TABLE OF CONTENTS

Exercise 1:
Start the Integration Server, Broker, and MWS

Overview

In this exercise, you will start the server components of the suite, and then open the Administrator console to confirm.

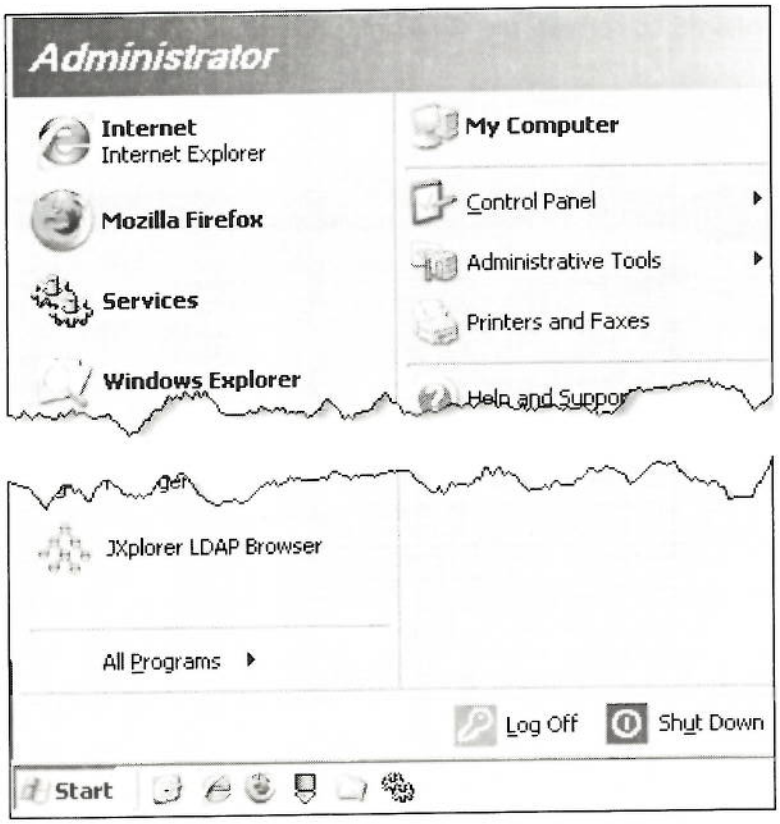
Steps

1. Before starting with the exercises, you need to set up your virtual machine appropriately. This is done by running the batch procedure **setup611.bat** from the folder **C:\TRAINING\611-41E** by using a command prompt window. Note: It is extremely important that you perform the execution in 2 steps. **First** change to the directory C:\Training\611-41E and **then** run the Batch procedure.

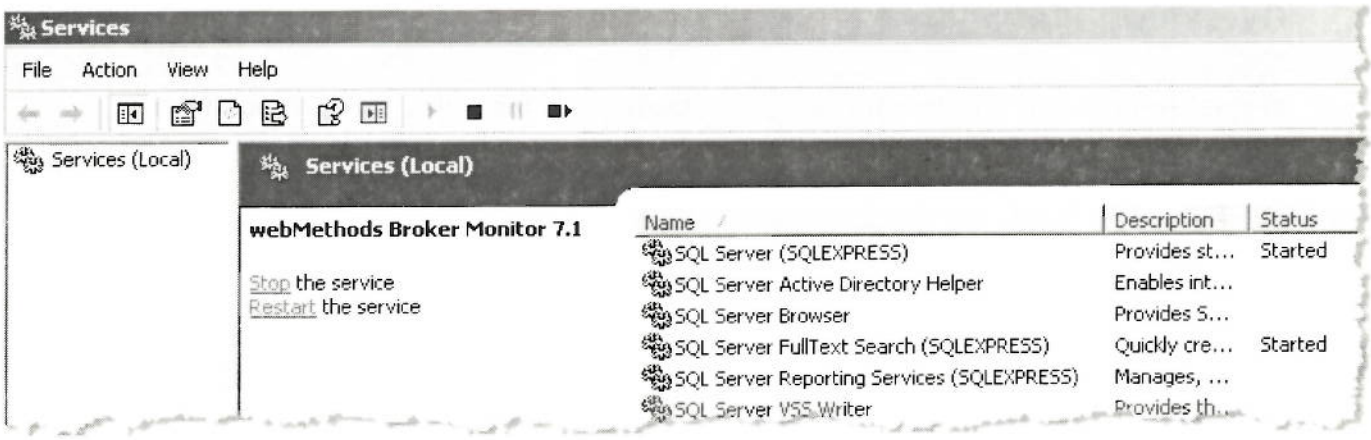
   ```
   C:\>cd /d C:\Training\611-41E
   C:\Training\611-41E>setup611.bat
   ```

   Check the output of the setup procedure if it produced any errors.

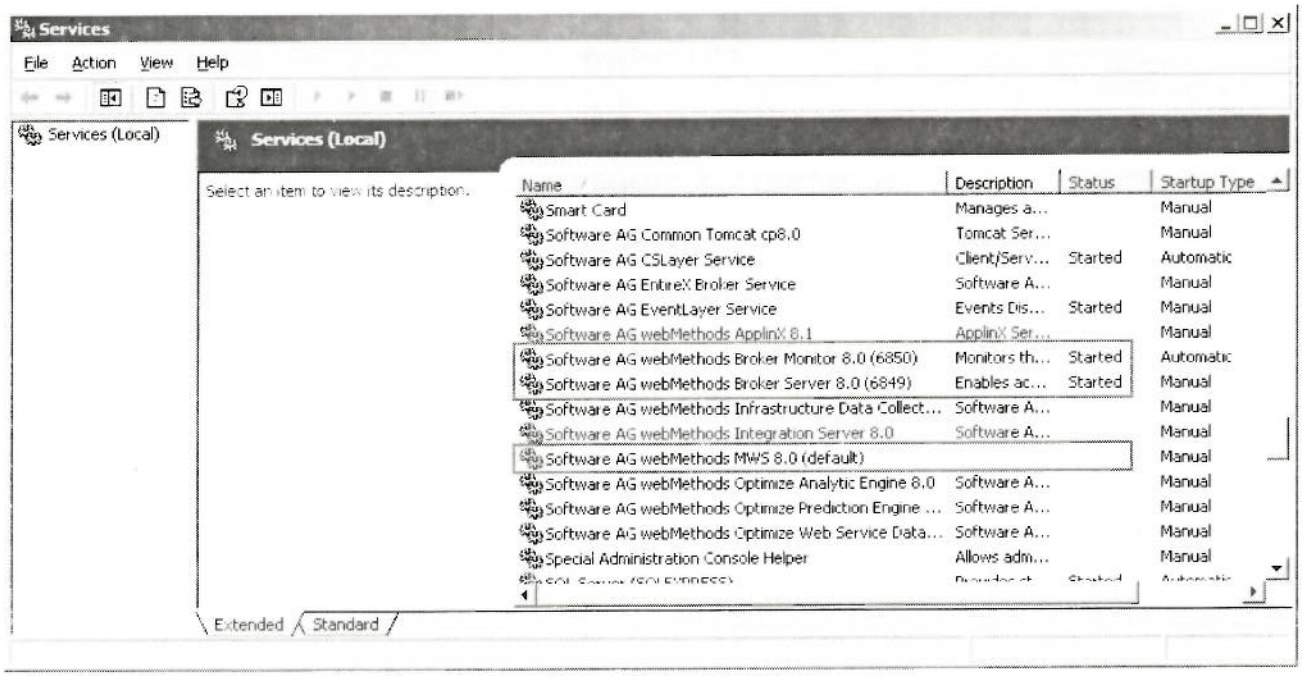
2. Start the **Services** administrative tool.



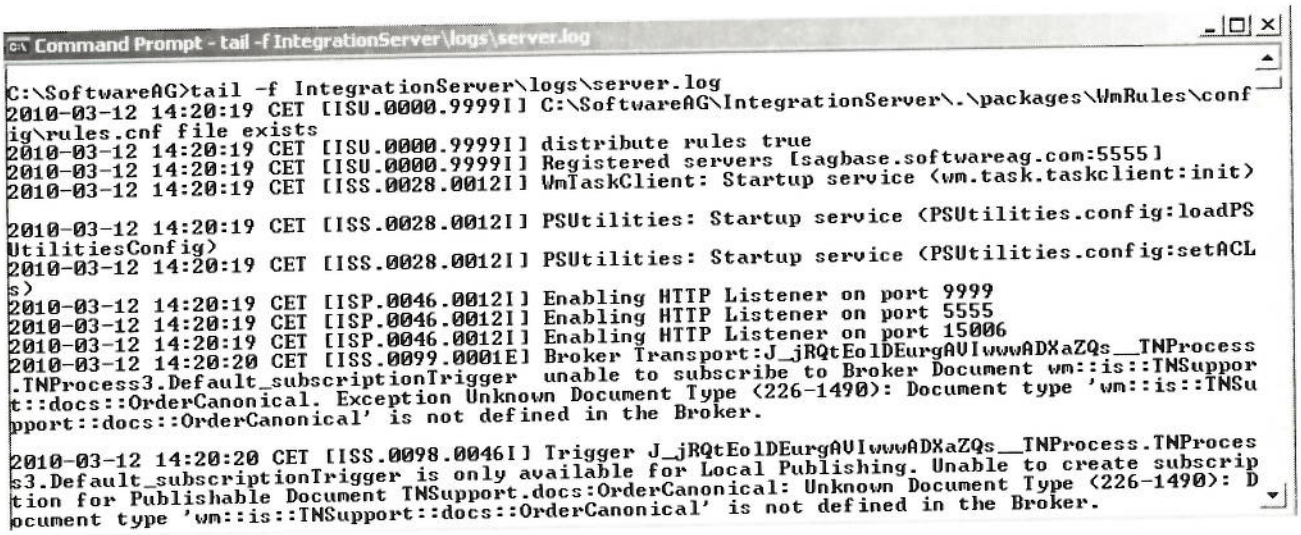
Make sure that the **SQL Server (SQLEXPRESS)** and the **SQL Server FullText Search (SQLEXPRESS)** services are started. If they are not started, then start them from the Services administrative tool.



3. Check the entry for "Software AG webMethods Broker Monitor 8.0". This service normally installs in Windows environments as an automatic service, and then is responsible for starting the "Software AG webMethods Broker Server 8.0" service. If this service is not started, **only** start the Broker Monitor service. *You should see the Broker Server service start immediately after the Broker Monitor - there is no need to start it manually.* Wait for both services to show Started. Press F5 to refresh the view.

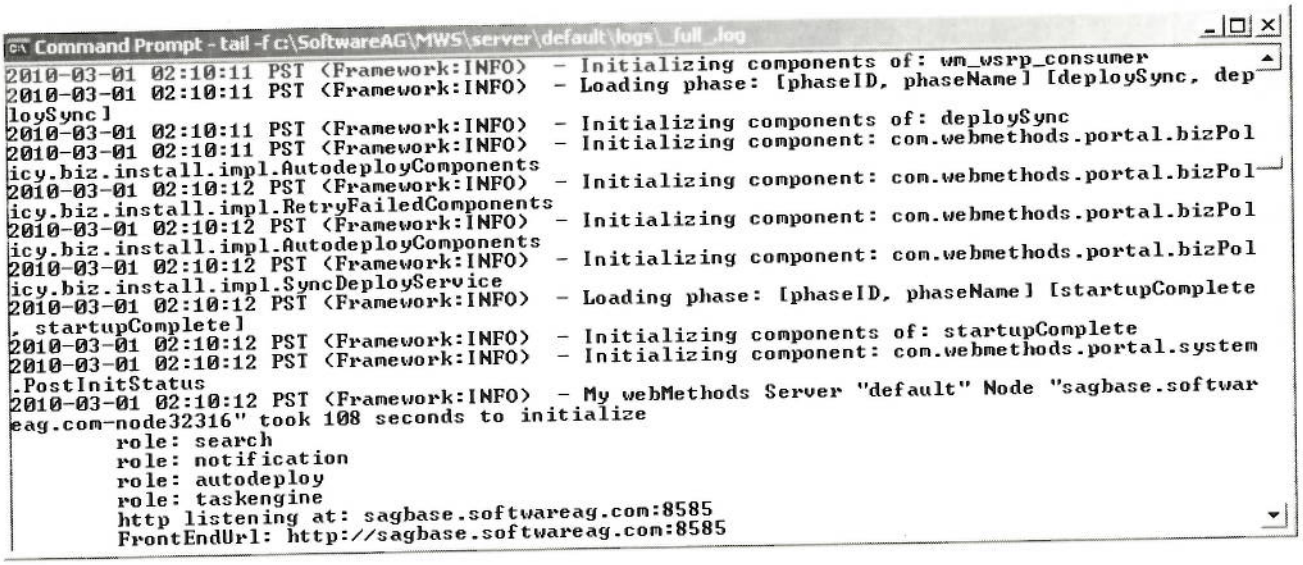


4. Check the entry for "**Software AG webMethods Integration Server 8.0**". IntegrationServer should be set to automatic start. If it is not started, **start the service**. The service will return almost immediately and show started, but behind the scenes, IntegrationServer will take a few minutes to start. Start a Command prompt window and use the 'tail -f' utility to monitor the servers logfile, which is stored at **...\IntegrationServer\logs\server.log**. Wait for the server to complete its startup, which is indicated by a line containing the message "**Enabling HTTP Listener on Port 9999**". Note that it is easy to miss these lines, as Integration Server will print some more startup messages after this point.

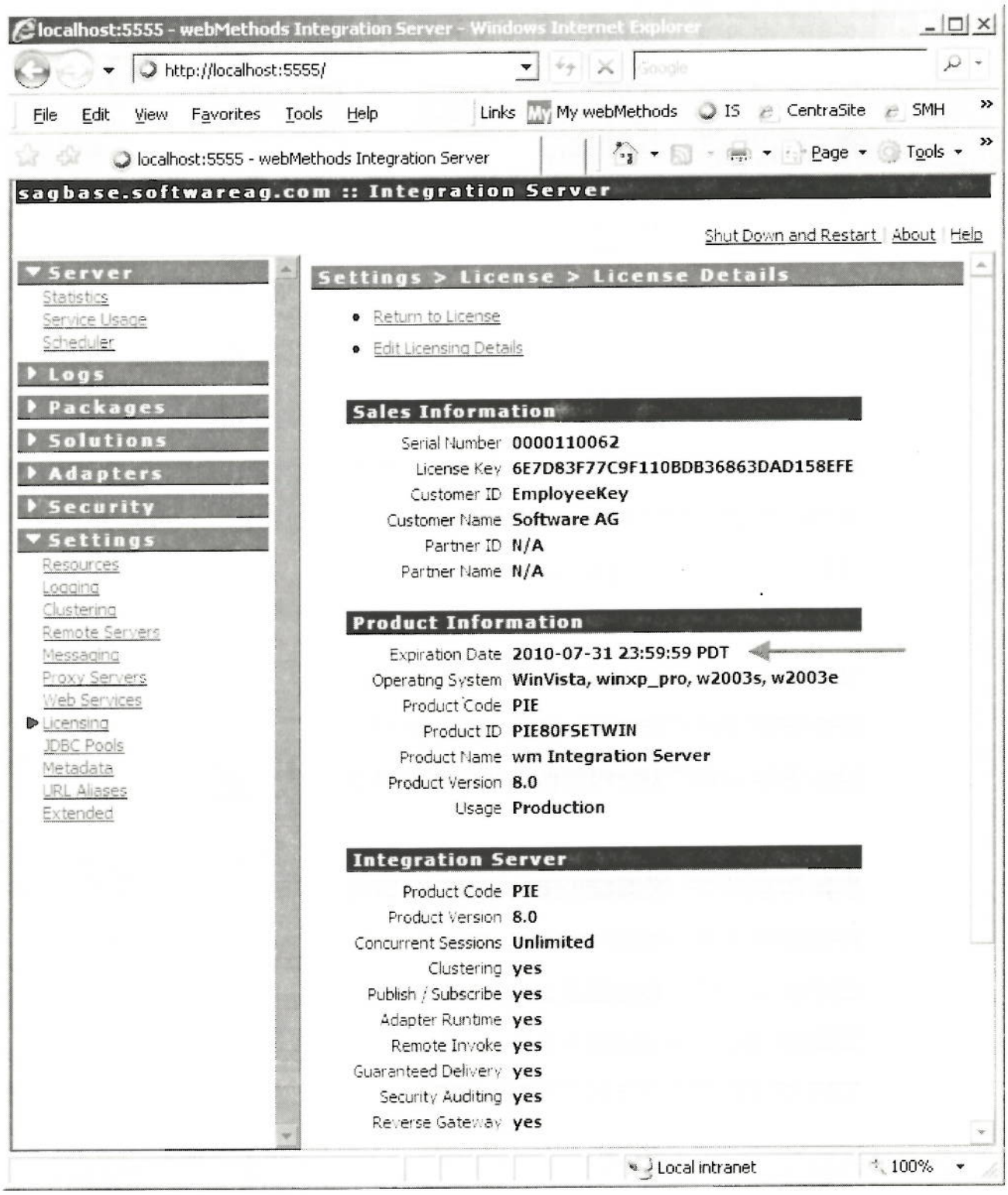


5. Verify that IS has started by using a browser to access http://localhost:5555. Login as **Administrator | manage**.

6. Check the entry for "Software AG MWS 8.0 (default)". The MWS should be set to manual start. If it is not started, **start the MWS service**. The service will return almost immediately and show started, but behind the scenes, MWS will take a few minutes to start. Start a Command prompt window and use the 'tail -f' utility to monitor the servers logfile, which is stored at **...\MWS\server\default\logs_full_.log**. Wait for the server to complete its startup, which is indicated by a line like "...Server... took 108 seconds to initialize".

