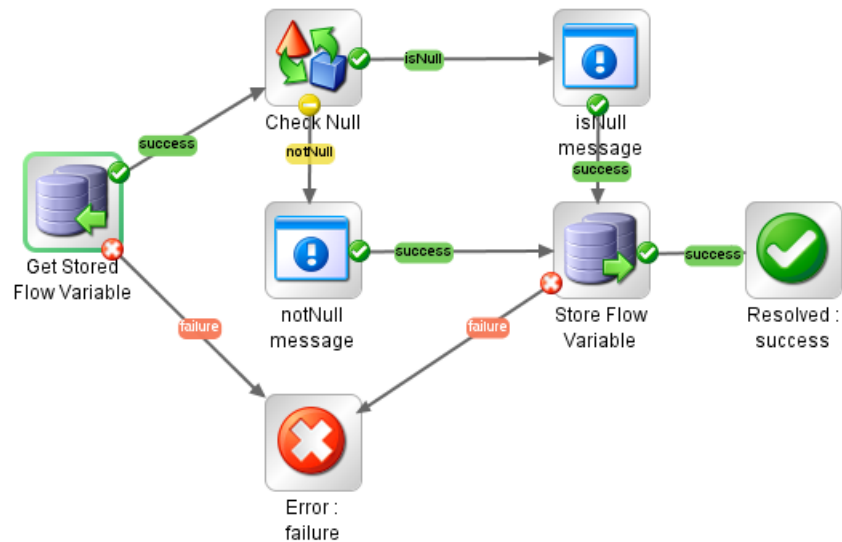
- Assemble the Data Persistence flow as shown below.

In your Data Persistence folder, create a new flow named **Data Persistence**.

Get Stored Flow Variable and Store Flow Variable are in Integrations/Hewlett-Packard/Operations Orchestration/Cross Run Data Persistence.

Check Null is in Utility Operations/Flow Variable Manipulation.

notNull Message and isNull Message are renamed Display operations in Utility Operations.



2. Configure the Get Stored Flow Variable step.

Inputs tab:

Step Name: Get Stored Flow Variable

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
keys	<input checked="" type="checkbox"/>	Single Value	Value: value1,value2,value3
createFlowVariables	<input type="checkbox"/>	Single Value	Value: true

keys input:

Step Name: Get Stored Flow Variable

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > keys

Name: keys Input Type: Single Value

Input Data Flow

Assign from Variable: keys

Otherwise: Use Constant

Assign to Variable: keys

Otherwise: Use Constant

Constant Value: value1,value2,value3

createFlowVariables: set to **true**. (Setting to **false** clears the stored flow variable.)

3. Add a Result to Get Stored Flow Variable

Add a result named ResultTable based on the Result Field ResultTable.

Step Name: Get Stored Flow Variable

Inputs Results Display Description Advanced Scriptlet

Step Results

Add Result Remove Result

Name	From	Assign To	Assignment Action	Filters
ResultTable	Result Field: ResultTable	Flow Variable	OVERWRITE	No Filters

4. Configure the Check Null step.

Set keyName to **value1**.

Step Name: Check Null

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > keyName

Name: keyName Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Use Constant

Assign to Variable: <not assigned>

Otherwise: Use Constant Value: value1

5. Configure the notNull message step.

Step Name: notNull message

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input

Assign To Input	Required	Type	From
field1	<input type="checkbox"/>	Single Value	Prompt User
field2	<input type="checkbox"/>	Single Value	Prompt User
field3	<input type="checkbox"/>	Single Value	Prompt User
field4	<input type="checkbox"/>	Single Value	Value:

Inputs tab:

- The field1 through field3 inputs should Prompt User. Assign from Variable should be <not assigned> and Assign to should be **value1** through **value3** respectively. Field4 is not used. The example shows the field1 input.

Step Name: notNull message

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > field1

Name: field1 Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Prompt User

Assign to Variable: value1

Otherwise: Prompt For: User Message: Enter value1:

Display tab:

- Display a message that shows the values for value1 through value3.

Step Name: notNull message

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Message

Prompt Width: 0 Height: 0

Prompt Text:

```
The following persistent flow variables were stored in the last flow run:
value1: ${value1}
value2: ${value2}
value3: ${value3}
```

6. Configure the isNull message step.

Inputs tab:

Step Name: isNull message

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
field1	<input type="checkbox"/>	Single Value	Prompt User
field2	<input type="checkbox"/>	Single Value	Prompt User
field3	<input type="checkbox"/>	Single Value	Prompt User
field4	<input type="checkbox"/>	Single Value	Value:

- Configure field1 through field 3 to Prompt User and assign to variable **value1** through value3 respectively. The example shows the field1 input.

Step Name: isNull message

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > field1

Name: field1 Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Prompt User

Assign to Variable: value1

Otherwise: Prompt For: User Message: Enter value 1:

Display tab:

- Display a message showing the values for value1 through value3:

Step Name: isNull message

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Message

Prompt Width: 0 Height: 0

Prompt Text:

```
The following persistent flow variables will be set:
value1
value2
value3
```

7. Configure the Store Flow Variable step.

Inputs tab:

- Remove the key1 input and add inputs for value1 through value3/

Step Name: Store Flow Variable

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
value1	<input type="checkbox"/>	Single Value	Value: \${value1}
value2	<input type="checkbox"/>	Single Value	Value: \${value2}
value3	<input type="checkbox"/>	Single Value	Value: \${value3}
key1	<input type="checkbox"/>	Not Assigned	No Assignment

- Configure each input. The example shows the value1 input.

Step Name: Store Flow Variable

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > value1

Name: value1 Input Type: Single Value

Input Data Flow

Assign from Variable: value1

Otherwise: Use Constant

Assign to Variable: value1

Otherwise: Use Constant Value: \${value1}

- Configure the Resolved: success step.

Display tab:

- Display a message informing that value1 through value3 were set. Also display the Result Table (\${resultTable}). The resultTable flow variable was defined as a result in Get Stored Flow Variable and shows how persistent flow variables are stored in the system.

Step Name: Resolved : success

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Success

Prompt Width: 0 Height: 0

Prompt Text:

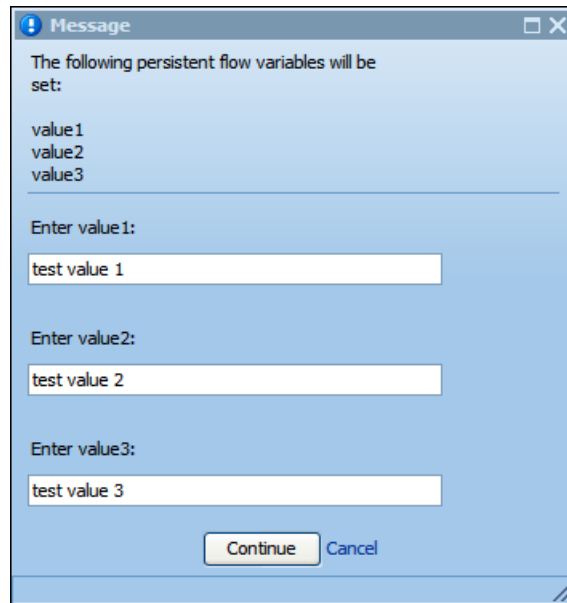
Persistent variables value1, value2, and value3 were set successfully.

Must be a \

\${ResultTable}

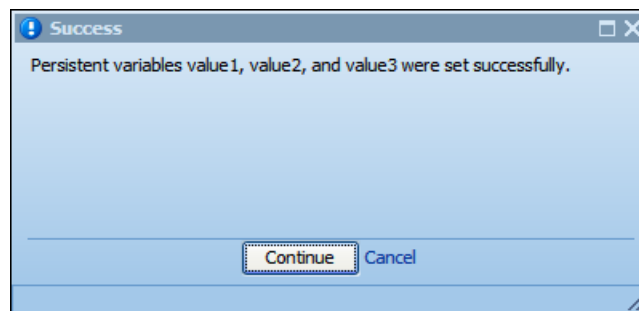
- Test the flow.

