

HP Operations Orchestration Studio Advanced Authoring

Version 9.00

© 2010 Hewlett-Packard Development Company,
LP



Student Guide



© Copyright 2011 Hewlett-Packard Development Company, L.P.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

This is an HP copyrighted work that may not be reproduced without the written permission of HP. You may not use these materials to deliver training to any person outside of your organization without the written permission of HP.

Printed in USA

HP Operations Orchestration Studio: Advanced Authoring

Student Guide Version 9.00 Rev. B

May 2011

HP Restricted — Contact HP Education for customer training materials.



Contents

Working With Operations and Subflows	3
Looping and Iteration	23
Filtering Flow Data	37
Parallel Processing	56
Using Responses, Rules, and Transitions to Control Flow Execution	72
Remote Action Service	90
Controlling Access to OO Objects	114
Persisting Data Across Flow Runs	127
XML Processing	141
Working With File Systems	153
Working With Email	165

HP Operations Orchestration Studio: Advanced Authoring

Student 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

Refer to the HP Operations Orchestration Advanced Authoring Lab Guide.

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

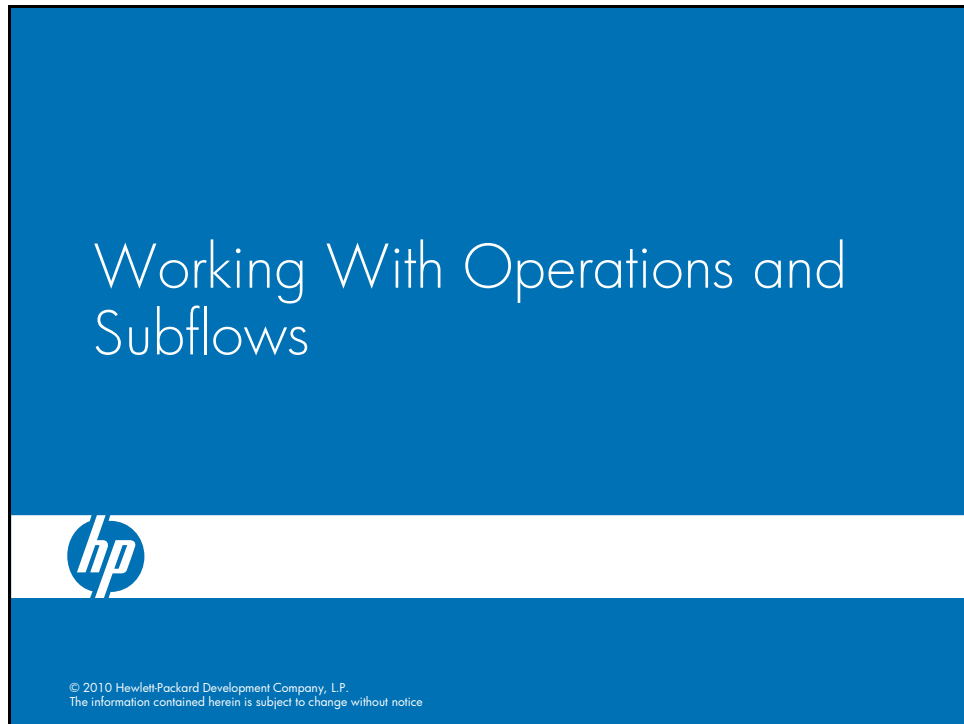
Exercise 8: Persisting Data Across Flow Runs

Exercise 9: XML Processing

Exercise 10: Working With File Systems

Exercise 11: Working With Email

Working With Operations and Subflows



In this module you will learn how to work with Operations and Subflows to control and add content to the OO Library.

Objectives

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

- Explain how operations and subflows are used as steps in a flow
- Modify operation properties
- Create new operations
- Use subflows in a parent flow
- Explain best practices for working with subflows
- Assign step results in a subflow to output fields for use in the parent flow
- Work with a subflow's Properties editor

2

7.5 Rev B



In this module you will learn how to expand your Operations Orchestration content library by adding new operations and subflows. Both operations and subflows can be the basis of steps in a flow, and for this reason they are functionally equivalent in many respects. Learning how to manage both is a key skill in OO.

This module will begin with operations, then focus on subflows. Here you will learn:

- How operations and subflows are used as steps in a flow
- How to modify an operation's properties
- Create new operations, both "local" and iAction-based
- Use subflows in a parent flow
- Explain the best practices for working with subflows
- Use step results in a subflow to assign data to flow output fields that can then be accessed in a parent flow
- Work with a subflow's properties editor

Working With Operations

- A great deal of the content in the OO Library consists of Operations
- Most operations refer to iActions deployed to a Remote Action Server (RAS) while others work locally
- You can modify operation properties or if needed create a new operation
- Most operations are located in the following folders:
 - Integrations
 - Operations
 - Utility Operations
- Steps in flows are simply instances of operations or subflows in the library – one operation can be used in any number of steps

3



The Operations Orchestration consists of individual operations and flows that are built from operations and subflows stored in the Library.

There are essentially two types of operations:

“Local” operations that run natively in OO and do not access a deployed Java or .NET iAction

RAS-based flows that locate a deployed iAction on a Remote Action Server

This module will cover both cases.

Most operations are located in the following folders in the OO Library:

- Integrations
- Operations
- Utility Operations

The Accelerator Packs folder in the Library contains deployable subflows that can be used as is, with modification, or as subflow steps in parent flows.

Examining Operation Properties

To open the Properties Editor:

- Double-click the operation in the Library
- Right-click a step in a flow and select Open Operation

Local Operation

Inputs Summary defines operation parameters

RAS Operation

Inputs Summary locates the iAction, archive, and Remote Action Service to execute

The slide shows the properties editor of the two types of operations available in Operations Orchestration.

The Local Operation, so called because it executes locally to the OO installation, simply does whatever it is designed to do on the on the Central host.

The RAS operation, on the other hand, executes an “iAction” deployed to the Remote Action Service. At runtime, the RAS field determines the RAS Reference or URL to the Remote Action Server. Then the iAction to be executed, specified in the Action Class field, is located in the Archive specified within the operation. iActions can be either Java (.jar) or .NET (DLL) files located in the repository of the RAS host.

Most content in OO is RAS-based so understanding the structure of a RAS operation and the use of the RAS Reference is key to understanding how OO works.

Inputs and Outputs

- Steps inherit inputs from those defined in the operation
- The operation:
 - Takes inputs
 - Does something
 - Produces outputs
- Operation outputs are written to “output fields”
- Flow authors create flow variables from outputs by creating results in the Step Inspector’s Results tab
- All operations produce a success code, output string (stdout), error string (stderr), and failure message
- Additional outputs are defined within the local operation or by the iAction the operation points to
- You can select an operations “primary output”

5

7.5 Rev B



This slide outlines the basic process of providing inputs and handling outputs.

Inputs for an operation are defined within the operation’s properties window. Those inputs are reflected in the step when the operation is referenced in a flow.

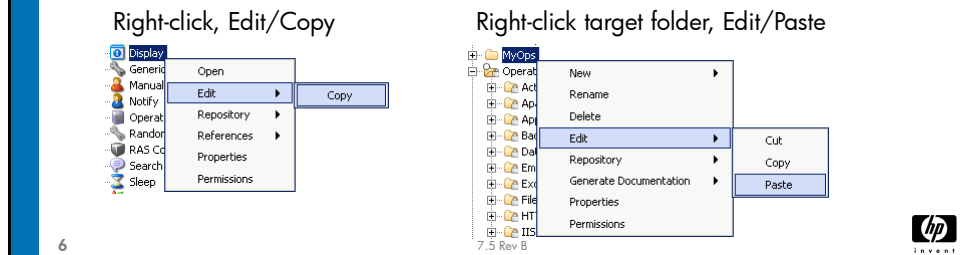
At runtime, OO takes the step inputs and sends them to the operation, which in turn produces outputs. In OO outputs are written to “output fields” which are available to the step but not necessarily written to flow variables. The flow author captures and output and writes it to a flow variable in the step’s Results tab.

All operations produce a success code, and output string (stdout), and error string (stderr) and a failure message. Additional outputs are defined in the iAction that the operation executes.

It’s important to understand the Primary Output, which is essentially the returnResult for a step. Changing the Primary Output to suit the unique needs of a flow is a key skill in OO.

Common Changes to Operation Properties

- It's common practice to change the following operation properties:
 - Primary Output
 - Icon
 - Default RAS Reference (covered in RAS section)
 - Responses (add/remove/modify, covered in Responses section)
- You must work with an operation copy to change its properties



This slide covers some of the most common tasks for working with operations. Authors commonly modify:

- Primary Output
- Icon
- Default RAS Reference
- Responses

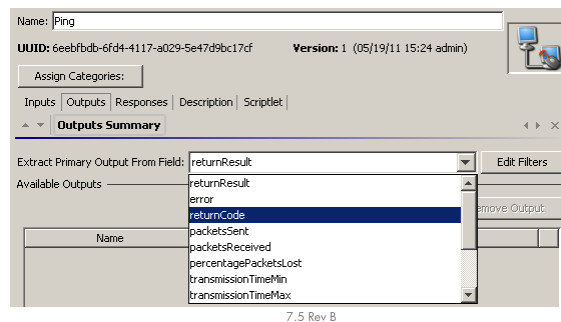
It's critically important to work with a copied version of an operation if you need to make any changes to its properties. Since a single operation can be shared among many flows, changing its properties will be reflected in all flows that refer to it. So if you were to add a Response, for example, all flows that refer to that operation would fail with an Unhandled Responses error. For that reason, most operations are locked in the Library.

To copy an operation, create a target folder in the Library. Then right-click operation and select Copy. Then right-click the target folder and select Paste. The operation is copied and is checked out if you are working in the public repository.

Once you change properties, use that operation only in those flows that require the changed properties.

Changing the Primary Output

- Problem: Ping operation returns the full output string, all you want is the success code
- Solution: Change the primary output
 1. Select Outputs tab in Properties
 2. Select the primary output from the pull-down menu
 3. Save the operation, use it in your flow



One of the more common – and useful – changes to make to an operation is to change the primary output.

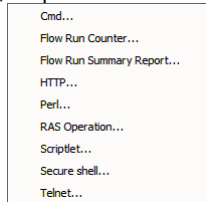
In this example, the Ping operation returns the full output string as its primary output. In your flow, all you are interested in is the success code – did the Ping succeed or not?

To change the primary output, simply open the Properties editor of a copied version of the operation, select the Outputs tab, and choose the primary result from the pull-down menu.

The primary output becomes the default output for the operation and you can use that to write more efficient flows. For example, in a step that follows the Ping step, you can assign a value for an input by selecting “Use Previous Step Result” and the primary output of the previous step will be assigned to that input.

Creating a New Operation

- You can create the following new operations:
 - **Cmd**: Shell commands
 - **Flow Run Counter**: Counts the number of flow runs
 - **Flow Run Summary Report**: Summary report of flow/steps
 - **HTTP**: Put or get operations
 - **Perl**: Call a Perl script
 - **RAS Operation**: Invoke IActions
 - **Scriptlet**: Create a scriptlet used in multiple flows
 - **Secure Shell**: Secure communications
 - **Telnet**: Use telnet protocol
- To create a new operation, right-click a folder, select New Operation, then select the type



8

7.5 Rev B



Operations Orchestration allows you to create new operations in the library. The slide lists the various types of operations you can create.

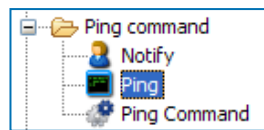
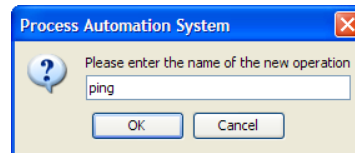
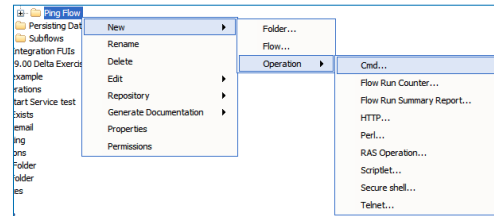
Note that in most cases, you should use content that is already available in the Library rather than creating new content. This is especially true for Flow Run Counter and Summary Report, HTTP, Scriptlet, Secure Shell and Telnet.

But still there are times when it is easier and more convenient to simply create a new operation

To create an operation, simply right-click a folder, select New Operation, and then choose the type to create.

Example: Create a Command-Line Ping Operation

- Right-click the folder where you will be working, select New Operation
- Select the type of operation
 - **Cmd**: Shell commands
- Name the Operation
- The operation's Properties Editor opens



9



In this example you are creating a local cmd operation that executes a Ping command.

Note that any command issued with a cmd operation will execute only on the local Central server.

You simply create the operation using New/Operation and select cmd as the type. Then give it a name and the operation appears in the folder where you created it.

Set cmd Properties

- Double-click to open the Properties tab
- In this example of a cmd operation:

ping \${host} is the shell command to be executed. For example:
`ping hp.com`

Arguments are command line arguments. For example example:
`ping hp.com -n 5`

The operation prompts the user to enter the host to ping and stores in **host** flow variable

When you create a new operation, its Property editor opens and you can then define the command, arguments, and other parameters for the command.

The cmd operation is essentially the same as opening a command or terminal window on the local Central host and executing a command at the command line.

You also need to define any inputs that will be required. At runtime, the inputs are evaluated and then the command is executed against those inputs. Outputs are then provided to the calling step.

Creating New Operations

- Oftentimes you are better off locating content already in the Library than creating new operations
- Here's where to find various operation types and the corresponding Library content:

Operation	Library Location
cmd	Utility Operations/Command Line Builder
Flow Run Counter	Integrations/Hewlett-Packard/Operations Orchestration/Flow Run Counter
Flow Run Summary Report	Integrations/Hewlett-Packard/Operations Orchestration/Flow Run Summary Report
HTTP	Operations/HTTP Client (lots of operations there)
Secure Shell	Operations/Remote Command Execution/SSH
Telnet	Operations/Remote Command Execution/Telnet

11



In most cases you are better off using content already present in the OO library because it has the benefit of reflecting any content changes that may be present.

The SSH and Telnet operations are good examples. The content already present in the Library (in Operations/Network/Remote Command Execution) has important parameters that are not available when you create a new SSH or Telnet operation "by hand."

The slide shows the location of content in the library for a few commonly used types of operations.

What is a Subflow?

- A *subflow* is a flow that serves as a step in a *parent flow*
- Using subflows allows you to simplify flow design by:
 - Separating parts of a flow into manageable pieces
 - Focusing a subflow on a single, well-defined task
 - Testing/debugging portions of a flow individually
- Subflows should be designed for reusability
 - Focus on commonly used tasks
 - Design inputs and outputs to facilitate reusability
- Variables in a subflow are not automatically available to the parent flow
 - Create a step result in the subflow and assign a flow variable in the parent flow based on the step result
- Debugging tools allow you to step into and out of subflows during testing

12



Similar to an operation, a Subflow takes inputs, processes them, and returns outputs to a parent flow.

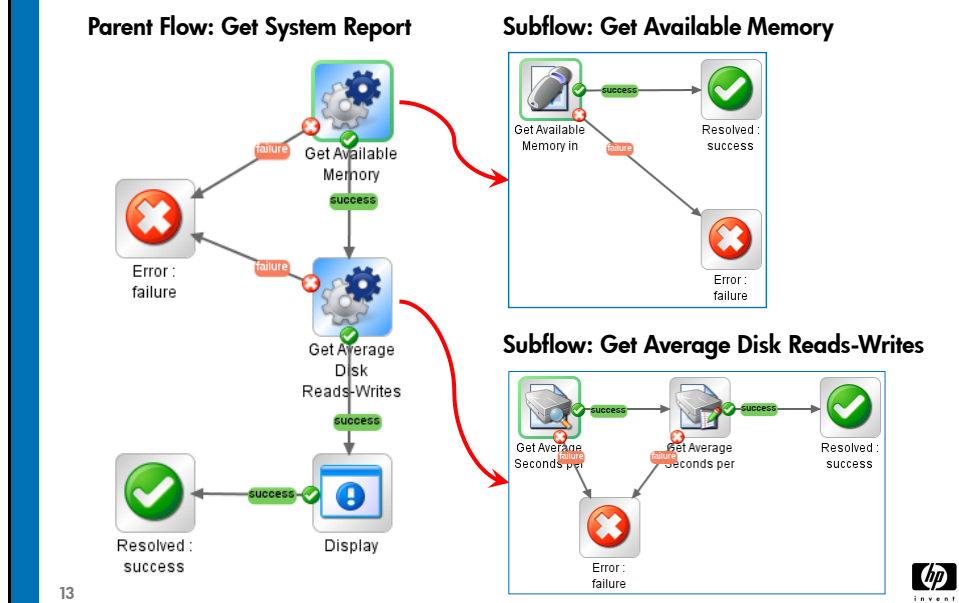
Using subflows is one of the easiest and most effective ways of building a custom library of content in OO. Additionally, the Accelerator Packs folder contains many predefined flows that can be used as-is, copied and modified, or used as subflows.

Subflows allow you to simplify flow design by breaking various tasks into manageable pieces. It's always a good idea to design subflows for reusability.

Recall that operations write outputs to "output fields." You often need to do the same thing with your subflows – write results from various subflow steps to flow output fields that can then be accessed by the parent flow. This is one of the important ways that subflows can essentially mimic native operations in OO.

The OO debugger is designed to give you complete step-by-step execution control over subflows.

Example: Subflows are Steps in Parent Flow

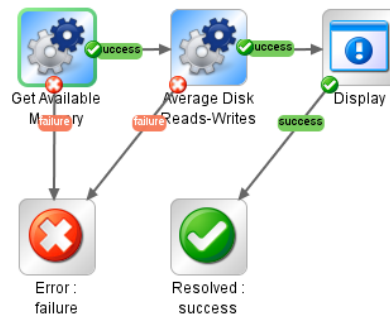


In this example, the two subflows basically get available memory and disk performance statistics. Those values are passed up to the parent flow, which displays a little performance report.

Passing Data From a Subflow to a Parent Flow

- In this flow, local variables defined in the subflows (Get Available Memory and Average Disk Reads-Writes) are not available in the parent flow – they are local to the subflow steps
- The solution is to use a Result to assign the local subflow variable to the parent flow's global context

Parent Flow: Get System Report



14



New flow authors are often confused when they define a variable in a subflow but that variable can't be "seen" in the parent flow.

That is because flow variables defined within a subflow, while global to the subflow, are local variables to the parent flow, meaning they are only available within the context of the subflow step in which they are defined.

To make subflow variables globally available in a parent flow, you use results in the subflow and parent flow to "export" data from the subflow and "import" it into the global context of the parent flow.

Assigning Results in Subflows

- In Subflows, Assign Results to Flow Output Fields:

The slide displays three screenshots of the HP Flow Director Inspector and two flow diagrams illustrating result assignment in subflows.

Inspector Screenshot 1: Get Available Memory in Megabytes

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

Inspector Screenshot 2: Get Average Seconds per Disk Read

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

Inspector Screenshot 3: Get Average Seconds per Disk Write

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

Flow Diagram 1: A subflow step 'Get Available Memory in' (green icon) has a 'success' path leading to a 'Resolved: success' node (green checkmark) and a 'failure' path leading to an 'Error: failure' node (red X).

Flow Diagram 2: A subflow step 'Get Average Seconds per' (green icon) has a 'success' path leading to a 'Resolved: success' node (green checkmark) and a 'failure' path leading to an 'Error: failure' node (red X).

To make flow variables available to the parent flow, the first step is to create a result in the subflow and assign it to a Flow Output Field. Previously you have assigned outputs to Flow Variables. Assigning them to a Flow Output Field means they can be subsequently picked up by a result in the parent flow.

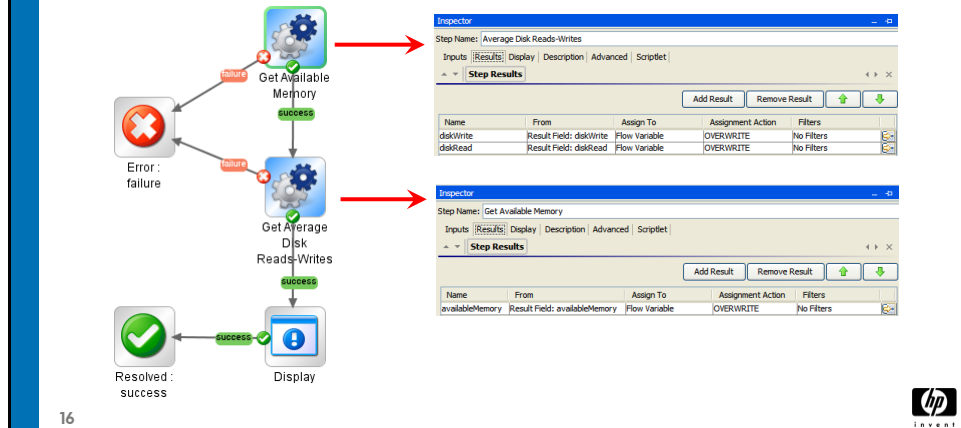
In the examples shown on the slide, the Get Available Memory step has a result named availableMemory, based on the output field "value", and that result is assigned to a Flow Output Field.

Similarly the diskRead and diskWrite results in the two disk operations are assigned to Flow Output Fields.

Assigning the result in the subflow step(s) to Flow Output Fields is the first step in making the output data globally available in the parent flow.

Assigning Results in Parent Flow

- Once results are assigned to Flow Output Fields in subflow steps, open the Step Inspector for each subflow step and assign those results to flow variables
- Flow data is now globally available in the parent flow

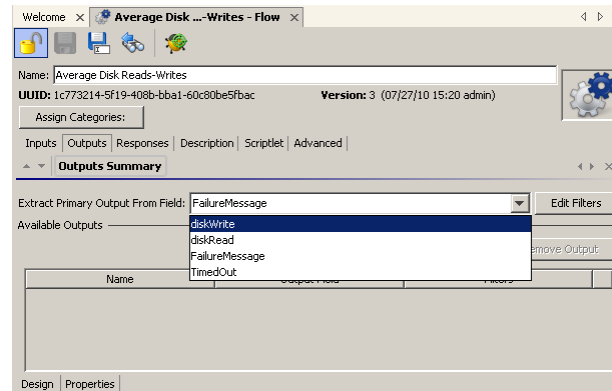


Once the results are defined and assigned to Flow Output Fields in the subflows, you create results for the corresponding subflow steps in the parent flow and assign the subflow results to Flow Variables, as shown on the slide.

Once you assign the Flow Output Field results defined in the subflows to Flow Variables in the parent flow, that data is now globally available for use in the parent flow and can be passed to subsequent steps in the flow.

Working With Subflow Properties

- Flows/subflows have a Properties Editor
- Select the Properties tab at the bottom of the Studio window
- You can define Flow Inputs and Outputs, including Primary Output
- Example: Specifying the Primary Output for a subflow



17

7.5 Rev B



You can modify the properties of a subflow. The Properties Editor for a subflow is the same as the Properties Editor for an operation.

Here you can specify “flow inputs,” set the primary output, change responses, set the default description, and other parameters that define the behavior of the subflow.

To access a flow’s Properties Editor, select the Properties tab at the bottom of the display. To return to the design panel of your flow, select the Design tab.

Using Debug Tools With Subflows



Step Over (F6)

- Use to run the flow one step at a time



Step Into (F5)

- Step into a subflow for step-by-step execution



Step Out (F7)

- Step out of a subflow and back into the main flow



Play (F11)

- Run everything including subflows

Best Practice:

- Debug the subflow(s) before the main flow

18



Debugging a flow that contains subflows can be different from flows that do not contain subflows.

When your flow contains one or more subflows, you have the option of “stepping into” a subflow and executing it in step-by-step mode. You can also “step out” of a subflow and return to the parent flow. These options apply only when executing the flow in step-by-step mode.

The Step Over button simply advances from step to step in the parent flow regardless of whether it is based on a subflow or an individual operation.

Any any time, you can always press the Play button to run the remaining steps to completion.

A best practice is to debug subflows and make sure they are working properly before using them as steps in a parent flow.