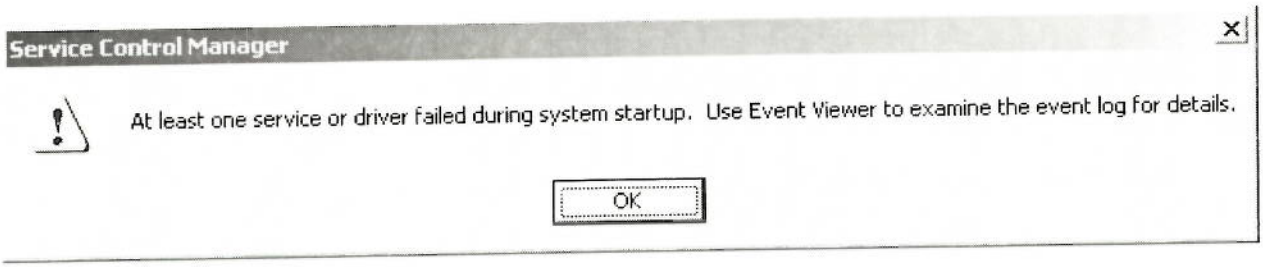


7. In the **Administrator** console's **Settings** area, check the Integration Server license key by selecting the **Licensing** link. Verify that the license key will not expire during class. *If the license key is expired, or due to expire before class is complete, ask your instructor for a new license key!*



8. Verify that the My webMethods Server has started by using a browser to access http://localhost:8585. Login as **Administrator | manage**.

Note: When you get a message box after startup telling you that "At least one service or driver failed during startup. ..." this is most of the time caused by a lock implemented as a lockfile in "**...\IntegrationServer\LOCKFILE**". This lockfile is used to prevent two instances of IntegrationServer to run from the same directory tree.



To fix this problem, check if IntegrationServer failed to start by opening the Services Administration tool

This will show a blank status for the entry "**Software AG webMethods Integration Server 8.0**". If this is the case, delete the file "**...\IntegrationServer\LOCKFILE**" and start the service again.

Check Your Understanding

1. What is the URL to access the Integration Server?
2. What is the URL for MWS?
3. Why is the Broker Monitor set to Automatic start, but not the Broker Server?

This page intentionally left blank

Exercise 2:
Packages and Folders

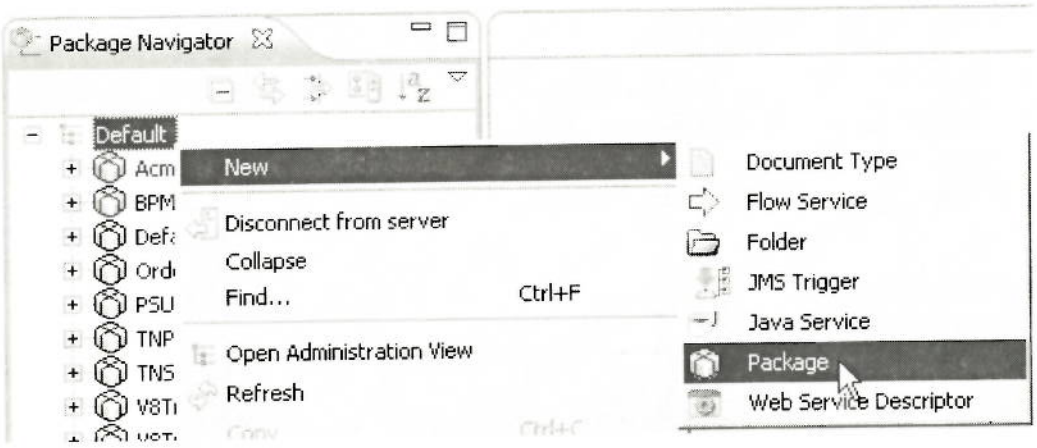
Overview

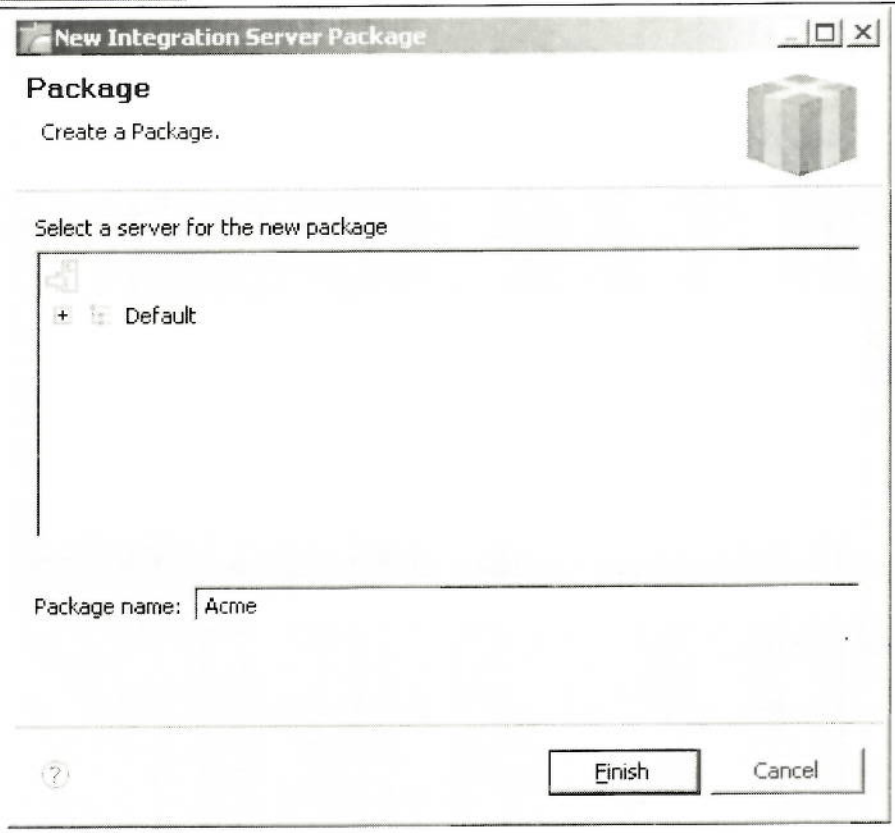
In this exercise, you will create a package and folders to store the Integration Server development work you will perform in future exercises.

Steps

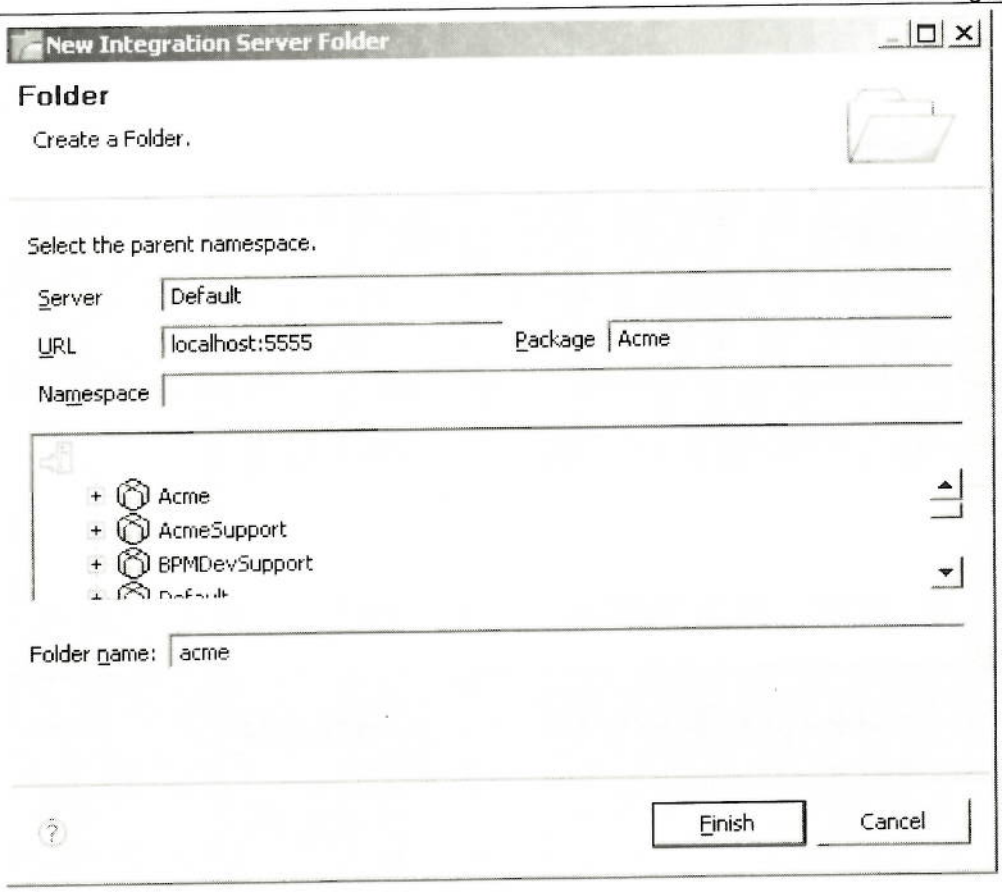
To begin development, we need a package and folder organization to store our future development work.

1. Open Software AG Designer, keep the default workspace, and then check the box beside "Use this as the default and do not ask again".
2. Create a new package called **Acme**.





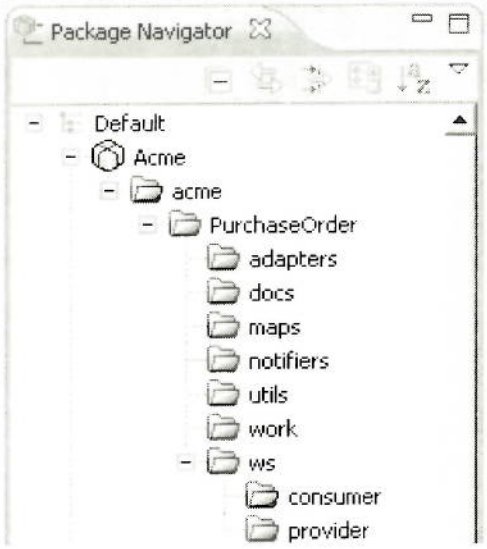
3. In the **Acme** package, create a folder called **acme**.



4. In the **acme** folder, create a folder called **PurchaseOrder**.
5. In the **acme.PurchaseOrder** folder, create 7 new folders, named as follows:
   a. adapters
   b. docs
   c. maps
   d. notifiers
   e. utils
   f. work
   g. ws

*Note: To place these folders all in the correct spot, each time right-click on the **acme.PurchaseOrder** folder and select **New -> Folder**. If you mis-type the name for a folder, right-click on the folder and select Rename.*

6. In the **acme.PurchaseOrder.ws** folder, create 2 new folders, named as follows:
   a. consumer
   b. provider



Check Your Understanding

1. If you place the folders in the wrong parent folder, how could you correct it?
2. Why is a consistent folder structure in all packages important?

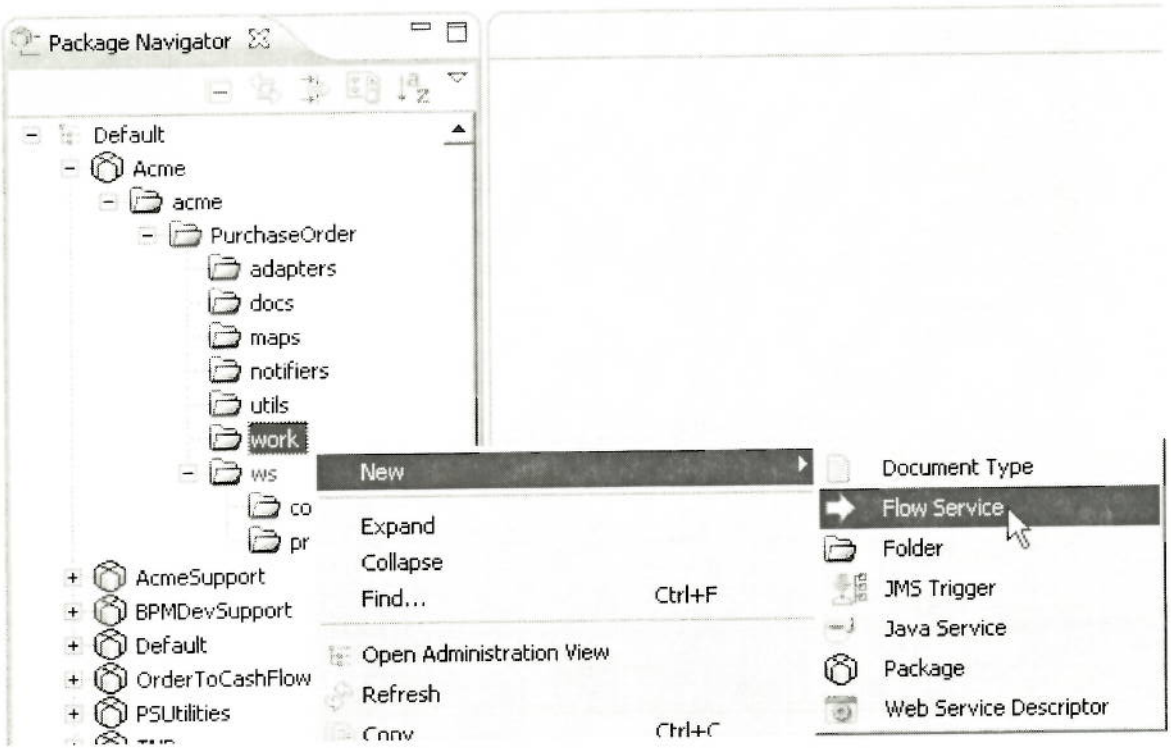
Exercise 3:
Create a Service

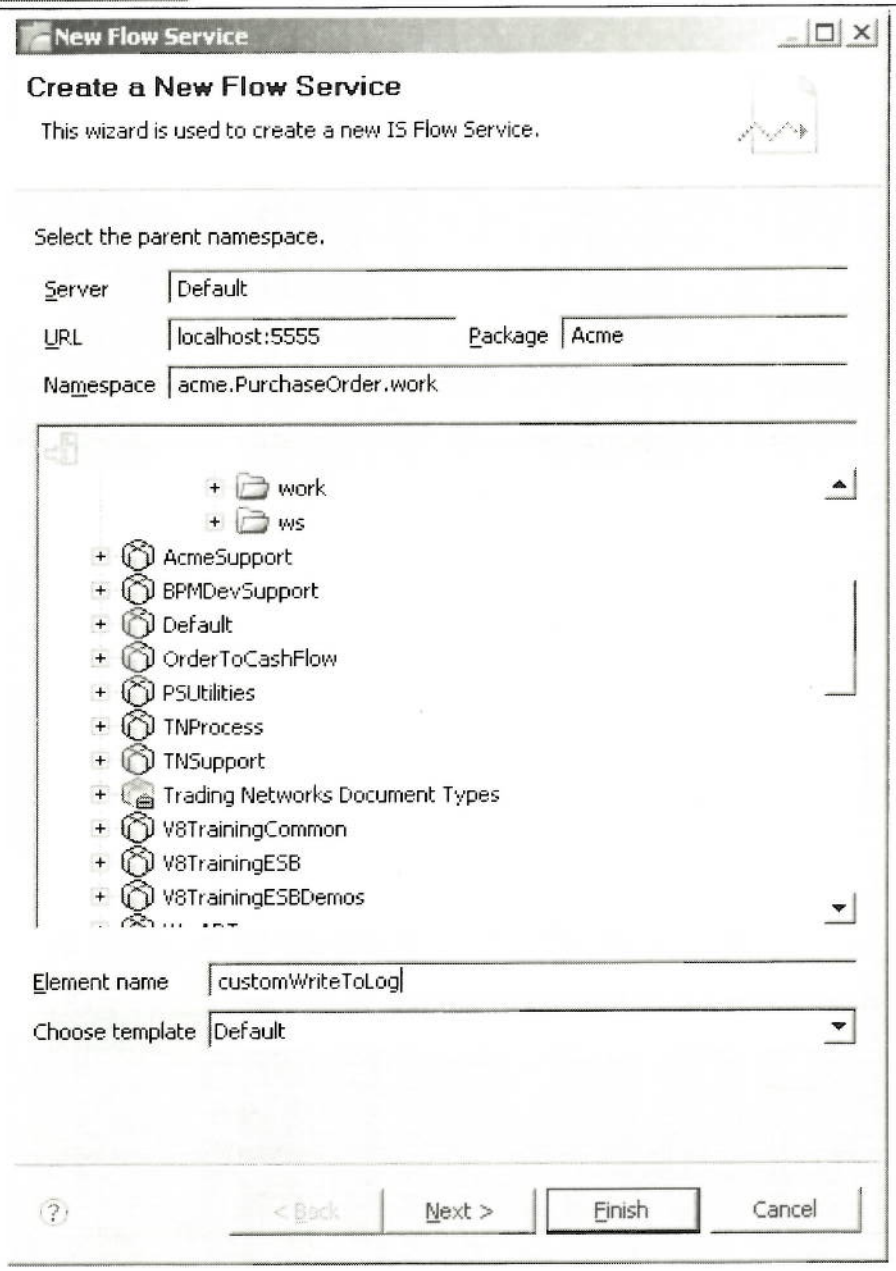
Overview

In this exercise, you will create and run a service.

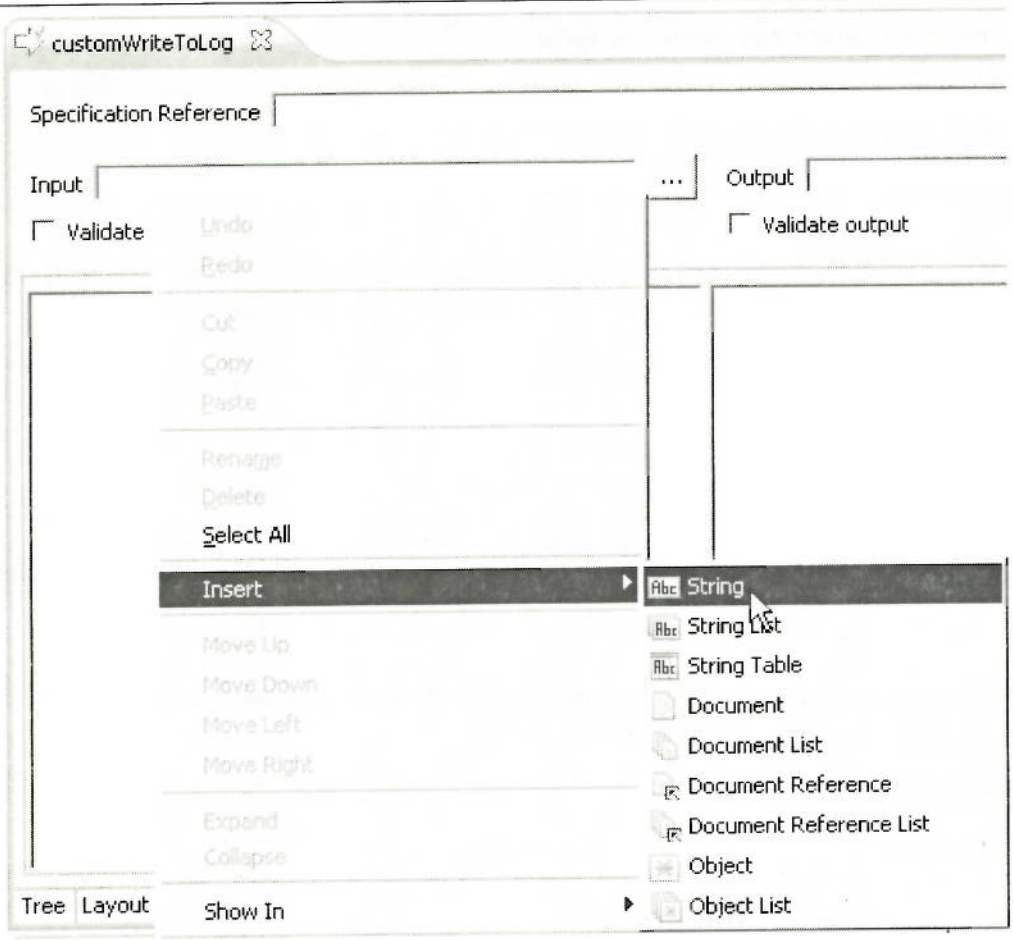
Steps

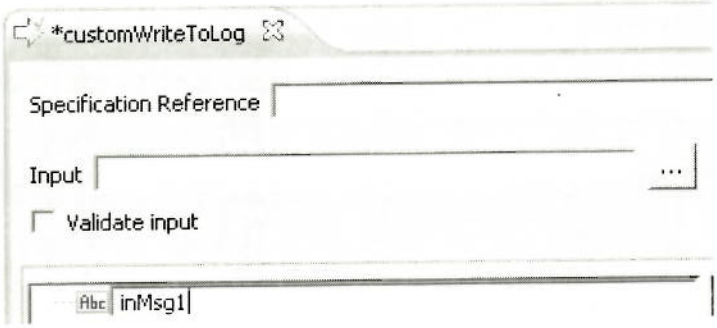
1. In the **Acme** package, in the **acme.PurchaseOrder.work** folder, create a flow service called **customWriteToLog**.

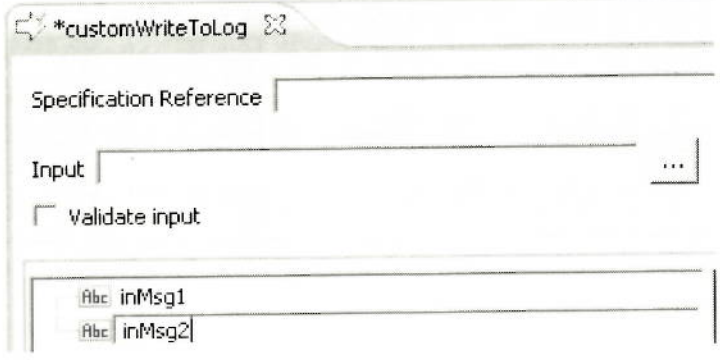




2. Add two String inputs to the new service, called **inMsg1** and **inMsg2**. To bing up the menu below, do a right click in the Input panel.







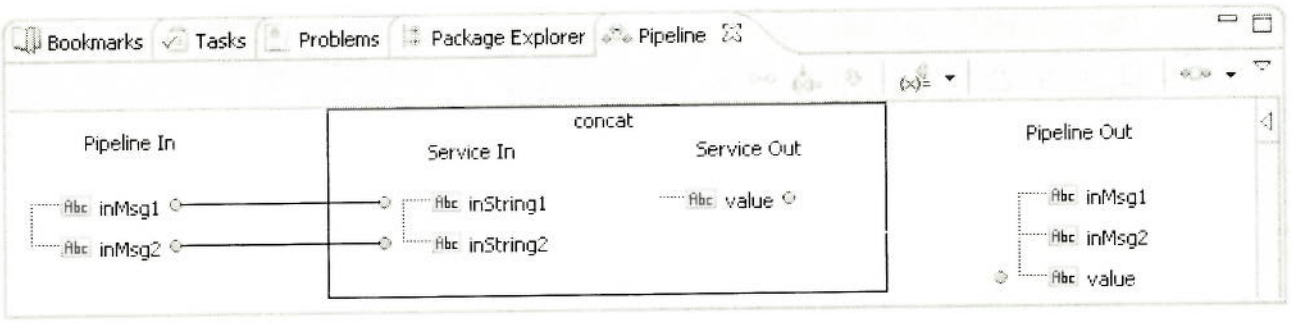
3. Add three service steps to the new service, as follows:
   a. **pub.flow:debugLog**
   b. **pub.string:toUpper**
   c. **pub.string:concat**

   The easiest way to insert these service invocations is to switch to the "Tree" tab on the bottom of the service editor and drag in the 3 services from the WmPublic package.

4. Using the Move Up and Move Down arrow icons, re-order the services to the following:
   a. **pub.string:concat**
   b. **pub.string:toUpper**
   c. **pub.flow:debugLog**

   Please note the (currently disabled) "Move Left" and "Move Right" Icons next to the "Move Up" and "Move Down" icons. You will need them in later exercises as well.

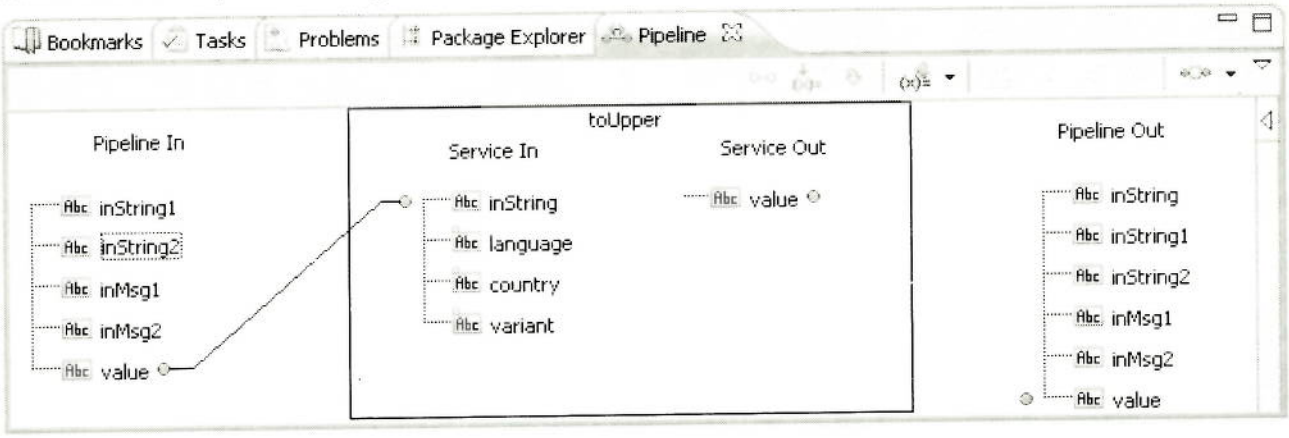
5. Map the data flow of your input through the services, as follows:
   a. pub.string:concat
      i. **inMsg1** should map to **inString1**
      ii. **inMsg2** should map to **inString2**



In order to complete this task, you have to switch to the pipeline view at the bottom of the IDE and And select the pub.string.concat service invocation on the customWriteToLog editor view. Then drag the inMsg1 argument to the inString1 parameter of the concat service.

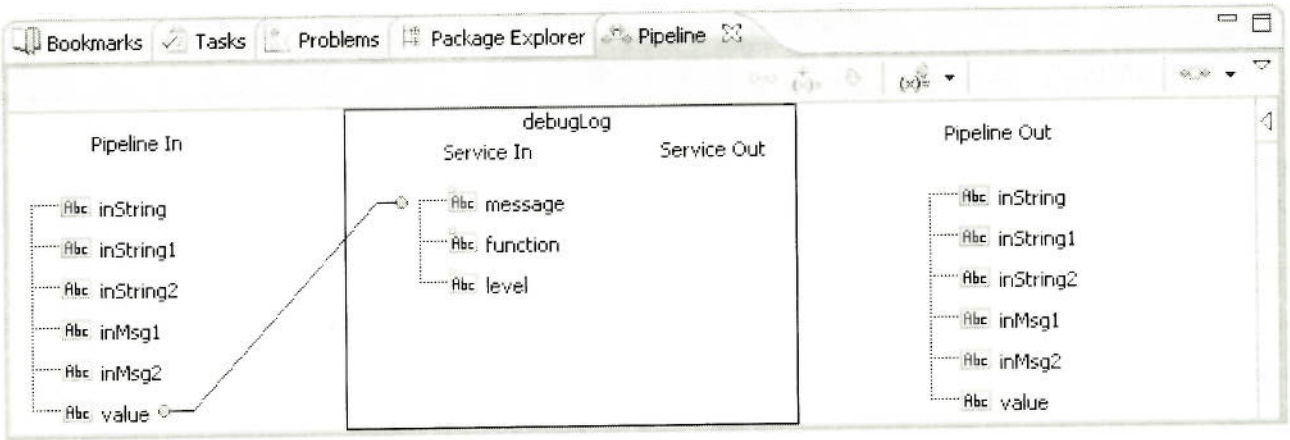
b. pub.string.toUpper

i. **value** should map to **inString**



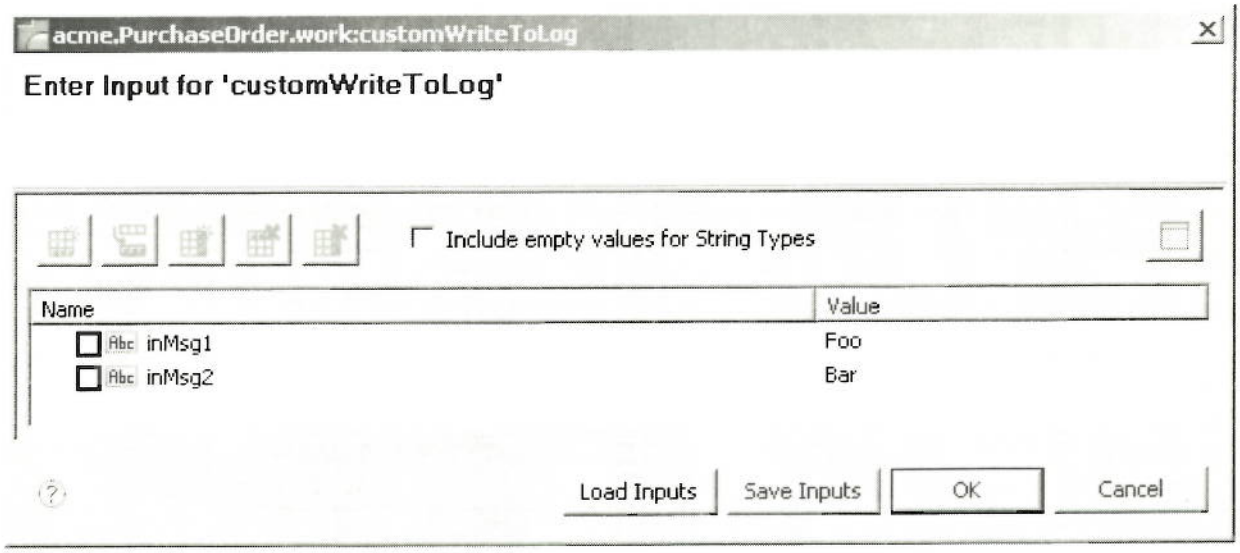
c. pub.flow:debugLog

i. **value** should map to **message**

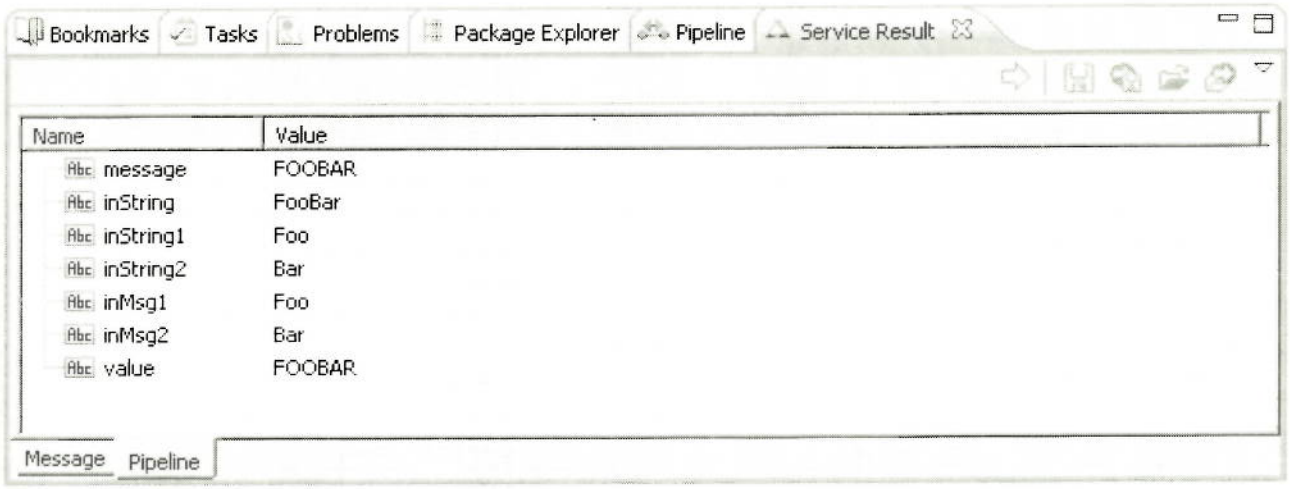


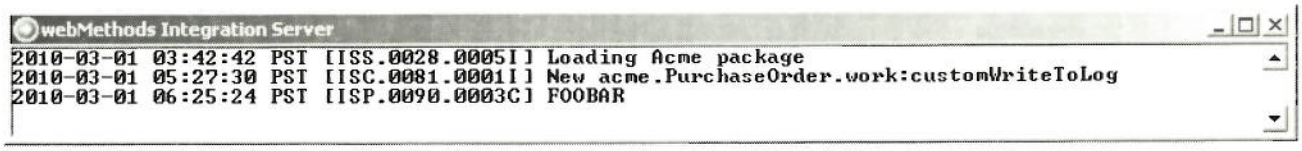
6. Save and run the service. For the first run of your service you must choose the "Run As" ➡ "Run Flow Service" from the context menu of the service. For further invokes you can simply hit the green arrow button in the upper toolbar: 

Then provide string inputs of your choice.

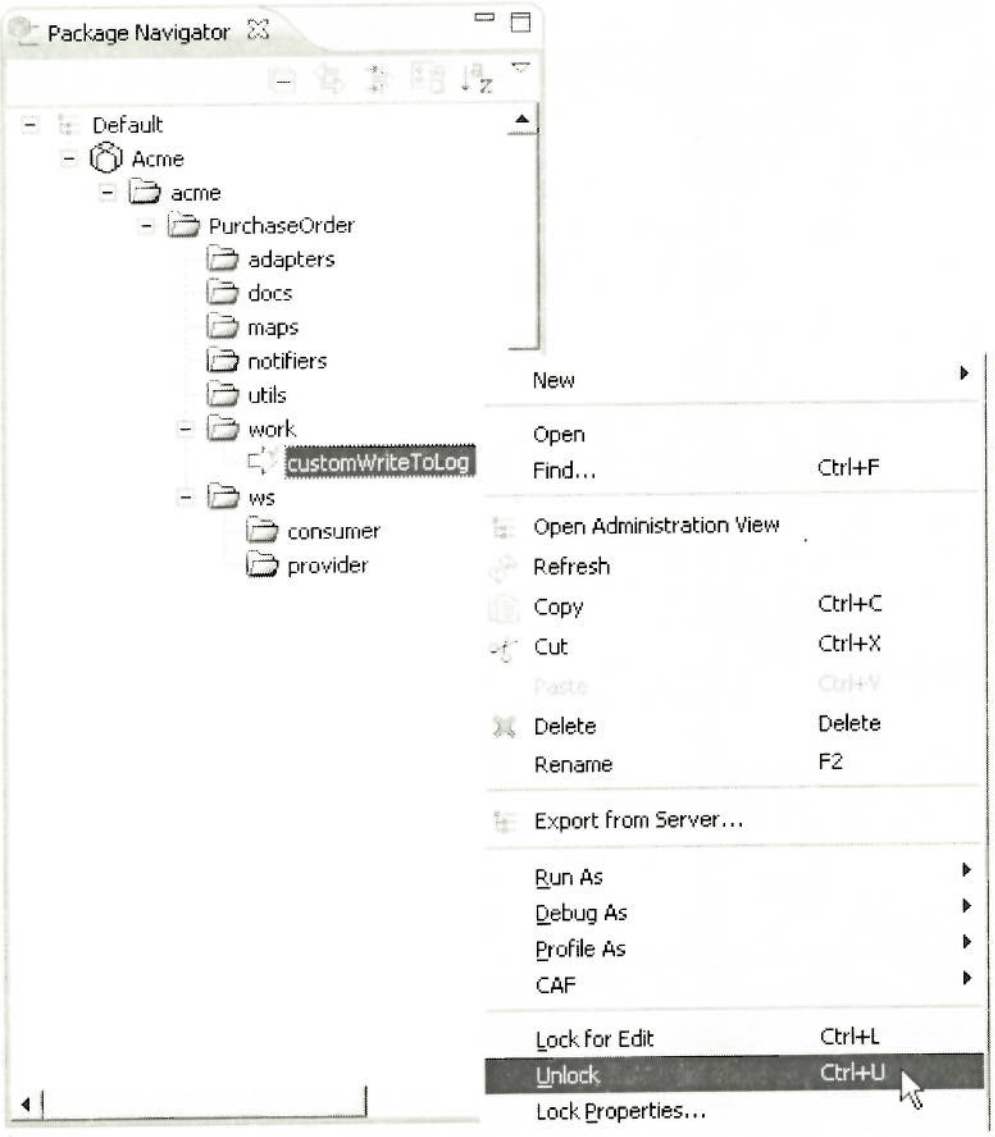


7. Check the Service Result view in Designer and then confirm that the service executed successfully by checking the Server log





8. Unlock your service, as you are now done with your development on this service.



Check Your Understanding

1. Why is the order of the services important? the ~~input~~ output of one service is to ~~output~~ input to the other service
2. How many inputs can the pub.string:toUpper service accept? 4
3. Where would the server log appear if the server is not running through the Command Prompt? within the IS log file, which is located in Integration\Server\logs\server.log

This page intentionally left blank

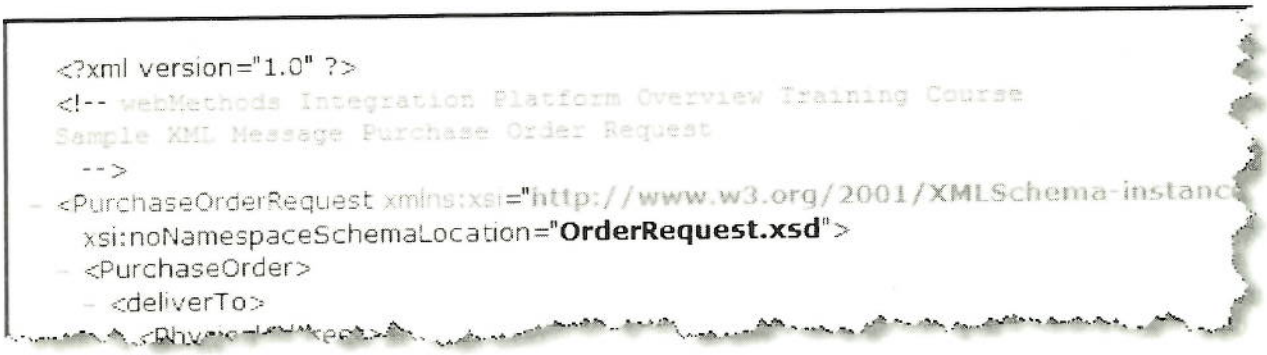
Exercise 4:
Document Types

Overview

Before we can write services to deal with complex structures, we need to create the document types that represent those structures inside the Integration Server. In this exercise, you will create and use document types in Designer. One document type will be created manually, and the other imported from an existing XML schema.

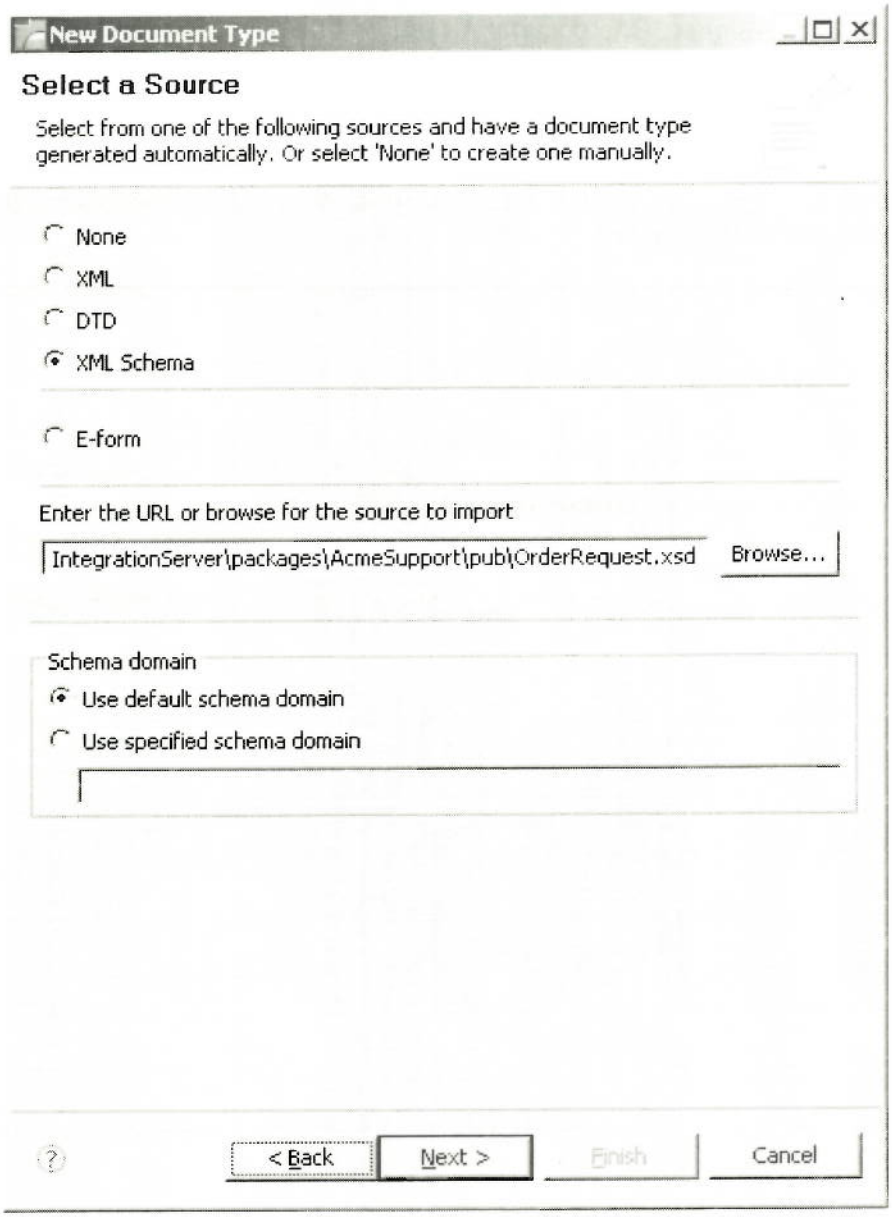
Steps

1. Use a text editor to look at the XML document **...\IntegrationServer\packages\AcmeSupport\pub\PORequest.xml**. What is the Root node?

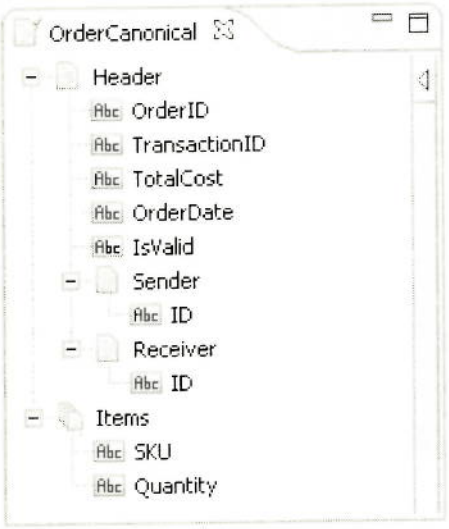


2. In the **acme.PurchaseOrder.docs** folder, create a **request** folder. In the **request** folder, create a new Document Type called **OrderRequest.** To create the new Document Type you can right click on the folder and select new then Document Type OR select File ➡ New ➡ Document Type from the File menu. Choose by importing from an existing XML schema when asked for a document source. When prompted to select a root node, choose the root node you found in Task 1 above. The schema can be imported from the following location:

   ...\IntegrationServer\packages\AcmeSupport\pub\OrderRequest.xsd



3. In the **acme.PurchaseOrder.docs** folder, create a new Document Type called **OrderCanonical**. You will create this document type manually (None) with the following structure:
   a. Header (Document)
      i. OrderID (String - be sure to indent under Header)
      ii. TransactionID (String - make sure it is indented under Header)
      iii. OrderDate (String - make sure it is indented under Header)
      iv. TotalCost (String - make sure it is indented under Header)
      v. IsValid (String - make sure it is indented under Header)
      vi. Sender (Document - make sure it is indented under Header)
         1. ID (String - be sure to indent under Sender)
      vii. Receiver (Document - make sure it is indented under Header)
         1. ID (String - be sure to indent under Receiver)
   b. Items (Document List - should NOT be indented under anything)
      i. SKU (String - be sure to indent under Items)
      ii. Quantity (String - be sure to indent under Items)



4. Save your document types.

Check Your Understanding

1. Why are additional documents created in the acme.PurchaseOrder.docs.request folder when the OrderRequest schema is imported? There are complex type include
2. What is the benefit of using a schema for import over using a DTD? more detail definition are available
3. What are the two ways to indent a document type element under another? right clicking on the parent and inserting field
   use the controls on the tool bar

With the document types we just created will assist a future exercise that will push a purchase Order Request to a Order Canonical

Purchase Order Request → ACME Order Canonical → 

This page intentionally left blank

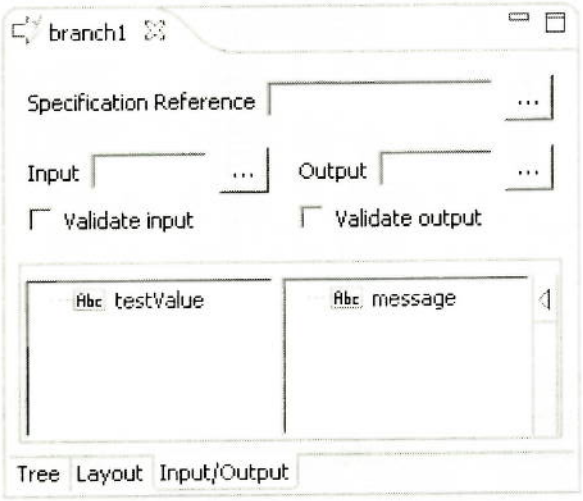
Exercise 5:
Flow Services - BRANCH

Overview

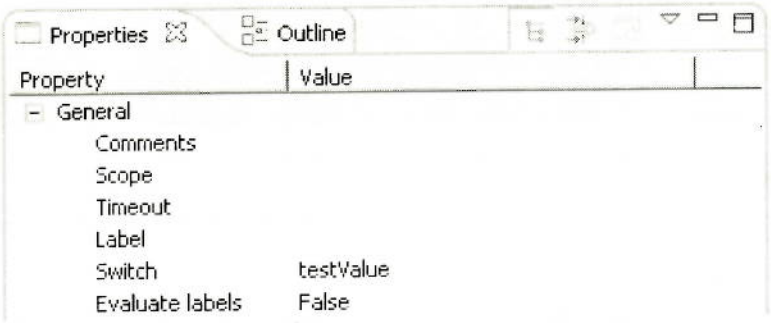
In this exercise, you will create business logic using two different Branch steps: one to test for contents of a variable, and one to evaluate labels associated with logic.

Steps

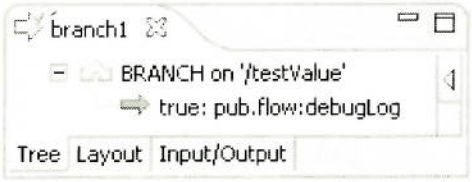
1. In the **acme.PurchaseOrder.work** subfolder, create a new Flow service called **branch1.**
2. Define the inputs and outputs of branch1 as input String called **testValue** and output String called **message.**



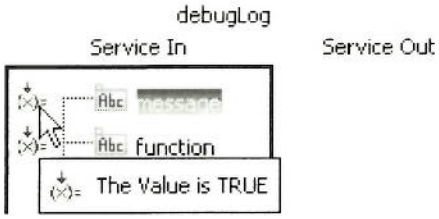
3. In the next steps you will add a **Branch** statement to the flow service that conditionally writes a message to the Server Log, based on the contents of the testValue variable. For example, if testValue = true, write a message saying "The value is TRUE" to the server log.
   a. Add a **Branch** statement to your service. Specify **testValue** as the switch property.

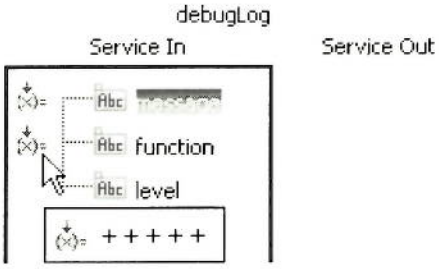


   b. Add a **pub.flow:debugLog** statement below the Branch (be sure to indent it under the Branch so that it becomes part of the Branch logic). In the debugLog properties, set the Label for the service to be **true** (all lower-case).

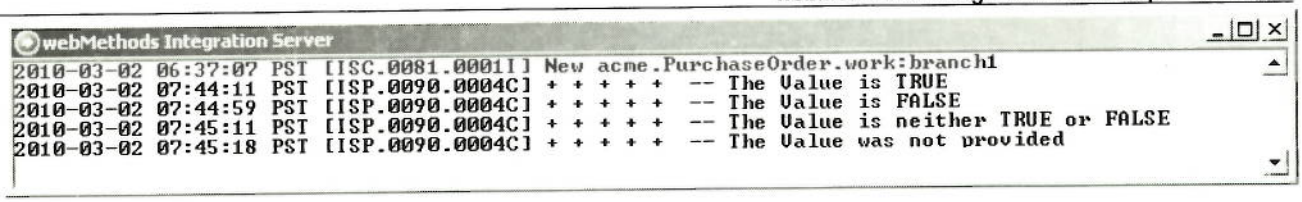


c. On the Pipeline tab for the debugLog statement, set the value of the variable **message** to be "The value is TRUE". As an eyecatcher set the value of **function** to "+ + + + + ". Of course, do not enter the Quotes.





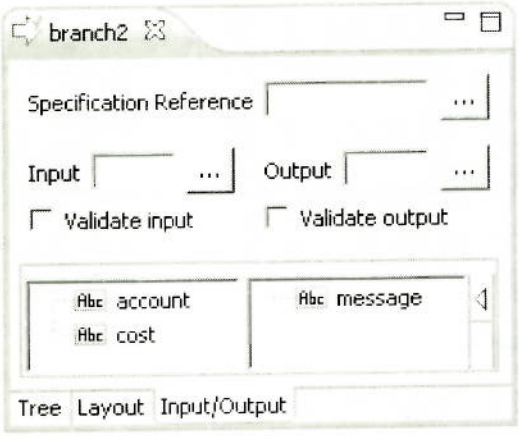
4. In the next section you add three more pub.flow:debugLog statements to your Branch (be sure to indent all of them under the Branch so that they become part of the Branch logic). Set the Label property and the variable message for each of them as follows:
   a. pub.flow:debugLog (already completed in Task 3, listed here as an example)
      i. Label = true
      ii. Message = The value is TRUE
   b. pub.flow:debugLog
      i. Label = false
      ii. Message = The value is FALSE
   c. pub.flow:debugLog
      i. Label = $default
      ii. Message = The value is neither TRUE or FALSE
   d. pub.flow:debugLog
      i. Label = $null
      ii. Message = The value was not provided
5. Test your service by running it several times and providing different values each time for **testValue**. The output should appear in the **Service Result view**, and you can look in the Server Log.



The screenshot above was produced by running the service with the input values "true", "false", "maybe" and by not filling out the message input value.

Note: If your service does not work, or does not output the correct message, run your service using the debugger so that you can step through the code.

6. In the same acme.PurchaseOrder.work folder, create another Flow service called **branch2** with two input Strings, **account** & **cost**, and an output String **message**.

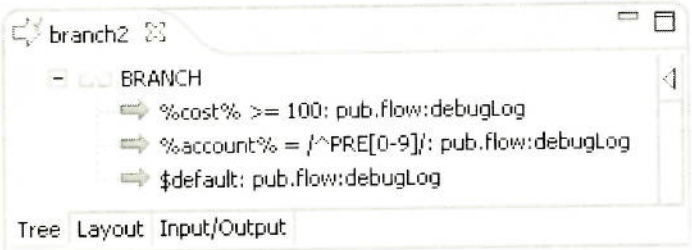


7. In the next section you write **Branch** and **pub.flow:debugLog** code to write a message (based on the value of the input fields) to the Server Log. In this service, we want to evaluate labels, so be sure to leave the Branch **switch** parameter empty and set **Evaluate Labels** to True. The structure for this service should be as follows:

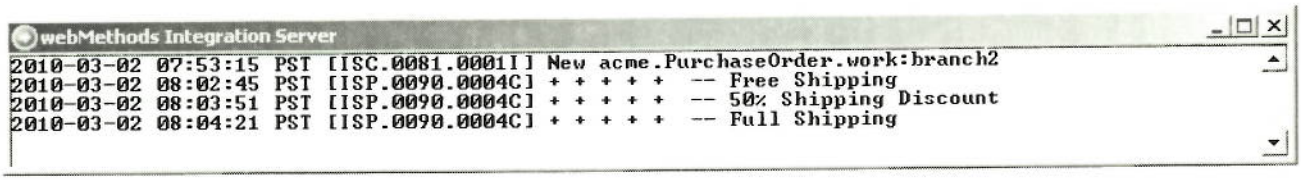
   a. If the contents of variable **cost** are >= **100** then write **Free Shipping** to the server log.
   Note: %cost% >= 100 evaluates the contents of cost at run-time.

   b. If **account** starts with **PRE0** thru **PRE9** then write, **50% Shipping Discount** to the server log.
   Note: you can test this in one step with a regular expression such as %account% = /^PRE[0-9]/

   c. Otherwise, write **Full Shipping** to the server log.



8. Save and test the service. Check your results in the server log.



The screenshot above was produced by running the service using the following inputs:

run	cost	Account
1	100	Foo Inc.
2	42	PRE42 Inc.
3	42	Bar Inc.

Check Your Understanding

1. When are regular expressions useful in branch?
2. Can you combine a switch variable with Evaluate labels=True?
3. What are the special test values that can be used as labels in a branch statement?

Exercise 6:
Building Flow Services - LOOP

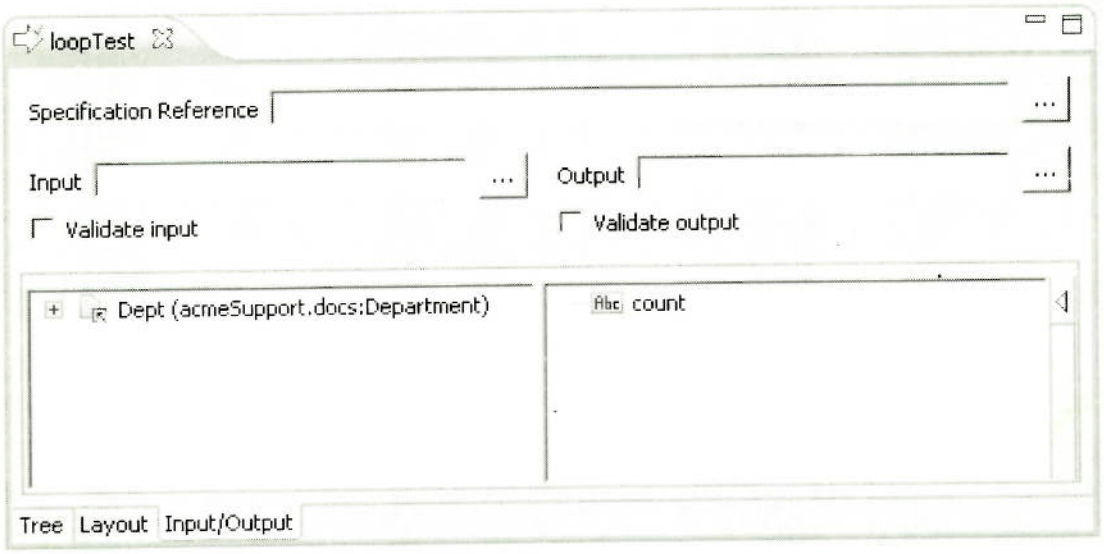
Overview

In this exercise, you will create business logic to process a list of employees using a Loop step in a Flow service.

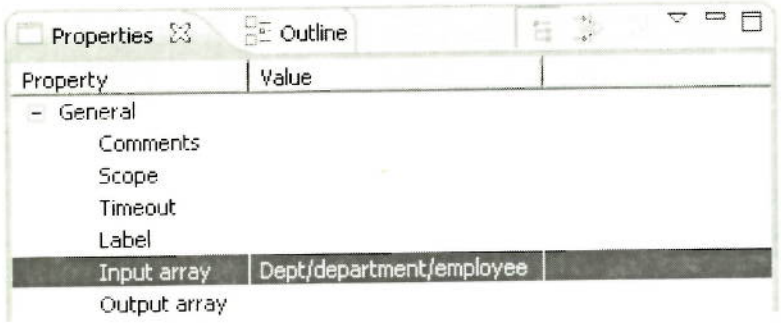
Steps

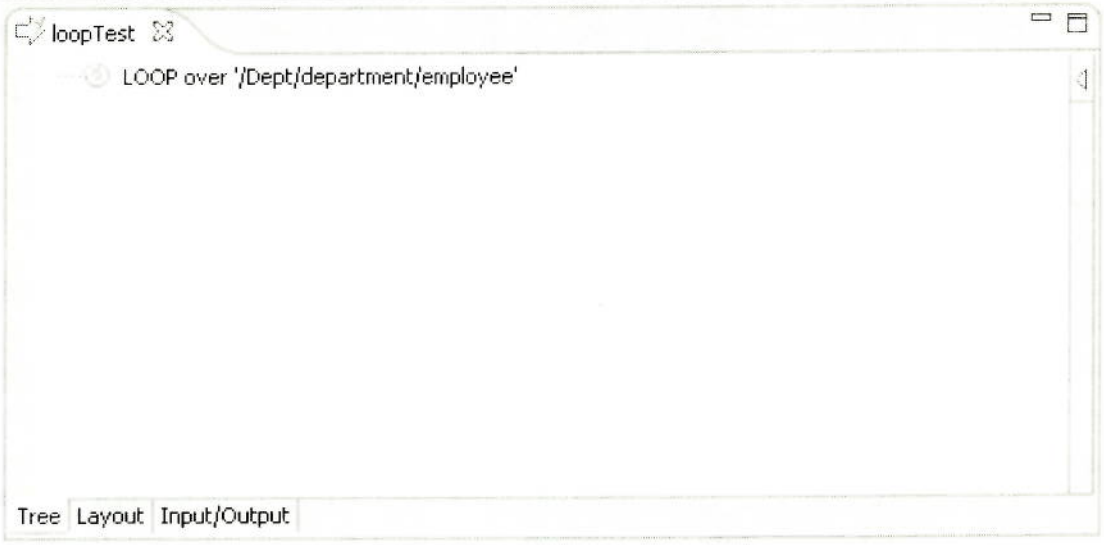
1. In the **acme.PurchaseOrder.work** folder, create a new flow service called **loopTest**. Set the input to be a Document Reference to **acmeSupport.docs:Department** called **Dept**, and set the output to be a single **String** field called **count**.

   Note: Do not use the Input or Output entry fields. Their purpose will be discussed later.

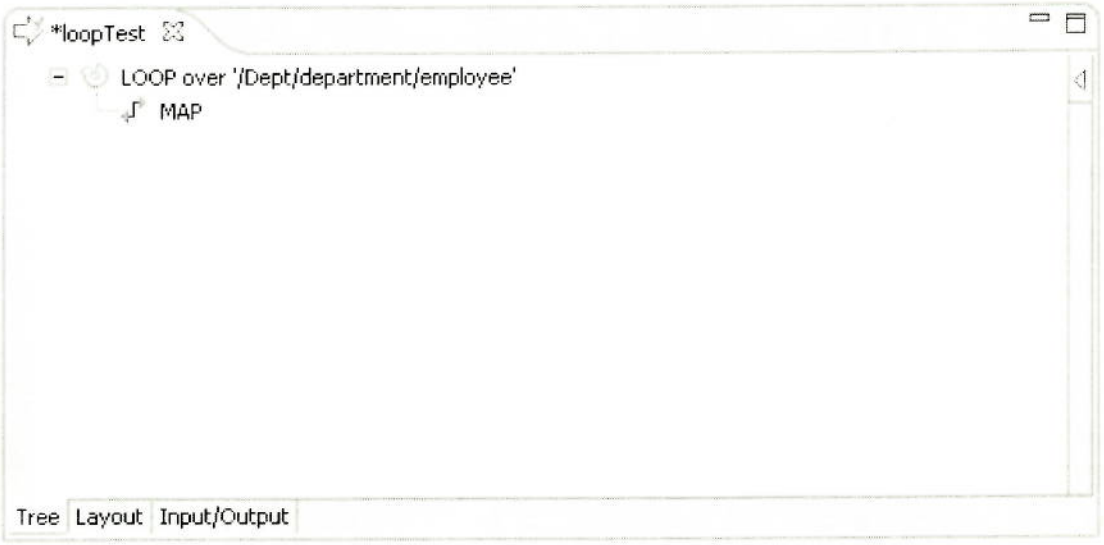


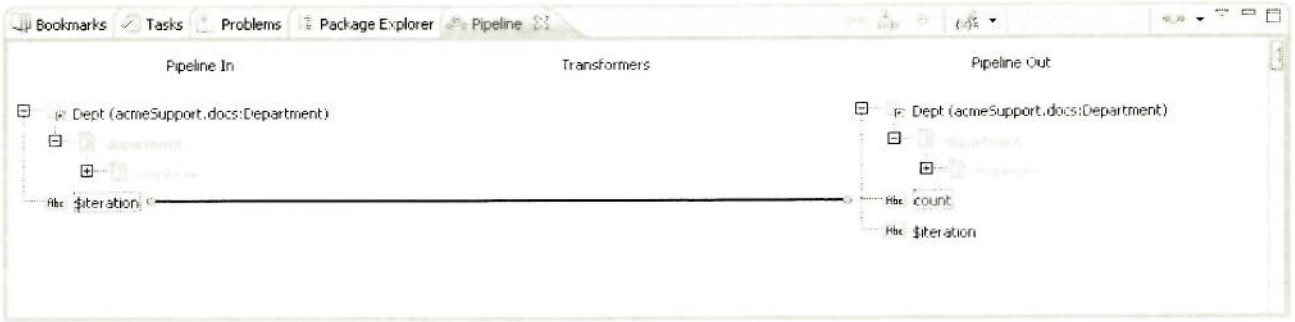
2. On the **Input/Output** tab of your service, expand the **Dept** variable. Find Dept/department/employee and right-click to **Copy**.
3. In your service, add a **LOOP** statement and in the **Input Array** property, paste in Dept/department/employee. Press enter to see the Input array displayed in the Loop.



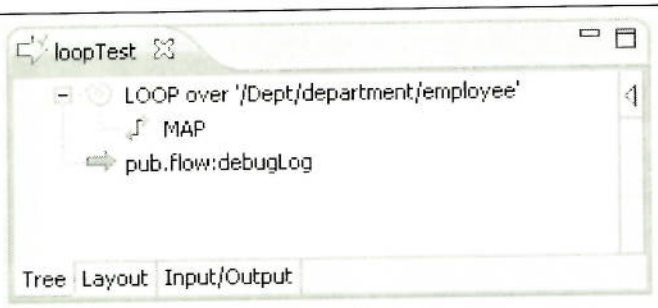


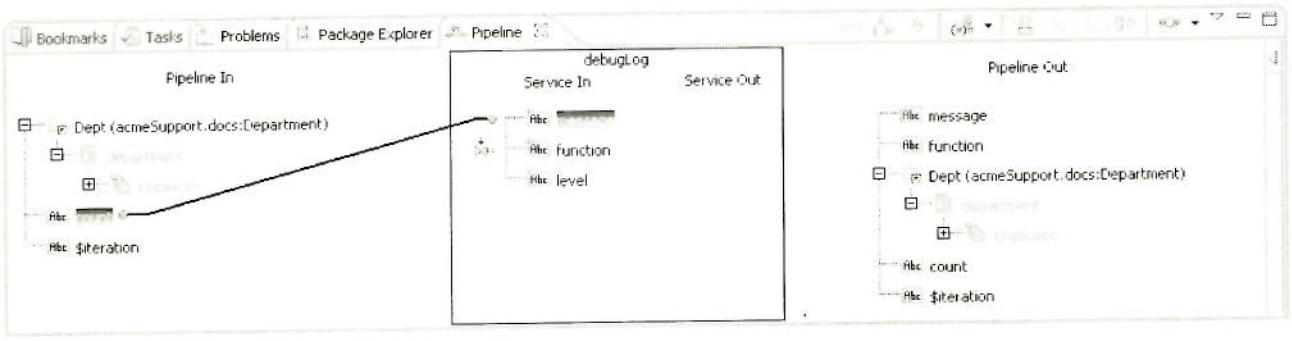
4. Add a **MAP** statement under the Loop step (be sure it is indented under the Loop). Map **$iteration** to **count**.





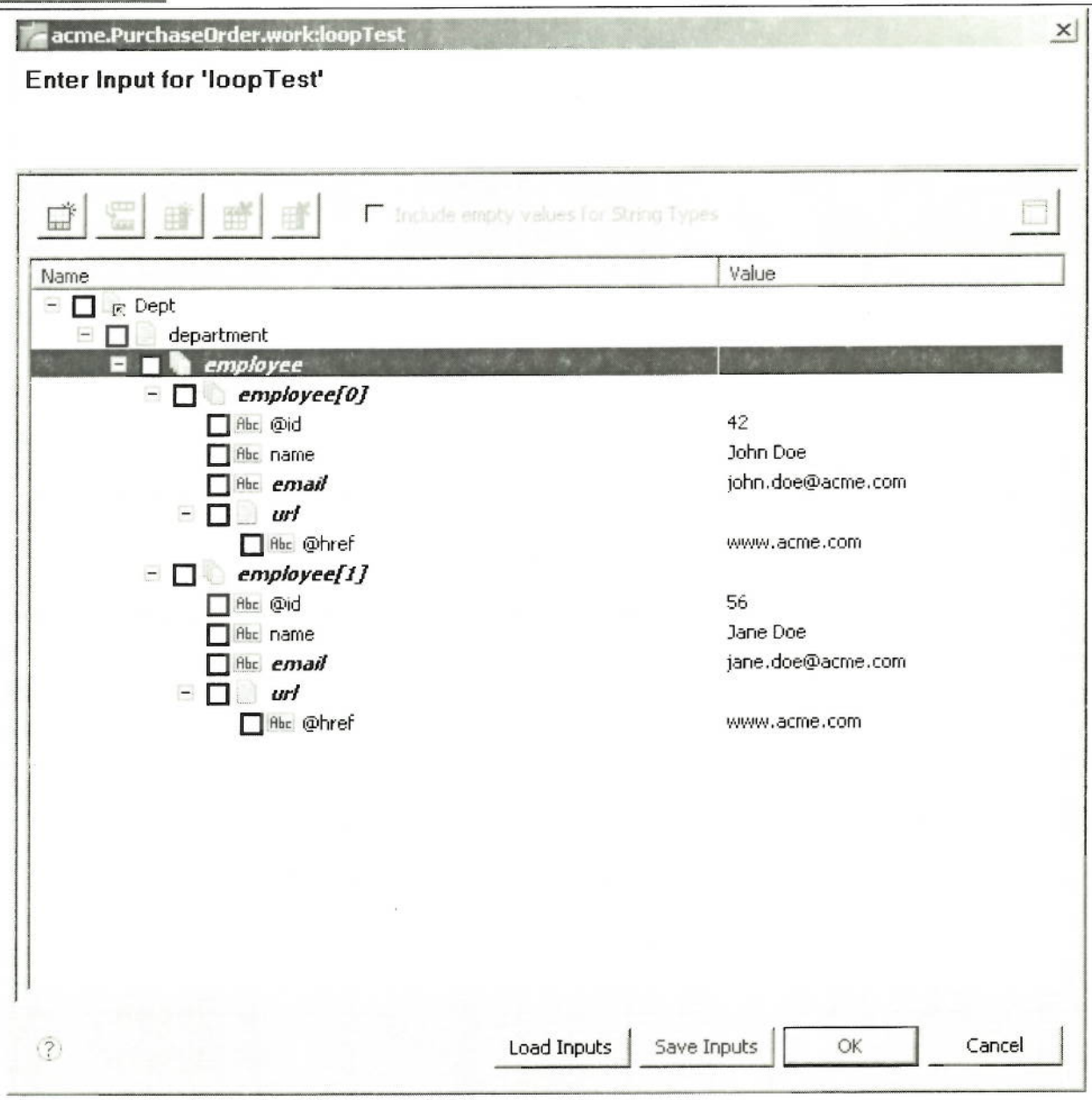
5. Add a **pub.flow:debugLog** below the MAP step (make sure it is NOT indented under the Loop). Map **count** to **message** and set **function** to an eyecatcher like **"+ + + + + "**.

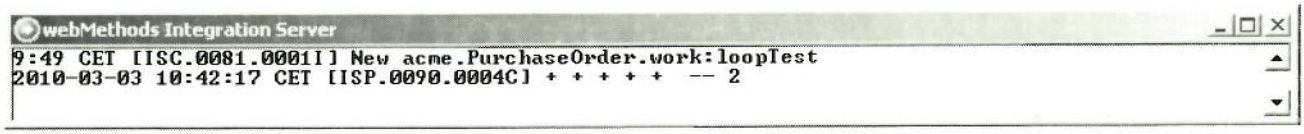




Note: When a Loop statement is added, a new variable called $iteration appears in the Pipeline. This variable keeps track of the number of iterations of the loop.

6. Save and run the service. When the service requests it's input, expand the input box, and use the **Add Row** button to add **two** employees. Type some data for each employee. Click the **OK** button. You should see a count of **2** in the Server log.





Check Your Understanding

1. What would happen if the MAP and debugLog steps were not indented under the Loop?
2. How many employees could you have added? Does Loop have a limit?
3. Why do you want to use document references rather than creating the document in the service input?

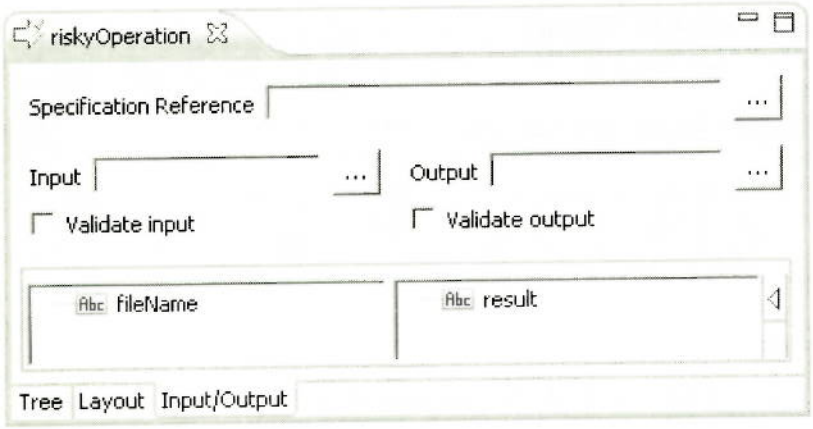
Exercise 7:
Building Flow Services – SEQUENCE

Overview

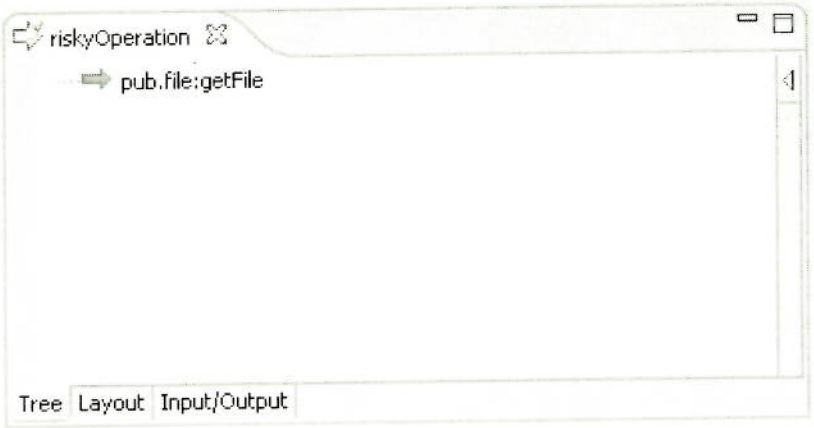
In this exercise, you will create business logic for a Try/Catch using sequence in a Flow service. We will create a service that will return an exception, and then embed that service in a Try/Catch sequence.

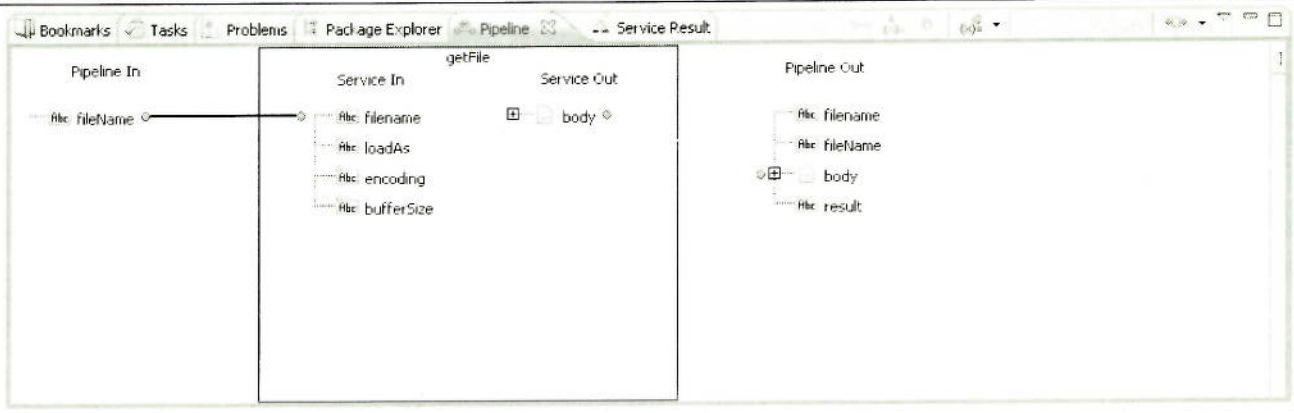
Steps

1. In the **acme.PurchaseOrder.work** folder, create a new Flow service called **riskyOperation**. Set the input to be a String called **fileName** and the output to be a String called **result**.

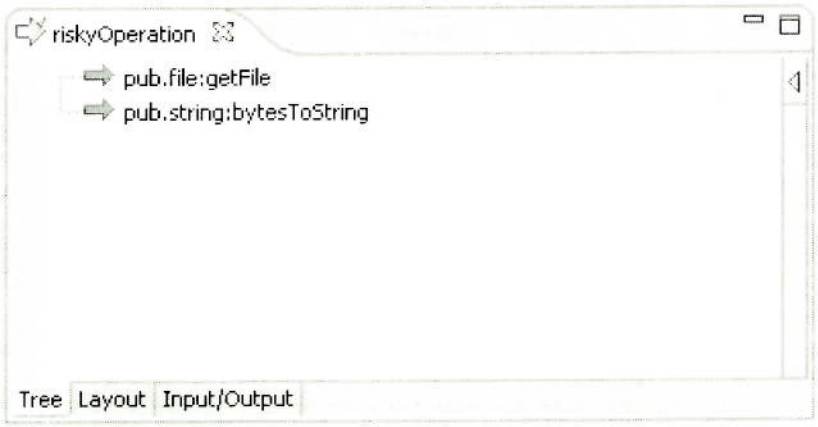


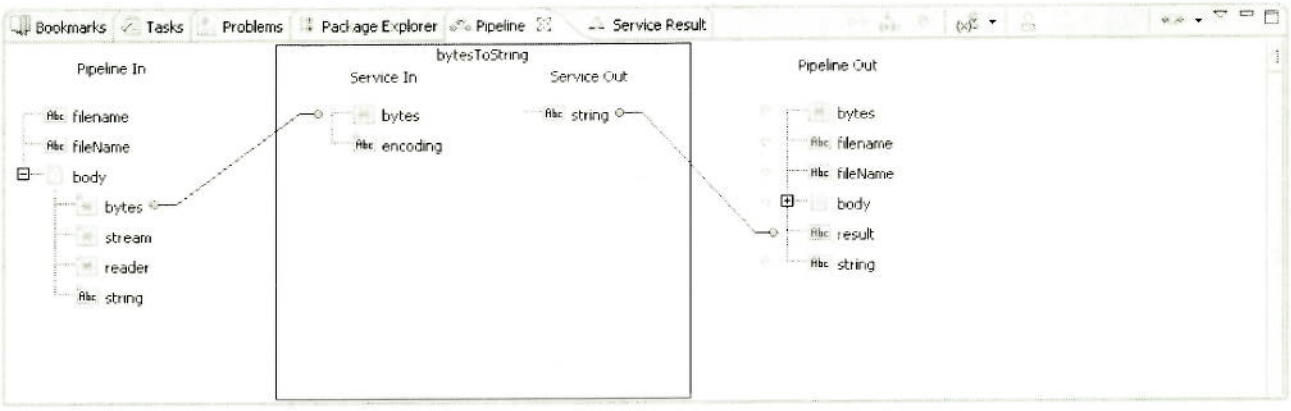
2. Add an invocation of the service **pub.file:getFile** to your service. Map **fileName** to **filename**.



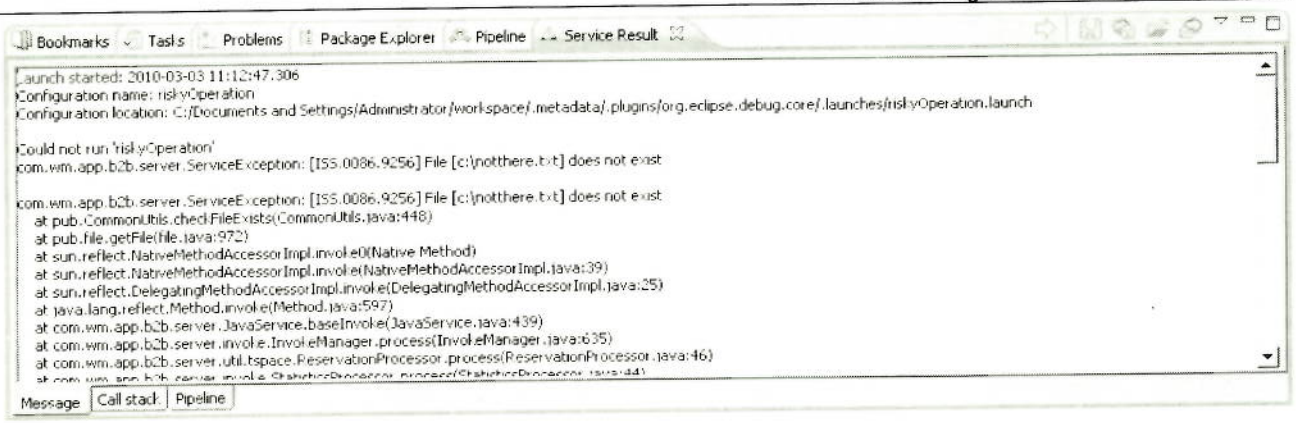


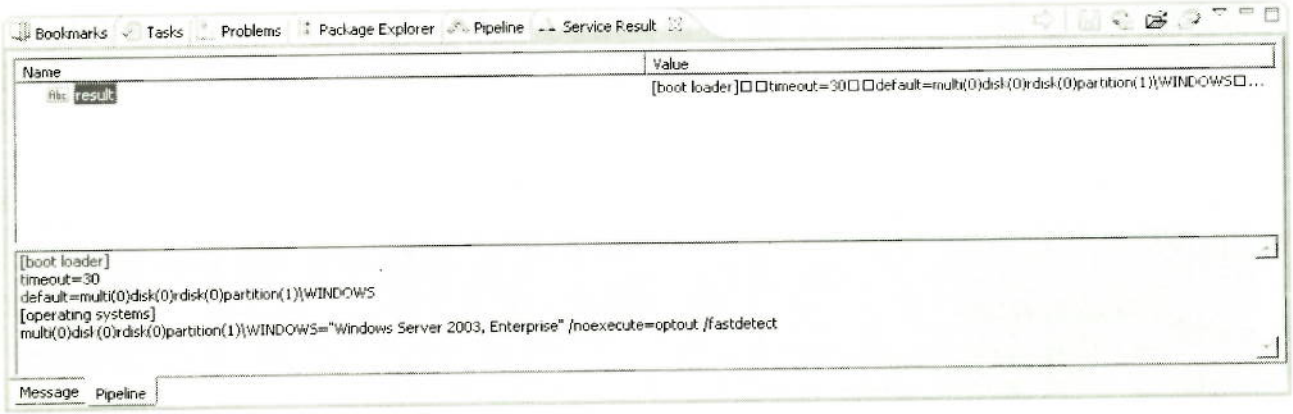
3. Add an invocation of the service **pub.string:bytesToString** to your service.
   a. Map **body/bytes** to **bytes** and **string** to **result**.
   b. Drop all other variables from the Pipeline Out.



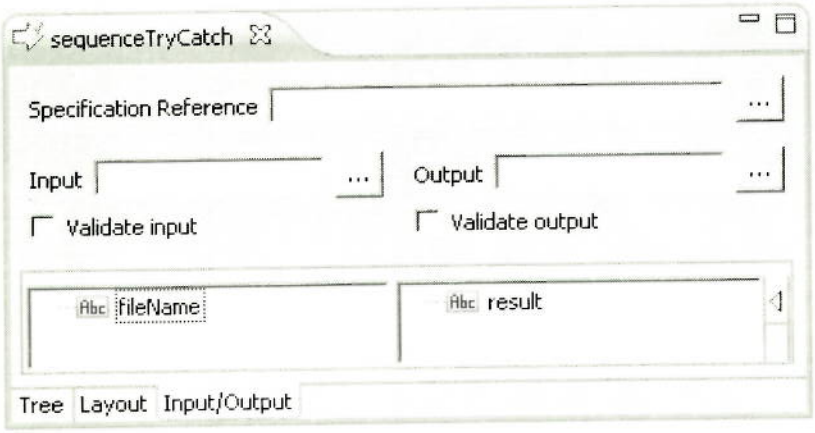


4. Save and run the service. Provide it with a **fileName** of **c:\notthere.txt** (or any other file that does not exist!) What result do you receive? Also try the service with an existing file like **c:\boot.ini**. What result do you receive then?



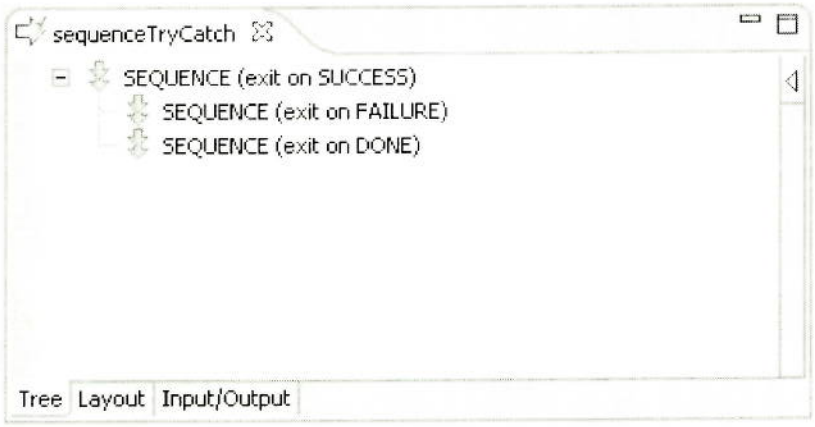


5. Now create another service in the **acme.PurchaseOrder.work** folder called **sequenceTryCatch.** We will use this service to allow us to catch the exception that **riskyOperation** raises if the file name is not correct. Define a single input String for the service called **fileName** and a single output String called called **result.**

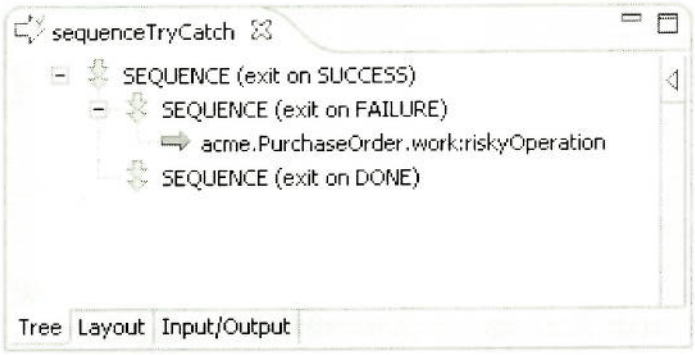


6. In the sequenceTryCatch service, add three **SEQUENCE** steps to create a Flow try/catch block, as follows:
   a. SEQUENCE set to Exit on Success
      i. SEQUENCE set to Exit on Failure (be sure it is indented under the first SEQUENCE)
      ii. SEQUENCE set to Exit on Done (be sure it is indented under the first SEQUENCE)

While creating the individual SEQUENCE statements, set their comment property to reflect the "exit on" condition.



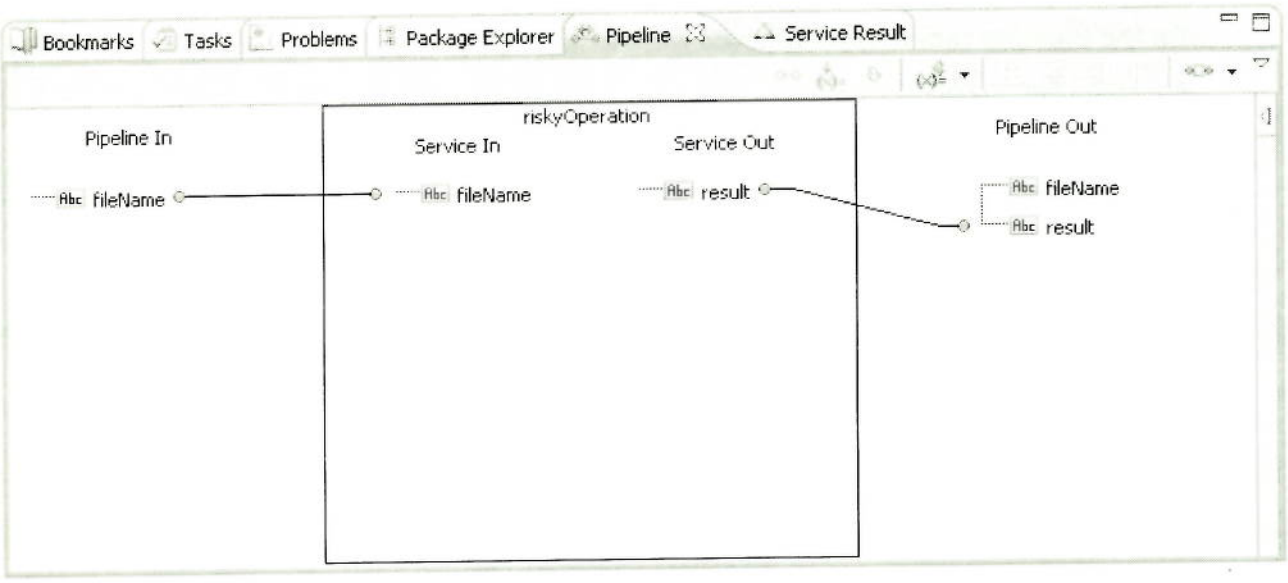
7. In the SEQUENCE Exit on Failure, add the service **acme.PurchaseOrder.work:riskyOperation** (be sure it is indented under the SEQUENCE Exit on Failure).



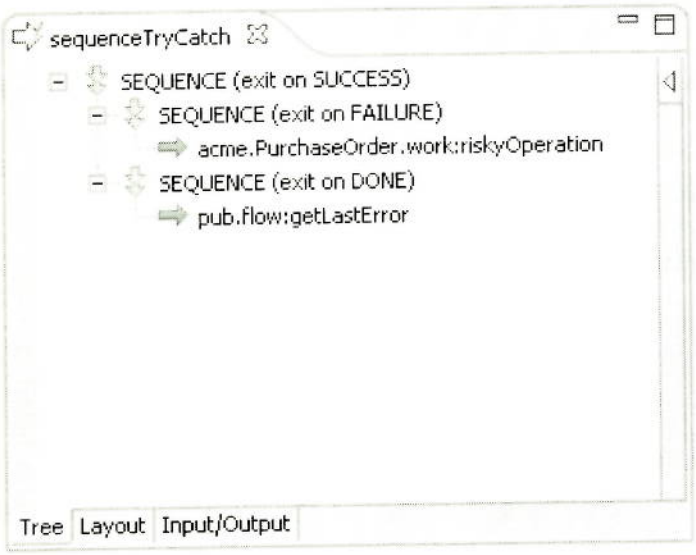
On the Pipeline tab, map as follows:

a. **fileName** to **fileName**

b. **result** to **result**

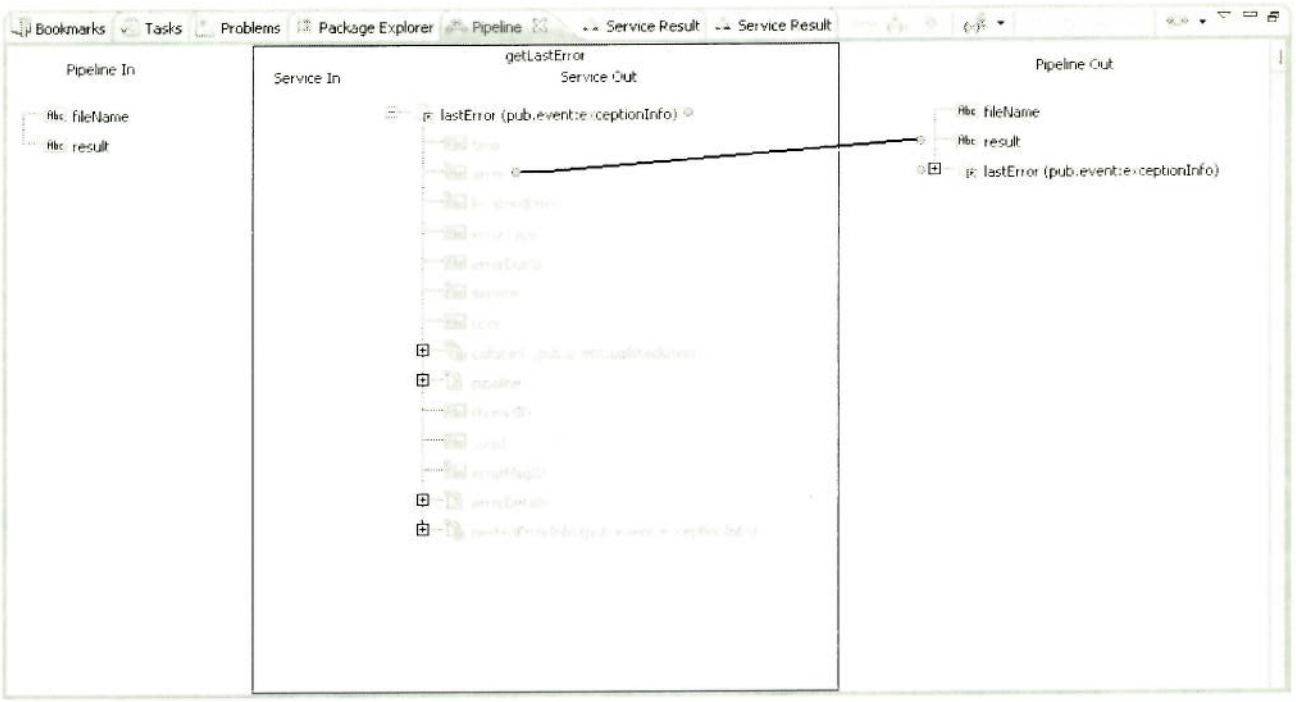


8. In the SEQUENCE Exit on Done, add the service **pub.flow:getLastError** (be sure it is indented under the SEQUENCE Exit on Done).

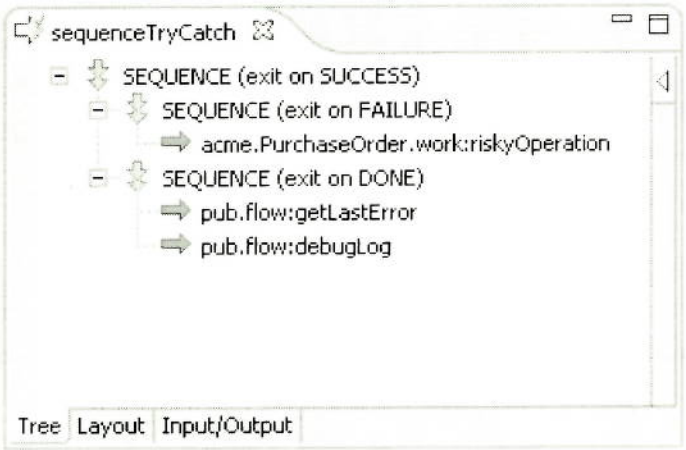


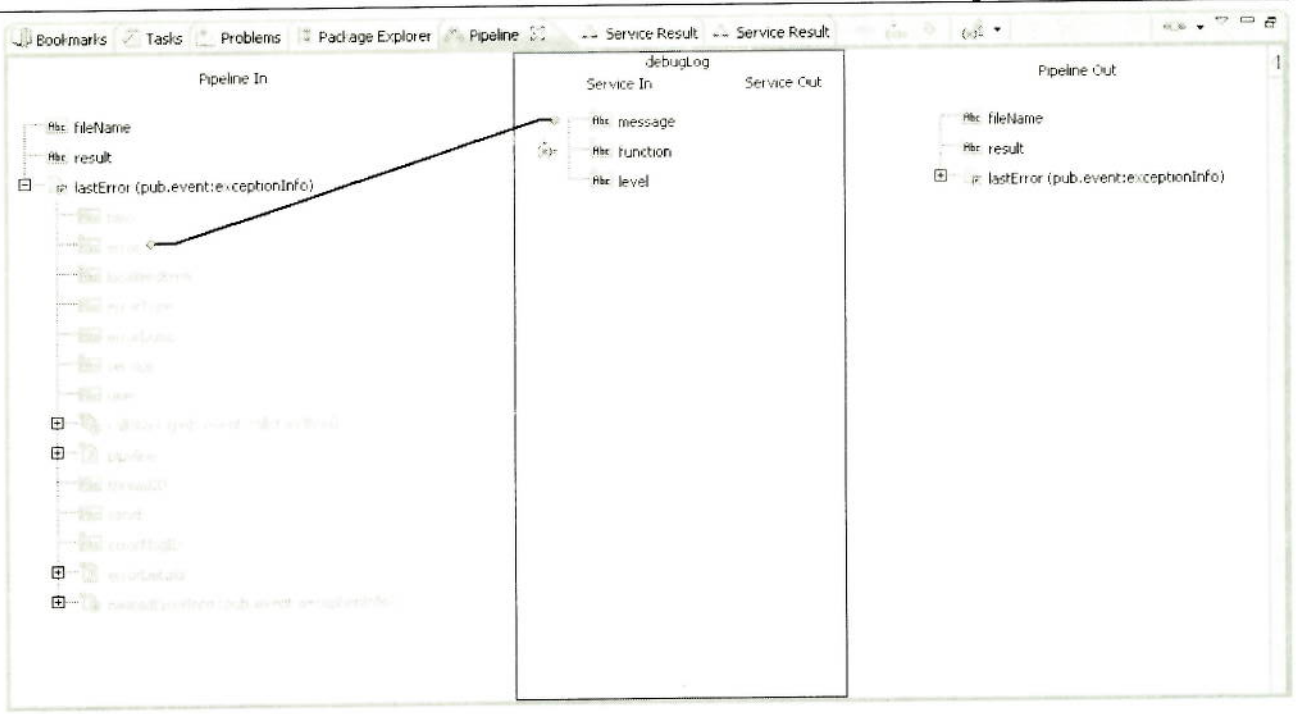
On the Pipeline tab, map as follows:

a. **lastError/error** to **result**



9. Add an invocation of the **pub.flow:debugLog** service below the invocation of **pub.flow:getLastError**. Map **lastError/error** to the **message** input parameter of the **debugLog** service. Set function to the usual eyecatcher.





10. Save and run your service. Check the Results tab and the server log.
    a. To fail, provide **c:\notthere.txt** as the file. Verify you see an error message in the server log.
    b. To succeed, provide **c:\boot.ini.** Successful execution will show the file contents in the Results tab and no error message in the server log.

    Try using the debugger in both cases to see the decision path.

Check Your Understanding

1. Rather than using a service you know will fail, how can you throw an Exception in Flow?
2. What happens if the riskyOperation service works (doesn't fail?)

This page intentionally left blank

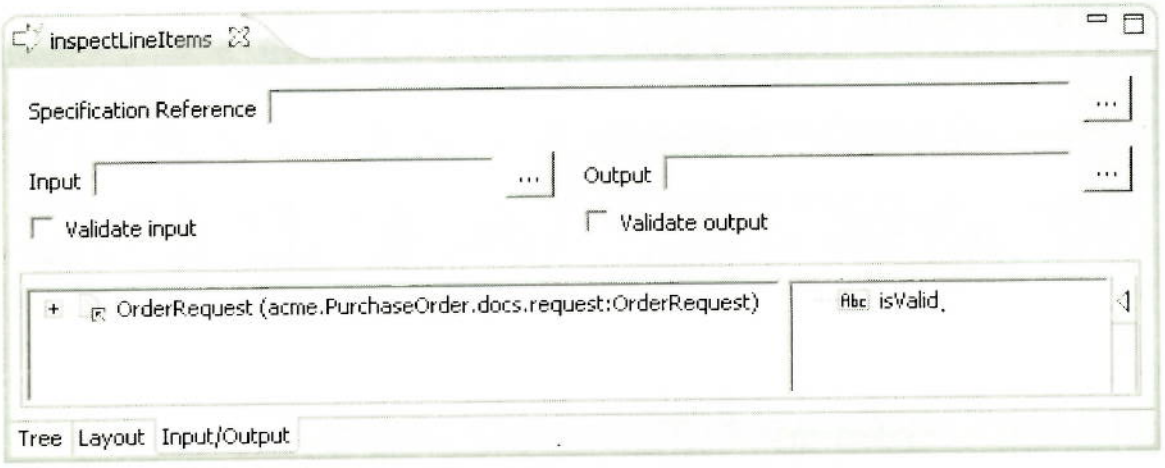
Exercise 8:
Validation Service

Overview

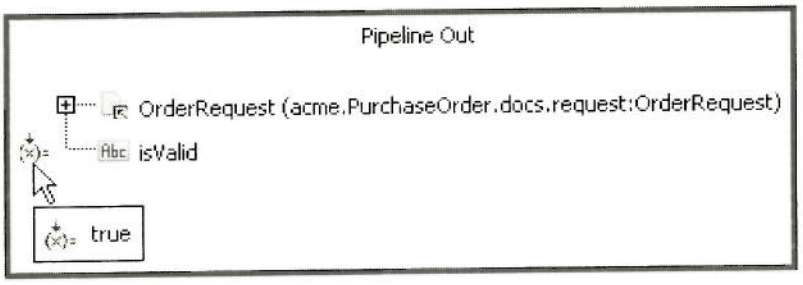
In this exercise, you will create business logic to validate an inbound Purchase Order. For now, this means to verify that the line items in the Purchase Order have a valid quantity. If this is not the case, you will flag the order as invalid for follow up by a customer service representative.

Steps

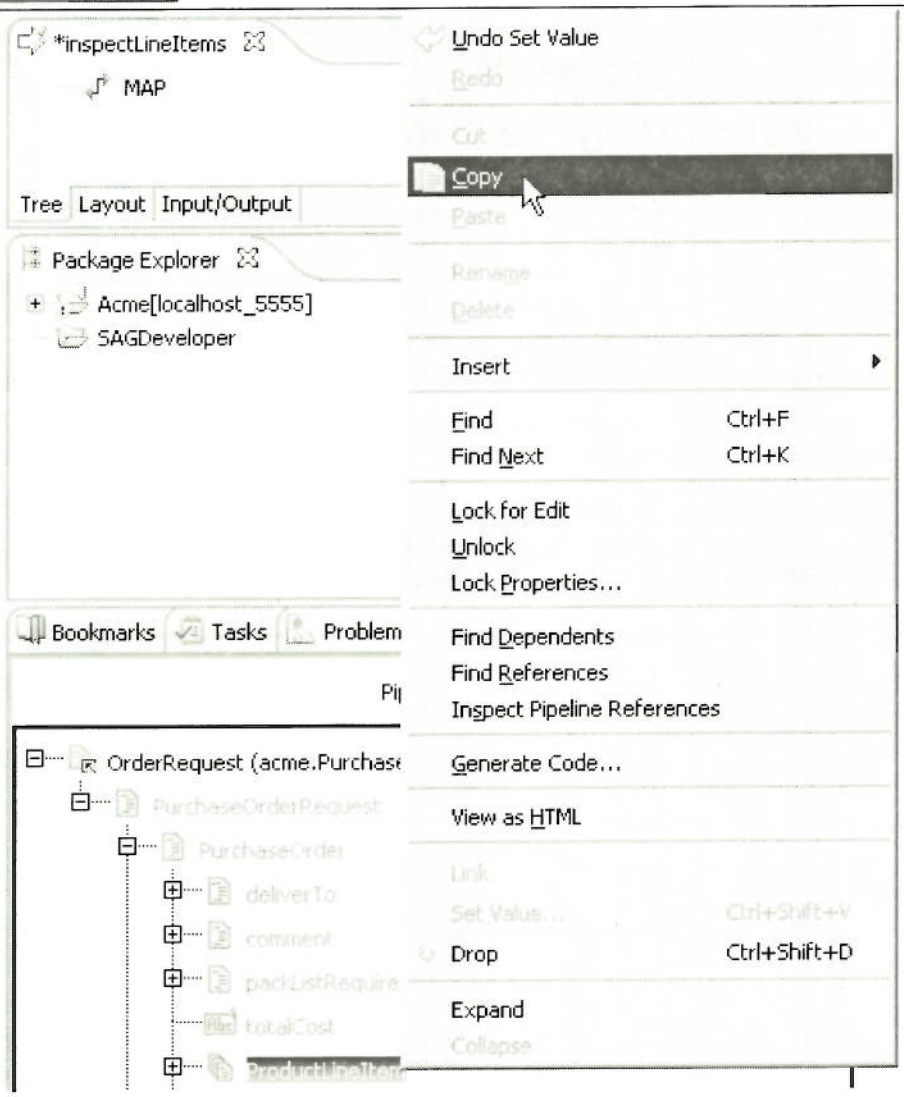
1. In the **acme.PurchaseOrder.utils** folder, create a service called **inspectLineItems.** For input, specify a document reference to **acme.PurchaseOrder.docs.request:OrderRequest.** Call this input Variable **OrderRequest**. As output variable, create a String called **isValid.**

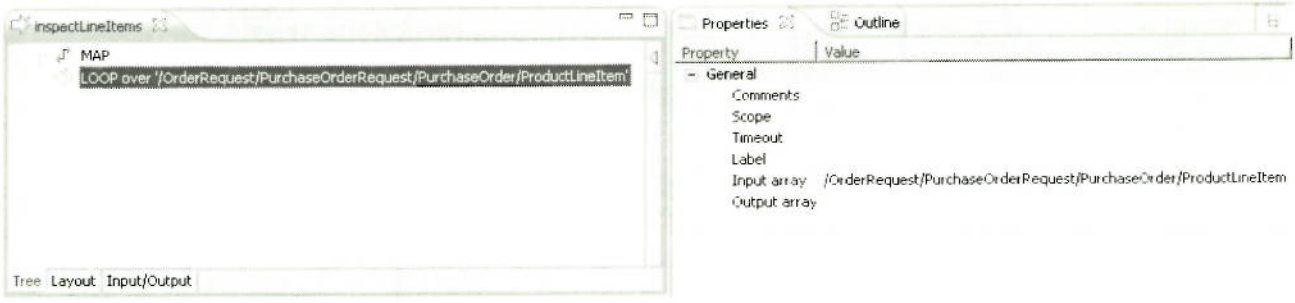


2. In the service, add a MAP step to initialize the **isValid** variable to **true.**



3. In the Pipeline tab, locate and copy the **OrderRequest/PurchaseOrderRequest/ PurchaseOrder/ProductLineItem** field. Then add a **LOOP** step to your service and set the **Input array** property by pasting the copied value.





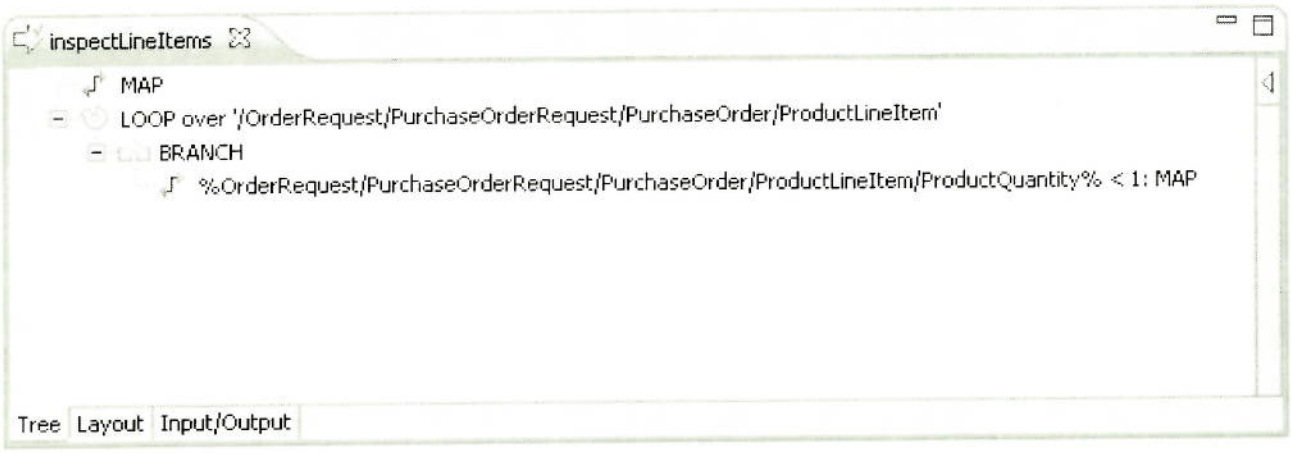
4. Under the LOOP step in your service, add a **BRANCH** step (be sure it is indented under the LOOP). Leave the **Switch** property empty and set **Evaluate labels** = **True**.

5. Now you will add code under the BRANCH. Add a MAP step below the BRANCH step (be sure it is indented under the BRANCH). In the Pipeline tab, locate and copy the **/OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/ProductQuantity** field.

   Then type: "**%% < 1**" (six characters, no quotes, around the smaller sign are spaces) into the Label property of the MAP step.

Then click the mouse between the two percent signs and paste the copied value. You should end up with %OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/ProductQuantity% < 1

6. In the Pipeline of the MAP step, set **isValid** = **false**. Your service should look like this:



7. Save and run the service. For input, load the file **...\IntegrationServer\packages\AcmeSupport\pub\order_request_input.txt.**

   When using this input file, **isValid** should be **true** in the Results.

   Run again and change one of the **ProductQuantity** values to 0 (in the **ProductLineItem** array). **isValid** should be **false** in the results.

8. **For extra credit**, stop processing after the first invalid **ProductLineItem**

Check Your Understanding

1. Why did we set isValid to true at the very beginning?
2. Is there another way we could have validated this particular value with writing Flow or Java?

This page intentionally left blank

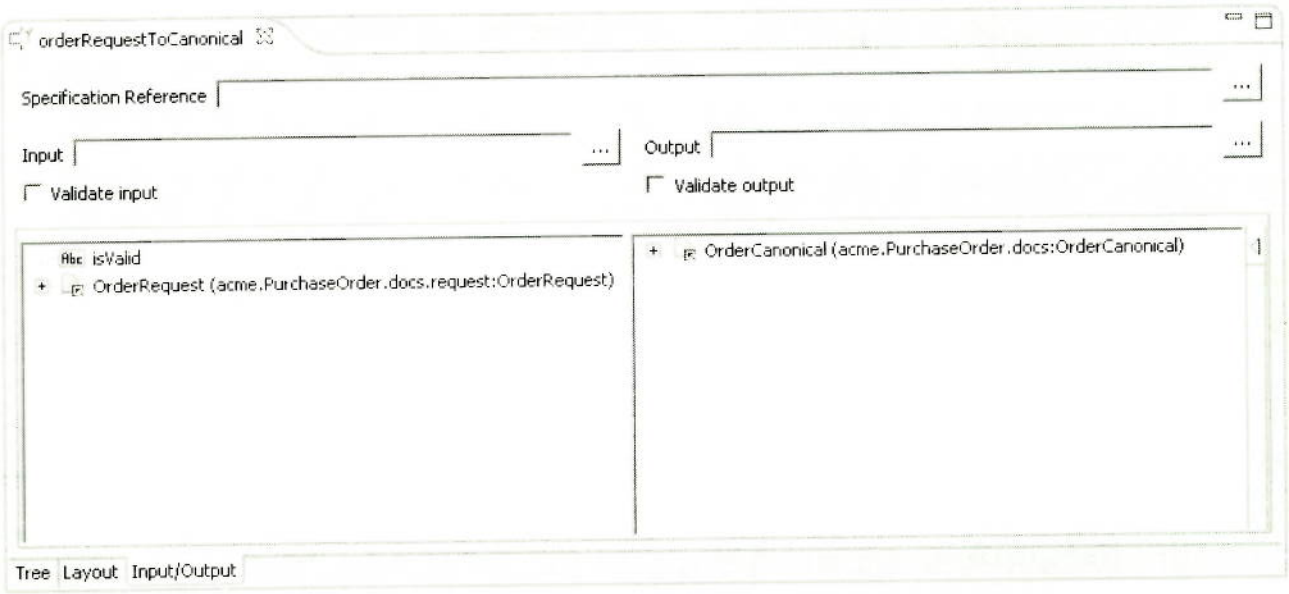
Exercise 9:
Mapping Service

Overview

In this exercise, you will create a service that maps from one data format to another using the document types you created in a previous exercise.

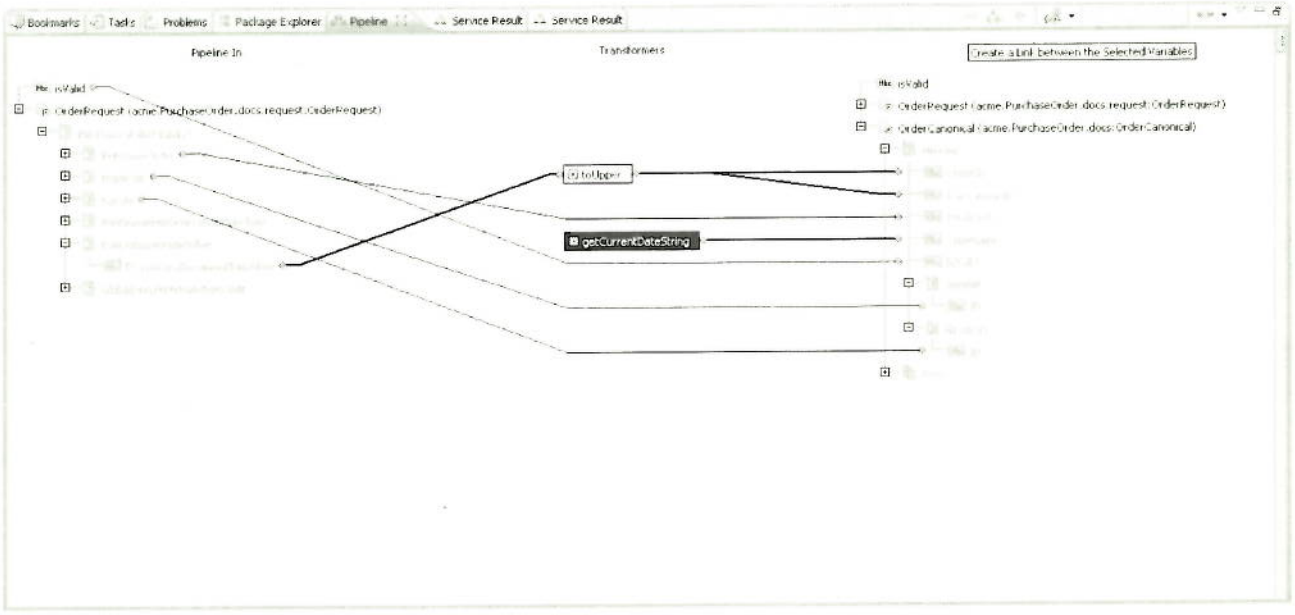
Steps

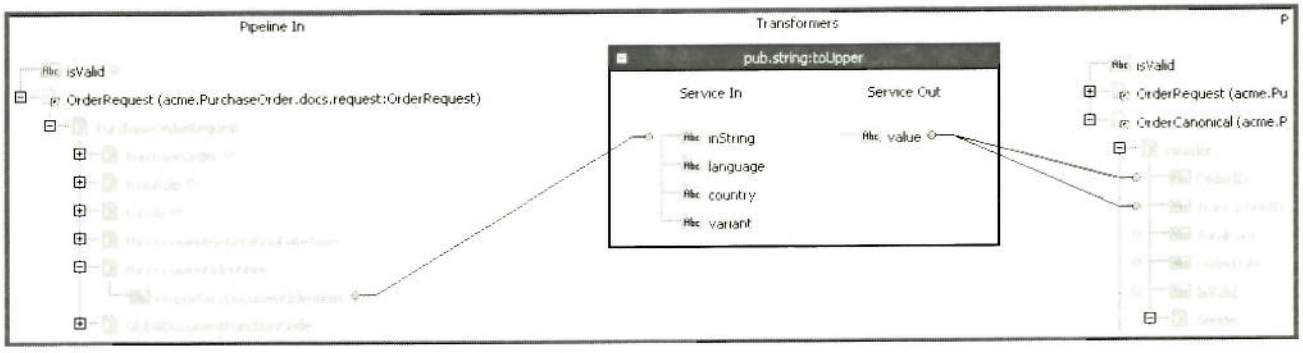
1. In the **acme.PurchaseOrder.maps** folder, create a Flow service called **orderRequestToCanonical**.
   a. Set the input to be a **String** variable called **isValid** and a document reference to **acme.PurchaseOrder.docs.request:OrderRequest** named **OrderRequest**.
   b. The output should be a document reference to **acme.PurchaseOrder.docs:OrderCanonical** named **OrderCanonical**.

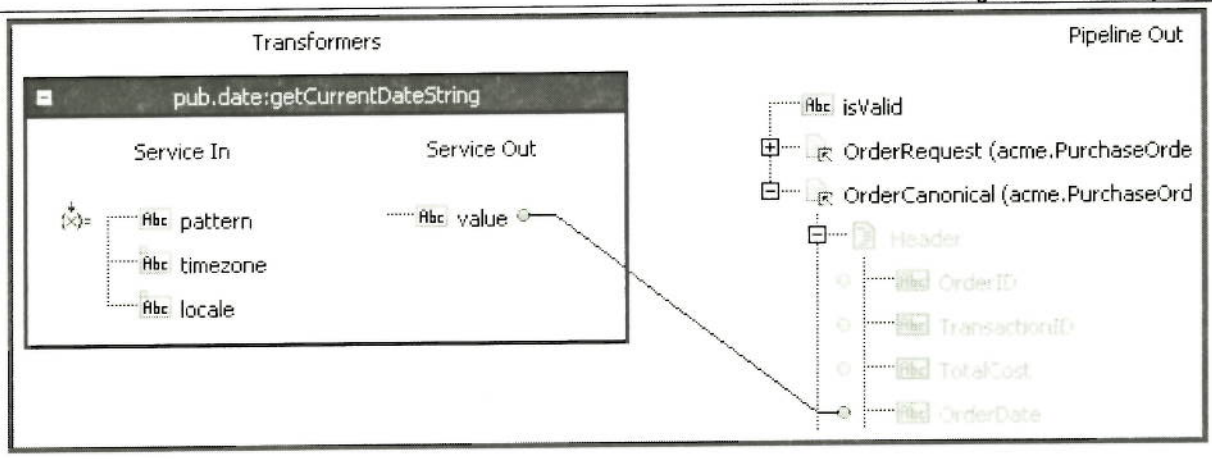


2. Add a **MAP** statement to your service. In the MAP statement, map the following variables from left to right on the Pipeline tab.
   a. OrderRequest/PurchaseOrderRequest/PurchaseOrder/totalCost to OrderCanonical/Header/TotalCost
   b. isValid to OrderCanonical/Header/IsValid
   c. OrderRequest/PurchaseOrderRequest/fromRole/PartnerRoleDescription/DUNS to OrderCanonical/Header/Sender/ID
   d. OrderRequest/PurchaseOrderRequest/toRole/PartnerRoleDescription/DUNS to OrderCanonical/Header/Receiver/ID

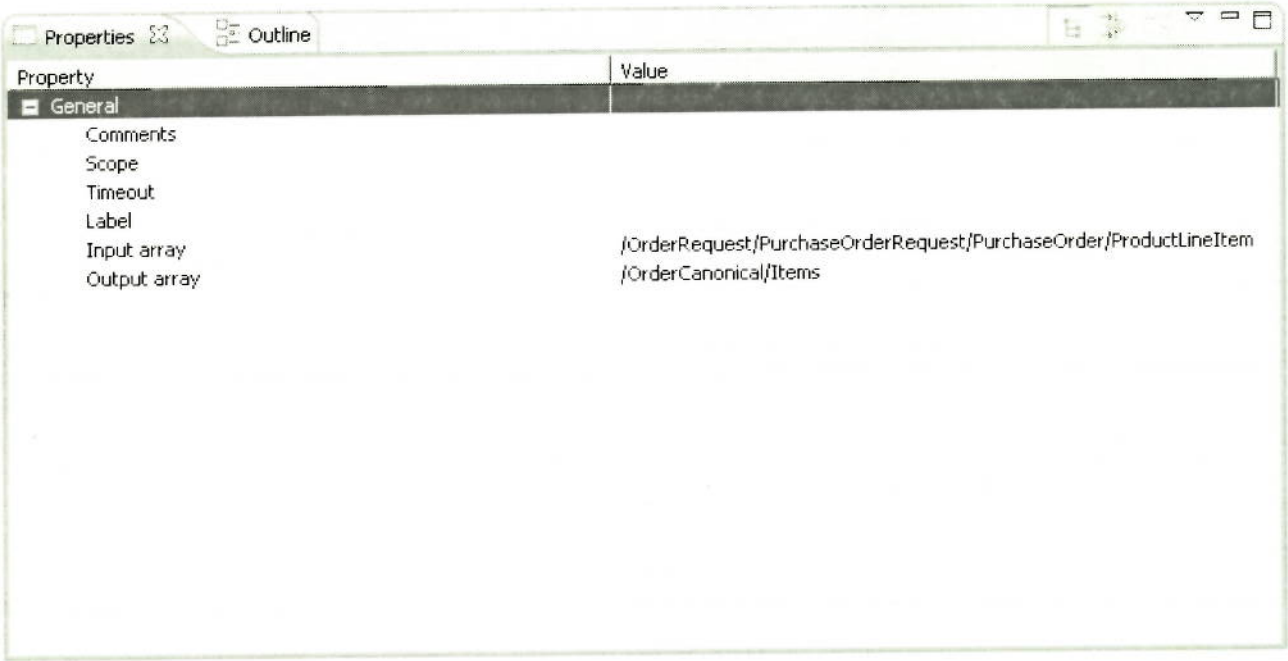
3. Add the service **pub.string:toUpper** as a transformer in your MAP step. Map the following variables from OrderRequest to the transformer and from the transformer to OrderCanonical:
   a. **OrderRequest/PurchaseOrderRequest/thisDocumentIdentifier/ ProprietaryDocumentIdentifer** to the transformer **inString**
   b. The transformer value to **OrderCanonical/Header/OrderID**
   c. The transformer value to **OrderCanonical/Header/TransactionID**
4. Add the service **pub.date:getCurrentDateString** as a transformer in your MAP step. Map the following variables in the transformer and the OrderCanonical document.
   a. Set the transformer **pattern** to **MMMM dd, yyyy**
   b. Map the transformer **value** to **OrderCanonical/Header/OrderDate**



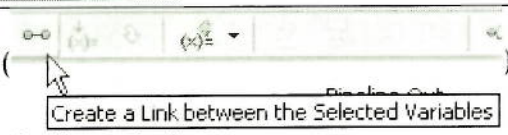




5. Add a **LOOP** step to the service. Set the input array to be **/OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem**. Set the output array to be **/OrderCanonical/Items.**

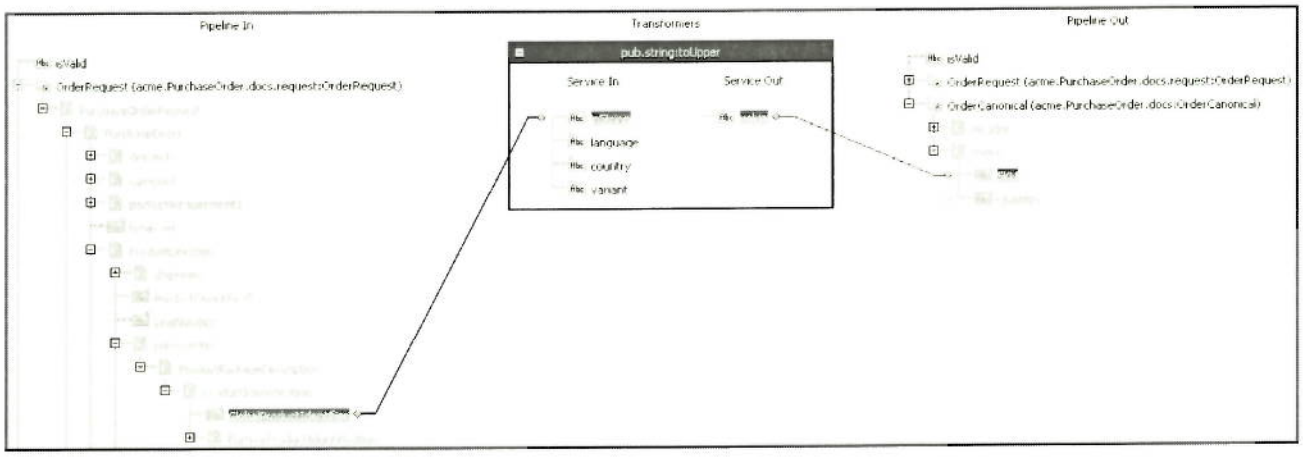


6. Add a **MAP** step in the LOOP (be sure it is indented under the LOOP statement). Map **/OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/ProductQuantity** from the **OrderRequest** variable to the **OrderCanonical**'s variable **/OrderCanonical/Items/Quantity** member.
Hint: When mapping such deeply nested structures, it is often easier to create a map line in two steps. First select the **from** and the **to** fields by clicking them with the mouse. In the second step connect the two selected fields by clicking on the "Create a Link..."

(

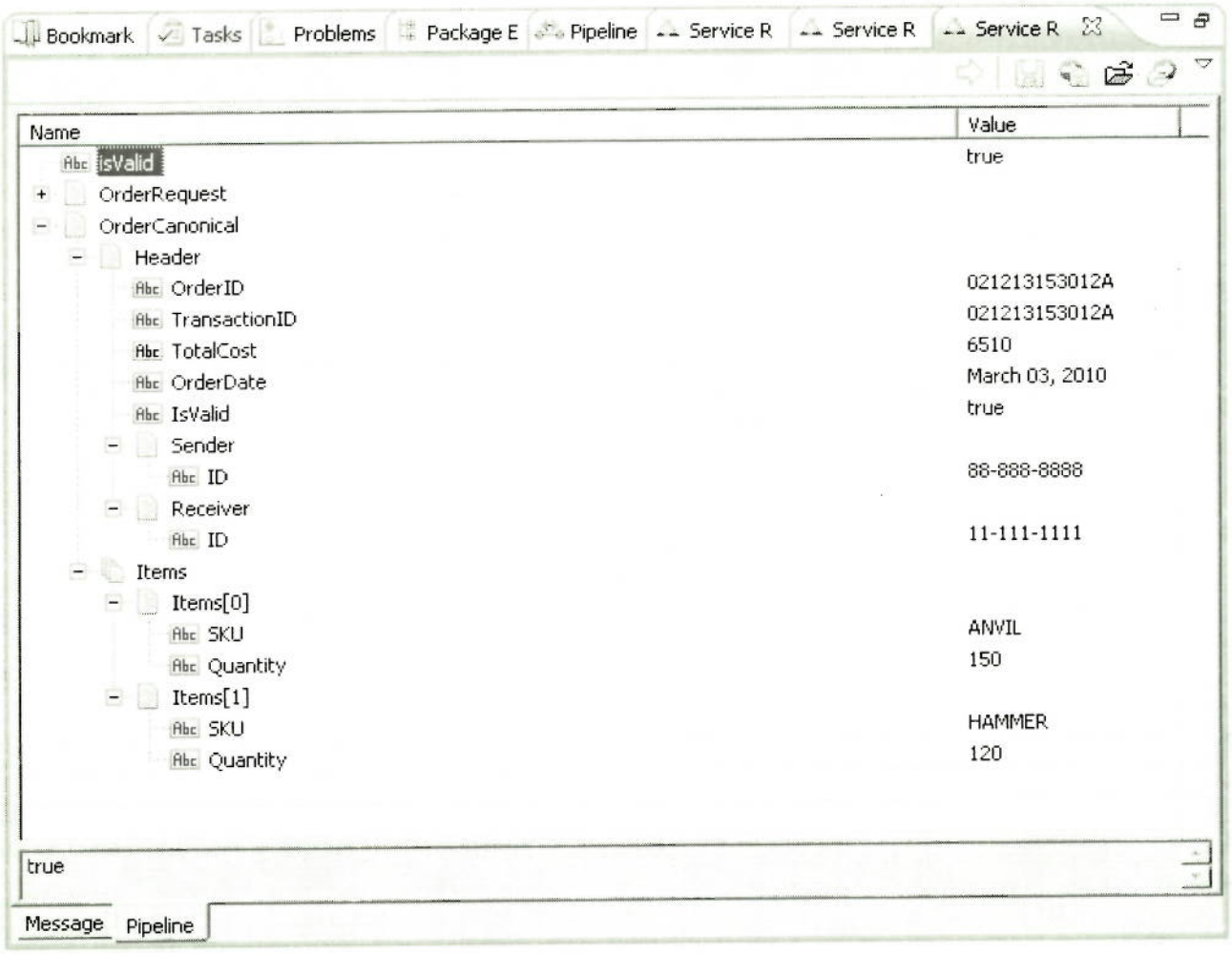
) button. This button is located in the titlebar of the pipeline window.

7. Add the service **pub.string:toUpper** as a transformer in the MAP step. Map the following variables:
   - **/OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/productUnit/ProductPackageDescription/ProductIdentification/GlobalProductIdentifier** to the transformer's **inString**
   - Transformer **value** to **/OrderCanonical/Items/SKU**



8. Save the service and run. Use the Load button and the input file **...\IntegrationServer\packages\AcmeSupport\pub\order_request_input.txt** (Do not forget to set **isValid** to true or false when you test!).

   Check the **Results** panel. Collapse **OrderRequest** and look at **OrderCanonical.** This variable must be completely populated. Especially check the date and the uppercase **OrderID**, **TransactionID**, and **SKU** values.



Check Your Understanding

1. How is a transformer different from a normal service?
2. What if the transformer you want to use is not in the transformer drop-down list?
3. Why did we need to LOOP over ProductLineItems? Why not just map from ProductLineItems to Items?

Check Your Understanding

1. What exactly is each line of the Java code doing in the endsWith service?
2. Is the service thread safe? What would you have to do if not?
3. How could the cursor handling be improved?