When you run the flow for the first time you are prompted to enter values for value1 through value3. Note that those values are not set yet.



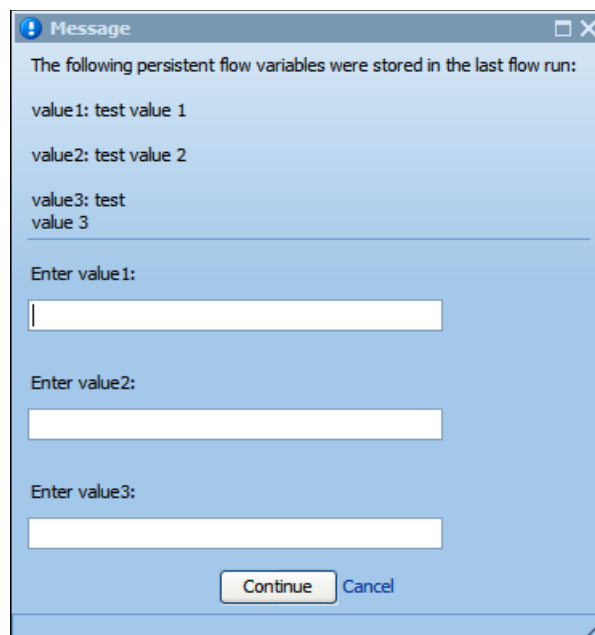
A blue message dialog box titled "Message" with a standard Windows-style title bar (minimize, maximize, close buttons). The text inside reads: "The following persistent flow variables will be set:" followed by a list: "value1", "value2", and "value3". Below this list is a horizontal line. Under the line are three input fields, each preceded by a label: "Enter value1:", "Enter value2:", and "Enter value3:". The first two input fields contain the text "test value 1" and "test value 2" respectively. The third input field contains "test value 3". At the bottom right of the dialog are two buttons: "Continue" (highlighted with a yellow border) and "Cancel".

At the end of the flow run you are notified that the values were set.



A blue success dialog box titled "Success" with a standard Windows-style title bar. The text inside reads: "Persistent variables value1, value2, and value3 were set successfully." Below this text is a horizontal line. At the bottom right of the dialog are two buttons: "Continue" (highlighted with a yellow border) and "Cancel".

Run the flow again, this time noting that the you are shown the persistent values for value1 through value3. You can also change those values, and the new values will be persisted.



A blue message dialog box titled "Message" with a standard Windows-style title bar. The text inside reads: "The following persistent flow variables were stored in the last flow run:" followed by a list: "value1: test value 1", "value2: test value 2", and "value3: test value 3". Below this list is a horizontal line. Under the line are three input fields, each preceded by a label: "Enter value1:", "Enter value2:", and "Enter value3:". The first two input fields are empty. The third input field contains "test value 3". At the bottom right of the dialog are two buttons: "Continue" (highlighted with a yellow border) and "Cancel".

Exercise 9

XML Processing

In this exercise you will work with the operations in the OO library for managing XML documents.

In Studio, solutions to this exercise are in My Ops Flows/Exercise Solutions/Advanced Authoring/XML Processing.

Objectives

By the end of this exercise you will be able to:

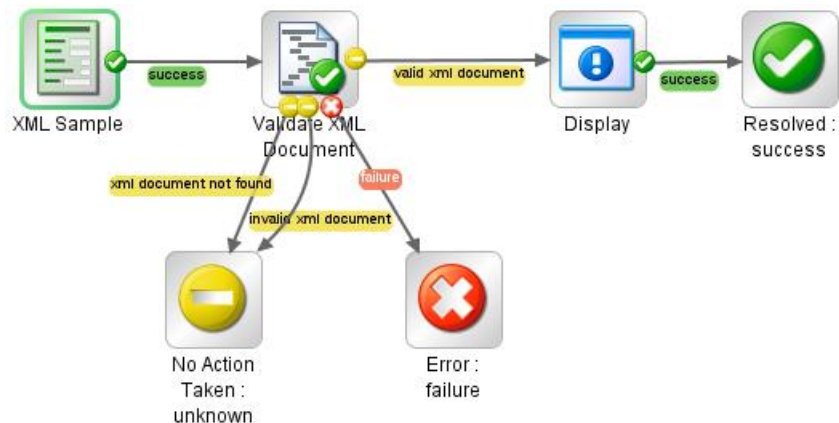
- Build a simple XML validation flow
- Use an XML filter to extract data from an XML document
- Use content in the OO Library that extracts data from an XML document

Detailed Instructions

Task 1: Build a simple XML flow

In this task you will build a flow that uses an XML filter that executes an XPath expression to extract data from an XML document.

1. Create a folder in My Ops Flow named **XML Processing**.
2. Build the Filter XML Flow shown below. The XML Sample operation that generates a sample XML document is in the My Ops Flows/Operations folder. Validate XML Document is in Utility Operations/XML Processing. Display is in Utility Operations.



3. Open the Step Inspector for Validate XML Document by double clicking. Make the following changes:
 - a. Remove the xmlLocation input (not used):

Inspector

Step Name: Validate XML Document

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
xmlDocument	<input type="checkbox"/>	Single Value	Prompt User
xmlLocation	<input type="checkbox"/>	Single Value	Prompt User

- b. Open the Input Editor for xmlDocument by clicking the yellow arrow button and assign the value from and to the input xmlDocument:

Inspector

Step Name: Validate XML Document

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > xmlDocument

Name: xmlDocument Input Type: Single Value

Input Data Flow

Assign from Variable: xmlDocument

Otherwise: Prompt User

Assign to Variable: xmlDocument

'Otherwise: Prompt User' Configuration

Prompt For: ☒ Text ☐ Selection

User Message:

4. Open the Display tab of the Display step to show the contents of xmlDocument:

Inspector

Step Name: Display

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

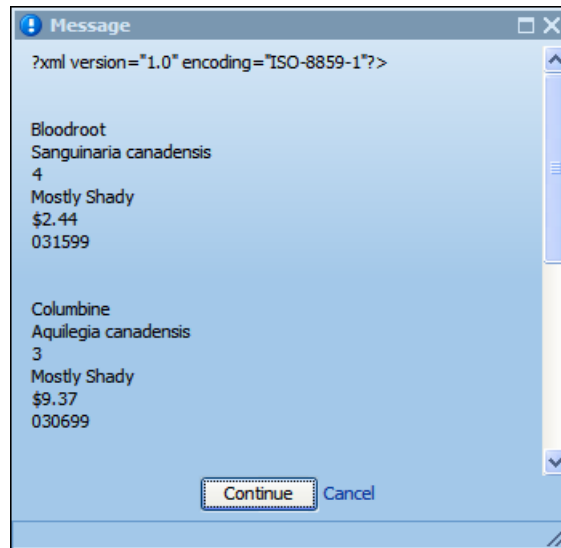
Prompt Title: Message

Prompt Width: 0 Height: 0

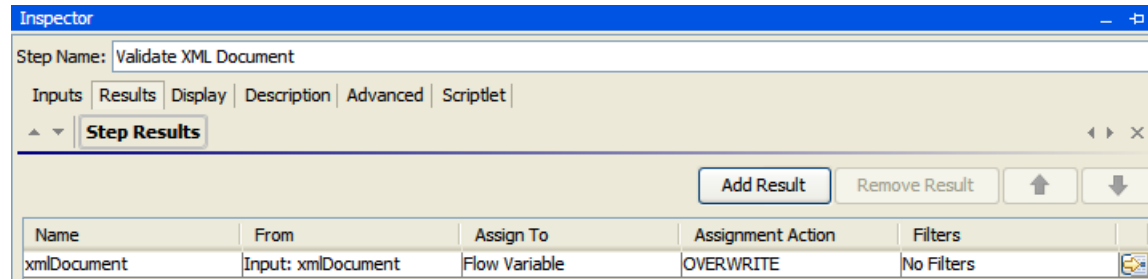
Prompt Text:

\${xmlDocument}

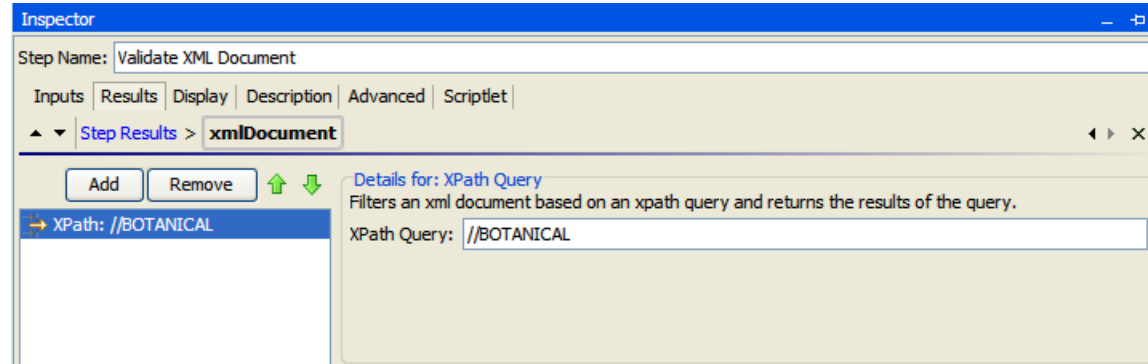
5. Save changes and test the flow – you will see the contents of the XML document.



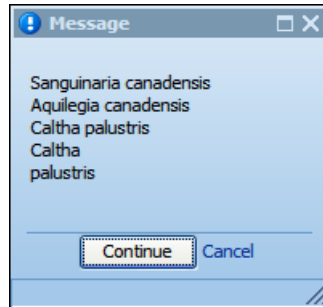
6. Add an XPath filter to the Validate step. Open the Step Inspector for Validate XML Document and select the Results tab. Add the result and specify the filter:
 - a. Add a result named xmlDocument



- b. Open the filter editor and add an XPath Query filter (select from the pull down list) with a value of `//BOTANICAL`. You can also use XPaths like `//COMMON`, `//LIGHT`, and `//AVAILABILITY`:



7. Save changes and test the flow – you will see a listing for the XPath you used.



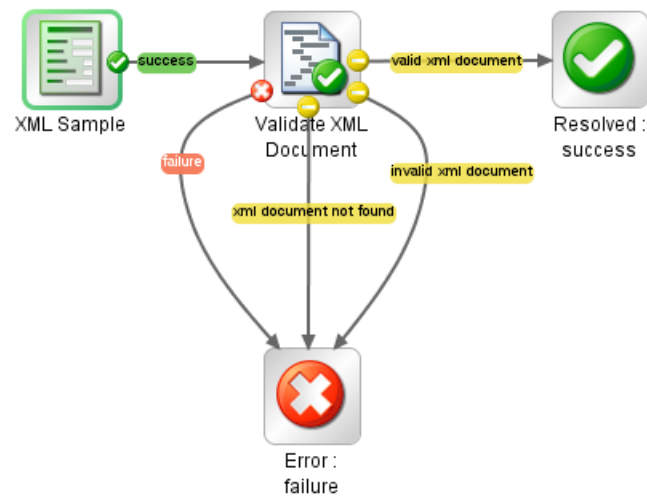
Task 2: Build a simple XML validation flow.

In this task you will use content in the OO Library in a flow that uses an XPath expression to extract data from an XML document.

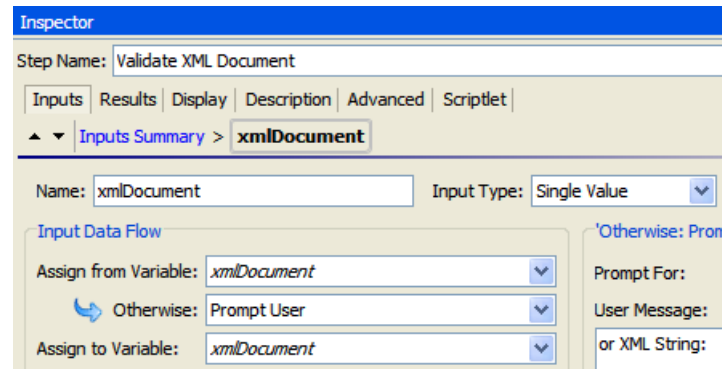
1. Create a new folder and flow.

In My Ops Flows create a new folder named **XML Processing**, and in that folder create a new flow named **Validate XML**. Assemble the components as shown.

Use the XML Sample operation in you're my Ops Flows/Operations folder. Validate XML Document is in Utility Operations/XML Processing.



2. Modify Validate XML Document.
 - a. Open the Step Inspector for Validate XML Document by double-clicking.
 - b. Remove and remove the input for xmlLocation – since the XML document is fed to this step by a flow variable (xmlDocument) defined in the XML Sample step, the xmlLocation input is not need. In practice you would use one or the other depending on the source of your XML document.
 - c. Modify the xmlDocument input to pick up its value from the flow variable xmlDocument. When done save your changes.



3. Test the flow.

You should see that the flow ends with Resolved: Success in the Studio debugger, indicating that the XML document was successfully validated. Examine the output in the debugger's Run Tree and Step Result Inspector.

Task 3: Add XPath Evaluator

In this task you will add an XPath Evaluator to your flow. The XPath Evaluator allows you to extract data from the XML document using an XPath.

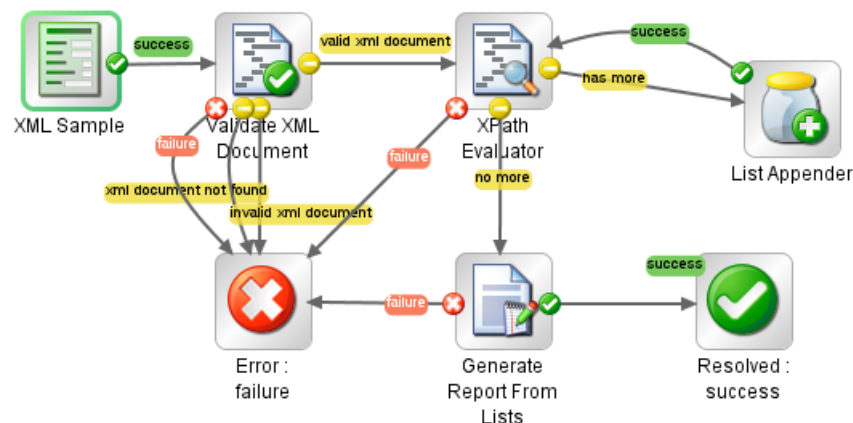
1. Duplicate your Validate XML Document flow and rename it **Process XML**.

This gives you a working copy that you can modify.

2. In your Process XML flow, assemble the components as shown.

Path Evaluator is in Utility Operations/XML Processing.

List Appender and Generate Report from Lists are in Utility Operations/Containers/Lists.



3. Modify XPath Evaluator.

In the Step Inspector, open the input editor for xml:

- Assign from Variable: **xmlDocument**
- Assign to Variable: **xml**

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Inputs Summary > **xml**

Name: Input Type:

Input Data Flow

Assign from Variable:

Otherwise:

Assign to Variable:

'Otherwise: Prompt User' Co

Prompt For: ☒ Text

User Message: Enter a value for xml

Save changes and close the Step Inspector when done.