Summary: Subflows

- A subflow is a flow that executes at a step in a parent flow
- Subflows allow you to simplify design and build re-usable components that perform more complex tasks than individual operations
- Variables used in a subflow may not be immediately available to the parent flow
 - Create a Step Result in the subflow
 - Assign a flow variable in the parent flow based on the subflow's step result(s)
- Subflows should be fully tested and debugged before incorporating into a parent flow
- The Studio debugger allows you to step into and out of subflows during debugging

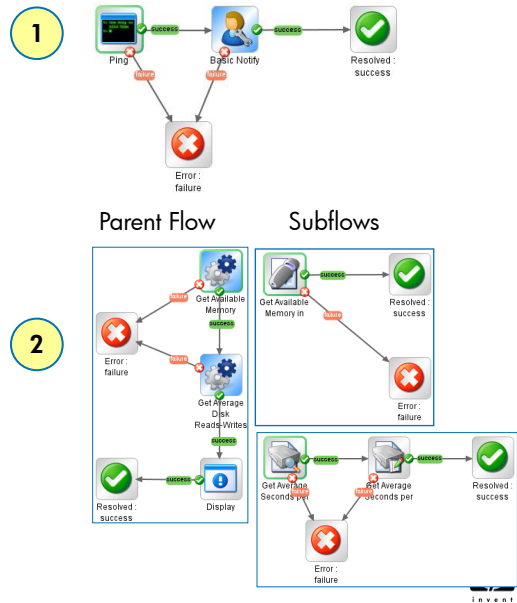
19



To summarize, subflows are a powerful tool that allow you to expand your library of OO content. It's important to remember when working with subflows that you are creating content that is analogous to an operation, meaning you need to be mindful of things like its primary output, when to write data to flow output fields, and other aspects of authoring.

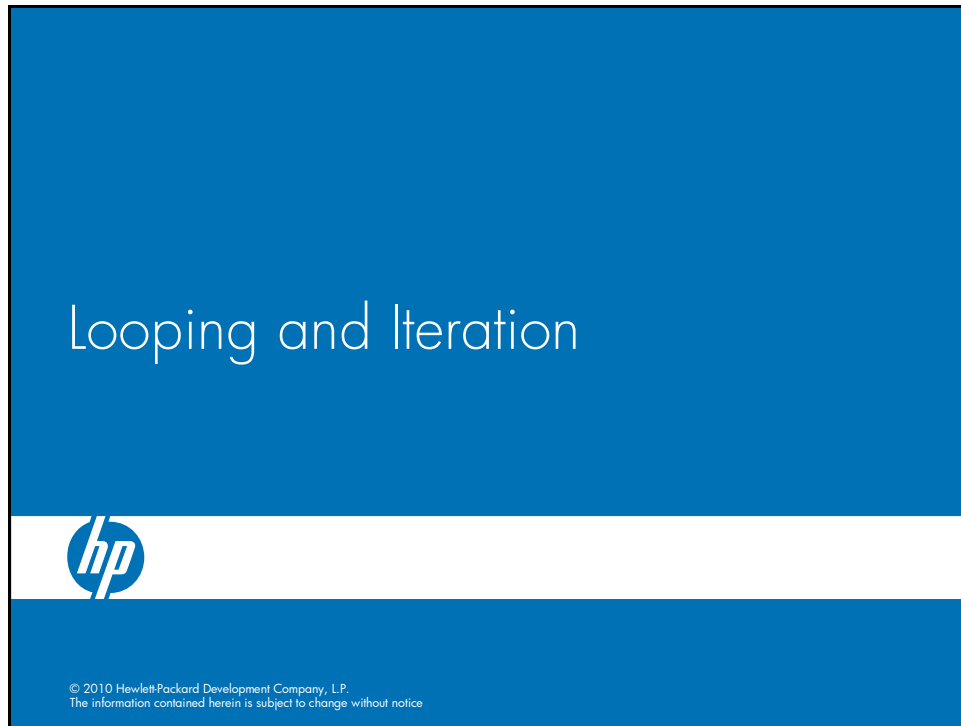
Exercise: Working With Operations and Subflows

- Create a new operation and use it in a flow
- Create a flow based on two subflows



In the exercises for this module you will create a cmd operation and use it in a flow. Then you will create two subflows and incorporate them into a parent flow.

Looping and Iteration



In this section you will learn how Operations Orchestration handles looping, iteration, and related tasks like compiling lists based on iterative processes.

Objectives

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

- Explain how OO handles iteration and related tasks such as list compilation
- Locate content in the OO library for performing looping, iteration, and related tasks
- Author a flow that uses iteration and list compilation to compile a list of random numbers
- Author a flow that uses the Counter operation to compile an incremented list

2



Review the objectives for this section. The OO Library contains a folder of content designed specifically for handling looping and iteration, along with related tasks. The focus of this section is on that content in the OO library.

In the exercise you will build a number of flows that use the looping and iteration capabilities.

Looping and Iteration in OO

- OO iteration operations in Utility Operations/Looping:



Counter - *Counts from one number to another number*



Loop - *Loops a number of times*



List Iterator - *Iterates through a list of values*



Selection List Iterate - *Iterates through a selection list*

- These operations are often paired with list operations:



List Appender – *Appends items to a list*



List Prepend – *Adds items to the front of a list*

3

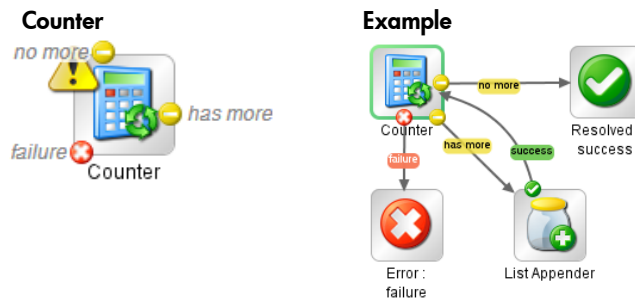


The Operations Orchestration looping and iteration content is located in Utility Operations/Looping. Here you will find:

- Counter – Counts from number a to number b
- Loop – Simply loops through a process for the specified number of times
- List Iterator – iterates through a list of values
- Selection List Iterate – iterates through a selection list defined in the Configuration folder of the OO library
- The iteration and looping operations are often (usually) paired with operations that generate lists based on the results of the iterative process:
- List Appender - Adds items to a list, which is usually a named flow variable based on a context key
- List Prepend – adds items to the front of a list

Working With Iterating Operations

- Operations like Counter, Loop, and other iterating OO content have the same general format:
 - Has More: This response links to the step or steps that are repeated
 - The step(s) in turn link back to Counter
 - No More: This response links to the step to execute when all iterations are completed



It is very important to learn how to identify and deal with content in the library that iterates. A lot of different operations in the OO library (XML operations or file system operations, for example) are iterative functions. All iterative content has the same general form:

1. A has more result that links to other steps to be executed repeatedly or iteratively
2. A no more result that the flow follows once all of the iteration conditions have been satisfied

Whenever you see has more and no more results on a step, you know you are dealing with iterative content.

The general rule is to link the has more result to whatever steps you want to repeat iteratively, and the no more result to the next steps to continue with once the iteration is complete.

The slide shows a simple example: The Counter's has more result links to the List Appender repeatedly for the specified number of counts. When the counting is completed and the list is compiled, the no more result links to the Resolved return step, where the flow ends.

Notice that the step(s) to be iterated link back to the calling step, in this case, Counter.

Working With Lists

- List Appender and List Prepend are often used in conjunction with an iterating operation
- Lists are often compiled and then assigned to a flow variable
- You specify the delimiter to put between items in the list
- The compiled list can then be handled in many ways:
 - A source of values for a multi-instance step
 - A selection list
 - A simple list that can be iterated through later in your flow

5

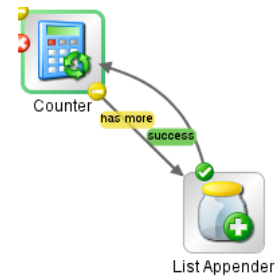


Iterative content is very commonly linked to list compilation steps like List Appender and List Prepend.

In the List operations you typically create a named flow variable, which contains the list. The list is delimited by whatever delimiter you choose when you set up the flow variable to contain the list.

Compiling the List: List Appender

- To use List Appender:
 - Success response on List Appender links back to the iterating step, Counter in this example
- Configuring List Appender Inputs:
 - keyName: the name for the context key – this becomes the name of the flow variable that contains the list
 - resultText: The value to add to the list, often Result of Previous Step
 - Delimiter: The separator for list items



Inspector

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: numberList
resultText	<input checked="" type="checkbox"/>	Single Value	Result of previous step
delimiter	<input type="checkbox"/>	Single Value	Value: ,

In this example, List Appender is defining a flow variable named numberList. This flow variable is the constant value of the keyName input – that constant value becomes the name of the flow variable that contains the list. Think of it this way: keyName is the container for the list items, and numberList is the label used to refer to that list of items.

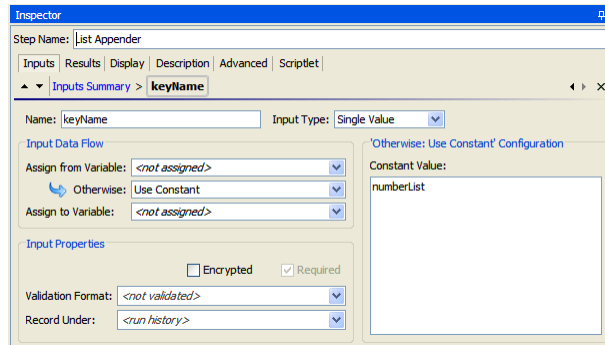
The resultText is the text to be written to the list. This is typically the result of the previous step.

The delimiter is simply the character or characters you want to separate values with. For example, using a comma as the delimiter creates a CSV (comma-separated value) list that when written to a file could be opened by a spreadsheet program.

As a reminder, you would simply link the has more result on the iteration step (Counter in the example) to the List Appender, which in turn is linked back to the Counter step. Note that the iterative process called by Counter could contain multiple steps. In that case, the last step in the process is the one you would link back to Counter.

Assigning the keyName

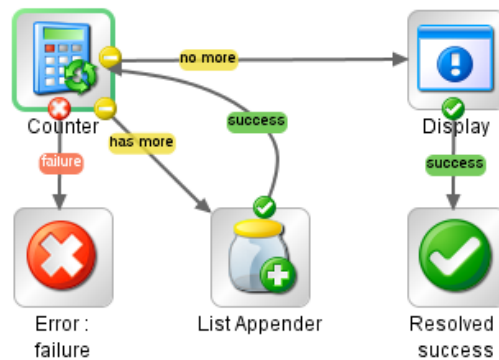
- To assign a name for the context key, open the keyName input editor and give it a constant value which is the name of the flow variable for your list
- Assign from and Assign to are not assigned
- Think of this as putting a label on a jar – the context key is the jar and the value assigned to keyName is the label
- Subsequently you refer to the context key with the flow variable named numberList



The slide shows more about defining a constant value for keyName. That constant value becomes the named flow variable that references the items stored in the keyName context key. Once defined in this way, you would refer to numberList, not keyName, to access the values stored in the list.

Example: Loop and List Iterator

- Counter prompts the user to enter a number
- Counter then counts to that number
- List appender adds each number to a list
- Display shows the list to the user

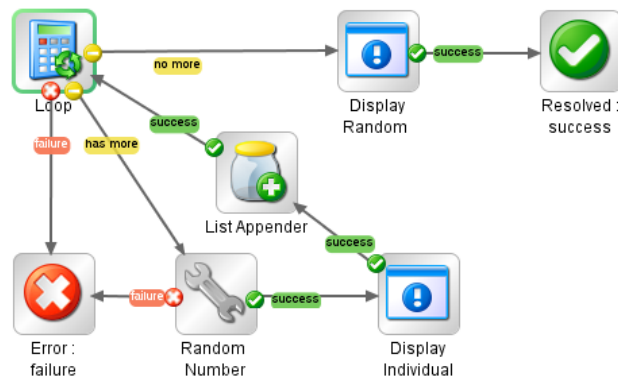


8

In this simple example, the Counter step prompts for a number to count to. It then iterates that number of times, adding values to List Appender until counting is finished. At that point, the no more result is active and the flow proceeds to the Display step, which displays the compiled list.

Example: Compile a List of Random Numbers

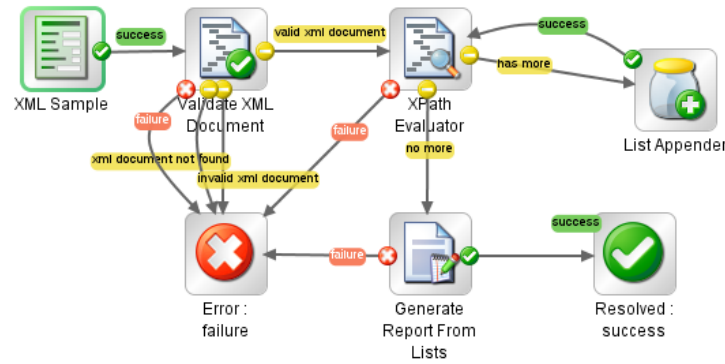
- Loop iterates a specified number of times
- Random Number is called at each iteration
- The random number is displayed then appended to a list
- When the list is complete the entire list is displayed



This example shows a number of steps comprising the iterative process. Here Loop generates a random number, displays that number, and then appends it to a list. When the list is complete the flow follows the no more path to Display Random, which displays the list of random numbers. The point here is to show how to link an iterative process that contains multiple steps back to the calling step, which in this case is Loop.

Example: Process XML

- Validate step checks that it received a valid XML document
- XPath Evaluator iterates through the document and extracts elements that match an XPath Expression
- List Appender compiles “hits” to a list
- Generate Report generates a report when iteration is complete



10



The XML Processing content in the OO Library's Utility Operations XML Processing folder is often iterative in nature. For example, the XPath Evaluator iterates through an XML file searching for items that match an XPath expression, appending found items to a list. When the list compilation is complete, it generates a report based on the retrieved data.

Other Iterating OO Content

- Many operations in the OO library iterate:
 - Read From File
 - XML operations – XML Element Filter, XML Get Attributes, XPath Evaluator
 - Other content
- The basic method of handling these operations is the same:
 - Has More response links to a subsequent step
 - No More response links to the step to use when iteration is complete
 - Results are often compiled to a list

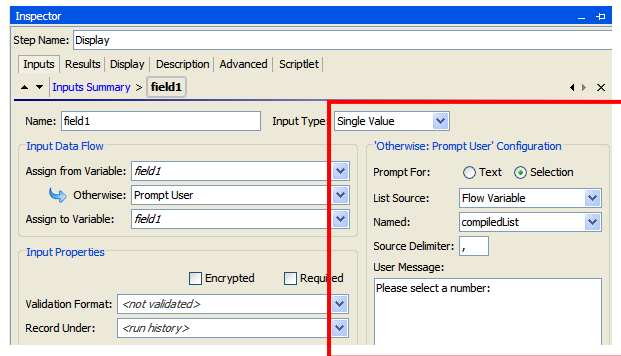
11



This slide reviews the other content in the OO library that uses iteration – remember that the basic way of handling iteration is the same for all content iterates.

Compiling a Selection List

- Another useful task is to compile a selection list
- For user selection input:
 - In the Input Editor, select Prompt For a Selection
 - List Source: select Flow Variable
 - Named: is the name of the flow variable that contains your list items and delimiter is the item delimiter in that list



12



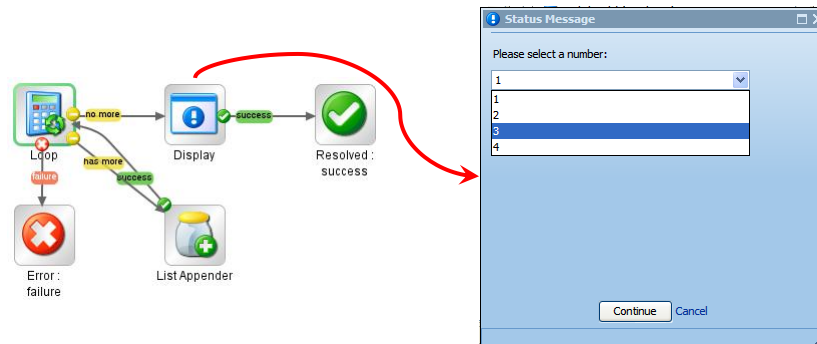
Another useful task is to compile a selection list based on a list compiled from an iterative process.

In this example, the Input Type is a Single Value based on a Selection. The List Source is a flow variable named compiled List, and the source delimiter is a comma. The User Message is the message displayed to the user when the list is presented.

When you run this flow, the user will be presented with a pull-down menu labeled “Please select a number”. Opening that menu shows the list of compiled numbers. Subsequent actions of the flow can be specified depending on the selection the user makes.

Example: Using a Selection List

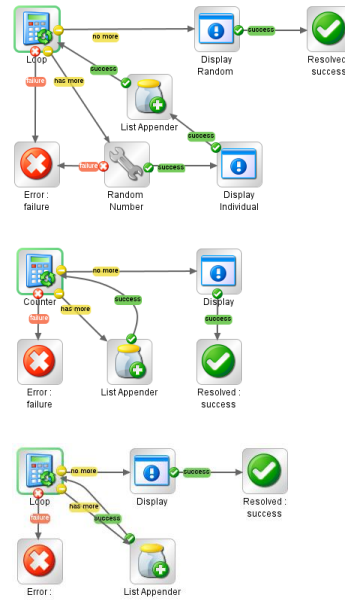
- Loop and List appender compile a list of numbers
- Display presents the numbers as a selection list
- The same technique can be used to present any selection to a user



Here is an example of using a selection list compiled from a Loop and List Appender combination.

Exercise: Looping and Iteration

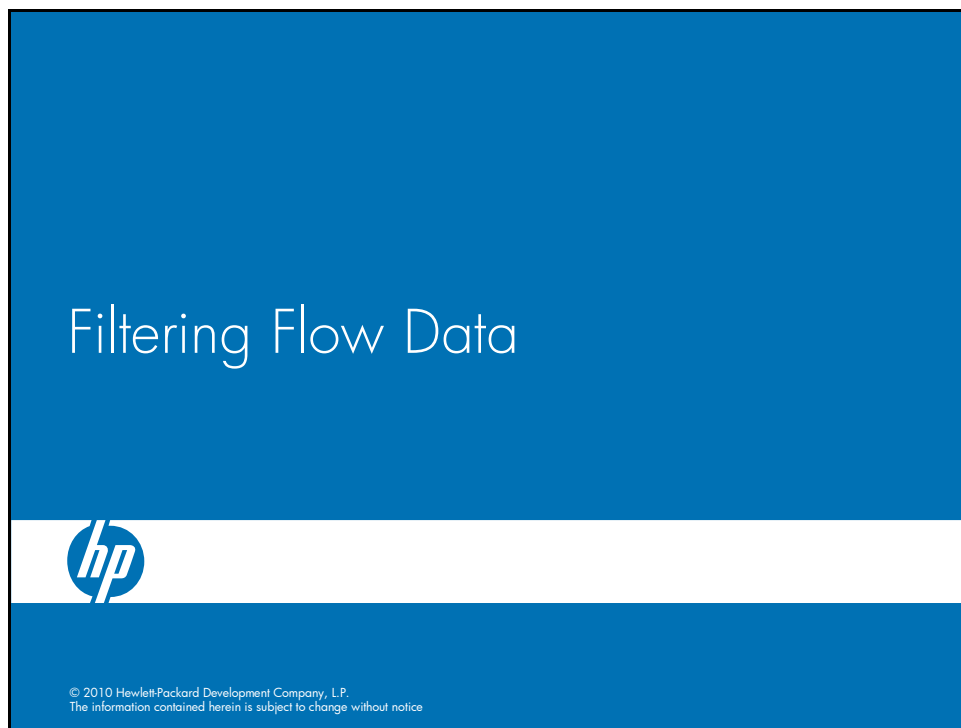
- Author flows that:
 - Use Loop to compile a list of random numbers
 - Use Counter to compile a list
 - Use a Selection List



14

In the exercise you will author three flows that use looping and iteration to perform a variety of tasks.

Filtering Flow Data



In this section you will learn how to extract data from complex output using OO filters.

Objectives

By the end of this module, you will be able to:

- Use filters to extract data from inputs and results
- Use regular expressions to filter flow data and store it in a flow variable
- Access the Filter Editor to create and test your regular expression
- Use scriptlets to manipulate flow data
- Describe the unique methods available in OO
- Create, test, and deploy a scriptlet
- Save a scriptlet for later use
- Use a pre-defined scriptlet in a step

2



Review the objectives. The goal of this module is to learn how to apply pre-defined filters to results. Additionally you will learn how to use regular expressions and scriptlets to filter flow data.

Filtering Results

- A result is part of the operation's output
 - success code
 - output string
 - error string
 - failure message
- Operations generate other outputs which are not automatically assigned to flow variables
- Use Results to assign outputs to flow variables
- Results can be filtered to extract only the needed data
- Evaluating results lets you:
 - Conditionally determine the next step in a flow
 - Pass data to other steps in the flow or to other flows
- Filtering results allows you to extract and modify parts of the result

3



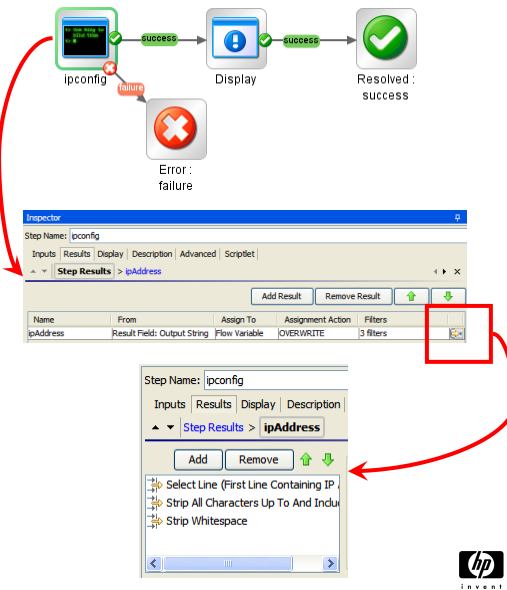
The output of an operation is referred to as a result.

Each operation produces a success code, output string, error string, and failure message. By using the Results tab in the Step Inspector, you can grab any output and assign it to a result, which is usually assigned to a flow variable. Once assigned to the flow variable it is available to subsequent steps in the flow.

Evaluating results allows you to determine the next step in a flow, and filters extract and/or modify portions of the result. Results can often be large blocks of data, so learning how to filter results is a necessary skill in Operations Orchestration.

Example: Filtering a Result

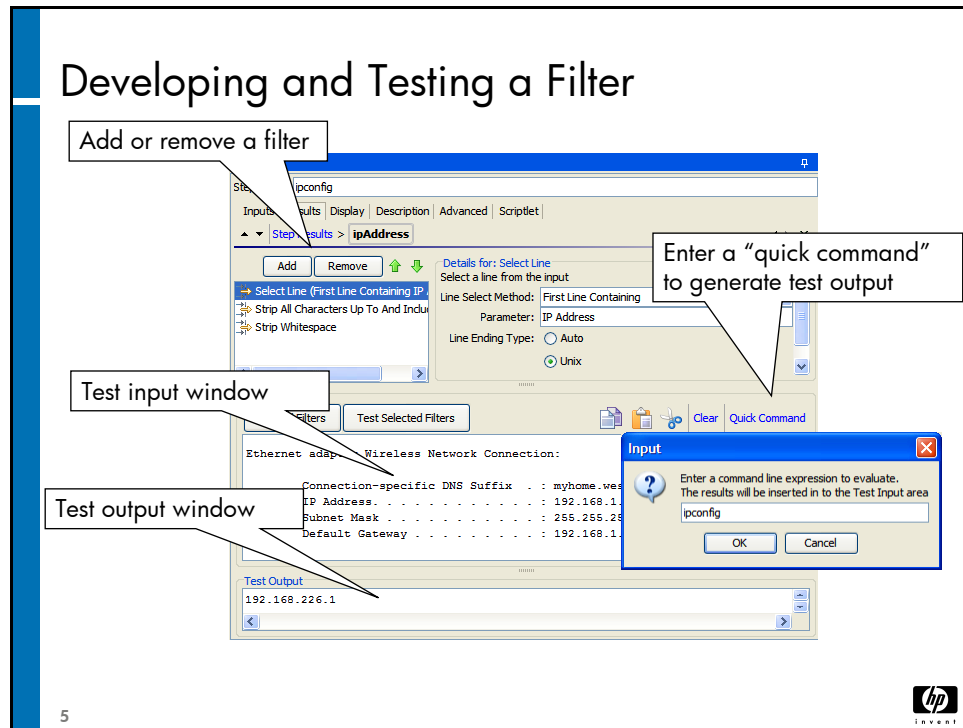
- Ipconfig step uses three native OO filters to extract just the IP address from raw output
- The ipAddress result is the flow variable that is filtered to contain just the IP address
- To access the filter editor:
 - Click the Results tab, create your result if necessary
 - Click the Filter Editor to the right of the result
 - Add and test filters



In the example, the ipconfig step, which produces a lot of output, is using three filters to extract an IP address.