4. Modify List Appender to compile the list of returned elements.
Open the Step Inspector and use the Input Editor to specify the inputs.

- keyName: Constant Value listItems

▲ ▼ Inputs Summary > **keyName**

Name: Input Type:

Input Data Flow

Assign from Variable:

Otherwise:

Assign to Variable:

'Otherwise: Use C

Constant Value: listItems

- resultText: Use Previous Step Result.

▲ ▼ Inputs Summary > **resultText**

Name: Input Type:

Input Data Flow

Assign from Variable:

Otherwise:

Assign to Variable:

'Otherwise: Use

Constant Value: listItems

- delimiter: Constant Value comma (,)

Name: Input Type:

Input Data Flow

Assign from Variable:

Otherwise:

Assign to Variable:

'Otherwise: Use

Constant Value: ,

5. Modify Generate Report From Lists.

Generate Report From Lists creates a report from one or more lists. You need to add an input for each list. You also need to specify the delimiter for your lists.

- a. In the Step Inspector add an input for listItems and set the delimiter to comma (,).

listItems:

Name: Input Type:

Input Data Flow

Assign from Variable:

Otherwise:

Assign to Variable:

Prompt For:

User Message:

delimiter:

Name: Input Type:

Input Data Flow

Assign from Variable:

Otherwise:

Assign to Variable:

Constant Value:

If the'

- b. In the Results tab add a result named returnResult based on the Result Field returnResult:

Inputs Results Display Description Advanced Scriptlet

Step Results

Add Result Remove Result

Name	From	Assign To	Assignment Action	Filters
returnResult	Result Field: returnResult	Flow Variable	OVERWRITE	No Filters

6. Modify the Resolved: success step to display the report.

Open the Step Inspector and in the Display tab, display the returnResult flow variable returned by the Generate Report From Lists step.

Step Name:

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title:

Prompt Width: Height:

Prompt Text:

7. Test the flow.

When you run the flow you will be prompted to enter an XPath expression. Use any of the following:

- * or node() (returns everything)
- //COMMON (returns common plant names)
- //BOTANICAL (returns plant botanical names)
- See http://www.w3schools.com/XPath/xpath_syntax.asp for a nice tutorial on XPath expressions.

The specified data is displayed in a report.

Exercise 10

Working With File Systems

In this exercise you will work with file systems. You will list contents of a directory, write information to a file, read a file, and write a stock quote to disk.

In Studio, solutions to this exercise are in My Ops Flows/Exercise Solutions/Advanced Authoring/File System.

Objectives

By the end of this exercise you will be able to:

- List the contents of a directory
- List only subfolders of a parent directory
- Write a listing of directory contents to a file
- Read a file and display its contents
- Get a stock quote from the Internet and write it to a file

Detailed Instructions

Task 1: List contents of a directory.

In this task you will create a simple flow that lists the content of a directory on your local system.

1. Create a new folder and flow.

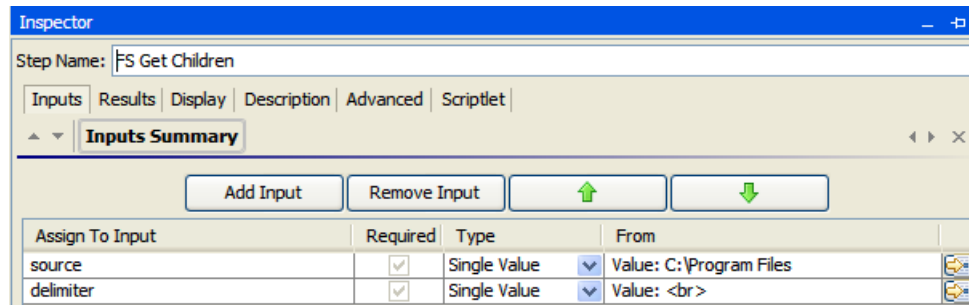
Name the folder File System and the flow List Directory. Then assemble the components as shown. FS Get Children is in Operations/File System/Cross Platform.



2. Configure FS Get Children inputs.

In the Step Inspector, configure the source and delimiter inputs. Set source to a Constant Value of a directory, like **C:\Program Files**.

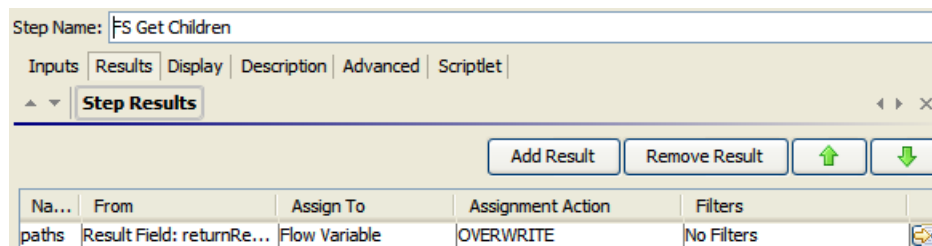
Set delilmiter to **
**.



Leave the Step Inspector open.

3. Add a result.

In the Results tab add a result named **paths** based on the returnResult. When done save your changes and close the Step Inspector.



4. Display the directory paths and number of paths found.

Open the Step Inspector for Resolved: success and in display the paths and count:

Number of paths:

`${count}`

Paths in \${source}:

`${paths}`

5. Test the flow.

Run the flow in the Studio debugger. When prompted to enter a path, use something like C:\Program Files. The full paths of each file and subdirectory in Program Files is returned along with the number of items found.

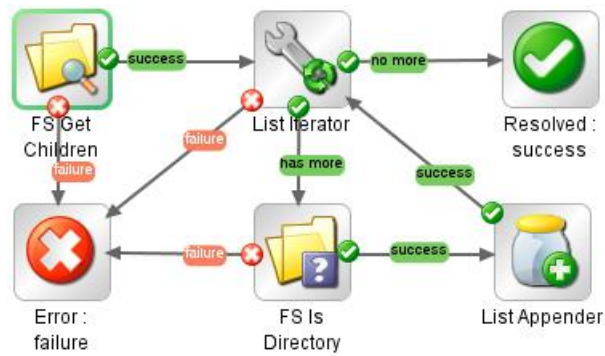
Task 2: List only subfolders.

In this task you will modify the List Directory flow to return only subfolders.

1. Modify the List Directory flow to save only directories.

You can duplicate your List Directories flow and rename it **List Directory Subfolders** if you wish.

Assemble the components as shown:



FS Is Directory is located in Operations/File System/Windows Only, List Iterator is in Utility Operations/Looping, and List Appender is located in Utility Operations/Containers/Lists.

2. Modify FS Get Children inputs and results.

Inputs:

Set source to Prompt User for input, and set the delimiter to a comma.

Assign To Input	Required	Type	From
source	<input checked="" type="checkbox"/>	Single Value	Prompt User
delimiter	<input checked="" type="checkbox"/>	Single Value	Value: ,

Results:

Add a result named **list** based on the Result Field: returnResult.

Name	From	Assign To	Assignment Action	Filters
list	Result Field: return...	Flow Variable	OVERWRITE	No Filters

3. Configure List Iterator inputs.

With the Input Editor for the list input, set Assign From Variable to **list** and Assign To Variable to **<not assigned>**.

Inspector

Step Name: List Iterator

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Inputs Summary > **list**

Name: list Input Type: Single Value

Input Data Flow

Assign from Variable: list

Otherwise: Prompt User

Assign to Variable: <not assigned>

Otherwise: Prompt For: User Message:

4. Configure FS Is Directory inputs.

Set the Source input to use **Result of Previous Step** and assign the value to **Source**. Assign From Variable should be <not assigned>.

Inspector

Step Name: FS Is Directory

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Inputs Summary > **Source**

Name: Source Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Use Previous Step Result

Assign to Variable: Source

Otherwise: Use

If the 'assign

5. Configure List Appender.

- a. Set the keyName to a constant value of **filteredPathList**. This is the filtered list of directory names.

Inspector

Step Name: List Appender

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Inputs Summary > **keyName**

Name: keyName Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Use Constant

Assign to Variable: <not assigned>

Otherwise: Use

Constant Value: filteredPathList

- b. Set resultText to **Assign from Variable: source**. Leave Assign to Variable: <not assigned>.

Inspector

Step Name: List Appender

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Inputs Summary > resultText

Name: resultText Input Type: Single Value

Input Data Flow

Assign from Variable: source

Otherwise: Prompt User

Assign to Variable: <not assigned>

'Otherwise: Prompt User' Cor

Prompt For: Text

User Message: If the 'assign fro

String to Append :

- c. Set the delimiter to `
`. This HTML break tag will produce a nicely formatted list of directory names.

Step Name: List Appender

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Inputs Summary > delimiter

Name: delimiter Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Use Constant

Assign to Variable: <not assigned>

'Otherwise: Use

Constant Value:

6. Configure the Display tab in Resolved: Success.

Set the Display tab to display the filteredPathList variable:

Step Name: Resolved : success

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: msg

Prompt Width: 0 Height: 0

Prompt Text:

\${filteredPathList}

7. Test the flow in the Studio debugger.

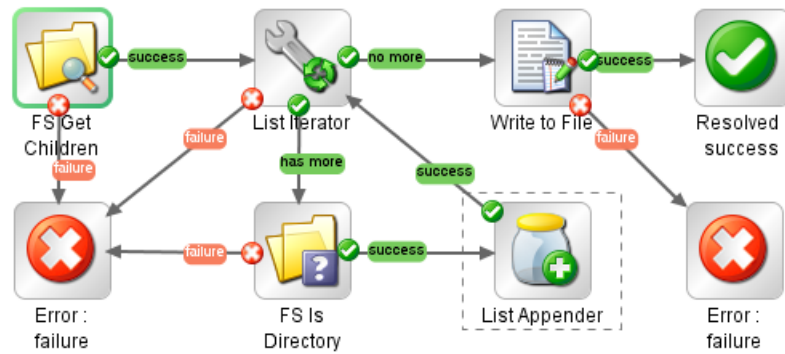
Provide a source directory, like C:\Program Files. You should see a list of subdirectories in Program Files.

Task 3: Write the list to a file

1. Add a "Write to File" step to your flow.

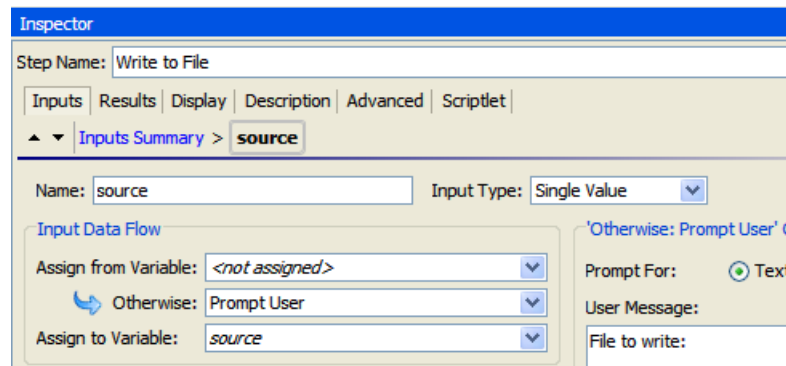
You can duplicate your existing flow and modify that version if you wish. The Write to File operation is in Operations/File System/Windows Only.

Assemble the components as shown.

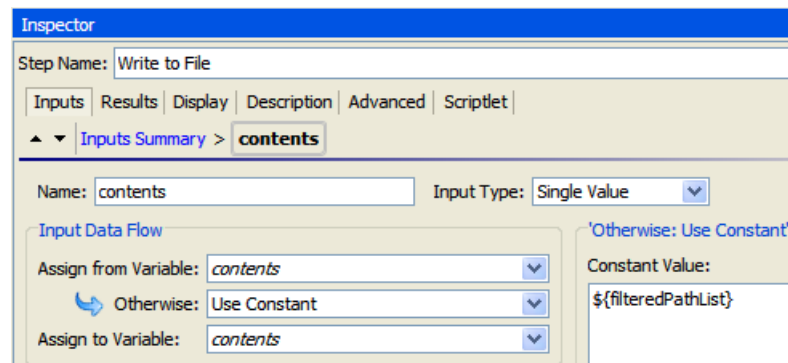


2. Modify Write to File inputs.

- Set the delimiter input to Constant Value but leave the Constant Value blank.
- Leave the source input to Prompt User but change the prompt to **File to write:** and set Assign from Variable to **<not assigned>**.



- Set the contents input to a Constant Value of **\${filteredPathList}**.



- Modify the Display tab in the Resolved: Success.
Set the message to **\${source} successfully written.**

Inspector	
Step Name:	Resolved : success
Inputs	Results Display Description Advanced Scriptlet
<input checked="" type="checkbox"/> Always prompt user before executing this step	
Prompt Title:	msg
Prompt Width:	0
Prompt Height:	0
Prompt Text:	<pre> \${source} successfully written. </pre>

4. Test the flow.

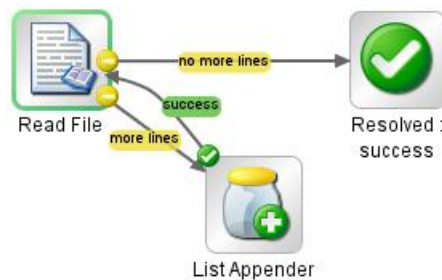
Run the flow in the Studio debugger – you should see a pop-up window telling you the file was successfully written, and the file should appear at the location you specified.

Task 4: Read a file

In this task you will build a flow that reads and displays the contents of the file created by the flow you authored in Task 3. Then you will filter the output to show only certain lines in the file.

1. Build the Read File flow.

Create the flow in the same folder as the flow you created in task 3. Name it Read Directory List From File. The Read File operation is in Operations/File System/Windows Only and List Appender is in Utility Operations/Containers.



2. Modify ReadFile inputs.

Remove the Filter, user, and password inputs – they are not used in this flow. Then set the value of the delimiter input to **
** or whatever delimiter you used in your previous flow.

Inspector

Step Name: Read File

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
Source	<input checked="" type="checkbox"/>	Single Value	Prompt User
Store	<input checked="" type="checkbox"/>	Single Value	Value: True
delimiter	<input type="checkbox"/>	Single Value	Value:
eofReset	<input type="checkbox"/>	Single Value	Value: True
Filter	<input type="checkbox"/>	Single Value	Prompt User
user	<input type="checkbox"/>	Single Value	Prompt User
password	<input type="checkbox"/>	Single Value	Prompt User

3. Modify List Appender.

Modify inputs as shown:

Step Name: List Appender

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
keyName	<input checked="" type="checkbox"/>	Single Value	Value: MyFile
resultText	<input checked="" type="checkbox"/>	Single Value	Result of previous step
delimiter	<input type="checkbox"/>	Single Value	Value:

- keyName: set to MyFile.

Step Name: List Appender

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > keyName

Name: keyName Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Use Constant

Assign to Variable: <not assigned>

Otherwise: Use Constant Value: MyFile

- resultText: set to Use Previous Step Result.

Step Name: List Appender

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > resultText

Name: resultText Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Use Previous Step Result

Assign to Variable: <not assigned>

- delimiter: Set to Use Constant with a single return character in the Constant Value field. The field will appear blank with the cursor on the second line.