Filters are applied to results. Clicking the yellow arrow at the right of the result opens the Filter Editor.

In the Filter Editor you have a simple interface for setting up your filters and testing them, either individually or all at once.



The filter editor is a compact application for assigning and testing filters. A key feature is the ability to use sample output for testing the filter.

To generate sample data, you can either:

Click Quick Command and enter the command you want to test against, like ping or ipconfig. This executes the specified command and places the output in the test input window.

You can also simply copy data, from a terminal window or text file for example, and paste it into the test input window.

Once you have your test data, you can begin adding and testing filters. Test output is shown in the lower window.

It is a common practice to use as many filters as needed to get the desired result. For you can initially filter your data to extract a single line (like the first or last line of output, or a line that starts with a particular string) and then apply a second, third, or fourth filter to extract the data you want.

The Advanced Authoring course covers using Regular Expressions and Scriptlets to filter data.

Define and Test the Filter

- Click Add, select the filter
- Select filter parameters
- Test

The screenshot illustrates the process of defining and testing a filter in the Hp Operations Orchestration Studio. The main window shows the 'Inspector' tab for a step named 'ipconfig'. The 'Step Results' section is expanded, showing the 'ipAddress' filter. A 'Select Filter' dialog is open, allowing the user to choose a filter to add. The 'Extract Number' filter is selected. The 'Test Filter Input' section shows the raw output of the 'ipconfig' command, which includes network configuration details. The 'Test Output' section shows the filtered output, which is the last line of the raw output: '192.168.226.1'.

The slide shows an example of defining and testing a single filter against ping output that was generated with a Quick Command.

In this example the filter is extracting the last line of output.

What is a Regular Expression?

- OO provides a mechanism for filtering operation output or step results using *regular expressions* (or regex)
- This section focuses on how to incorporate a regular expression into a filter
 - Constructing regular expressions is a large topic and is not covered in this course
 - Refer to online resources to learn how to construct regular expressions

7



A Regular Expression is a mechanism for filtering OO data (output) using regular express (regex) syntax that confirms to a number of popular and widely used standards, like Perl5.

This module focuses on incorporating a regular expression into an OO filter. It does not tell you everything you need to know about regex, which is a very large topic.

However there are many resources on the Internet that you can use to learn more about regex.

Example

Filter IP address from ping output

Problem: Need to store IP address in a flow variable

1. Add a result named IPAddress, open the Filter Editor
2. In the Filter Editor, click Add
3. Select Regular Expression

The screenshot shows the 'Inspector' window for a step named 'NRAS Ping'. The 'Results' tab is selected, showing a table with columns: Name, From, Assign To, Assignment Ac..., and Filters. A result named 'IPAddress' is listed, with 'From' set to 'Result Field: output', 'Assign To' set to 'Flow Variable', and 'Filters' set to 'OVERWRITE'. A yellow circle with the number '1' points to the 'Filters' column.

The 'Filter Editor' window is open, showing a list of filter types. A yellow circle with the number '2' points to the 'Add' button. The 'Test Filter Input' section shows a table with columns: PID, PPID, PUID, W2PID, TTY, UID, STIME, and COMMAND. The 'Test Output' section is empty.

The 'Select Filter' dialog box is open, showing a list of filter types. A yellow circle with the number '3' points to the 'RegularExpression' option.

8

In the example shown on the slide, you are using a regular expression to filter an IP address from ping output.

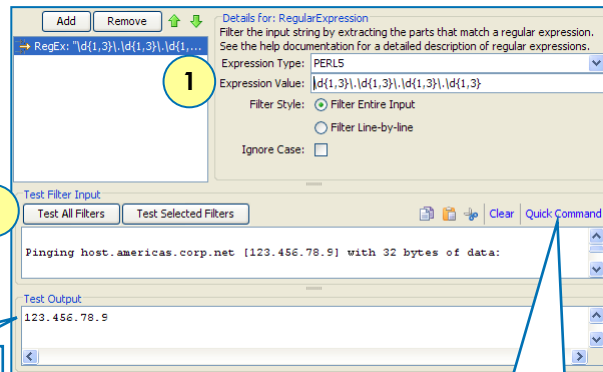
You use the filter editor to do this, which was introduced in the Essentials class.

To create a regex filter, just select Regular Expression as a filter type in the Filter Editor.

Testing the Regular Expression

1. Enter your regular expression into Expression Value
2. Click Test All or Test Selected Filters

Filtered output – this value is stored in the Flow Variable *IPAddress*



Use Quick Command if needed to generate sample output

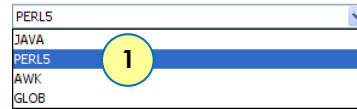
The slide shows a regular expression (Expression Value) that is being tested against ping output that was generated with a Quick Command.

The Test Output indicates that the regex was successful.

Regular Expression Filter Options

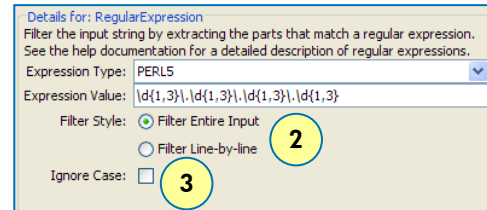
1. Regex type

- Java
- Perl 5
- Unix AWK
- GLOB (simple)



2. Filter Style

- Filter Entire Input: Returns the 1st instance
- Filter Line-by-line: Returns all instances



3. Ignore Case

It's important to note the options available for working with regex in OO.

You can select the type from those shown on the slide – these are presented as a pull-down menu.

The Filter Style allows you to filter the entire output, which returns the first instance it finds, or line-by-line, which returns all instances.

You can also ignore the case of filtered data.

Constructing a Regular Expression

Wildcards

Wildcard	Uses
^	Matches the beginning of a string
\$	Matches the end of a string
.	Any character except newline
\b	Word boundary
\B	Any except a word boundary
\d	Any digit 0-9
\D	Any non-digit
\n	Newline
\r	Carriage return
\s	Any white space character
\S	Any non-white space character
\t	Tab
\w	Any letter, number or underscore
\W	Anything except a letter, number or underscore

Modifiers

Modifier	Effect
*	Match zero or more
+	Match one or more
?	Match zero or one
{n}	Match exactly n occurrences
{n,}	Match n or more occurrences
{n,m}	Match between n and m occurrences
[abc]	Match either a, b, or c
[^abc]	Match anything except a, b or c
[a-c]	Match anything between a and c
a b	Match a or b
\	Escape a special character (for example \. Means '.' not match anything)

11



This slide is basically a small “cheat sheet” for constructing a regular expression.

It's important to refer to resources found on the Internet for detailed information on regex.

Summary: Regular Expressions

- Regular Expressions provide a powerful way to filter operation output or step results and store filtered data in a global or local variable
- You create and test regular expressions in the Filter Editor
- OO supports Java, AWK, Perl5 (the default), and GLOB regular expression formats

12



Review the summary for regular expressions.

Scriptlets

What is a Scriptlet?

- A *Scriptlet* is a Javascript (Rhino) or Perl (Sleep) script for manipulating flow data
- Scriptlets:
 - Offer greater flexibility and capability than simply filtering data
 - Allow you to extend the capability of operations
 - Provide a familiar coding environment
 - Allow you to create local/global variables and store values for use in the flow
 - Can be stored in the Library for later use
 - Can be shared among different flows by using the Scriptlet operation
- Limitation
 - Scriptlets should be small, 64k or less

13



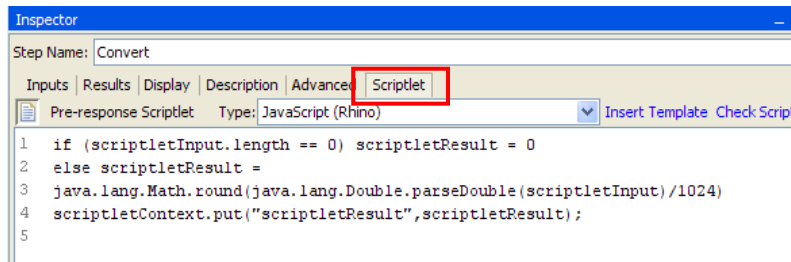
A Scriptlet is a Javascript (Rhino or Javascript-within-Java) or Perl (Sleep) script that you can use to manipulate flow data and variables.

Scriptlets offer the greatest amount of flexibility in managing flow data but also require that you know how to use at least one of the supported scripting languages.

Review the advantages of using scriptlets from the slide and note the 64k size limitation, which means very large scriptlets cannot be used in OO.

Accessing the Scriptlet Editor

- To access the Scriptlet editor, click the Scriptlet tab
- Scriptlets can be assigned to any step in a flow



14

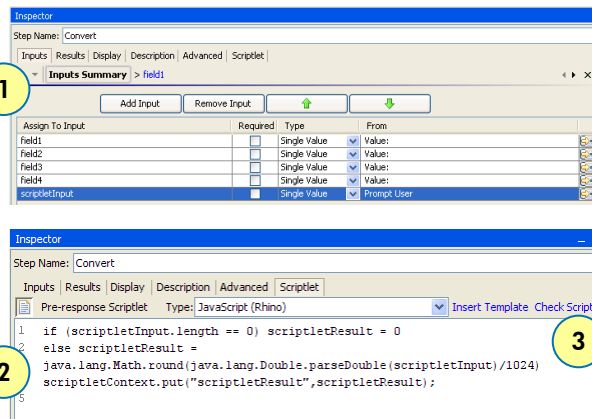


All steps have a Scriptlet tab which is where you access the Scriptlet editor. Thus you can assign a scriptlet to any step in a flow.

Using Scriptlets

Example: Convert bytes to Kbytes

1. Convert step prompts user for a value, stores it in *scriptletInput* flow variable
2. Scriptlet:
 - Checks whether *scriptletInput* is empty
 - If not, divides by 1024
 - Creates a local variable, *scriptletResult*, and stores the result there
3. Select Check Script to validate your script



The example shows a simple scriptlet that converts a number of bytes into kilobytes (kbytes).

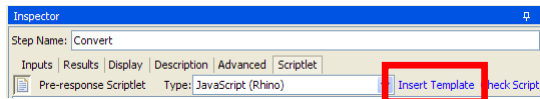
The scriptlet:

- Checks whether *scriptletInput* is empty
- If not it divides the value by 1024
- Creates a local variable, *scriptletResult*, and stores the result there

Once you have entered your scriptlet you can use the Check Script button to see if your code is valid.

Scriptlet Methods

Select **Insert Template**
To see a summary of methods



<code>myData = inputName;</code>	Access an input defined in Inputs panel
<code>myContextData = scriptletContext.get("myContextKey");</code>	Get the context key from the context
<code>myContextData = scriptletContext.getLocal("myContextKey");</code>	Get the context key from the local context
<code>code = parseInt(scriptletRawResult['Code']);</code>	Get the return code from a command line or script
<code>data = scriptletRawResult['Output String'];</code>	Access the output of an operation
<code>error = scriptletRawResult['Error String'];</code>	Access the error string from an operation
<code>scriptletResponse = "Success";</code>	Set the response of an operation
<code>scriptletResult = "Your Result Here";</code>	Set the result of an operation
<code>scriptletContext.putGlobal("Output", data);</code>	Create a global variable
<code>scriptletContext.put("LocalVariable", "LocalValue");</code>	Create local variable


16

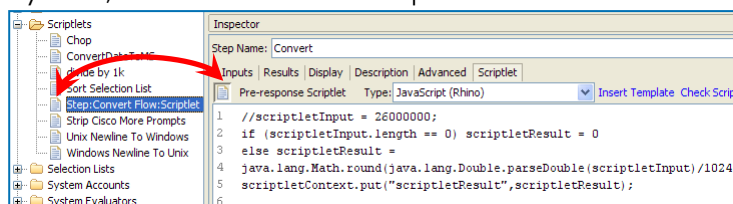


OO has unique methods for managing flow data.

To see a summary of these methods, click the Insert Template button which is basically a short primer on the available methods and what they do.

Saving and Reusing Scriptlets

1. To save a scriptlet to the Configuration/Scriptlets folder
 - Select the scriptlet
 - Click the Scriptlet icon  and drag to the Configuration\Scriptlets folder
 - Name the scriptlet.
2. To reuse a scriptlet from the Configuration folder
 - Open the Configuration\Scriptlets folder.
 - Drag the scriptlet you want to use from the folder into the scriptlet editor.
 - To change from a Configuration\Scriptlets scriptlet to one that you write yourself, click Switch to a custom scriptlet



17



Scriptlets can be saved and reused.

To save a scriptlet in the Library, simply drag the icon in the Inspector window into the Scriptlets folder in your Configuration folder in the Library.

To assign a predefined (saved) scriptlet, drag it from the library onto the icon in the Inspector window.

Summary: Scriptlets

- Scriptlets allow you to use Javascript or Perl to manipulate flow data
- Use the Scriptlet tab in a step to access the Scriptlet editor, where you can write, validate, and test your scriptlet
- Scriptlets can be saved in the Library for later use
- You can create reusable operations around scriptlets so they can be used by multiple flows

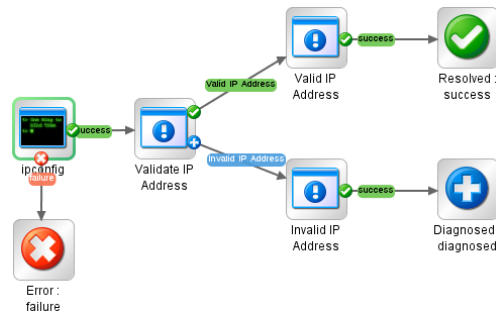
18



Review the summary and ask if there are any questions before proceeding to the exercise.

Exercise: Filtering Data

- Author flows that filter and modify a flow variable using:
 - Native OO filters
 - A Regular Expression
 - A Scriptlet
- Develop a flow that selects an execution path based on filtered output in a Validate step

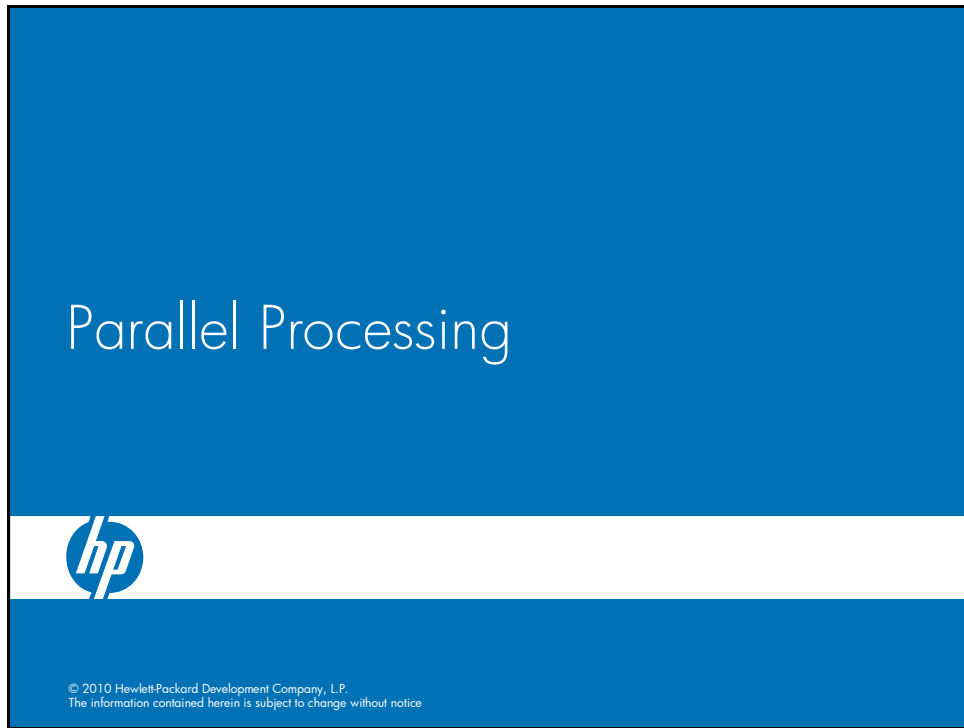


19



In the exercise you will develop a filters that uses a regular expression and a scriptlet.

Parallel Processing



In this section you will learn how to incorporate parallel processing into your flows to increase efficiency and execution speed.

Objectives

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

- Explain how to use the following multi-processing capabilities in your flows:
 - Multi-instance step
 - Parallel-split step
 - Nonblocking step
- Describe how flow data is handled in multi-processing operations
- Create a flow that uses a multi-instance step
- Create a flow that uses a parallel-split step

2



Review the objectives for this section. Here you will learn how to use three methods of parallel processing: Multi-instance steps, Parallel-split steps, and Nonblocking steps.

Working With Parallel Processing

- Parallel Processing lets you author flows that run in simultaneous execution paths:
 - Multi-Instance Step
 - A multi-instance step starts many simultaneously running instances of a step
 - Use for applying the same actions on multiple targets simultaneously
 - Parallel-Split Step
 - A parallel-split step divides flow execution into parallel, simultaneously executing paths
 - Nonblocking Step
 - A nonblocking step allows the flow run to continue while the step executes
- Parallel Processing gives you many design options to meet unique, specific needs

3



With parallel processing you can author flows that execute along simultaneous execution paths. There are three types as listed on the slide. Each of these is examined in detail in this module.

Parallel processing opens up many design possibilities and allows you to design more efficient flows.

Multi-Instance Steps

What is a Multi-Instance Step?

- A multi-instance step starts many simultaneously running instances of a step
- Use for applying the same actions on multiple targets simultaneously
- You can add many inputs to a multi-instance step
- To create, right-click and select Toggle Multi-Instance
- Step icon changes
- Provide multiple inputs in a list
- A new result, Group Done, is added to the step
- Success result “loops” to next step(s), only Group Done links to the success return step

4



A Multi-Instance step is one where multiple instances of a step run simultaneously.

This is useful for applying the same actions across multiple targets simultaneously.

Specifying a multi-instance step is basically a toggling exercise – you simply right-click a step in the Design panel and select Toggle Multi-Instance.

When you do this, the icon changes with a new response, Group Done.

You can also provide lists of inputs, like hostnames or IP addresses, for specifying multiple targets.

In other words, a multi-instance step is a very easy and convenient way to basically iterate the same actions to a series of targets. Once all instances are completed, the flow proceeds down the path specified by the Group Done response.

Multi-instance applies only to a step and not to an underlying operation, so turning on multi-instance has no effect on other flows that reference the same operation.

Using Multi-Instance Steps

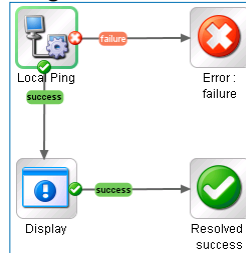
Single Instance:

- Local Ping success response links to Display, which links to Resolved return step

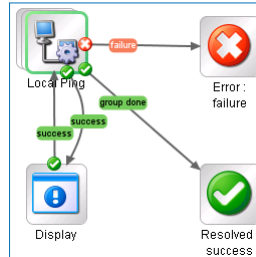
Multi-Instance

- Local Ping icon changes
- Group Done response is added
- Success response iterates to Display until all instances are complete
- Group Done response links to Resolved return step when all instances are complete

Single Instance



Multi Instance



5



The example shows how to use a Multi-Instance step to ping a number of target hosts simultaneously.

Simply right-click the Local Ping step and select Toggle Multi-Instance. Note the changes to Local Ping. The changes require you to reconnect transitions to accommodate the new Group Done response. You also need to provide a list of inputs to the multi-instance step.

At run time, the multi-instance iterates through all the hosts specified in the input list until it reaches the Group Done condition, when it then proceeds to the Success return step.

Toggling Multi-instance Steps

- Right-click step, select Toggle Multi-instance to activate or Toggle Single Response to deactivate
- You may need to change inputs to a list
- If using a flow variable compiled from List Appender operation, scriptlet, or other function, make sure to use the correct delimiter

Specify list and delimiter

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

Input Properties

Input Delimiter: , Encrypted: [] Required: [x]

Validation Format: <not validated>

Record Under: <run history>

Otherwise: Use Constant Configuration

Constant Value: oocentral75,localhost

Toggle Multi-instance

Toggle Single Response

Toggle Nonblocking

Debugging

Properties

Open Operation

Cut

Copy

Paste

Remove Step

Specify list and delimiter

Provide delimited list

The slide shows specifically how to toggle Multi-Instance on and off, and how to specify the list of inputs for the multi-instance step. You basically specify the list and delimiter.

The list can be a constant value or a flow variable with multiple entries separated by the specified delimiter.

Nonblocking Steps

What is a Nonblocking Step?

- A nonblocking step allows the flow to continue with subsequent steps while the non-blocking step is still executing
- To specify nonblocking, right-click the step and select Toggle Non-Blocking
- Icon changes to a lightening bolt and it acquires one result, Done
- Nonblocking steps are automatically checkpointed
 - When you restore a run it starts at the point immediately preceding the non-blocking step
- Best practice: Use only on steps that do not directly impact subsequent steps



A non-blocking step is simply a step that allows the flow to proceed while it is executing.

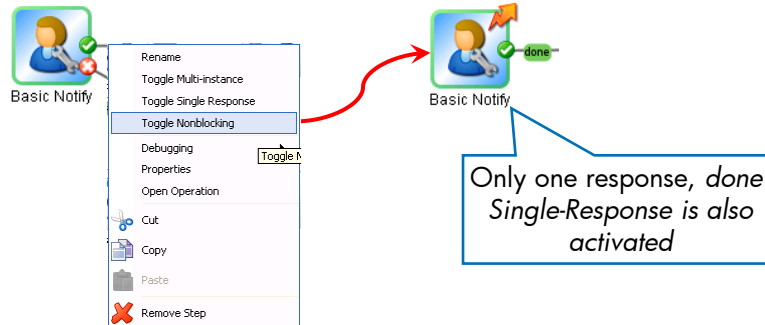
This is very useful, for example, when a step is a subflow that does not depend on the parent flow, and visa versa. You can simply set the subflow step as a nonblocking step and the flow run will proceed while the subflow is executing.

It's important to remember that a nonblocking step should only be used when subsequent steps do not directly require results returned from the nonblocking step.

If the flow run fails because of a service interruption or some other factor, the restored flow run begins immediately preceding the nonblocking step. In other words they are automatically checkpointed.

Using Nonblocking Steps

- Right-click the step, select Toggle Nonblocking
- Step changes:
 - Lightning bolt icon indicates non-blocking
 - Only one response, done
 - Type changes to Single Response



8



To use a nonblocking step, simply select the step by right-clicking and select Toggle Nonblocking.

When you toggle nonblocking on, the step gets only one response, done. Therefore you will need to reconnect transitions appropriately after specifying the step as nonblocking.

Parallel-Split Steps

What is a Parallel-Split Step?

- A parallel-split step splits flow execution into two independent paths
- Each path is called a lane, steps within each lane are called lane steps
- Best practice: Use for simultaneously running two separate sets of operations in a single flow
- To create, drag the parallel-split icon to your design panel, add steps
- Parallel-split steps converge when lanes are finished

9



A parallel split step is the most complex of the three parallel execution steps.

In a parallel-split step, flow execution splits into two or more separate execution paths.

Each path is called a lane, and steps within each lane are called lane steps.


A parallel-split is useful if you want to establish two or more entirely separate sets of operations within a flow and allow them to run simultaneously.

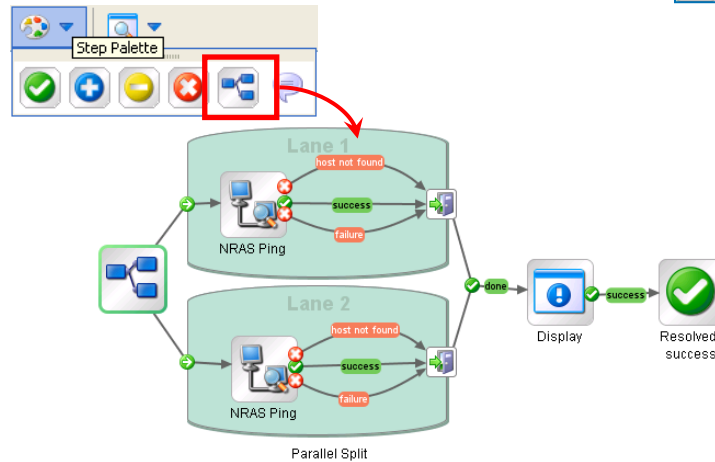
A parallel-split step increases runtime overhead because each lane uses a separate thread in addition to the thread being used by the flow itself. This is an important performance and load balancing consideration. If you have a lot of flows that use parallel-split steps you may need to increase the thread pool.

There are limitations on what you can do inside each lane. For example, you cannot pass a flow variable from one lane to the next at run time, and all lanes converge when completed – there are no return steps within the lanes.

Using a Parallel-Split Step

1. Drag the parallel-split icon onto your design panel
2. Add step(s)

- Lanes cannot have return steps, wire everything to 



To use a parallel-split step, open the Step Palette and drag the parallel-split onto your design panel.

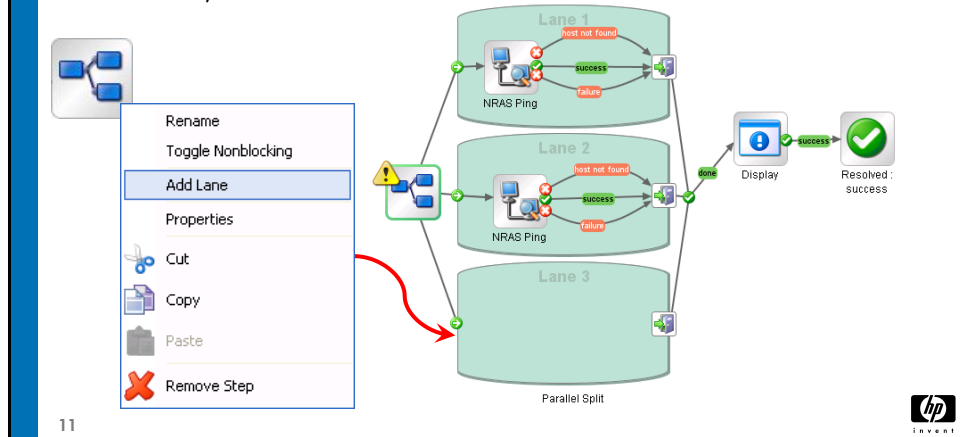
You can then add steps within each lane. Those steps will execute in parallel at runtime.

Note the way the steps converge, with responses linking to the “exit door” on each lane, which then links to a single Success response.

You need to be mindful of the limitations when working with a parallel-split step. In some instances it might make more sense to use subflows rather than parallel-split lanes if the level of complexity required exceeds the limitations presented by using the parallel-split step.

Adding and Changing Lanes

- Right-click the parallel-split icon
- Select Add Lanes from menu
- To change lanes, right-click lane and select Move, Rename, or Remove



It's easy to add, change, move, or remove lanes.

Just right-click on the parallel-split icon and take the appropriate action.

The example shows how to add a lane to an existing parallel-split step.

Data Handling in Parallel-Split Steps

- Each lane obtains a copy of the flow variables in the global context; values associated with those flow variables are available for any of its steps
- Lane steps can obtain data from the local and global contexts and save data to local context
 - Use the `scriptletcontext.putglobal()` method to write data to the global context
 - Data are not merged back to the global context until all the lanes have finished
- If lane steps write to a single flow variable, the last step to write to the flow variable assigns the final value
- A lane step cannot pass values to a step in another lane

12



Review the data handling considerations when working with parallel-split steps.

If your data requirements exceed the limitations imposed by the parallel-split step then you should consider using subflows rather than the parallel-split step to accomplish the task.

You need to use the `scriptletcontext.putglobal()` method to write data to the global context from within a lane.

If a lane writes to a single flow variable at runtime, the last step to write to the flow variable assigns the final value.

Finally, you cannot pass values to a step in another lane.

Considerations When Using Parallel-Split Steps

- Only one output/response: **All the lanes have completed**
- Flow permissions apply to parallel lanes
- Gated transitions in lanes cannot be handed off; transitions in lanes cannot be configured as handoffs
- Parallel-split steps and their lane steps are checkpointed, and you cannot remove the checkpoints
 - If you restore a run it restores to the point just before the parallel-split

13



Review the considerations listed on the slide. If you have an application that exceeds the limitations of working with parallel-split steps then consider using subflows instead.

Debugging Parallel Processing

Processes that run in parallel in Central run serially in the Studio debugger

- Parallel-Split: Lanes always execute serially in studio but begin at the same time in Central
- Multi-instance: The instances are executed serially, allowing you to examine how long it takes each instance to finish
- Nonblocking: These do not behave as nonblocking steps in the Debugger - steps that follow a nonblocking step do not execute until the nonblocking step has completed

14



It's important to note that there are differences in how flows run in Studio as compared to Central.

In the Studio debugger, parallel lanes execute serially, not simultaneously as they do in Central. Therefore the flow will behave differently and take longer to execute in Studio.

The same is true for multi-instance steps – each instance executes serially in Studio. So if you need to time a multi-instance step you should do so in Central.

Nonblocking steps do not behave as nonblocking in Studio, the flow run waits for the nonblocking step to complete before proceeding.

Summary: Parallel Processing

- Multi-Instance Step:
 - A multi-instance step triggers multiple simultaneous instances of a step
 - Useful for applying the same action(s) to multiple targets
- Nonblocking Step:
 - Allows a flow run to continue while the step executes
- Parallel-Split Step
 - Divides a flow run into two separate, simultaneous paths
 - Use for simultaneously executing two or more sets of operations simultaneously
 - Each lane obtains flow variables from the global context; use `scriptletcontext.putglobal()` to write to the global context
 - Only one output/response: **All the lanes have completed**

15

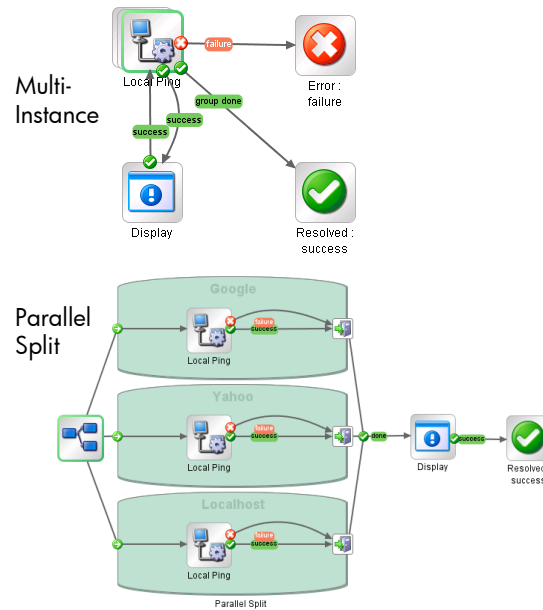


Review the summary for this section.

Exercise 1: Parallel Processing

Author flows that use:

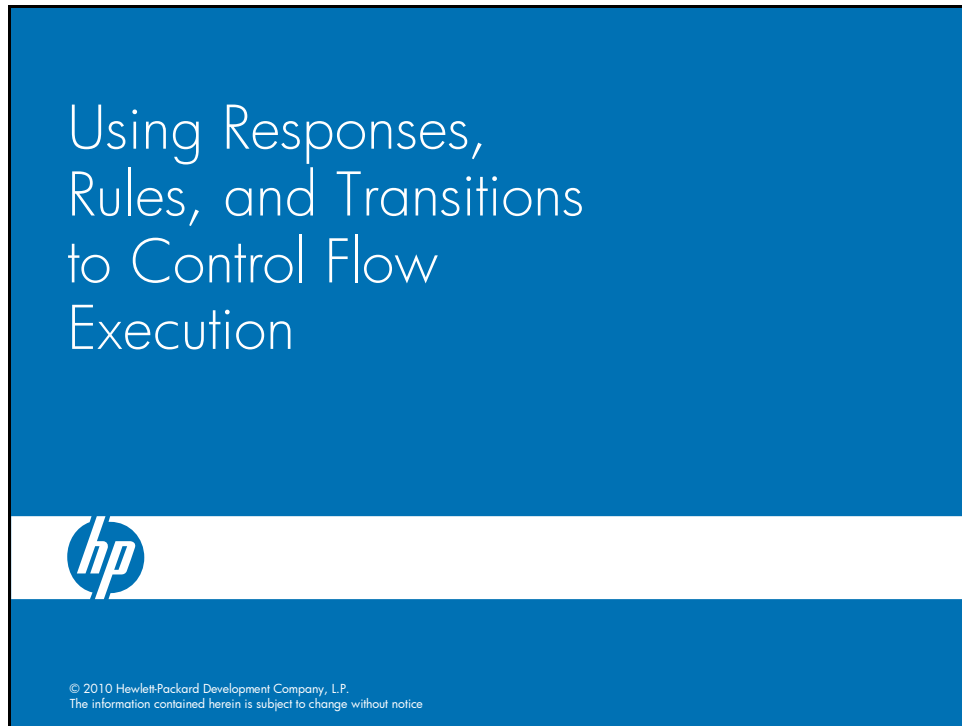
- Multi-Instance Step
- Parallel Split Step



16

In the exercise you will work with all three types of parallel processing – nonblocking, multi-instance, and parallel-split steps.

Using Responses, Rules, and Transitions to Control Flow Execution



In this section you will learn how to use responses to direct the execution path of your flow depending on conditions that you specify in your flow.

Objectives

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

- Add responses to steps in a flow
- Use Rules to define response behavior
- Use transitions to:
 - Connect responses to steps
 - Provide descriptive information to users
 - Control who can execute steps in a flow
 - Establish ROI values for steps and flow runs

2



In this section, you will see how to modify the properties of an operation to add responses – success, failure, no action, and diagnosed – and then how to determine rules to be followed that will cause a particular response to be taken at runtime.

Additionally you will learn more about how to use transitions to control flow execution, establish a return-on-investment value, and provide descriptions to users.

Controlling Execution: Responses and Transitions

- Working with Responses and Transitions is an important authoring skill
- Responses and Transitions determine the path of flow execution
- Every step in a flow, except a return step, has at least one response
- You can add and/or modify responses to alter flow behavior
- Every response must have a transition that links one step to the next
- In addition to simply linking steps, transitions can be used to:
 - Provide additional information to flow users
 - Control who can execute particular steps in a flow
 - Establish an ROI (Return-On-Investment) value to steps; cumulative step ROIs equal the total ROI for a flow

3



Working with responses and transitions is a very important skill and allows you to gain full control over how your flow executes based on conditions that are detected or become available at runtime.

These two topics are presented together because every response on a flow step requires a transition to connect it to a subsequent step.

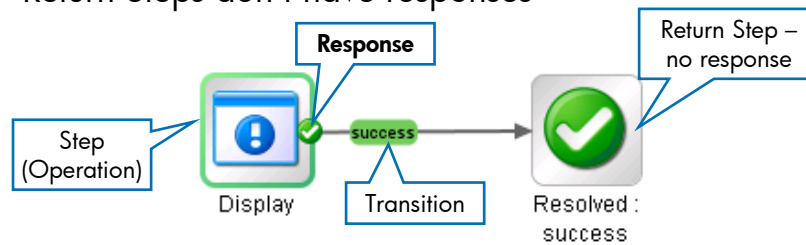
By adding responses and/or applying rules to new or existing responses you can gain full control over the behavior of your flow.

Additionally, modifying transitions allows you to further control flow execution while providing valuable information to users at runtime.

Responses

What is a Response?

- A Response is the outcome of an operation
- Responses direct the flow by linking to the next step
- Responses are predefined for operations
- You can add, remove, or modify responses based on your specific needs
- Return Steps don't have responses



As you have already seen, a response is the outcome of an operation. A response directs flow execution by linking to the next step. Responses are predefined for operations in the OO library but you can add, remove, and change responses by modifying the Responses tab in the operation's properties editor. Additionally you can control what rules to follow so that a particular response is selected at runtime based on conditions present in your flow at that moment.

As you can see on the slide, every response must be connected by a transition to a subsequent step. Note that return steps do not have responses because they are the endpoints of flow execution.

Available Responses



Resolved or Succeeded

- Successful – flow continues to the next step



Diagnosed or Evaluated

- A condition was diagnosed or evaluated



No Action Taken

- No action taken at this step



Error, Failed

- The step failed

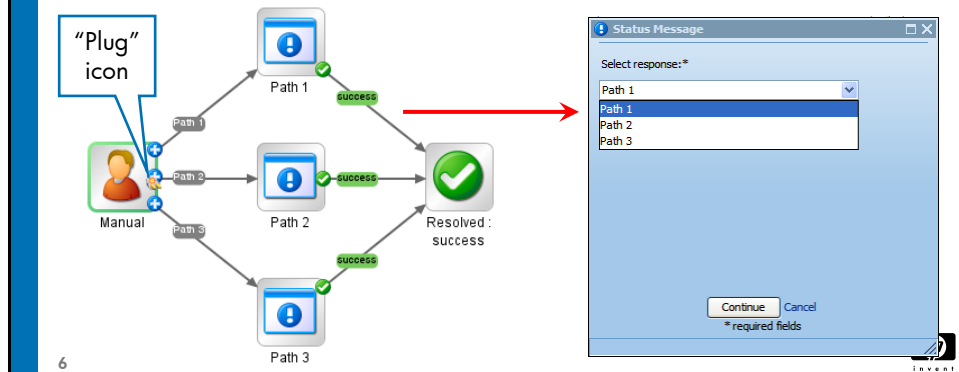
5



The available responses are shown on the slide. Even though the behaviors listed on the slide – success, failure, etc. – are predefined for operations in the library, as a flow author you have complete control over determining what constitutes success, failure, etc. By assigning rules to a response – which you will learn how to do shortly – you have complete control over defining the behavior associated with a response.

Manually Controlling Flow Execution

- The Manual operation (in Utility Operations) allows you manually control flow execution
- Simply drag the plug icon on Manual to a step – the response and transition take the name of the step it connects to
- Add as many responses and transitions as you need
- At runtime, the user is given a menu to select the flow execution path



To begin learning how this all works, you can use the Manual operation in the OO library and build a small flow that demonstrates how to direct the path of a flow depending on a condition present at runtime, in this case, a user selection.

Simply drag Manual onto your design panel and add a few more operations, as shown on the slide – these are simply “do nothing” Display operations used for illustration purposes.

To wire this together, simply drag the plug icon on Manual to a subsequent step. This automatically adds a named response to the step you connect it to.

When you run the flow, you will be presented with a menu. Selecting an item from the menu selects the execution path, depending on which response is “activated” when the selection is made.

The lab guide has an exercise where you build this flow.

Using Responses to Direct a Flow Run

- Responses allow you to direct flow execution at runtime
- Filters and rules applied to flow data can be used to select a step's response – the flow executes down that path
- You can add as many responses and rules as needed to control flow execution
- You use an operation's Properties editor to add responses and optionally apply filters and rules

7



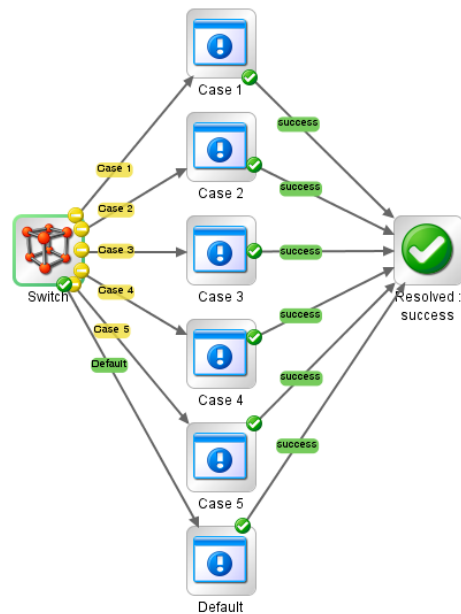
Responses allow you to direct flow execution at runtime.

The Manual operations is a very simple way to illustrate this because it automatically adds responses when you connect the plug icon to a subsequent step.

In practice you will be editing the properties for an operation and specifying responses there. Additionally you will use the Rule Editor to define what conditions will cause that response to be selected at runtime.

Example: OO Switch

- Switch step has six responses
- At runtime, user is presented with a menu – Case 1 through Case 5
- When the user makes a selection, filters in Switch select the correct execution path
- Path selection can be accomplished without user interaction simply by applying a rule to a flow variable



In this example, the Switch operation has six responses, five “no action” and one “success”.

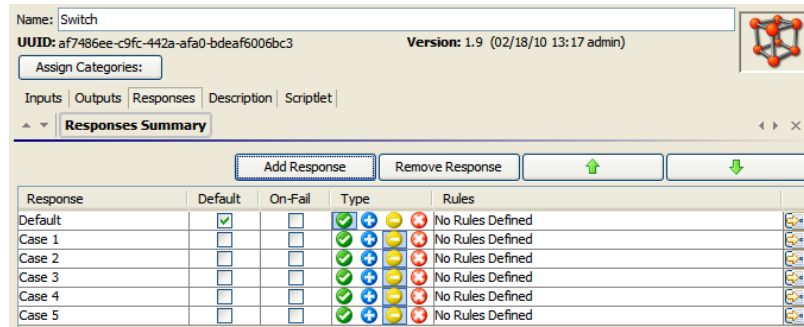
At runtime, the user makes a selection from a menu – the choices are Case 1 through Case 5. Then depending on which selection the user makes, that particular path is selected at runtime.

Note that the condition that directs flow execution can be anything that can be detected or measured in the system at runtime, not just a user selection – the value of a flow variable, the result of a previous step, etc. All of these can be used to define the rules that will cause a particular response to be selected, depending on the rules you have defined for the response.

In this example, the Switch step is simply a modified Display operation whose properties were modified to include the new responses and rules.

Adding Responses to an Operation

- Open the Operation Properties
 - Double-click in Library, or right-click operation in Design Panel and select Open Operation
- Select Responses tab, click Add Response
- Name the response
- Select the response Type
- Specify Rules and Filters



To add responses to an operation, double-click the operation in the Library or select Open Operation in the Design panel to open the properties editor.

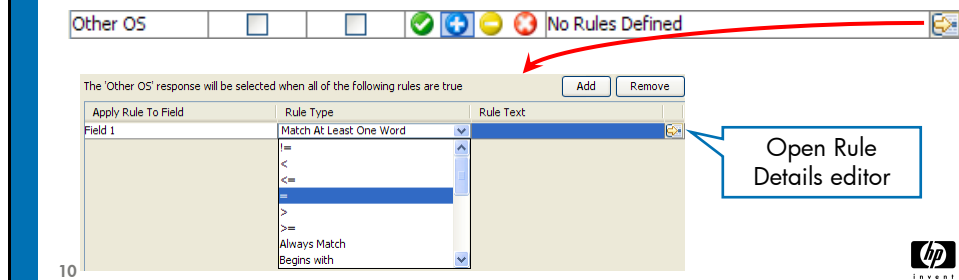
A word of caution – make a copy of an operation before adding responses. If you add a response to an operation that is shared among different flows, every flow that calls that operation will have an Unhandled Response error and will not run.

Working on a copied version of the operation avoids that problem.

To add the response, select the Responses tab and add the responses you need. Initially no rules are defined, as shown on the slide.

Define Rules for a Response

- Rules let you determine what happens next
- To open the Rules Editor, click the arrow
- Specify:
 1. What field to apply the rule to
 2. Type of rule to use
 - Can also be a regular expression or scriptlet
 3. Rule text

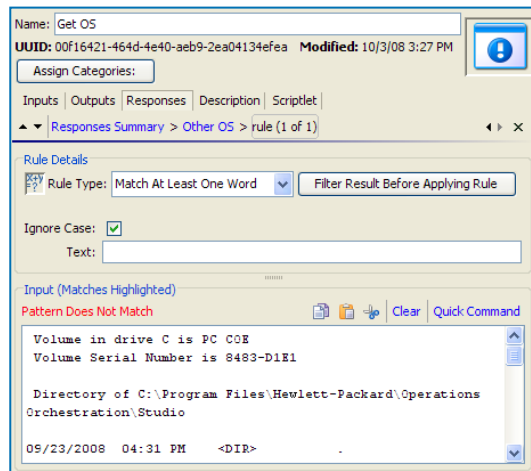


To define rules, just open the Rules editor and add the rules you want – for example, in this case the field Field 1 is being tested to see whether it exactly matches a text string. Field 1 would be visible in the Inputs of the Step Inspector.

You can apply more than one rule and you can also open the Rule Details editor to filter complex output and test for a particular value. The rule can also be a regular expression or scriptlet result.

Rule Details Editor

- Click the arrow icon (🔍) in the Rules editor to open the Rule Details editor
- Allows you to specify advanced options for the rule
- Click Filter Results to apply advanced filters or regular expressions
- Click Scriptlet tab to use a scriptlet for the rule
- Use Quick Command to generate sample data for defining filters

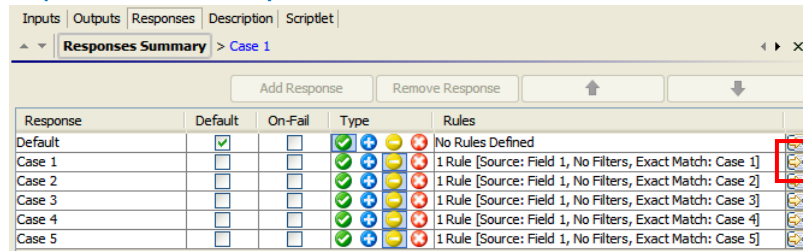


11

The Rules Detail Editor is very similar to the Filter Editor – it allows you to enter test output, either by copying and pasting or with Quick Command – and develop a filter to extract the value you want to test for in the rule.

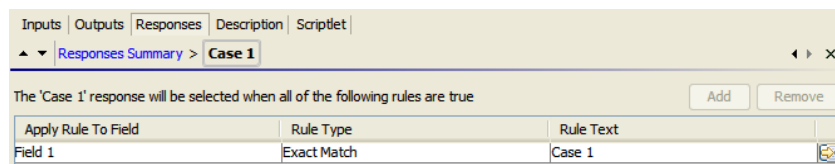
Response Summary and Rule Details

Response Summary



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]
Case 4	<input type="checkbox"/>	<input type="checkbox"/>		1 Rule [Source: Field 1, No Filters, Exact Match: Case 4]
Case 5	<input type="checkbox"/>	<input type="checkbox"/>		1 Rule [Source: Field 1, No Filters, Exact Match: Case 5]

Rule Details



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

Once defined, the rules are summarized on the Responses tab. You can always click the Rule Editor to change a rule. In the example shown, the Case 1 response will be selected if the value for Field 1 is an exact match of the text "Case 1" which is provided by user selected at runtime. If the user provides a different value – Case 3 for example – then the Case 3 response is selected.

Working With Transitions

What is a Transition?

- A transition does four things:
 1. Links steps in a flow
 2. Tells Central users what happened in a step
 3. Controls who can execute a particular step
 4. Establishes an ROI (Return on Investment) value for a step
- Every response must have a transition

13



You have already worked a lot with transitions. Every time you connect one step to the next, you are simply connecting a response to a subsequent step with a transition. In other words, the transition is the arrow that connects steps in a flow.

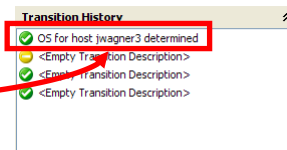
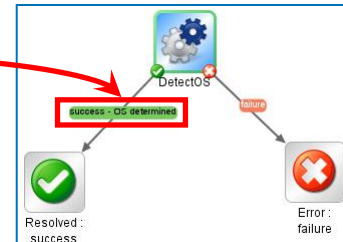
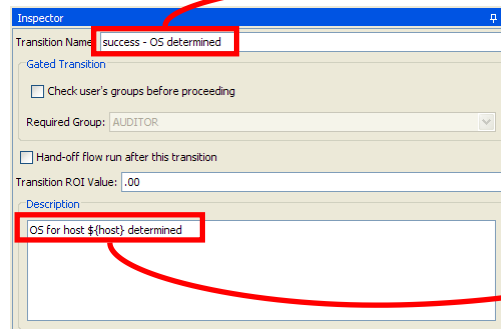
But there's much more to a transition than that. With a transition you can tell Central users what happened in a step, control which group can execute a particular step, and establish an ROI value for a step.

Every response must have a transition that connects it to a subsequent step. The exception is return steps which do not have responses and thus serve as endpoints for a particular execution path.

Transition Name and Description

Telling Users What Happened

- Add a descriptive transition name
- Use the Description to provide transition detail upon execution
 - Replaces **<Empty Transition Description>** in Transition History at runtime
 - Can refer to variables in the Description:
 - OS for host **`${host}`** determined



14

One important role the transition plays is to inform the user what happened at a particular step. You use the Transition Name and Description for this purpose.

The Name is the label that appears on the transition in the design.

The Description appears in the Transition History when debugging the flow and also shows up appropriately in the reporting for the flow run.

By using the Transition Name and Description you can add a lot of value to your flows by making them more descriptive, which is a big benefit when the flow moves into production.

Gated Transition

Controlling Who Performs the Next Step

- Use a Gated Transition to check the group
- Select Hand-off flow run to require a new group to proceed with the flow execution
 - Select “Check user’s groups before proceeding”, specify the Required Group
 - Only Central user(s) in the specified group can continue the flow run

The screenshot shows a window titled "Inspector" with a tab for "success - OS determined". Below the tab, the "Gated Transition" section is highlighted with a red border. It contains the following settings:

- ☒ Check user's groups before proceeding
- Required Group: AUDITOR (selected from a dropdown menu)
- ☒ Hand-off flow run after this transition
- Transition ROI Value: .00

15



A Gated Transition allows you to control who performs the next step in the flow. A gated transition checks the group at runtime and if a different Required Group is specified, the flow run is handed off to that group.

At runtime there are various ways of alerting users that a flow run is being handed off. Emails with execution links can be sent, for example, when the flow is handed off.

Specifying ROI

Establishing a Value for a Step

- ROI determines the value for a particular step
- Cumulative ROIs determine value for a flow run
- You can use different approaches for calculating ROI, for example:
 - You know that a particular step in a flow replaces a manual process that takes 15 minutes
 - You were paying someone \$20/hr to perform that step manually
 - Step value: $\$20 * 15/60 \text{ minutes} = \5
 - ROI values appear in Central run reports
 - You don't have to set an ROI value – you can leave the field blank

16



Finally you can specify an ROI for a particular step. The transition is where you specify the ROI.

The ROI is simply a monetary value. The flow author can use any method that makes sense to determine the value of a step. The cumulative step ROIs equals the value of the entire flow run.

Specifying an ROI is optional – you don't need to enter an ROI value for any step in a flow.

Summary: Responses and Transitions

Responses

- A Response is the outcome of an operation
- Responses direct the flow by connecting the response to the next step in the flow
- Responses are predefined for operations
- You can add, remove, or modify responses based on your specific needs
- To add or modify a response, Open the operation and select Responses
- Use the Rules Editor to define rules or filters for the response
- You can also apply a scriptlet to a response

Transitions

- Transitions connect the response of one step to a subsequent step
- Use transitions to:
 - Show Central users what happened at a step
 - Control who can continue with a flow as it executes
 - Establish a value of a step and for a flow run
- Every response in a step must have a transition that leads to another step

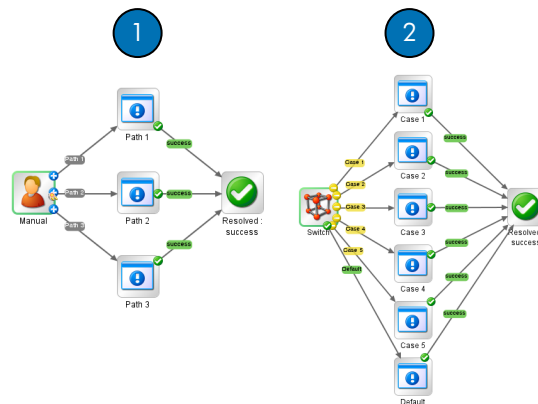
17



This slide summarizes what we covered in this section. Make sure to review the summary and remember to make a local copy of an operation before adding or changing its responses.

Exercise: Responses and Transitions

- Author flows that:
 - Use manual selection
 - Use responses and rules to direct flow execution



18

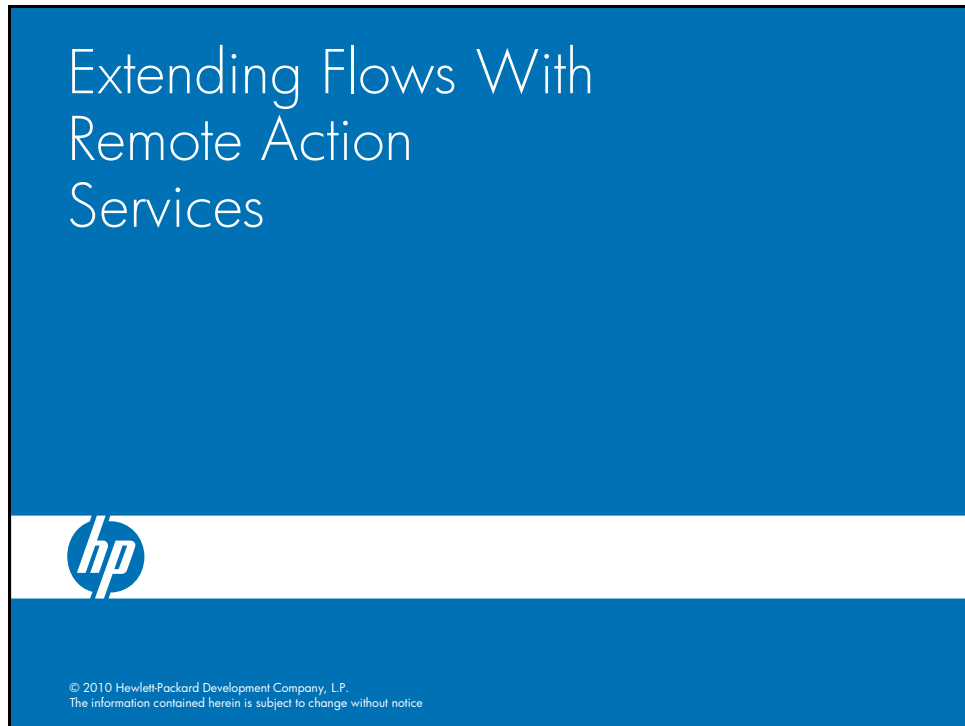


The exercise has two parts. In the basic exercise, students add responses to an operation that branch depending on values determined at runtime.

In the Advanced exercise Windows or Linux commands are executed depending on the type of target platform.

The Advanced exercise is optional.

Remote Action Service



This module will cover the OO Remote Action Service, a critical component of any OO deployment.

Objectives

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

- Explain the role of the Remote Action Service in Operations Orchestration
- Add and manage RAS References in Studio and Central
- Provide overrides to RAS references
- Explain how steps/operations locate iActions on a RAS server
- Install an iAction, import an operation, and use in a flow
- Explain the role of important RAS configuration files
- Disable client authentication
- Enable remote debugging



In this module you will learn about the role of the Remote Action Service in OO. You will learn how to manage RAS References in Studio and Central and how to provide override values that allow you to access a different Remote Action Service at runtime. You will learn what an iAction is in detail and how RAS operations locate iActions in the RAS repository. You will install an iAction and create OO content that accesses it. Finally this module covers some advanced topics like key configuration files, how to disable client authentication for debugging, and how to enable remote debugging.

What is Remote Action Service (RAS)?

- A Web Service that:
 - Listens for requests from Central/Studio to perform a Java or .NET “iAction” deployed to the RAS repository
 - Performs that action
 - Returns a result
- Most work in OO occurs through the RAS
- RAS capabilities change depending on what platform it is running on
 - All platforms: Java, Windows: .NET/C#
- Since the communication is done with SOAP (using https) it is very easy to tunnel through a firewall to the RAS
- Also because the communication is done via https the requests can be proxied (either https-based or SA Gateway)
- Every Central installation has a RAS
- You can also install a standalone RAS to extend OO services to remote networks and datacenters
- RAS service RSJRAS manages both Java and .NET iActions – .NET is available only on Windows servers



Remote Action is a web service that listens for requests from Central or Studio to execute Java or .NET iActions deployed to the RAS repository.

Nearly all of the work that happens in OO occurs through the RAS.

The architecture is based on SOAP calls to the Central web services interface using https, giving you an easy ability to tunnel through a firewall or proxy requests.

Every instance of Central has a default RAS. You can also install a standalone RAS allowing you to extend your flows to remote networks, Windows domains, or data centers behind a firewall.

A process/service named RSJRAS monitors both the Java and .NET iActions.

It's important to note that there is one RAS but it behaves differently depending on the platform you are on. On Windows, you get the full suite of deployed Java and .NET iAction content. On Linux or Solaris, you only have access to the Java content deployed to the RAS repository.

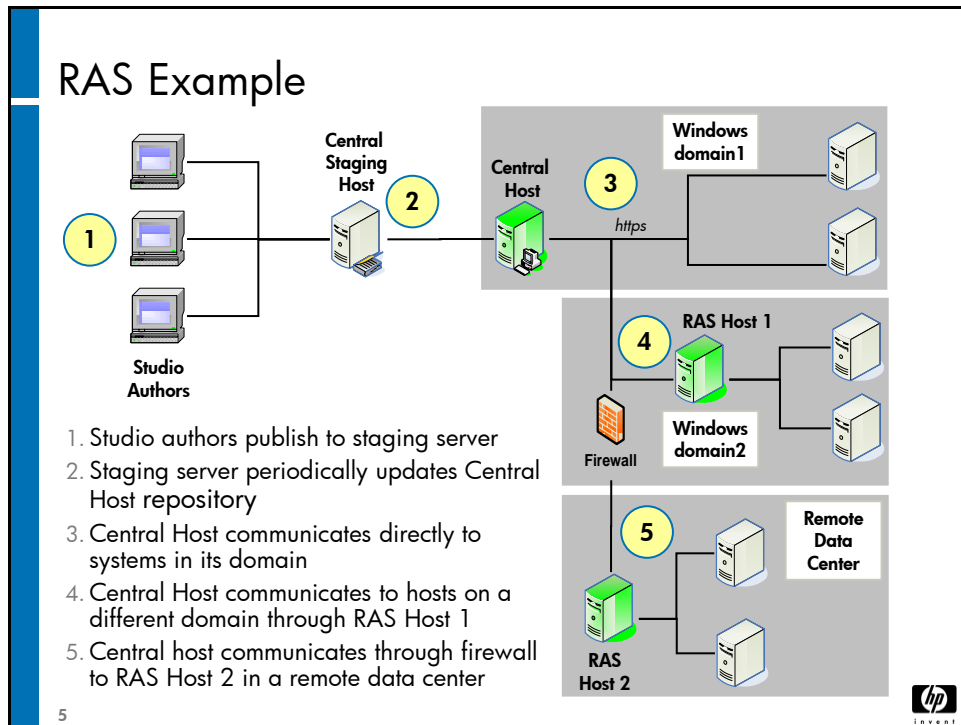
Supported Platforms

- RAS is supported on:
 - Windows
 - Linux
 - Solaris
- Java-based operations are supported on all three platforms
- .NET/C# based operations are only supported on Windows
 - The RAS will not attempt to load these if the platform is reported to be not windows



RAS is supported in Windows, Linux, and Solaris.

.NET content is only available on the Windows platform.



Here is a simple topology that shows a number of RASes on a network.

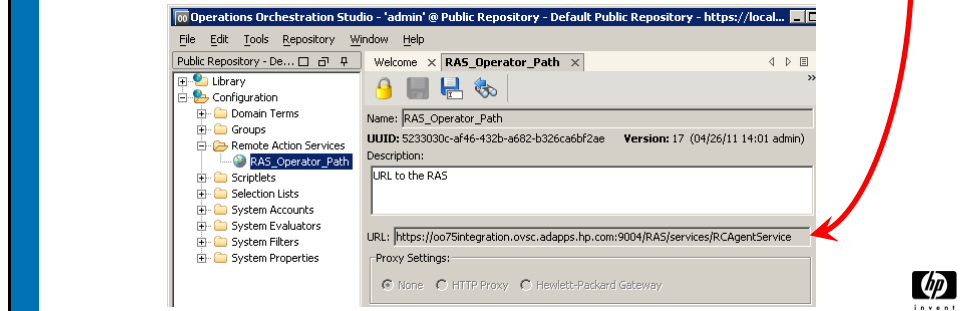
Here, studio authors (1) deploy content to a staging Central host (2). Periodically content is published to the Central production host (3) which has a default RAS that can interoperate with all of the hosts in its network or domain.

To access hosts in a second Windows domain (4) a standalone Remote Action Server is installed which can reach all of the hosts in that domain.

The data center behind a firewall (5) is also served by a standalone RAS.

RAS Reference

- How a step (operation) locates the RAS host that contains the deployed iActions
- Consists of a named URL to the RAS as its value
- Stored in Configuration/Remote Action Services
- Default: RAS_Operator_Path, you can add as many RAS references as needed
- URL format: `https://RASHOST:9004/RAS/services/RCAgentService`



To locate RAS hosts and repositories on your network, you use a RAS Reference.

A RAS Reference is a named configuration item that has the URL to a RAS server as its value.

In OO, you point to different RASes simply by accessing the named RAS reference in a step in a flow.

Each RAS operation has a default RAS defined – you can provide override values for that RAS reference at runtime, as explained later.

The RAS URL always has the same format:

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

Once a step locates a Remote Action Server, definitions within the referenced operation locate the actual iAction content to execute in the repository for that RAS.

How do Steps Know Which RAS to Use?

- The RAS reference is specified in the operation that the step refers to
- All RAS operations use the default RAS reference, RAS_Operator_Path, it's easy to change the default RAS reference
- It's possible for each step in a flow to use a different RAS
- Each "RAS operation" has:
 - A RAS reference, where the iAction is hosted
 - The name of the archive
 - The name of the iAction to execute with in the specified archive
 - Inputs required by the iAction
 - Outputs/primary output returned by the iAction



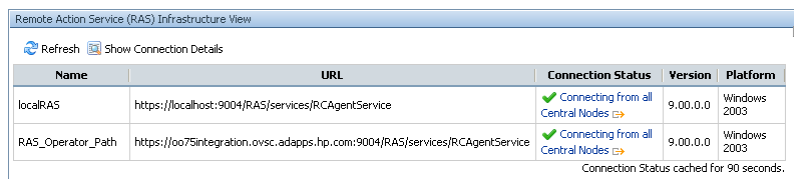
The RAS reference is simply the URL used to locate the RAS host on a network. You can easily proxy the RAS reference if the RAS host is behind a firewall.

All RAS operations have a RAS reference, and all installations of Central have a default RAS. The reference to the default RAS is RAS_Operator_Path.

All RAS operations contain the RAS reference, along with the name of the JAR file or DLL that contains the specified iAction to execute. The JARs and DLLs are stored in the RAS repository for that particular installation.

Viewing and Monitoring Deployed RASes

- Central: Administration/Administration tab



Remote Action Service (RAS) Infrastructure View

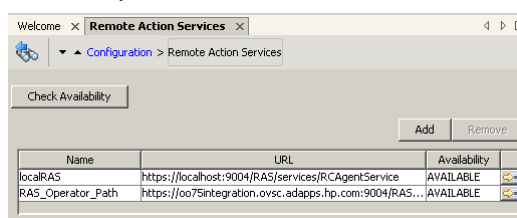
Refresh Show Connection Details

Name	URL	Connection Status	Version	Platform
localRAS	https://localhost:9004/RAS/services/RCAgentService	✓ Connecting from all Central Nodes	9.00.0.0	Windows 2003
RAS_Operator_Path	https://oo75integration.ovsc.adapps.hp.com:9004/RAS/services/RCAgentService	✓ Connecting from all Central Nodes	9.00.0.0	Windows 2003

Connection Status cached for 90 seconds.

- Studio:

- Select Configuration/Remote Action Services
- Here you can monitor and check availability



Welcome x Remote Action Services x

Configuration > Remote Action Services

Check Availability

Add Remove

Name	URL	Availability
localRAS	https://localhost:9004/RAS/services/RCAgentService	AVAILABLE
RAS_Operator_Path	https://oo75integration.ovsc.adapps.hp.com:9004/RAS...	AVAILABLE

Central and Studio have tools for viewing and monitoring RASes.

In Central, go to Administration/Node Administration to view the available RASes and their status.

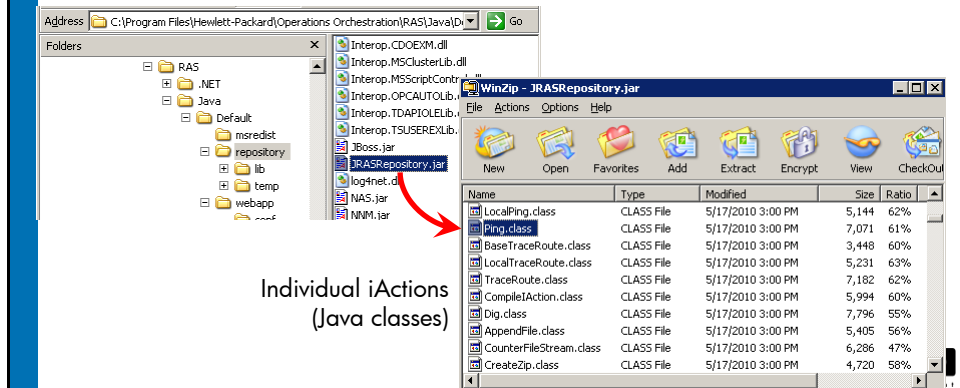
In Studio, go to Configuration/Remote Action Services. Here you can view deployed RASes, see their status, and check availability.

What is an iAction?

- Executable code in the RAS repository
- Can be Java (cross platform) or .NET/C# (Windows only)
- Example: **RAS_Operator_Path**

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

../RAS/Java/Default/repository – JARs (Java) and DLLs (.NET)



An iAction is executable code in the RAS repository. It can be either Java or .NET/C#.

iActions are stored in archives (JAR files or DLLs) in the RAS repository (RAS/Java/Default/repository in the OO home directory).

Example: Ping Operation

- All RAS operations have the same format
 - **Action Class:** The iAction to execute in the specified Archive
 - **Archive:** The JAR or DLL located in the RAS repository
 - **RAS:** The RAS Reference
 - **Override RAS:** A flow variable for assigning an alternate RAS to use at runtime (more on this later)
 - **Inputs:** iAction inputs – required inputs are checked

Name: Ping
 UUID: 8fdb451d-f629-4c50-a75a-ffc4c89e5729 Version: 1 (05/12/11 13:58 admin)

Assign Categories:

Inputs Outputs Responses Description Scriptlet

Inputs Summary

RAS Operation fields

Action Class: com/ibconclude/content/actions/cmd/ping/Ping.class

Archive: JARRepository.jar

RAS: /Configuration/Remote Action Services/RAS_Operator_Path

Override RAS: \${override.RAS}

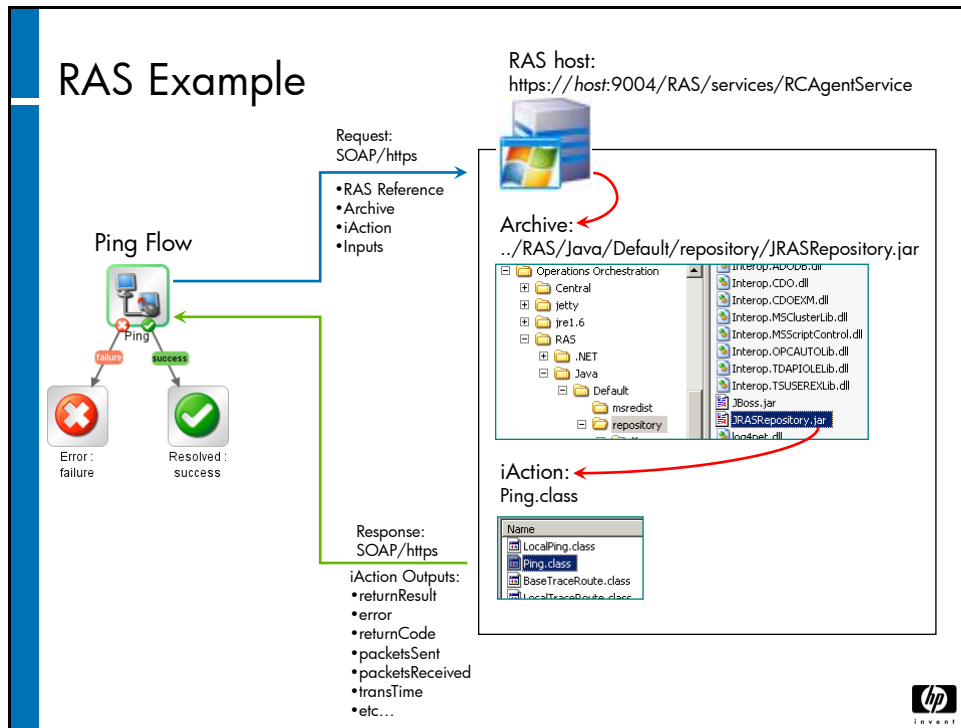
Inputs

Input	Required	Type	Template
targetHost	<input checked="" type="checkbox"/>	Not Assigned	No Assignment
packetCount	<input type="checkbox"/>	Not Assigned	No Assignment
packetSize	<input type="checkbox"/>	Not Assigned	No Assignment
protocol	<input checked="" 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

iActions are called out in the Action Class field of a RAS operation's properties. The iAction is located in the specified Archive, and the RAS field defines the default RAS reference which contains the URL to the RAS host.

The RAS operation also declares all of the inputs that the iAction requires.

Note that no definition is usually needed for an operation's inputs – inputs are typically configured with the Step inspector.



This slide shows a very simplified example of how it all works.

In this example, the Ping step in a flow formats inputs for a particular, and a SOAP request is formed and sent to the RAS host.

The RAS host in turn takes the information formatted in the SOAP request that contains all of the information needed to locate the iAction and send it the expected inputs.

The RAS server formats a SOAP response that is sent back to the calling step. That response contains all of the outputs that the iAction provides.

Using an Alternate RAS

- You can specify an alternate Remote Action Service, referred to as an “override” RAS, in Central and Studio
- In Central, you provide an “override” RAS URL to an existing named RAS Reference
- In Studio you can:
 - Embed a new RAS reference in a RAS operation
 - Supply the name of a RAS reference to an override input:
 - For Java iActions: \${overrideJRAS}
 - For .NET iActions: \${overrideNRAS}
- With an override in place, iActions on the specified RAS host are used instead of those in the default RAS
- This is an extremely useful capability, allowing you to decide at runtime where to execute iActions on a local or remote network



The RAS architecture is highly scalable, allowing you to access essentially any RAS on your local network or on a different Windows domain or behind a firewall.

There are a number of ways of providing a different RAS Reference at runtime:

You can embed a new RAS reference within the operation itself

You can supply the name of a different RAS reference and assign it to an “override” flow variable (overrideJRAS for Java iActions and overrideNRAS for .NET iActions).

With the override in place, iActions on the specified RAS are located and executed instead of those on the default RAS.

This is an extremely useful tool, allowing you to determine at runtime where to execute iActions on your network.

Central RAS Override

- As a Central admin go to Administration/Runtime Environment/Remote Action Service (RAS) Overrides
- Click the Add button, select the RAS you want to override, then provide a new RAS URL

- All references to selected RAS Reference are redirected to the override URL until you remove the override



You can also provide overrides in Central.

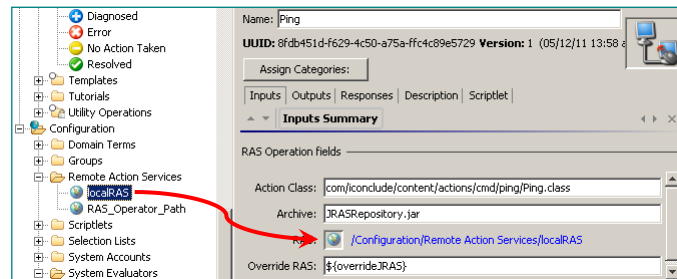
In Central the approach is different. The override value is a different URL that is applied to an existing named RAS reference.

You do this in the Administration/Runtime Environment/Remote Action Service (RAS) Overrides tab.

The override remains in place until you remove it. While it is in place, all references to a particular RAS are directed to the URL specified in the override.

Selecting RASes in Studio

- Change a RAS Reference to an operation:
 - Click the RAS reference in Configuration/Remote Action Services, drag it over the RAS globe icon in the operation's Properties editor
 - You must use a copy of the operation if it's source is in a sealed folder



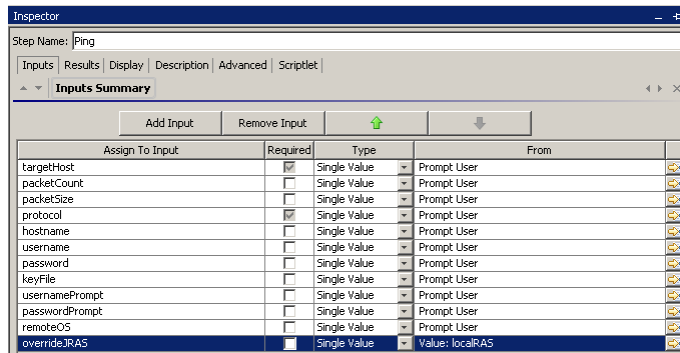
To change a default RAS for an operation in Studio, open the operation's properties editor, then click and drag a RAS reference from Configuration/Remote Action Services to the globe icon in the properties editor. Then save and check in the operation.

Always make sure you are working with a copied operation when you change the default RAS.

Changing RASes at Runtime

- You can select a RAS at runtime by supplying a value to the `${overrideRAS}` or `${overrideNRAS}` inputs
- The value for the input is simply the name of the RAS reference as it exists in the Configuration/Remote Action Services folder
- This technique allows you to “programatically” decide which RAS to use in a particular step
- Example: Ping step

The value **localRAS** is assigned to `overrideJRAS`, so this step will execute its action on that RAS



To provide a value for an override RAS, simply create an input for a step and name it `overrideJRAS` or `overrideNRAS`, depending on whether you are accessing Java or .NET iActions. Then simply the name of a RAS reference as the value of that input. Subsequently, the step is directed to the `overrideRAS`.

Deploying New iActions to a RAS

- Creating new iActions allows you to add functionality to OO
- Steps to deploy new content:
 - Follow the steps in the SDK Guide for packaging content into a JAR (Java Archive File) that can be deployed to the RAS
 - Copy the JAR file to the RAS repository, restart the RSJRAS service
 - In Studio, create an operation from the new RAS content
 - “Customize” the operation(s) as needed (change icons, modify responses, set primary outputs, add a description, etc)
- Once deployed you have a new RAS operation that you can use in flows



One of the main advantages of the RAS architecture is the ability to easily create and deploy new content to the RAS.

Follow the instructions in the OO SDK guide for creating and packaging a JAR file or DLL that contains the iAction(s) to deploy.

Once the new Java or .NET is packaged, copy the file to the RAS repository directory and restart the RSJRAS service.

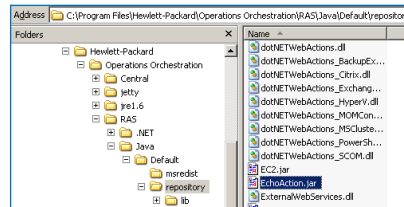
Then in Studio, select a folder and under the File menu select Create Operations from the RAS. You can then select the iActions you want.

The new content is imported as individual operations for the iActions you selected, and these operations can then be used in flows.

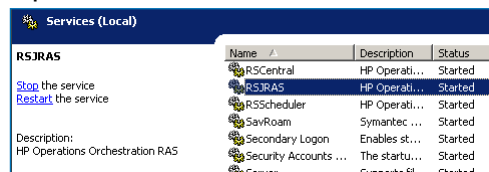
Example: Deploying a New iAction

To deploy a new iAction (EchoAction.jar):

1. Copy your Jar file to ../RAS/Java/Default/repository

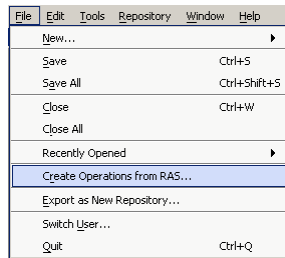


2. Open Services Control Panel, restart the RSJRAS service

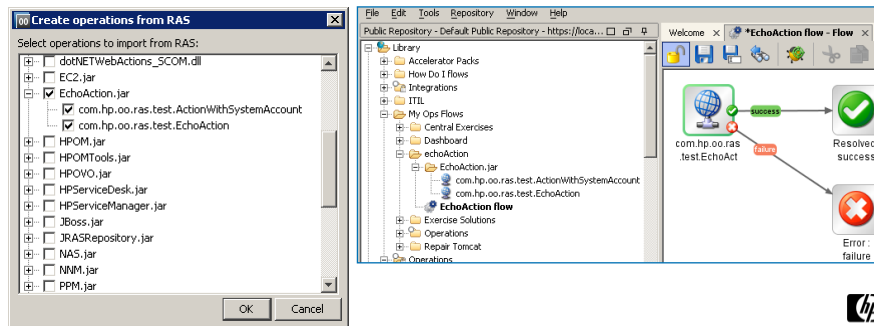


The slide shows the process of adding a new JAR file to the OO RAS repository and then restarting the RSJRAS service.

Create OO Operations From New iActions



- In Studio, create a new folder for your content, then highlight that folder in the Library
- Open File/Create Operations from RAS...
- Select the deployed content for creating a new operation
- New content imports into the selected Library folder – check in and use in flows



Once the new content is deployed to the RAS, this slide shows the steps to create operations from the deployed iActions.

Important File Locations

- ../RAS/Java/default/repository
 - This is where all of the iAction libraries live (the operations that do all of the work)
 - This is also where all of the 3rd-party libraries for .NET-based operations
- ../RAS/Java/default/repository/lib
 - This is the lib directory for non-action Java libraries. Sub-folders exist in this directory for libraries required by an iAction library.
- ../RAS/Java/default/webapp/conf
 - Directory where all of the configuration files live
- ../RAS/Java/default/webapp/logs
 - Directory where all of the log files are written



There are a number of important files for maintaining a RAS, as shown on this slide.

Important Files

- In %ICONCLUDE_HOME%/RAS/Java/default/webapp/conf:
 - **jetty.xml** – Controls how the jetty service is configured. Things like ports, https, client authentication, etc. are configured here
 - **iras.properties** – Configuration for http/https, port, heartbeat, and Kerberos
 - **log4j.properties** – Configuration for the logger
 - **raskeystore.jks** – Where all of the SSL certificates are stored
 - **wrapper.conf** – Configuration file for the jetty service. Controls things like the JRE to use, any system configuration properties that should be passed to the JVM, enable/disable remote debugging, JVM memory sizes, and default log levels



In the weapp/conf directory you will find the files listed above.

RAS Advanced Topics

Disabling Client Authentication

- Allows direct communication with the RAS with a web browser for troubleshooting
- Two options:
 - Add a client certificate to both the browser and raskeystore.jks
 - Remove SSL authentication from jetty.xml

```
>><Set name="NeedClientAuth">false</Set>
```
- Restart the RSJRAS service after editing the jetty.xml file



Disabling client authentication allows you to directly communicate with the RAS with a web browser for development and debugging.

The slide outlines the process for adding a client certificate, editing the jetty.xml file, and restarting the RSJRAS service.

RAS Advanced Topics

Enable Remote Debugging

- Useful when creating new iActions
- Connect a Java debugger to the RAS service by editing the wrapper.conf file:

```
# uncomment one of these to start it in debug mode on port 8070
#wrapper.java.additional.5=Xdebug -
  Xrunjdwp:transport=dt_socket,address=8070,server=y,suspend=n
#wrapper.java.additional.6=Xdebug -Xnoagent -Djava.compiler=NONE -
  Xrunjdwp:transport=dt_socket,address=8070,server=y,suspend=y
```
- The two lines simply trigger different debugger options
- Important – the numbers shown in the file must be sequential. If you want to use the second debug option, you would uncomment that line and change 6 to 5
- Restart the RSJRAS service after enabling remote debugging



You can also enable remote debugging as shown on the slide.

Note that the debug levels are cumulative – that is, to enable level 6 you would need to have all other levels through 5 also enabled.

Summary

- Remote Action Service (RAS) allows you to extend OO functionality to remote hosts and networks
- RAS installs with Central, and you can also install a standalone RAS on hosts on a different domain or behind a firewall to extend OO functionality
- Operations that use RAS require a RAS Reference, which is a URL that directs the flow to the RAS
- You can configure many RAS references in OO, which maintains a list of available RASes
- You can specify a proxy server in a RAS reference for communicating across a firewall or to use the SA Gateway
- You can import RAS operations into Studio to review available operations or use them in flows
- You can deploy IActions to a RAS to add functionality that is not available in a standard OO deployment

23



Remote Action Service is a key component of Operations Orchestration, and understanding how it works is key to your work as a flow author in OO.

Exercises

Remote Action Service

- View/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 a RAS

24



Refer to the Lab Guide for the Remote Action Service exercises.

Controlling Access to OO Objects



In this section you will learn how to control access to objects in the Operations Orchestration Library.

Objectives

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

- Control access to folders, flows, operations, and configuration items
- Assign Read/Write/Execute/and Link To permissions to groups for an object
- Observe how changes to access control are reflected in Central and Studio
- Use a Gated Transition to control which group can execute a step in a flow run
- Hand-off a flow run to another user

2

7.5 Rev C



Here you will learn how to control access to folders, flows, operations, and configuration items using properties in the OO Library.

You will learn about read, write, execute, and link to permissions and see how changes to access control are reflected in Central and Studio.

You will learn how to use a gated transition to control which group can execute a step in a flow and how to hand off a flow run to another group.

Controlling Access to OO Objects

- Permissions are access rights to OO objects
 - Folders, flows, operations, system accounts
- Permissions are assigned to groups, not individuals
- Available permissions:

Read	Author can see an object in Studio User can see an object in Central
Write	Author can modify or delete an object
Execute	User can start a flow run in Central or Studio User must have read-write access to all objects in the flow
Link to	Author can use an object in a flow, such as creating a step from an operation or subflow

3

7.5 Rev C



Permissions define access rights to OO objects: folders, flows, operations, and system accounts.

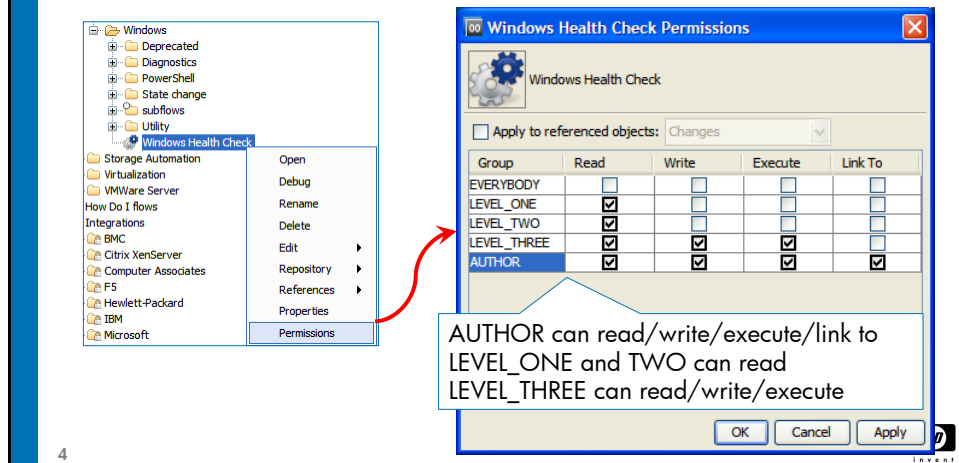
Permissions are always assigned to groups, not users, following the OO authentication and security model that states that capabilities in OO are assigned to groups, and members of a particular group inherit the capabilities assigned to the group.

The available permissions are:

- Read – The author can see an object in studio, and the user can see it in Central
- Write – The author can modify or delete an object
- Execute – the user can start a flow run in Central or Studio – that user must have read-write access to all objects in the flow
- Link to – An author can use an object in a flow, such as creating a step from an operation or subflow

Assigning Permissions

- Assign permissions to groups in Studio
- Right-click an object in the Library, select Permissions
- Default: Read/Write/Execute/Link To for EVERYBODY



To assign permissions, right-click the object in the library and select Permissions.

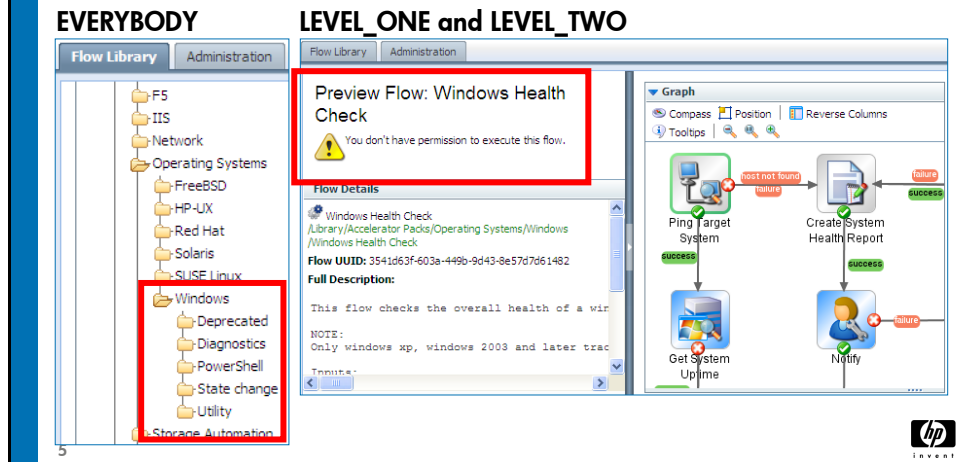
Then assign the permissions as needed to the available groups.

Note that all groups defined in Central will be displayed in the Permissions dialog box, allowing you to assign read, write, execute, or link to permissions to any group you choose in any combination that makes sense for your environment.

The default permission for objects in the Library is read-write-execute-link to for EVERYBODY, or essentially all users in the system.

Views in Central

- EVERYBODY: Flow is invisible in Central/Studio
- LEVEL_ONE and LEVEL_TWO: Flow is visible but can't be run
- LEVEL_THREE and AUTHOR: Can view and run the flow



The slide shows what happens when a particular object is hidden from a particular group's view – users of that group see a message saying they don't have permission to execute a flow in this example.

Controlling Flow Runs

- Use a Gated Transition to control who can run subsequent steps in a flow
- With a Gated Transition you can:
 - Specify which group can execute subsequent steps
 - Hand-off the flow run to a different user
- At runtime an email is generated and can be sent to whoever should complete the flow run
- The current run status is reflected in Current Runs where users can resume a flow run that has been assigned to their group

6



A Gated Transition allows you to control who can run subsequent steps in a flow.

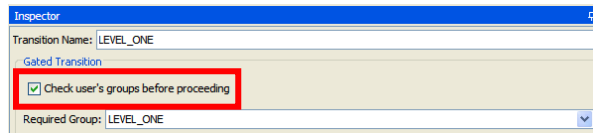
With a gated transition you can select the group that is permitted to execute the next step(s) in the flow or hand off a flow run to another user.

Email is the default method for handing off a flow run. The user receives an email with a link to continue the flow run. The user clicks the link, logs in if necessary, and continues the flow run.

This is all reflected in Central's Current Runs tab, which provides a nice interface for taking control of handed off flow runs, among other useful tasks.

Using Gated Transitions

- To require a group before allowing a flow run to proceed:
 1. Double-click a transition
 2. Select Check User's Groups Before Proceeding
 3. Select the group from the pull-down menu
- Transition turns red in the Design Panel

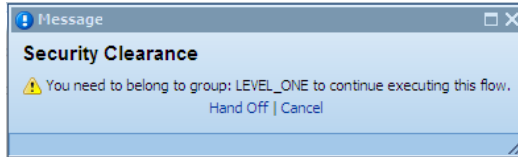


To specify a gated transition, double-click the transition and select Check user's groups. Then select the group that is allowed to complete the next step from the pull down menu.

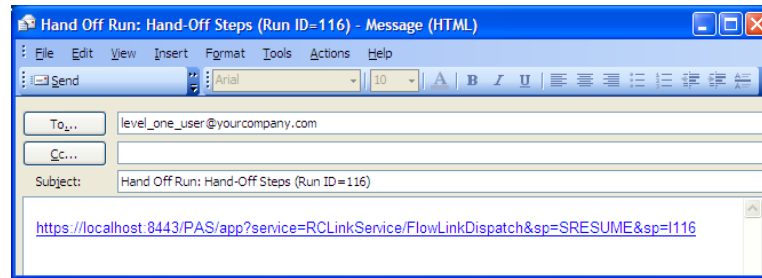
Once a gated transition is specified, its arrow becomes red when it is displayed in Central or Studio.

Central UI for Gated Transitions

- At runtime the user is prompted Hand Off to the specified group:



- Handing Off generates an email with the execution link:



When the flow is run, an email is generated from the Security Clearance dialog box. That email can be addressed to a user that belongs to the required group, and the email contains a link that will resume the flow run.

If the user is not logged into Central upon clicking the link, the user will need to log in and then proceed with the flow run. The flow run proceeds automatically once user authentication is completed.

Continuing the Flow Run

- Recipient clicks the link, logs in, and continues the flow run
 - Next Step or Run All continues flow run
- Flow Run is also reflected in Run Administration for the user handing off the flow and the recipient

The screenshot displays the 'Run Administration' tab in the Hp Operations Orchestration Studio. The interface shows the following details:

- Run Status:** READY
- Current Step:** Step 1
- Started:** 15:20:38 PDT 2009-10-09
- Options:** (icon)
- Controls:** Next Step, Run All, Pause
- Results Summary:**
 - Last Completed Step: **No results yet**
 - As of: 15:20:38 PDT 2009-10-09
 - Page: 1 of 1
- Table:**

Step	Response	Message
Step 1	✓	
- Display most recent:** 50 Items per page
- Update** button

On the right side, the **Flow** and **Graph** sections are visible. The **Flow** section shows 'Hand-Off Steps' with a description: '(no description available) /Library/My Ops Flows/Hand-Off/Hand-Off Steps'. The **Graph** section shows a flow diagram with three steps: Step 1 (computer icon), Step 2 (globe icon), and Step 3 (globe icon). Step 1 is highlighted with a red border. The flow is labeled 'LEVEL_ONE' between Step 1 and Step 2, and 'LEVEL_TWO' between Step 2 and Step 3.

The slide shows how the flow run continues in Central once it is handed off to a user who has logged into Central to continue the flow run.

Handing Off a Flow Run

- You can also hand-off a flow run without specifying a required group
- Select Hand Off Flow Run in the Transition Inspector
- Transition changes to an outline arrow
 - If handing off and specifying a group, icon changes to a red outline arrow

Transition Name: LEVEL_TWO

Gated Transition

☐ Check user's groups before proceeding

Required Group: AUDITOR

☒ Hand off flow run after this transition



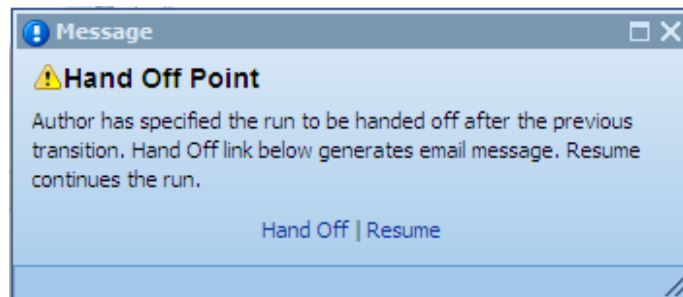
You can also simply hand off a flow run to another user without specifying a particular group.

Select Hand off in the transition's inspector window.

When this is checked, the arrow becomes a grey outline in Studio and Central.

Central UI for Handing Off a Flow Run

- At the hand-off transition, the user is prompted to either hand-off the flow run or resume
- Hand Off generates an email, Resume continues the flow run



11



When that transition is reached, the Hand Off Point is generated. At this point you can either hand off or continue the run.

Handing off generates an email, like the one generated when checking groups.

Summary:

- In Studio you can assign read, write, execute, and link to permissions to specified groups
- Permissions are reflected in the library views in Studio and Central for the specified groups
- A Gated Transition allows you to specify which group can execute the step(s) immediately following the transition
- At runtime a message and email are generated notifying the user that a required group has been specified
- You can also hand off a flow run without specifying a group
- Handoffs are reflected in Current Runs

12

7.5 Rev C



Permissions and gated transitions are powerful tools that allow you to implement an advanced security scheme for your flows.

Permissions pertain to the permissions specified in the flow library – read, write, execute, and link to.

Gated transitions control which groups can execute subsequent steps in a flow.

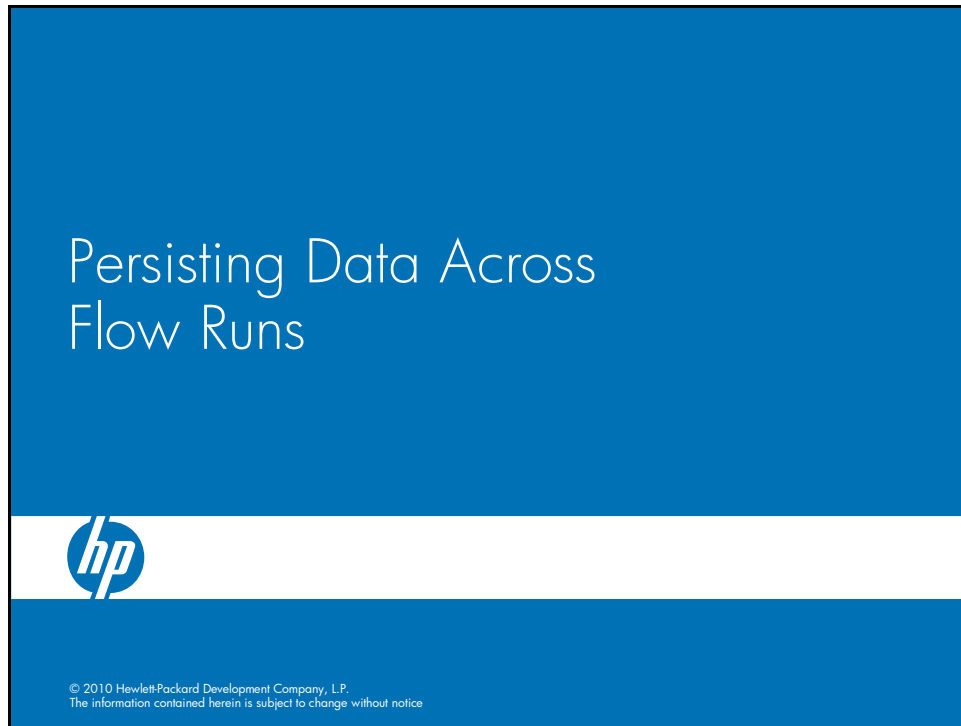
You can also hand off a flow run to another user.

Gated transitions are reflected in the Current Runs tab, which provides an interface for managing flow runs.

Exercise

- In the exercises for this section you will:
 - Control access to OO objects
 - Control access to steps

Persisting Data Across Flow Runs



In this section you will learn various ways of making flow data persistent, or available, across different flow runs.

Objectives

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

- Explain a number of ways of persisting data across flow runs
- Use System Properties to store data for use in flows
- Use data persistence content in the OO library to persist data
- Use System Properties in a flow
- Build a flow that uses data persistence content

2



In this section you will:

- Explain the different ways of persisting data across flow runs
- Use System Properties to store data for use in flows
- Use the data persistence content in the OO Library to persist data

Use defined System Properties in a flow

In the exercise you will build flows that use persistent data.

Strategies for Persisting Flow Data

- Persisting data means making data available across different flow runs
- There are a number of ways of persisting data:
 - Write the data to a database for later retrieval during a flow run
 - Write values to a file and then read those values during a flow run
 - Use System Properties to store values
 - Use data persistent operations in the OO library
- This section focuses on using System Properties and data persistence content to persist data

3



Persisting data means making it available to different runs of the same or different flows. There are a number of ways of accomplishing this.

You can write the data to a database or file and then retrieve it later

Use System Values to “permanently” store values that can be access by all flows

Use the data persistence content in the OO library to make data available across different flow runs

These are all valid ways of accessing data from one flow run to another. This section focuses on using System Properties and persistence content to make flow data persistent in the system.

System Properties

- A System Property is a configuration item located in the OO Configuration folder
- Adding a system property creates a flow variable in the global context with the value you specify
- You can use system properties to store a wide range of information that can be accessed by flows running on your instance of OO
- Using System Properties provides an efficient way of making data available to flow runs

4



System Properties are very widely used in OO to store parameters for later use in flows.

A System Property is defined in the System Properties folder in the Configuration folder of the repository.

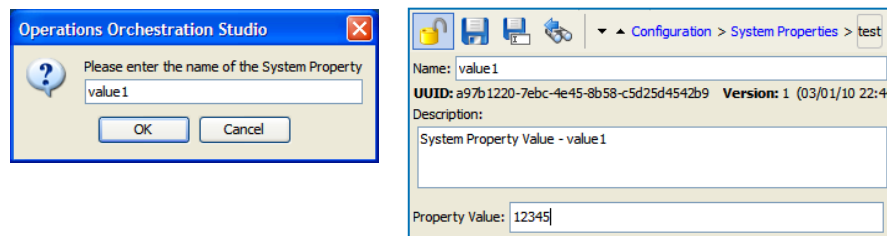
A System Property is basically a name-value pair. The name becomes the name of the flow variable, and the value stored in the System Property becomes the value of the flow variable.

Flow variables from System Properties are basically permanently available to all flows running in a particular instance of Operations Orchestration. Once defined, all you need to do is select the System Property when defining an input, and the associated value is assigned to whatever input you are defining.

Closely related are System Accounts, which allow you to store a user name and encrypted password, which can then be used for steps that require authentication. This is a convenient way of securely storing system accounts in OO. A System Account can be anything – an operating system username/password, credentials for accessing a particular host or program, a Central or data base user, or any other account that your flow needs to access.

Creating a System Property

- To create a System Property:
 - Right-click the Configuration/System Properties folder, select New
 - Name the property
 - Open the system property and enter the value and description
 - Save, check in



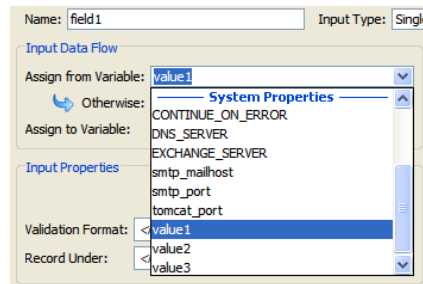
To create a System Property, right-click the System Properties folder in Configuration. Name the System Property and give it a value.

Once defined, as shown in the example on the slide, a global variable named value1 is defined in the system containing the value 12345. This value can then be referenced in flows, assigned to inputs, etc.

You need to check in a System Property after it is defined.

Assigning a System Property to a Flow Variable

- Since System Properties are simply available in the global context, they don't need to be created and have values assigned – simply refer to them.
 - For example, a system property named value1 containing the number **1** would simply be referred to as **\${value1}**
- You can also select the System Property in the pull-down menu in the Input Editor to assign it to a flow variable:



The slide shows how to assign a System Property to a flow variable – for Assign from, you simply scroll down to the System Properties section and select the defined property. The value associated with that property then becomes the value of the flow variable.

Limitations of System Properties

- There are a number of limitations to be aware of:
 - System Properties are globally available to all flows in your instance of OO
 - A System Property and its value are visible only in the Studio debugger because they are not created by the flow author like inputs or responses

7



System Properties have many advantages and a few limitations.

First, System Properties are globally available to all flows in your instance of OO. If this might cause a problem, don't use a System Property for the value you want to use.

Second, a System Property and its value are only available in the Studio Debugger because they are not created by a flow author the way inputs and responses are.

Content in the OO Library can be used to create and change values of System Properties. You can also use overrides in Central to provide temporary override values to defined System Properties.

Using Data Persistence Content

- The second way of persisting data covered in this module is using OO content in the Library to persist data across flow runs
- Two operations for persisting data are in:
 - Library/Integrations/Hewlett-Packard/Operations Orchestration/Cross Run Data Persistence



Store Flow Variable – *Stores a flow variable*



Get Stored Flow Variable – *Retrieves a stored flow variable*

8

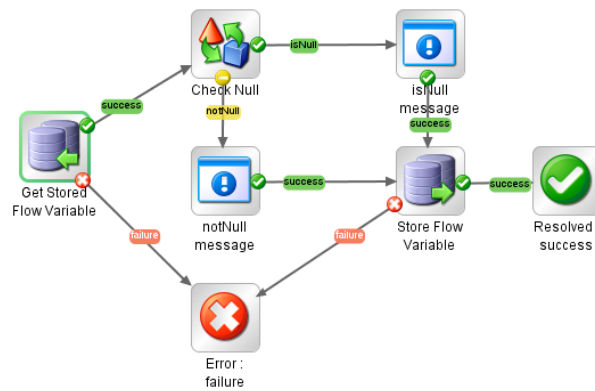


Another way of persisting data is to use data persistence content available in the Operations Orchestration folder in Integrations.

There are two operations: Store Flow Variable and Get Stored Flow Variable. Both are very simple to use, as shown in the following slides.

Example: Persisting Data

- This flow persists flow data across flow runs
 - Get Stored Flow Variable attempts to retrieve a stored flow variable
 - If that value is null (empty) it obtains a value and stores it with the Store Flow Variable step
 - Subsequently the stored value will be available



In this flow:

- Get Stored Flow Variable attempts to retrieve a stored flow variable.
- If that value is null (empty) it obtains a value at the notNull Message step and stores that value with the Store Flow Variable step
- Subsequently the stored value will be available and displayed the next time the flow is run.

This is one of the flows you will be authoring in the exercise.

Storing Flow Variables



Store Flow Variable – *Assign values to inputs*

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

- Inputs and their values are stored for later retrieval by the Get Stored Flow Variable step

10



This slide shows the Step Inspector for storing flow variables.

Retrieving Stored Flow Variables



Get Stored Flow Variable – *Retrieves a stored flow variable*

- Assign flow variables to keys input
- Create a result based on the Result Field: ResultTable, assign to a flow variable

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

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

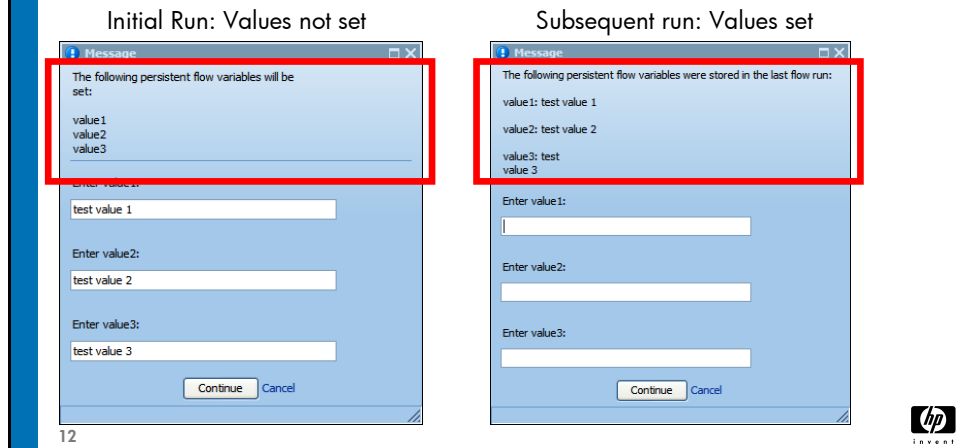
11



This slide shows the Step Inspector for Get Stored Flow Variable.

Running the Flow

- When you run the flow, initially values are not set
- Subsequently values are retrieved
- Values can be changed, new values are persisted



When you run the flow, the initial values for the flow variable shown are not set. At this point you would enter the values in the fields as shown.

The next time you run the flow, the values are set and can be changed by entering new values in the fields.

Summary

- This section covered two ways of persisting flow data: System Properties and persistence content in the OO library.
- System Properties allow you to define variables that are subsequently globally available to all flows in your instance of OO. Closely related are System Accounts which allow you to securely store authentication data.
- You can use the Input Editor for an input to assign the stored value to a particular flow variable.
- Persistence content in the Operations Orchestration folder in Integrations allows you to persist data from one flow run to the next.
- You can also modify the stored data using the persistence content.

13



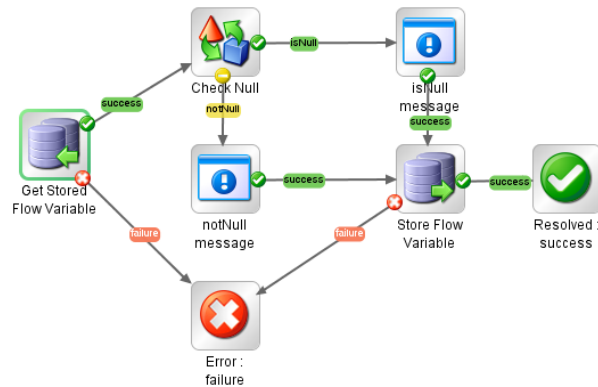
Review the summary for this section. You will find that when used correctly, System Properties are a very easy and convenient way to make data available to all flows in your library.

System Accounts, closely related to System Properties, allow you to securely store authentication data in OO for use in steps that require logins or other authentication parameters.

Persistence content is another way of persisting data across flow runs and can be very useful in many different situations, depending on your particular application.

Exercise: Persisting Flow Data

- In the exercise you will:
 - Use System Properties to store flow data
 - Write a flow that uses Store Flow Variable and Get Stored Flow Variable to persist data

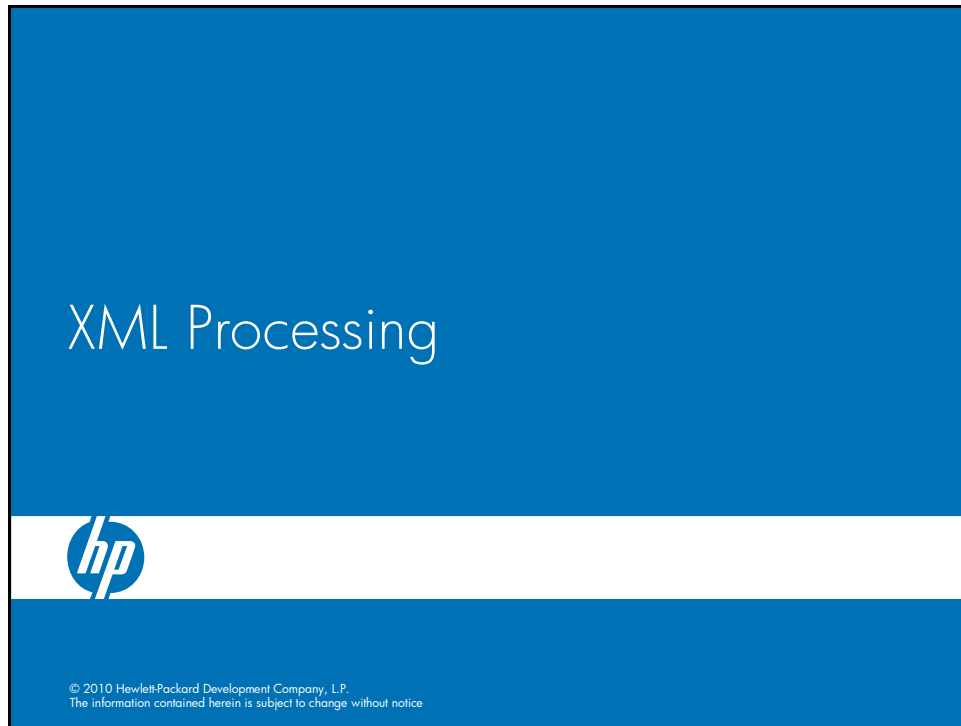


14



In the exercise you will use System Properties and persistence content to persist data.

XML Processing



In this section you will learn how to use OO's XML Processing capabilities to extract data from XML documents.

Objectives

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

- List and describe the operations available in OO for working with XML
- Explain how to use iterative operations to compile data extracted from XML documents
- Build a flow that validates and parses an XML document
- Build a flow that validates and parses XML data returned from a Web Service SOAP inquiry
- Use XML filters to extract data from an XML document

2



The goal of this module is to use the new XML operations available in the Library for processing XML content as well as the XML filters available in the Filter Editor.

XML Operations

- XML operations are located in Library/Utility Operations/XML Processing
- All require XML input, most require some kind of filtering criteria
- Listed operations iterate if more than one value is found

Operation	Description	Iterates?
Validate XML Document	Determines whether XML input is valid	No
XML Element Filter		Yes
XML Get Attributes	Retrieves attributes from root	Yes
XML Get Element Value	Retrieves a value from an element	No
XPath Evaluator	Evaluates XPath expressions	Yes

3



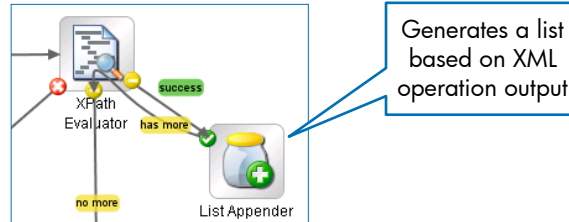
Operations Orchestration contains content in the library specifically designed for dealing with XML content.

The slide lists the operations, provides a brief description, and shows whether they are iterative operations or not.

Most of the XML operations iterate through the document so the process of working with iterative operations in OO apply to the XML Element Filter, XML Get Attributes, and XPath Evaluator.

Iterative Operations

- XML Element Filter, XML Get Attributes, and XPath Evaluator iterate if more than one attribute/element is found
- Use with an iterative operation
- Example: XPath Evaluator and List Appender



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

4



The iterative operations are typically paired with another operation that is intended to operate iteratively, like the List Appender.

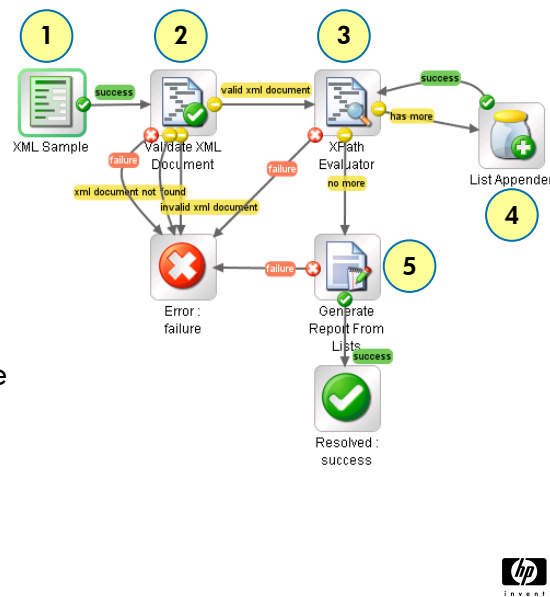
In the example on the slide, the XPath Evaluator operation iterates through the XML document and finds data that matches the XPath expression.

Whenever it finds data that matches the XPath expression it sends the result to the List Appender where it is added to a list. Control then goes back to XPath Evaluator and the process repeats until the No More condition is met. At that point the flow continues to the next step.

Example

This flow:

1. Generates XML data
2. Validates the document
3. Filters the specified element
4. Generates a comma-separated list
5. Writes a report to the Display



The example flow:

- Generates sample XML data and sends the document to the Validate step
- The Validate step checks whether it is a valid XML document. If it is control goes to the XPath Evaluator. If not, it goes to the error step.
- The XPath Evaluator prompts the user to enter an XPath expression and iteratively compiles a list using the List Appender operation.
- When there is no more data to evaluate, the compiled list goes to the Generate Report operation, which presents the user with a formatted report of the results of the XPath expression.
- Finally the flow ends at the Resolved step.

XML Filters

- You can also create OO filters for XML operations:
 - XML Get Attribute
 - XML Get Element
 - XML Get Element Value
 - XPath Query
- The XML filters provide an easy and very fast way to extract data from XML output
- To use:
 - In the Filter Editor, Add a filter
 - Select the appropriate XML filter from the list
 - Fill out the form
 - Test

6



The content in the OO library for dealing with XML was added in version 7.50. In version 9, XML filters were added. Using XML filters is much easier and faster than using the XML content in the OO library. While the XML content in the library can still be used if needed, using the XML filters will probably prove to be a better alternative for many or most applications involving XML.

XML Get Attribute

- To retrieve the attribute value for version:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CATALOG>
  <PLANT version="1.0">
    . . .
```

Single Match:

Details for: XML Get Attribute
Filters an xml document for the requested attribute value. For advanced XML filtering use the XPath Filter.

Element Path:

Include sub-elements: ☐

Attribute Name:

Result: ☒ Single Match
☐ As Table

Test Output

1.0

As Table:

Details for: XML Get Attribute
Filters an xml document for the requested attribute value. For advanced XML filtering use the XPath Filter.

Element Path:

Include sub-elements: ☐

Attribute Name:

Result: ☐ Single Match
☒ As Table

Test Output

Path, version
/CATALOG/PLANT[1], 1.0

7



The first filter, Get Attribute, does just that- it gets an attribute from an XML document named version in the specified path.

In this example, a sample plant catalog is being queried for an attribute value. The Single Match shows simply the value of that attribute. The As Table match shows the path and the value of the version attribute.

XML Get Element

- Returns complete element by:
 - Path
 - Child
 - Attribute

Path:

[Details for: XML Get Element](#)
Filters an xml document for elements given a path. For advanced XML filtering use the XPath Filter.

Element Path:	/CATALOG/PLANT		
Child Named:		Value:	
Attribute Named:		Value:	

Child:

[Details for: XML Get Element](#)
Filters an xml document for elements given a path. For advanced XML filtering use the XPath Filter.

Element Path:			
Child Named:	ZONE	Value:	
Attribute Named:		Value:	

Output:

[Test Output](#)

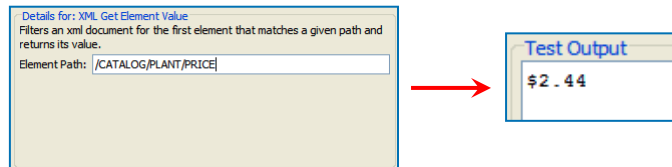
```
<PLANT version="1.0">
  <COMMON>Bloodroot</COMMON>
  <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
  <ZONE>4</ZONE>
  <LIGHT>Mostly Shady</LIGHT>
  <PRICE>$2.44</PRICE>
  <AVAILABILITY>031599</AVAILABILITY>
</PLANT>
<PLANT>
  <COMMON>Columbine</COMMON>
```

8

The XML Get Element filter takes either an element path, a named child, or a combination to produce filtered XML output, shown in the Output field on the slide.

XML Get Element Value

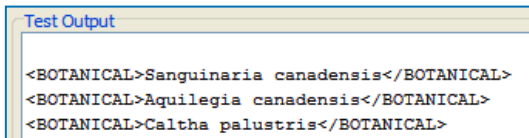
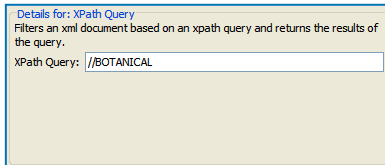
- Filters for the matching element
- Enter the full or relative path to the element
- Examples:
 - /CATALOG/PLANT/PRICE – First matching element value
 - /CATALOG/PLANT[3]/PRICE – Third matching element value



The Get Element Value takes the specified path and provides the value. Review the examples on the slide.

XPath Query

- The most versatile of the XML filters
- Enter an XPath
- Returns values that match the given XPath



10



XPath Query is the most flexible of the filters. It allows you to specify an XPath query of any complexity, and returns the values that match the query. As a reminder, you can do a Google search on XPath Query to see many different tutorials on how to construct an XPath. XPath expressions can be very general to very detailed and offer virtually unlimited searching capabilities.

Summary

- A number of operations are available for working with XML data:
 - Validate XML Document
 - XML Element Filter
 - XML Get Attributes
 - XML Get Element Value
 - XPath Evaluator
- Or you can use XML filters to extract data from XML output
- Filters are very easy and fast to use
- Available filters:
 - XML Get Attribute
 - XML Get Element
 - XML Get Element Value
 - XPath Query – recommended/preferred method

11



Review the summary. As a reminder, the content in the library for working with XML is very useful but in many or most cases can be replaced by easy-to-use filters applied to flow variables that contain XML output.

Exercise

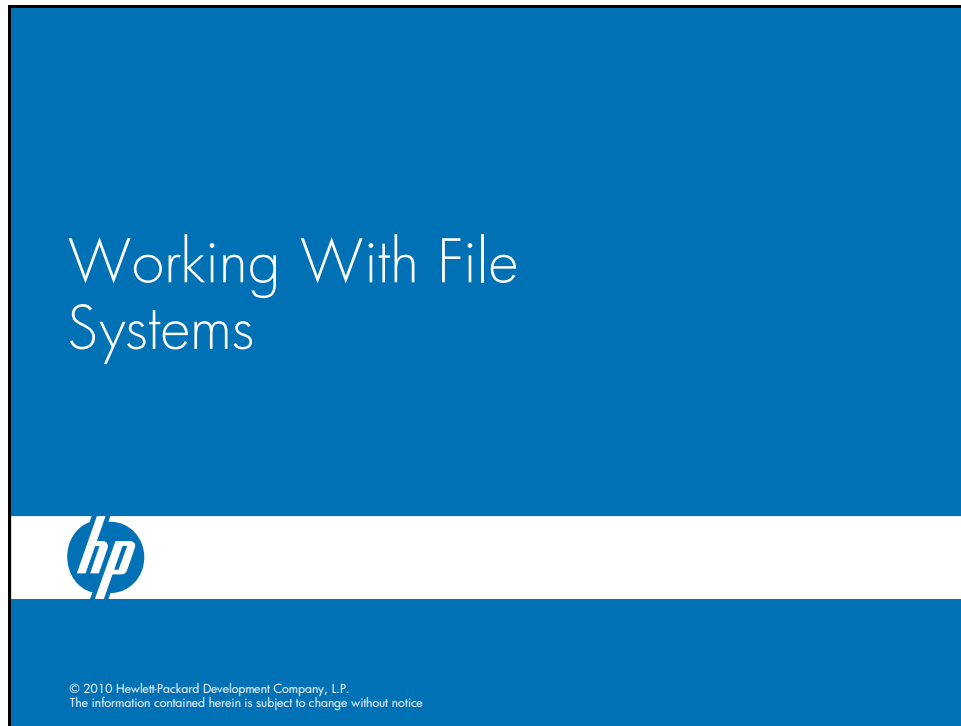
- Author a simple XML validation flow
- Author a flow that parses XML data using XPath Evaluator
- Use an XML XPath filter to retrieve data from an XML document

12



In the exercise you will author a flow that evaluates an XML document supplied by the XML Sample operation. The flow is similar to the one shown on the previous pages.

Working With File Systems



In this section you will learn how to use content in the OO library for working with file systems.

Objectives

By the end of this module you will be to:

- Locate and use the File System content in the OO library
- Differentiate between cross-platform and Windows-only file system content
- List the contents of a directory
- List only subfolders/subdirectories
- Write data to a file
- Read and filter data and store in an OO flow variable

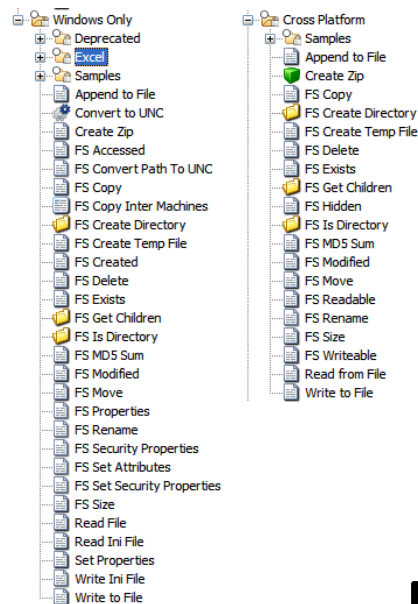
2



Review the objectives – basically you will learn how to read, write, and list external file systems with content in the OO library.

Working With File Systems

- OO has a large library of content for working with file systems
- File system content is in Operations/FileSystem
- It is divided into two broad categories: Windows Only and Cross Platform
- Windows Only contains more content including operations for working with Excel documents
- Cross Platform content works on any platform including Windows



Operations Orchestration has a large amount of content in the library for dealing with file systems. Just browsing the names of the operations tells you a lot about what they do – most of these are basically self-explanatory.

The important thing to note is the presence of the Windows Only and Cross Platform folders.

The Windows Only folder contains more content, with many operations for working specifically with Windows files like properties, Ini files, etc.

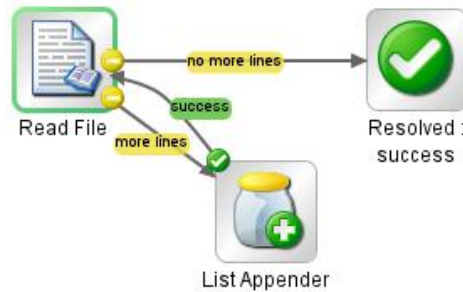
Additionally the Windows Only content supports windows authentication.

Many users prefer to use the Cross Platform content when possible because it is simpler to use, as fewer inputs, and bypasses the Windows authentication. Obviously you would use what ever content category is appropriate for your application, but generally speaking the Cross Platform content is a little easier to implement.

It's a good idea to look at the Samples folder for each platform category.

Reading a File

- Read File iteratively reads a file line-by-line
- As an iterative operation you need to tie the more lines response to some other step – List Appender in this case
- Read File also has a Filter allowing you to include only those lines that contain the filter you specify



4

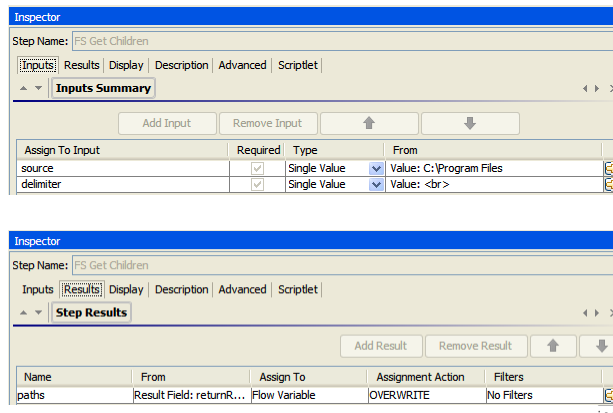
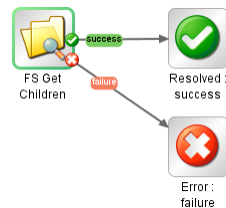


This example shows one of the simplest tasks, reading from a file.

The Read File operation iteratively reads each line of the file. To make the entire file available in a flow variable, you can simply append each line in List Appender. Optionally you can set filters in Read File to append only those lines that match the filtering criteria.

Listing Contents of a Directory

- FS Get Children retrieves all the contents of a directory
- Inputs are simply the source directory to evaluate (Source) and delimiter to place between each retrieved path
- Use a Result to assign the retrieved paths to a flow variable which can then be used in your flow



The FS Get Children operation lists the contents of a directory. This is not an iterative operation, it simply provides a directory listing.

You simply specify a source directory to list, and a delimiter to place between each item retrieved in the listing.

Use a result to assign the retrieved paths to a flow variable. The named result should be based on the Result Field returnResult.

List Only Subfolders of a Directory

- Another useful operation is FS Is Directory which acts as a filter which passes only paths that are directories
- The List Directory Subfolders flow:
 - Gets all paths and stores in a result named paths
 - Uses List Iterator to iterate through the list
 - Each item in the list is sent to FS Is Directory
 - If the path is a directory it is added to the list, otherwise it is ignored
 - The result is a listing of subdirectories in a parent directory

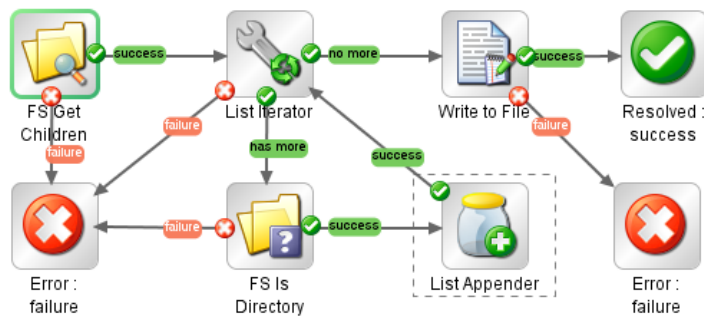


This example shows the use of FS Is Directory, which essentially acts as a filter that passes only those items that are directories. In the example above, FS Get Children compiles the list and the List Iterator goes through the list one item at a time, filtering each entry through FS Is Directory which passes only those items that are directories to List Appender, which compiles the list.

In other words, this simple flow returns a list of directories rather than the entire contents of a source directory.

Writing to a File

- The Write List to File flow writes the compiled list of directories to a file
- The cross-platform version of Write to File does not require username/password, the Windows version does



7



This example shows the directory listing flow from the previous slide but with an added step to write the list to a file. Writing to a file is very simple, you simply specify the full path to the file name you want to write and provide the flow variable content to write to the file, in this case the compiled list of directories.

Write to File Inputs

- Source: Full pathname of the file to write
- Contents: data to write to file
- Delimiter: Newline (\n, \r\n, etc)
- User/password: Windows account (Windows only)

Cross Platform

Inspector

Step Name: |Write to 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
contents	<input checked="" type="checkbox"/>	Single Value	Prompt User
delimiter	<input type="checkbox"/>	Single Value	Prompt User

Windows Only

Inspector

Step Name: |Write 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	Prompt User
Contents	<input checked="" type="checkbox"/>	Single Value	Prompt User
user	<input type="checkbox"/>	Single Value	Prompt User
password	<input type="checkbox"/>	Single Value	Prompt User
Delimiter	<input type="checkbox"/>	Single Value	Prompt User

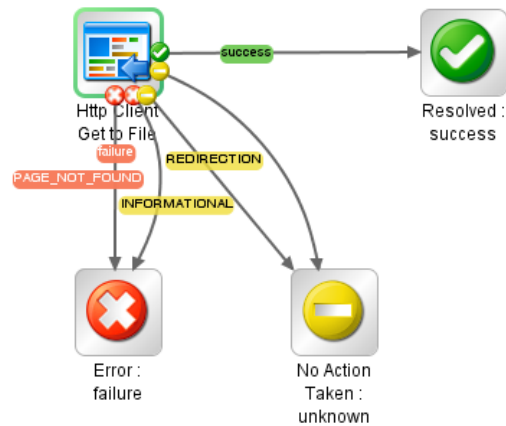
8



This slide shows the differences between the inputs for the Cross Platform and Windows Only versions of Write to File.

HTTP Get to File

- Other content in the OO library writes to disk
- HTTP Client Get to File does an HTTP Get operation and writes the results to disk



9

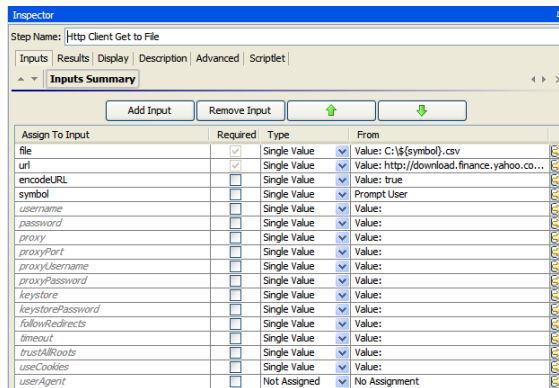


In this example, you are using an HTTP Get To File to perform an HTTP Get operation and write the result to a file. In the lab exercise you will use this flow to retrieve a stock quote from Yahoo Finance and write it to a file. You will need a clear Internet connection for this flow to work.

Inputs

Example: Get a stock quote, write to file

- File: File name to write – C:\\${symbol}.csv
- URL: Location to retrieve data
- Encode URL: true
- Symbol: Prompts user to enter a stock symbol
- Unused inputs are removed



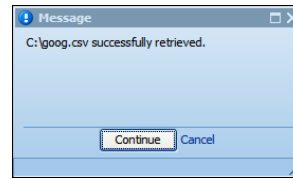
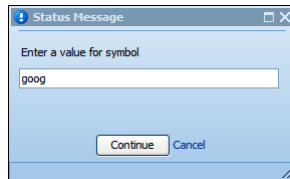
10



Here are the inputs required to get the stock quote with HTTP Client Get to File – the only inputs used are the file name to write (\${symbol}.csv), the URL to retrieve today's quote for \${symbol}, the symbol input to retrieve, and the encoding option (set to true). Unused inputs are simply removed.

Running the Flow

- Enter a stock symbol – stored in \${symbol}
- HTTP Client Get to File does an HTTP Get on this URL:
 - [http://download.finance.yahoo.com/d/quotes.csv?s=\\${symbol}&f=s1d1t1c1ohgv&e=.csv](http://download.finance.yahoo.com/d/quotes.csv?s=${symbol}&f=s1d1t1c1ohgv&e=.csv)
- File is saved to C:\\${symbol}.csv



11



When you run the flow you enter a stock symbol, and the flow gets the stock quote and writes it to a file named symbol.csv in the location you specify.

Exercise: Working With Filesystems

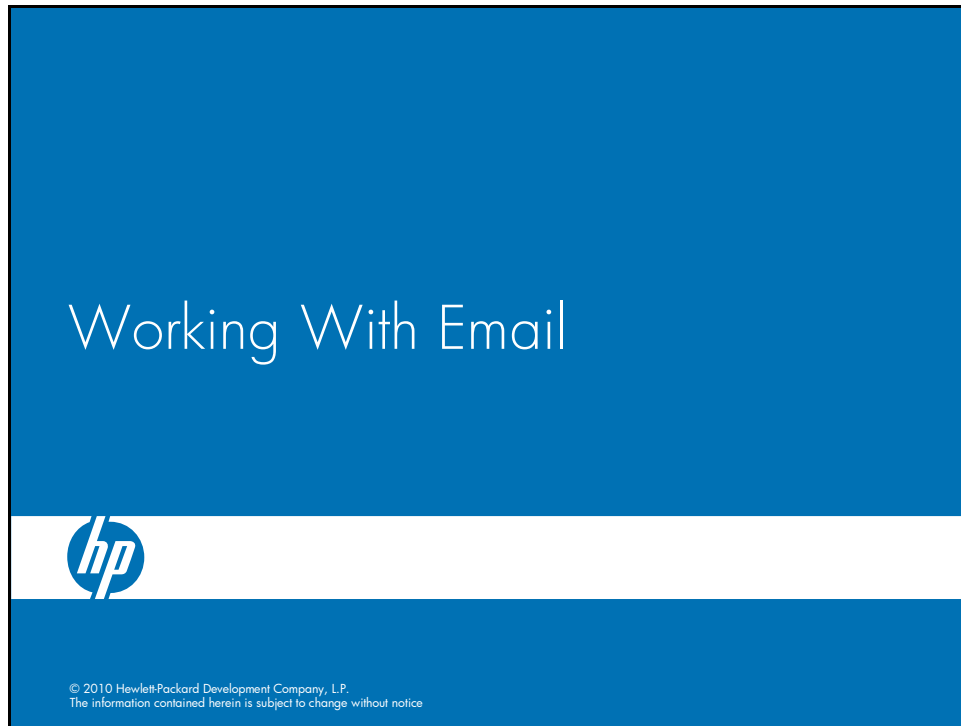
- Author flows to:
 - List contents of a directory
 - List only subfolders
 - Write a directory list to disk
 - Read a file
 - Write a stock quote to disk

12



In the exercise you will write a number of flows that deal with file systems.

Working With Email



This section covers another simple but very useful tool, sending emails from your flows, including details on the flow run itself.

Objectives

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

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

2



In this section and the accompanying exercise you will author the flows shown on the slide.

Working With Email

- Knowing how to send email from an OO flow is a very useful skill
- Emails are typically sent at steps that require user intervention, or at points when you need to provide information to a user
 - Flow run reports
 - Trouble tickets
 - Status reports
- OO email content can send email to the address you specify
- You can interact with POP3 or IMAP email
- Support for Microsoft Exchange Server is supported in the Integration library

3



Working with email is a very important skill for flow authors. Using email allows you to notify users of a particular condition at any point in a flow run. Email can be sent to anyone you specify, and support for Microsoft Exchange Server is supported in the Integration library.

For the exercises in this section you will need access to a mail server. A link to a free mail server is provided in the lab guide for this section, along with installation and configuration instructions. If you are working on the training VM provided for this class, a mail server is already installed and configured.

OO Content for Handling Email

- OO content for email is in Operations/Email:



Get Mail Message – *Retrieves an email from a server*



Send Mail – *Sends an email*



Get Mail Message Count – *Gets the number of messages on the server*



Test and Send Mail (subflow) – *Checks string values before sending*

- Exchange content is in Operations/Exchange
 - Content for managing Exchange servers

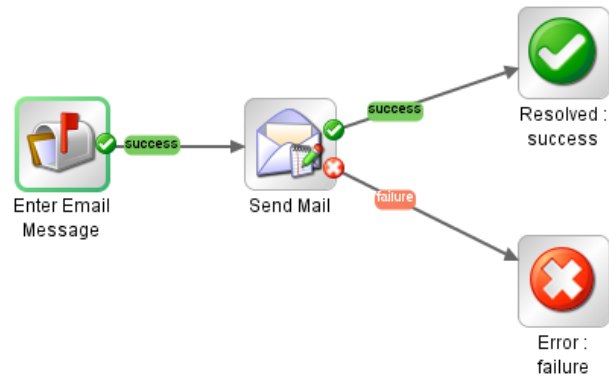
4



In Operations/Email you will find the content for handling email. The commonly used operations are shown on the slide. Exchange Server is not covered in this class but you can find out more by using Studio Help.

Example: Send an Email

- The Simple Send Email flow:
 - Prompts the user for an email message and subject
 - Sends the email to the specified server and email address



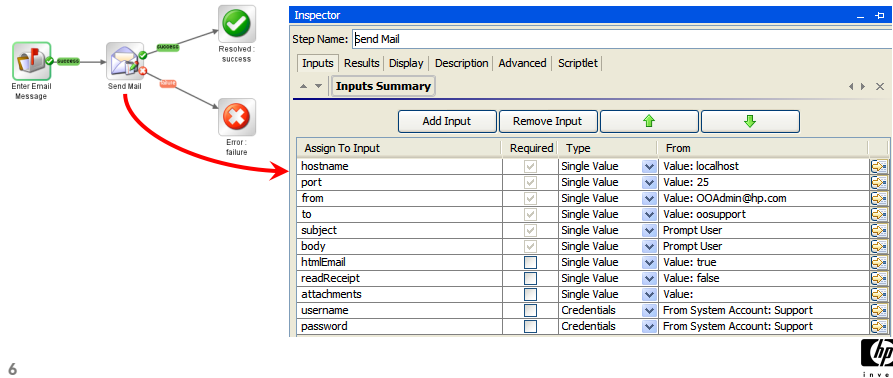
5



This simple flow captures an email subject and body and sends it to the recipient specified in the Send Mail step.

Configuring Send Mail Inputs

- Hostname: The Mail Server host
- Port: The SMTP port number
- From/to: The from/to address on the email
- Username/password: Stored in OO system accounts

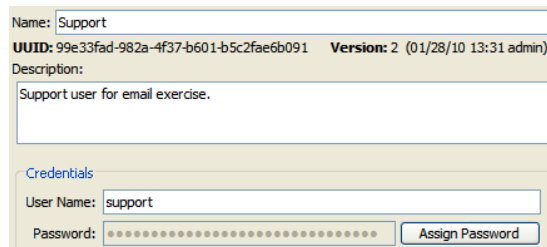


The slide shows the Send Mail inputs. Note that many inputs are not required. The inputs should look familiar – you need the name of a mail host, a port number, a From and To address, a subject and body, and other mail options as shown on the slide.

If you frequently send email to specific users, consider using System Accounts to store their information.

Using System Accounts

- It's a common practice to store email accounts in OO System Accounts
- To create a System Account:
 - Expand Configuration/System Accounts
 - Right-click System Accounts, select New
 - Enter the username and password (passwords are encrypted)
 - Subsequently the username and password can be used in email (and other) steps



The screenshot shows a configuration window for a system account. The 'Name' field is set to 'Support'. Below it, the 'UUID' is '99e33fad-982a-4f37-b601-b5c2fae6b091' and the 'Version' is '2 (01/28/10 13:31 admin)'. The 'Description' field contains the text 'Support user for email exercise.' Below the description is a section titled 'Credentials' which contains a 'User Name' field set to 'support' and a 'Password' field with masked characters. An 'Assign Password' button is located to the right of the password field.

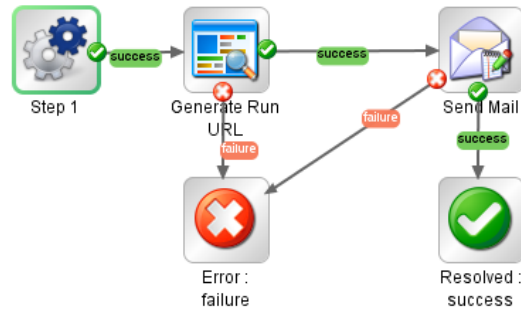
7



This slide shows setting up a System Account for an email user.

Example: Sending a Flow Run URL

- Emailing a flow run report is a common and useful task in OO
- This flow sends the recipient a URL to a flow run report
- Generate Run URL is in Integrations/Hewlett-Packard/Operations Orchestration
- It creates a flow variable named **IC_ReportURL** that can be included in the email body



8



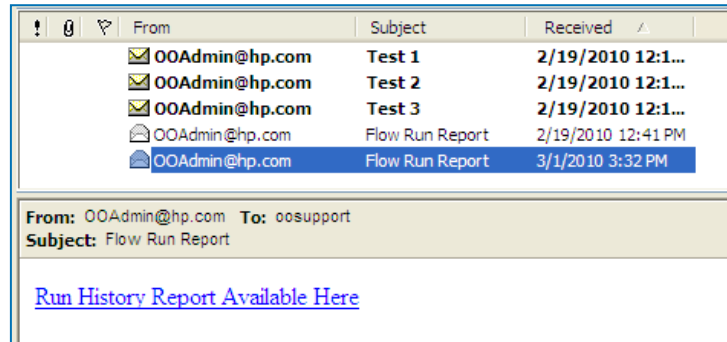
A Flow Run URL is a link to the flow run report in Central that is available at the completion of the flow run.

This is a very valuable tool to use, allowing you to send the flow run report to anyone who might need to see it.

Note that if you run this operation in a flow in Studio you will not see the actual flow run report – the flow must be run from Central to be able to view the actual run report. You can, however, test and debug the flow in Studio even though the actual run report URL is not provided.

Viewing the Report

- The Run URL is provided to the user in an email
- Clicking the link takes the user directly to the flow run report in Central
- Requires logging into Central if the user is currently not logged in

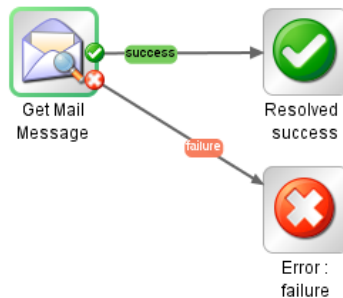


The recipient of the flow run report receives an email with a hyperlink to the run report in Central.

The user will need to log into Central to view the report.

Getting an Email

- You can also get an email from the mail server and handle it in OO
- Get Mail Message gets the message (or subject only) whose number is provided
- You can also use a Get Mail Message Count operation to get all numbers on a server
- Together these operations are a simple way to get message numbers and associated subjects which can then be used in an OO flow



10



You can also retrieve emails using content in the OO library. Get Mail Messages retrieves either an entire message or the subject only, and Get Mail Message Count retrieves message numbers on the mail server.

Together these two operations allow you to summarize email server status and provide summaries or entire emails to users when the flow is run.

Get Mail Message Inputs and Results

- Unused inputs can be removed
- To assign the email text to a flow variable use a Result based on the Result Field **Result**

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
trustAllRoots	<input type="checkbox"/>	Single Value	Prompt User from List - Selection List
enableSSL	<input type="checkbox"/>	Single Value	Prompt User from List - Selection List
keystore	<input type="checkbox"/>	Single Value	Prompt User
keystorePassword	<input type="checkbox"/>	Single Value	Prompt User

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

11

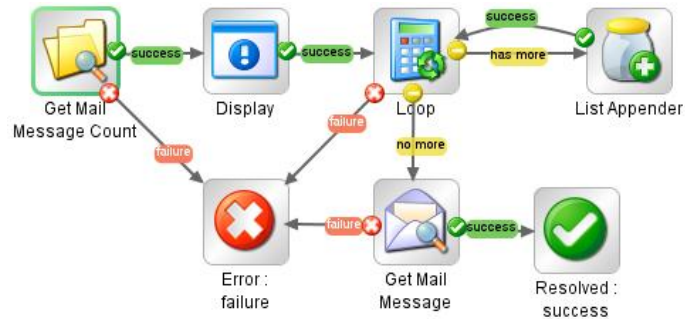


The slide shows the inputs for Get Mail Message – these are very similar to the other email content in the library, requiring a mail host and port, protocol, accounts, etc.

To assign email text to a flow variable, use a Result based on the Result Field: Result. The email flow variable can then be filtered or used in other ways in your flow.

Example: Simple OO Email Client

- This flow serves as a simple email client in OO
 - Get Mail Message Count retrieves the number of messages, shown to the user in the Display step
 - The Loop step compiles a list of message numbers
 - Get Mail Message prompts the user to select a message
 - The text of the message is displayed in the Resolved step



12



This example shows a simple email client implemented in OO. This flow gets the message count, presents a menu of message numbers to the user, then retrieves the select message and displays it to the user.

Configuration Considerations

- You need to know the ports your mail server uses as well as the protocols (IMAP/POP3)
- You may need to configure your email client with the System Accounts you are using in OO
- Outlook Express configuration example – System Account OOSupport:

General tab

Servers tab

Advanced tab

13



This slide reviews some of the configuration options to consider if you install your own email server and are configuring a client to interoperate with it.

In a production you will be interoperating with an established mail server – contact the system administrator for details.

Exercise: Working With Email

- Author these flows:
 - Simple Send Mail
 - Get Mail
 - Email Flow Run Report

14



In the exercise you will author flows that handle a variety of email tasks.