Step Name: List Appender

Inputs Results Display Description Advanced Scriptlet

▲ ▼ Inputs Summary > delimiter

Name: delimiter Input Type: Single Value

Input Data Flow

Assign from Variable: <not assigned>

Otherwise: Use Constant

Assign to Variable: <not assigned>

Constant Value:

4. Modify the Resolved: Success step.
 - Set the Display tab to display `${MyFile}`.

Step Name: Resolved : success

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: msg

Prompt Width: 0 Height: 0

Prompt Text: `${MyFile}`

Save changes and close editors when done.

5. Test the flow.

When prompted, enter the full path of the file written by your other flow, for example: C:\directory_list.txt.

You should see the complete file with each path on a separate line.

Task 5: Use HTTP GET to write a stock quote to disk

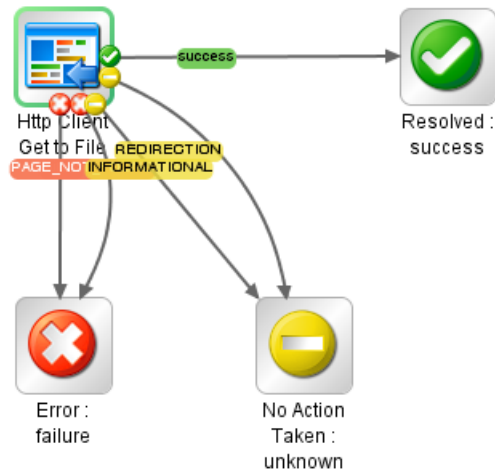
In this task you will use the HTTP Get to File operation to retrieve a stock quote from Yahoo Finance and save it to disk.

NOTE: This task may not work on the Surgient online training environment because of the security configuration applied to the virtual machine. If you have access to a local installation of Operations Orchestration you can try it on that instance.

1. Create a folder and flow.

In My Ops Flows, create a folder named HTTP Get. Then create a flow in that folder named HTTP Get. Assemble the components as shown.

HTTP Client Get to File is in Operations/HTTP Client.



2. Configure the HTTP Client Get to File step.
 - a. Inputs tab: Remove the unused inputs as shown – you will specify values for file, URL, encodeURL, and symbol as shown.

Inspector

Step Name: Http Client Get to File

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
file	<input checked="" type="checkbox"/>	Single Value	Value: C:\\${symbol}.csv
url	<input checked="" type="checkbox"/>	Single Value	Value: http://download.finance.yahoo...
encodeURL	<input type="checkbox"/>	Single Value	Value: true
symbol	<input type="checkbox"/>	Single Value	Prompt User
username	<input type="checkbox"/>	Single Value	Value:
password	<input type="checkbox"/>	Single Value	Value:
proxy	<input type="checkbox"/>	Single Value	Value:
proxyPort	<input type="checkbox"/>	Single Value	Value:
proxyUsername	<input type="checkbox"/>	Single Value	Value:
proxyPassword	<input type="checkbox"/>	Single Value	Value:
keystore	<input type="checkbox"/>	Single Value	Value:
keystorePassword	<input type="checkbox"/>	Single Value	Value:
followRedirects	<input type="checkbox"/>	Single Value	Value:
timeout	<input type="checkbox"/>	Single Value	Value:
trustAllRoots	<input type="checkbox"/>	Single Value	Value:
useCookies	<input type="checkbox"/>	Single Value	Value:
userAgent	<input type="checkbox"/>	Not Assigned	No Assignment

- b. Configure the URL input to use the following URL as a Constant Value:
 http://download.finance.yahoo.com/d/quotes.csv?s=\${symbol}&f=sl1d1t1c1ohgv&e=.csv
 - c. For file, enter **C:\\${symbol}.csv**. This will write the quote for the symbol the file to the specified location.
 - d. Set encodeURL to a Constant Value of **true**.
3. Configure the Resolved: Success step.
 Add a display message indicating that the file downloaded successfully:

Step Name: Resolved : success	
Inputs	Results
Display	Description
Advanced	Scriptlet
<input checked="" type="checkbox"/> Always prompt user before executing this step	
Prompt Title: Message	
Prompt Width: 0	Height: 0
Prompt Text:	
\${file} successfully retrieved.	

4. Test the flow.

Enter a stock symbol, like **hpq**, **goog**, or **aapl**. The file will be downloaded to the path you specified.

Exercise 11

Working With Email

In this exercise you will work with OO content that handles sending and receiving email. This is an important topic, particularly for integrations where it is a common practice to send emails to operators at different steps in a flow.

In Studio, solutions to this exercise are in My Ops Flows/Exercise Solutions/Advanced Authoring/Email.

Objectives

By the end of this exercise you will be able to:

- Author a simple Send Mail flow
- Author a simple Get Mail flow
- Author an Email Run Report URL flow

If you are working on the training virtual machine provided for this class, an email server has been pre-installed and configured. If you are working on a local installation of Operations Orchestration and need to install and configure a mail server, refer to *Installing an Email Server* at the end of this exercise.

Detailed Instructions

Task 1: Author a Simple Send Email flow

In this task you will author a simple flow that sends an email to the oosupport account.

1. Set up System Accounts.

Create new System Accounts named **Support** and **OO Support**, using username/passwords of support/support and oosupport/oosupport respectively.

In Configuration, right-click System Accounts and create the new accounts. The example below shows the inputs for the OO Support account.

Name:

UUID: 1e346832-b1bf-40ce-8627-4a17f8a0075f Version: 2 (01/28/10 13:31 admin)

Description:

Credentials

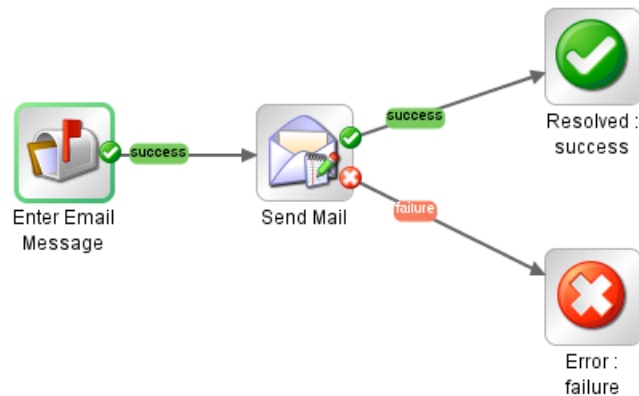
User Name:

Password:

You will use the System Accounts to identify mail users in your flows.

Check in the system accounts after they are created.

2. Create a folder and flow, then assemble the components of your flow.
 - a. Create a folder named **Email** in My Ops Flows, and in that folder create a new flow named **Simple Send Mail**.
 - b. Assemble the components as shown. The Enter Email Message step is based on a Display operation (Utility Operations/Display) that has been renamed and given a new icon. Send Mail is in Operations/Email/Send Mail.



3. Specify inputs for each step.

Enter Email Message:

- Add inputs named **emailBody** and **emailSubject** that prompt the user for input. You can remove the other unused inputs.

Inspector

Step Name: Enter Email Message

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
emailBody	<input type="checkbox"/>	Single Value	Prompt User
emailSubject	<input type="checkbox"/>	Single Value	Prompt User
field1	<input type="checkbox"/>	Single Value	Value:
field2	<input type="checkbox"/>	Single Value	Value:
field3	<input type="checkbox"/>	Single Value	Value:
field4	<input type="checkbox"/>	Single Value	Value:

Send Mail

Set the following inputs:

Step Name: Send Mail

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
hostname	<input checked="" type="checkbox"/>	Single Value	Value: localhost
port	<input checked="" type="checkbox"/>	Single Value	Value: 25
from	<input checked="" type="checkbox"/>	Single Value	Value: OOAdmin@hp.com
to	<input checked="" type="checkbox"/>	Single Value	Value: oosupport
subject	<input checked="" type="checkbox"/>	Single Value	Prompt User
body	<input checked="" type="checkbox"/>	Single Value	Prompt User
htmlEmail	<input type="checkbox"/>	Single Value	Value: true
readReceipt	<input type="checkbox"/>	Single Value	Value: false
attachments	<input type="checkbox"/>	Single Value	Value:
username	<input type="checkbox"/>	Credentials	From System Account: Support
password	<input type="checkbox"/>	Credentials	From System Account: Support
characterSet	<input type="checkbox"/>	Single Value	Prompt User from List - Selection List
contentTransferEncoding	<input type="checkbox"/>	Single Value	Prompt User from List - Selection List

- hostname: **localhost**
- port: **25**
- from: **OOAdmin@hp.com**
- to: **oosupport**
- subject: Assign from Variable: **emailSubject**
- body: Assign from Variable: **emailBody**
- htmlEmail: **true**
- readReceipt: **false**
- attachments: **No change**
- username: **From System Account: Support**
- password: **from System Account: Support**
- For username and password, set the Input Type to **Credentials**, Otherwise to **System Account**, and Account Named to **Support**.

Inspector

Step Name: Send Mail

Inputs Results Display Description Advanced Scriptlet

Inputs Summary > username

Name: username Input Type: Credentials

Input Data Flow

Assign from Variable: username

Otherwise: System Account

Assign to Variable: username

'Otherwise: System Account' Configuration

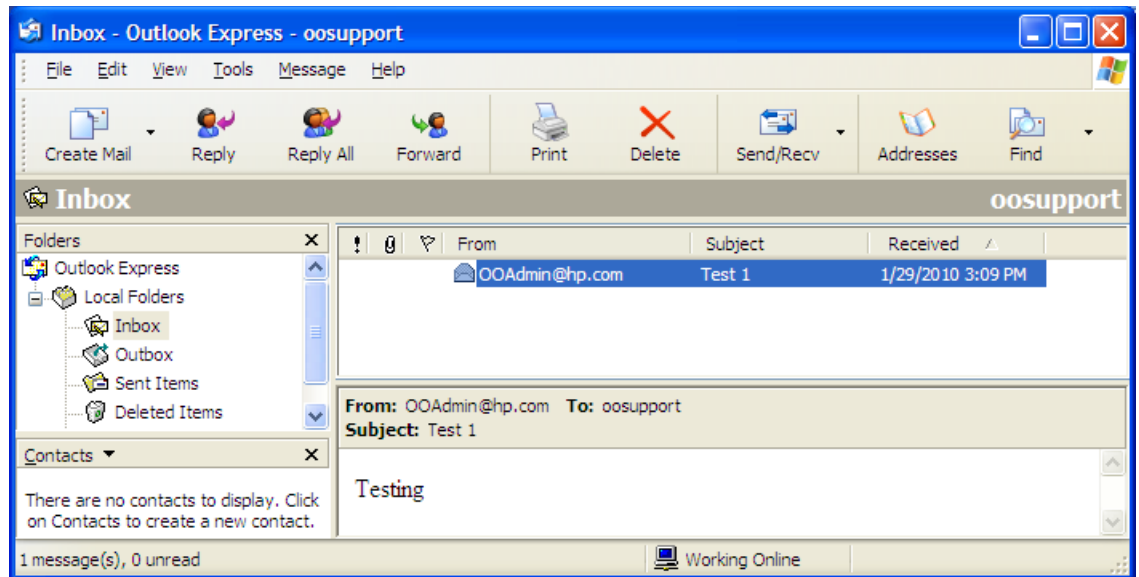
Account Named: Support

- Remove the characterSet and contentTransferEncoding inputs.

4. Test the flow.

Run the flow in the Studio debugger, entering the message body and subject. You should see a pop-up window telling you the message was processed by the email server.

Open Outlook Express as the oosupport identity. You should see the message you sent in your Inbox. Don't delete the message yet.



Task 2: Author a simple Get Mail flow

In this task you will author a simple flow that retrieves an email message from the server and displays it.

In this rudimentary mail reader, you need to provide the message number of the email you want to retrieve. The mail server queues messages in the order in which they are sent, so if you send 1 email in the previous task, then there is only one message in the queue.

1. Build the Simple Get Mail flow.



The Get Mail Message operation is in Operations/Email.

2. Assign Inputs and Results for Get Mail Message.

Inputs: Refer to the graphic to configure your inputs. The username and password inputs are the same System Accounts you used in the previous flow. If needed, delete the inputs shown in *italics* below.

Inspector

Step Name: Get Mail Message

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
host	<input checked="" type="checkbox"/>	Single Value	Value: localhost
port	<input checked="" type="checkbox"/>	Single Value	Value: 110
protocol	<input checked="" type="checkbox"/>	Single Value	Value: pop3
username	<input checked="" type="checkbox"/>	Credentials	From System Account: OO Support
password	<input checked="" type="checkbox"/>	Credentials	From System Account: OO Support
messageNumber	<input checked="" type="checkbox"/>	Single Value	Prompt User
folder	<input checked="" type="checkbox"/>	Single Value	Value: INBOX
subjectOnly	<input type="checkbox"/>	Single Value	Value: false
<i>trustAllRoots</i>	<input type="checkbox"/>	Single Value	Prompt User from List - Selection List
<i>enableSSL</i>	<input type="checkbox"/>	Single Value	Prompt User from List - Selection List
<i>keystore</i>	<input type="checkbox"/>	Single Value	Prompt User
<i>keystorePassword</i>	<input type="checkbox"/>	Single Value	Prompt User

Results:

Add one result named **email** based on **Result Field: Result**.

Inputs Results Display Description Advanced Scriptlet

Step Results

Add Result Remove Result ↑ ↓

Name	From	Assign To	Assignment Action	Filters
email	Result Field: Result	Flow Variable	OVERWRITE	No Filters

Modify the Display tab of the Resolved step to display the email: **\${email}**.

Step Name: Resolved : success

Inputs Results Display Description Advanced Scriptlet

☒ Always prompt user before executing this step

Prompt Title: Message

Prompt Width: 0 Height: 0

Prompt Text:

\${email}

- Test the flow in the Studio debugger. When prompted input message number 1, or another number based on the number of messages in your inbox. You should see the full contents of the email in a Display window.

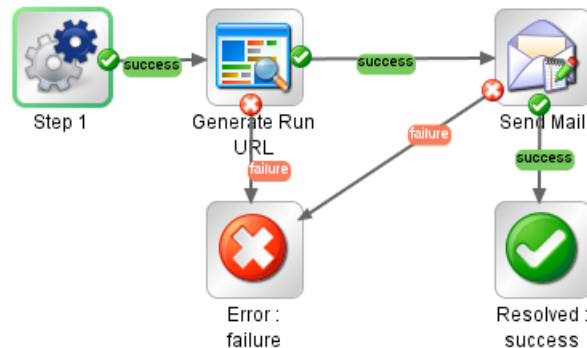
Task 3: Email a Flow Run Report URL

A Run URL is a URL that opens the report for a flow run in Central. It is commonly used to notify people that a flow has been run and a report is available.

The Generate Run URL operation will not actually send you to the report when you test a flow in Studio – you must check the flow in and run it in Central to be

directed to the report. However you can adequately test the flow in the Studio debugger.

1. Create a new flow named Sand Mail With Run URL and assemble components.



Generate Run URL is in Integrations/Hewlett-Packard/Operations Orchestration. Step 1 is simply a renamed Display operation (in Utility Operations) that has a new icon assigned - it simply generates a little activity for the flow to show up in the report.

2. Specify Send Mail inputs.

Use the same inputs for Send Mail as you did in the Simple Send Email flow, except for the body input – for the body, use a Constant Value of `${IC_ReportURL}`, which is the flow variable provided by Generate Run URL that contains the full URL to the run report. You can also provide a Constant Value for the subject input, like **Flow Run Report**. Refer to the graphic below.

Inspector

Step Name: Send Mail

Inputs Results Display Description Advanced Scriptlet

Inputs Summary

Add Input Remove Input ↑ ↓

Assign To Input	Required	Type	From
hostname	<input checked="" type="checkbox"/>	Single Value	Value: localhost
port	<input checked="" type="checkbox"/>	Single Value	Value: 25
from	<input checked="" type="checkbox"/>	Single Value	Value: OOAdmin@hp.com
to	<input checked="" type="checkbox"/>	Single Value	Value: oosupport
subject	<input checked="" type="checkbox"/>	Single Value	Value: Flow Run Report
body	<input checked="" type="checkbox"/>	Single Value	Value: \${IC_ReportURL}
htmlEmail	<input type="checkbox"/>	Single Value	Value: true
readReceipt	<input type="checkbox"/>	Single Value	Value: false
attachments	<input type="checkbox"/>	Single Value	Value:
username	<input type="checkbox"/>	Credentials	From System Account: Support
password	<input type="checkbox"/>	Credentials	From System Account: Support
characterSet	<input type="checkbox"/>	Single Value	Prompt User from List - Selection List
contentTransferEncoding	<input type="checkbox"/>	Single Value	Prompt User from List - Selection List

3. Test the flow in Studio and check for errors.

Run the flow in Studio then check the email in Outlook Express. You may need to select View Message in HTML in the View menu to properly format the email.

The flow will generate an email with a flow run URL, but the URL is incomplete when run in Studio.

4. Check the flow in and run in Central.

When you run the flow in Central, it sends an email with the run URL. Log out of Central when done.

5. Access the run report from the email.

Open the email in Outlook Express and click the URL. You may need to select View/View Message in HTML first to properly format the email.

Upon clicking the link, Central will start and you will be prompted to log in. After logging in you will automatically be placed at the run report for the flow run.

Installing an Email Server

If you are working on a local installation of HP Operations Orchestration and you need a simple email server for testing flows, you can use the instructions in this section to install and configure a freeware email server. If you are working on the training VM or you have another email server available, you can skip this section.

In this section you will install and configure the Argsoft freeware email server for use with Operations Orchestration.

Note – this installation is specifically for this exercise. Refer to the Argsoft documentation for further information.

1. Install ArGoSoft freeware mail server.

Download the installer from the FTP site for this class:

<ftp.usa.hp.com>

User: **hpoo_sp**

Password: **hpOO75sp**

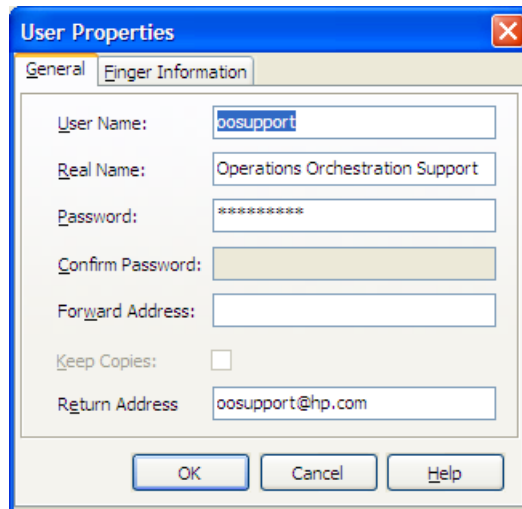
Look in the Extras/misc_installers directory and download the **agsoft.exe** installer. Follow the prompts to install the mail server.

You can also access the mail server directly from the Internet. Do a Google search on **argosoft download** to pick up the latest version.

Use SMTP port 25 and POP3 port 110.

6. Configure ArGoSoft user.

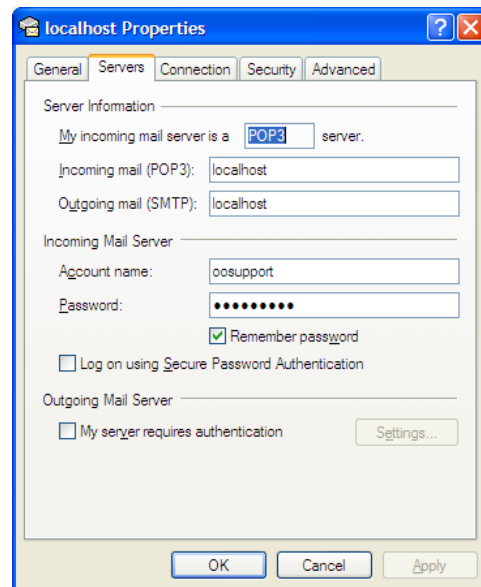
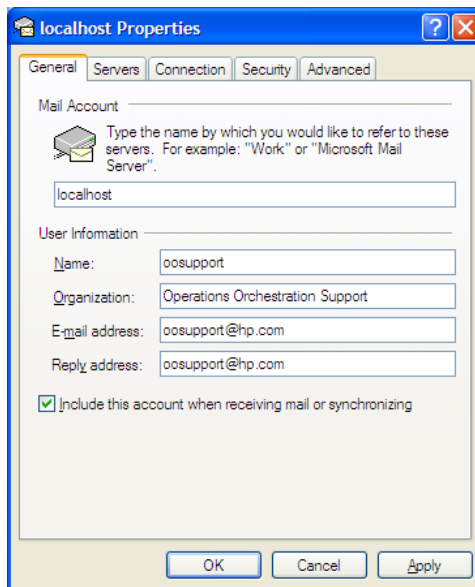
Create **support** and **oosupport** users with a passwords of **support** and **oosupport** respectively. The example shows **oosupport**.

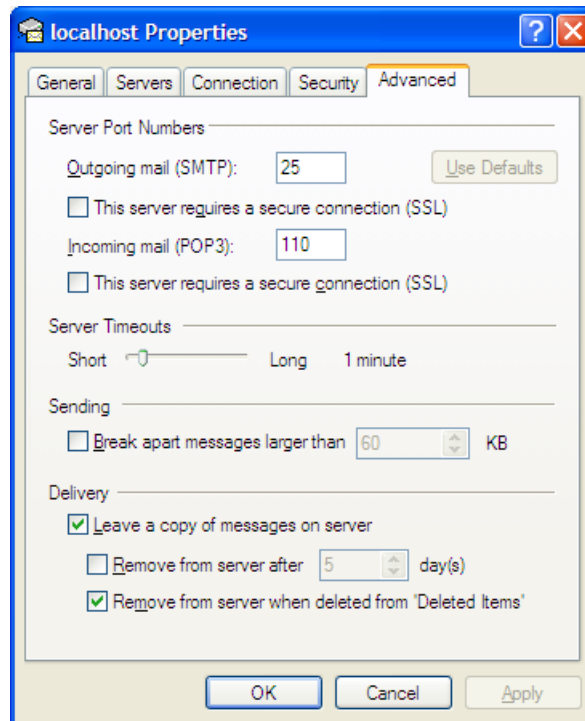


7. Configure Outlook Express.
8. This exercise uses Outlook Express as the mail client.

Open Tools/Accounts add a **localhost** mail account. You will need to modify the General, Servers, and Advanced tabs.

If you are using a different email client, you will need to modify the steps below.





Important – check **Leave a copy of messages on server** and **Remove from server when deleted from 'Deleted Items'**. This will keep messages on the mail server until you explicitly delete them.

9. Add an Outlook identity for oosupport.

This is the identity you will use in this exercise. Go to File/Identities/Add New Identity:

