webMethods 8
Integration Workshop

Exercise Guide

TRA611-42E

TABLE OF CONTENTS

Exercise 1: Start the Integration Server, Broker, and MWS 3
Exercise 2: Packages and Folders 9
Exercise 3: Create a Service 13
Exercise 4: Document Types 21
Exercise 5: Flow Services - BRANCH 25
Exercise 6: Building Flow Services - LOOP 29
Exercise 7: Building Flow Services – SEQUENCE 33
Exercise 8: Validation Service 41
Exercise 9: Mapping Service 45
Exercise 10: Create a Java Service 51
Exercise 11: Monitoring Services 53
Exercise 12: Invoking Services 57
Exercise 13: Create a Flat File Schema 73
Exercise 14: Create a Flat File Dictionary 79
Exercise 15: Web Service Descriptors and Custom Faults 83
Exercise 16: Broker Pub/Sub 91
Exercise 17: JMS Pub/Sub 95
Exercise 18: Create Adapter Services 99
Exercise 19: Adapter Notifications 107
Exercise 20: Use Services In a Business Process 113
Check Your Understanding: Anwers to the Questions 121

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

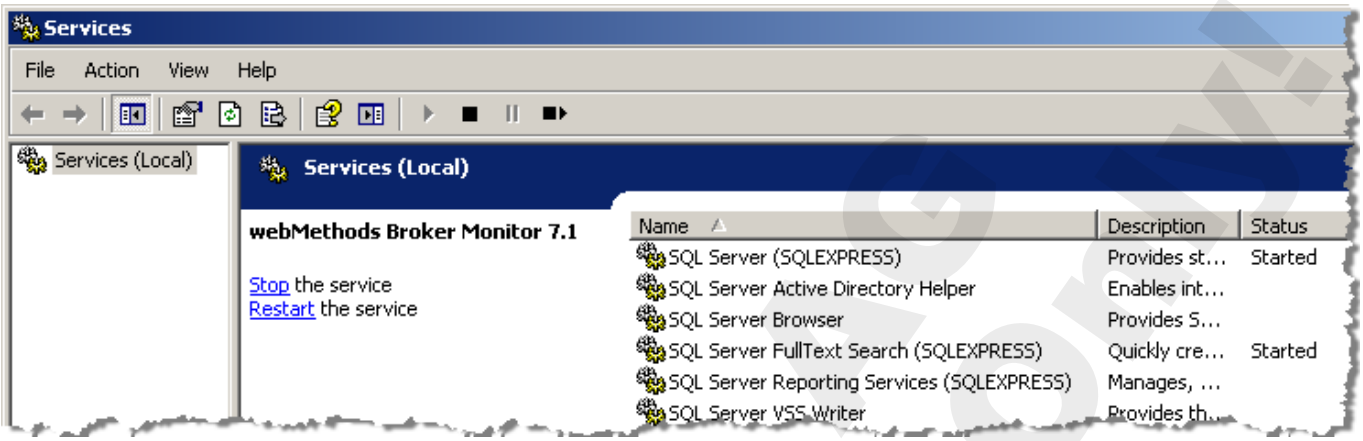
1. Before starting with the exercises, you need to set up your virtual machine appropriately. This is done by running the batch procedure **setup611.bat** from the folder **C:\TRAINING\611-41E** by using a command prompt window. Note: It is extremely important that you perform the execution in 2 steps.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

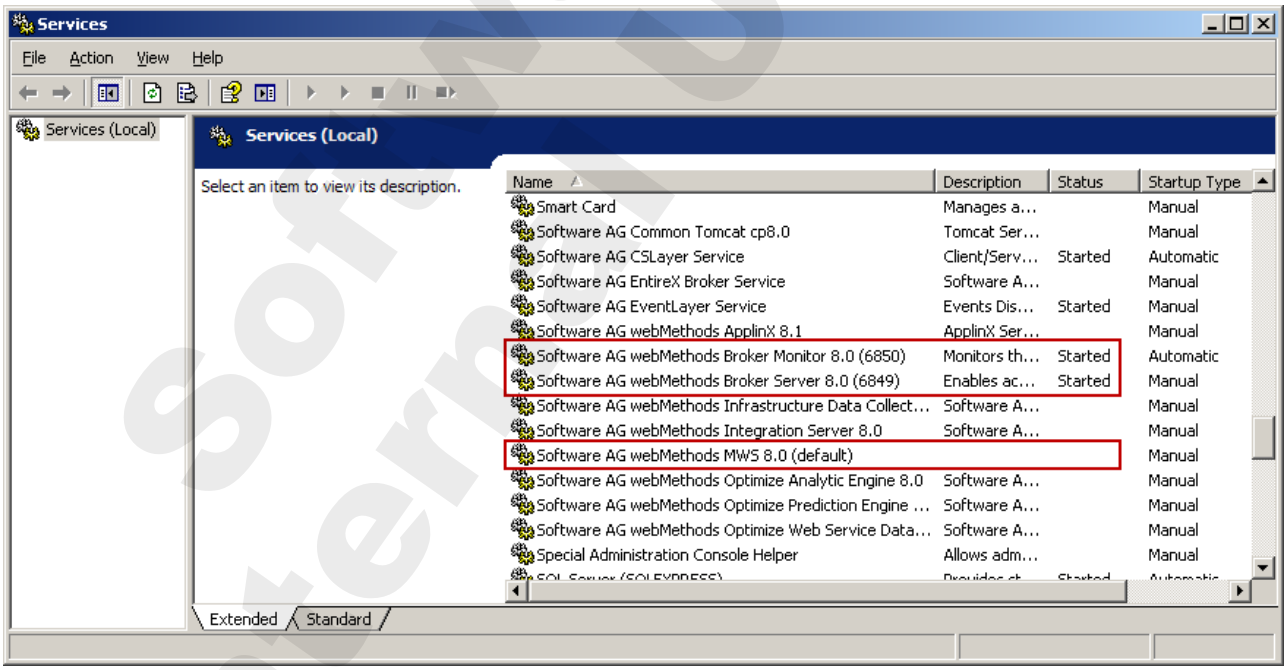
   Check the output of the setup procedure if it produced any errors.

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.

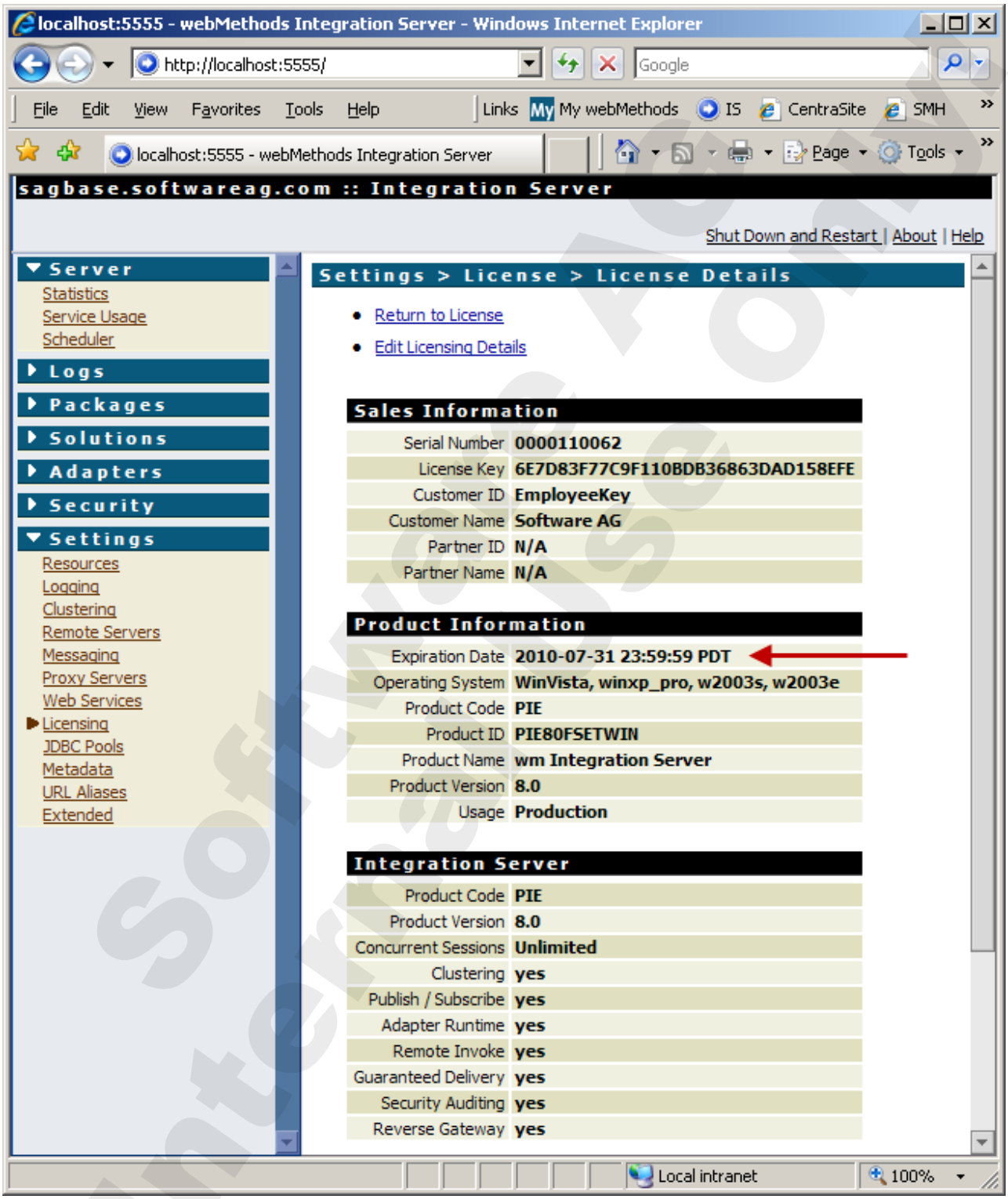
```
Command Prompt - tail -f IntegrationServer\logs\server.log

C:\SoftwareAG>tail -f IntegrationServer\logs\server.log
2010-03-12 14:20:19 CET [ISU.0000.9999I] C:\SoftwareAG\IntegrationServer\.\packages\WmRules\conf
ig\rules.cnf file exists
2010-03-12 14:20:19 CET [ISU.0000.9999I] distribute rules true
2010-03-12 14:20:19 CET [ISU.0000.9999I] Registered servers [sagbase.softwareag.com:5555]
2010-03-12 14:20:19 CET [ISS.0028.0012I] WmTaskClient: Startup service (wm.task.taskclient:init)

2010-03-12 14:20:19 CET [ISS.0028.0012I] PSUtilities: Startup service (PSUtilities.config:loadPS
UtilitiesConfig)
2010-03-12 14:20:19 CET [ISS.0028.0012I] PSUtilities: Startup service (PSUtilities.config:setACL
s)
2010-03-12 14:20:19 CET [ISP.0046.0012I] Enabling HTTP Listener on port 9999
2010-03-12 14:20:19 CET [ISP.0046.0012I] Enabling HTTP Listener on port 5555
2010-03-12 14:20:19 CET [ISP.0046.0012I] Enabling HTTP Listener on port 15006
2010-03-12 14:20:20 CET [ISS.0099.0001E] Broker Transport:J_jRQtEolDEurgAUIwwwADXaZQs__TNProcess
.TNProcess3.Default_subscriptionTrigger  unable to subscribe to Broker Document wm::is::TNSuppor
t::docs::OrderCanonical. Exception Unknown Document Type (226-1490): Document type 'wm::is::TNSu
pport::docs::OrderCanonical' is not defined in the Broker.

2010-03-12 14:20:20 CET [ISS.0098.0046I] Trigger J_jRQtEolDEurgAUIwwwADXaZQs__TNProcess.TNProces
s3.Default_subscriptionTrigger is only available for Local Publishing. Unable to create subscrip
tion for Publishable Document TNSupport.docs:OrderCanonical: Unknown Document Type (226-1490): D
ocument type 'wm::is::TNSupport::docs::OrderCanonical' is not defined in the Broker.
```

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".

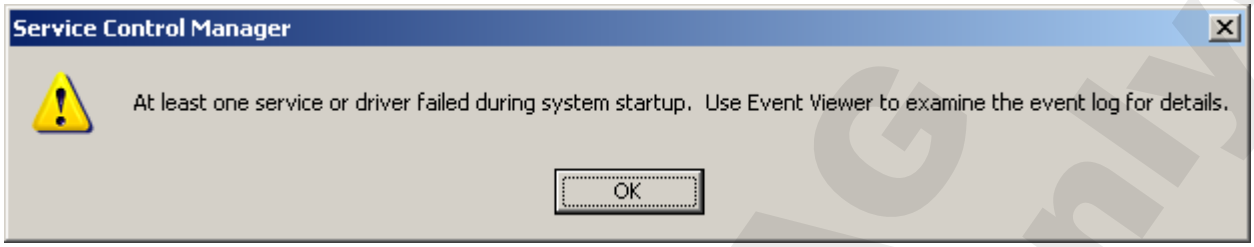
```
Command Prompt - tail -f c:\SoftwareAG\MWS\server\default\logs\_full_.log
2010-03-01 02:10:11 PST (Framework:INFO)  - Initializing components of: wm_wsrp_consumer
2010-03-01 02:10:11 PST (Framework:INFO)  - Loading phase: [phaseID, phaseName] [deploySync, dep
loySync]
2010-03-01 02:10:11 PST (Framework:INFO)  - Initializing components of: deploySync
2010-03-01 02:10:11 PST (Framework:INFO)  - Initializing component: com.webmethods.portal.bizPol
icy.biz.install.impl.AutodeployComponents
2010-03-01 02:10:12 PST (Framework:INFO)  - Initializing component: com.webmethods.portal.bizPol
icy.biz.install.impl.RetryFailedComponents
2010-03-01 02:10:12 PST (Framework:INFO)  - Initializing component: com.webmethods.portal.bizPol
icy.biz.install.impl.AutodeployComponents
2010-03-01 02:10:12 PST (Framework:INFO)  - Initializing component: com.webmethods.portal.bizPol
icy.biz.install.impl.SyncDeployService
2010-03-01 02:10:12 PST (Framework:INFO)  - Loading phase: [phaseID, phaseName] [startupComplete
, startupComplete]
2010-03-01 02:10:12 PST (Framework:INFO)  - Initializing components of: startupComplete
2010-03-01 02:10:12 PST (Framework:INFO)  - Initializing component: com.webmethods.portal.system
.PostInitStatus
2010-03-01 02:10:12 PST (Framework:INFO)  - My webMethods Server "default" Node "sagbase.softwar
eag.com-node32316" took 108 seconds to initialize
        role: search
        role: notification
        role: autodeploy
        role: taskengine
        http listening at: sagbase.softwareag.com:8585
        FrontEndUrl: http://sagbase.softwareag.com:8585
```

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

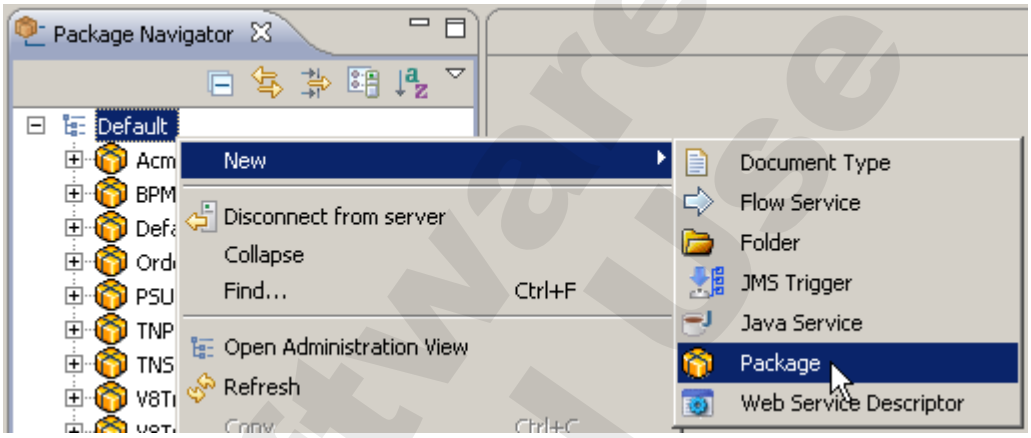
Overview

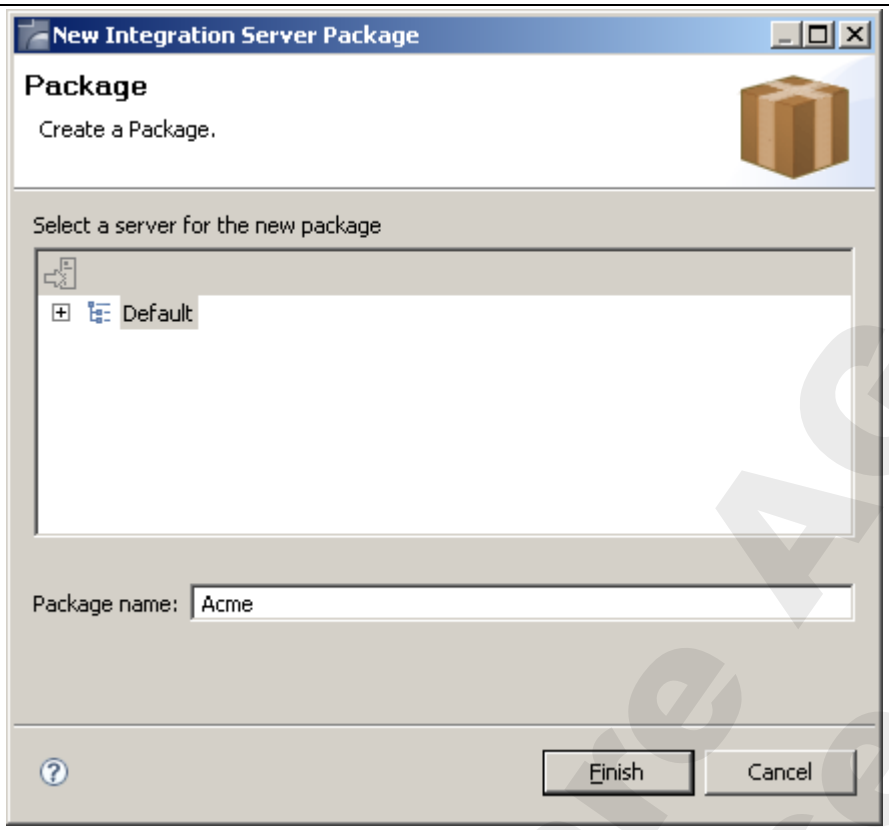
In this exercise, you will create a package and folders to store the Integration Server development work you will perform in future exercises.

Steps

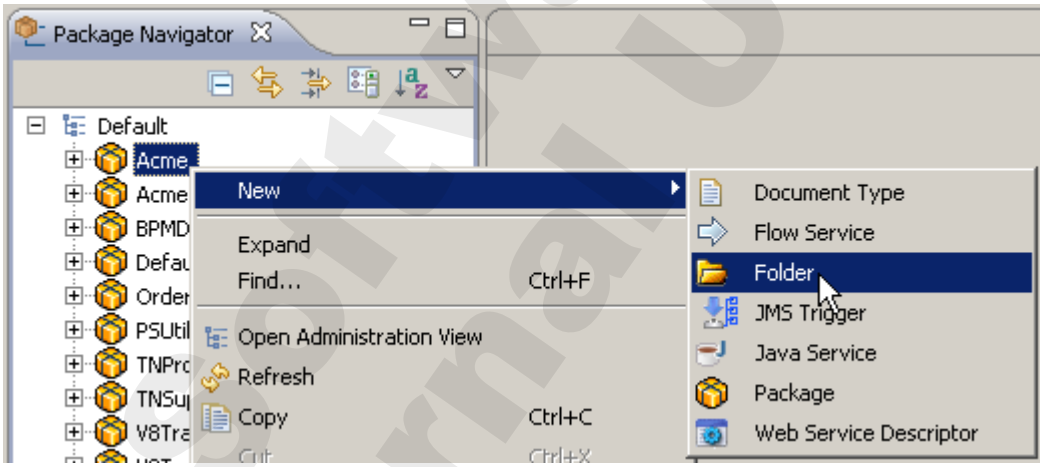
To begin development, we need a package and folder organization to store our future development work.

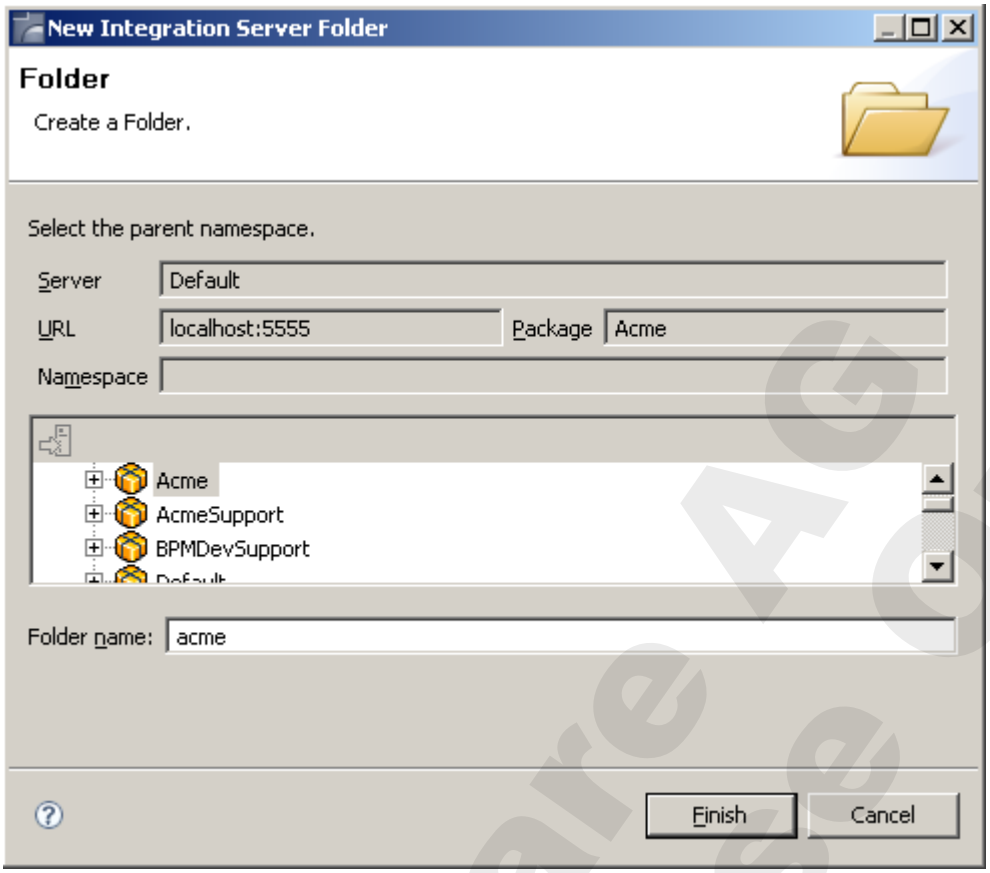
1. Open Software AG Designer, keep the default workspace, and then check the box beside "Use this as the default and do not ask again".
2. Create a new package called **Acme**.





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

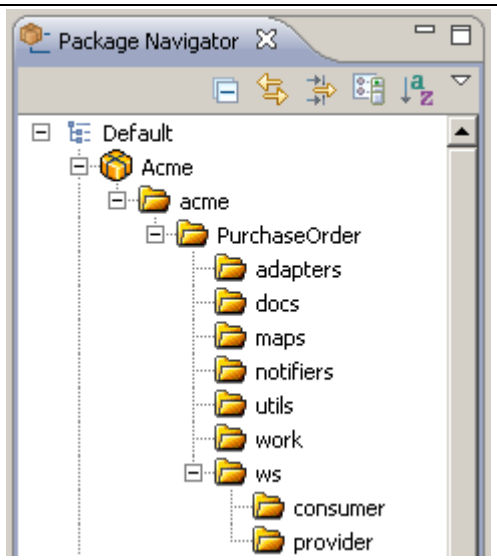




4. In the **acme** folder, create a folder called **PurchaseOrder**.
5. In the **acme.PurchaseOrder** folder, create 7 new folders, named as follows:
   a. adapters
   b. docs
   c. maps
   d. notifiers
   e. utils
   f. work
   g. ws

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?

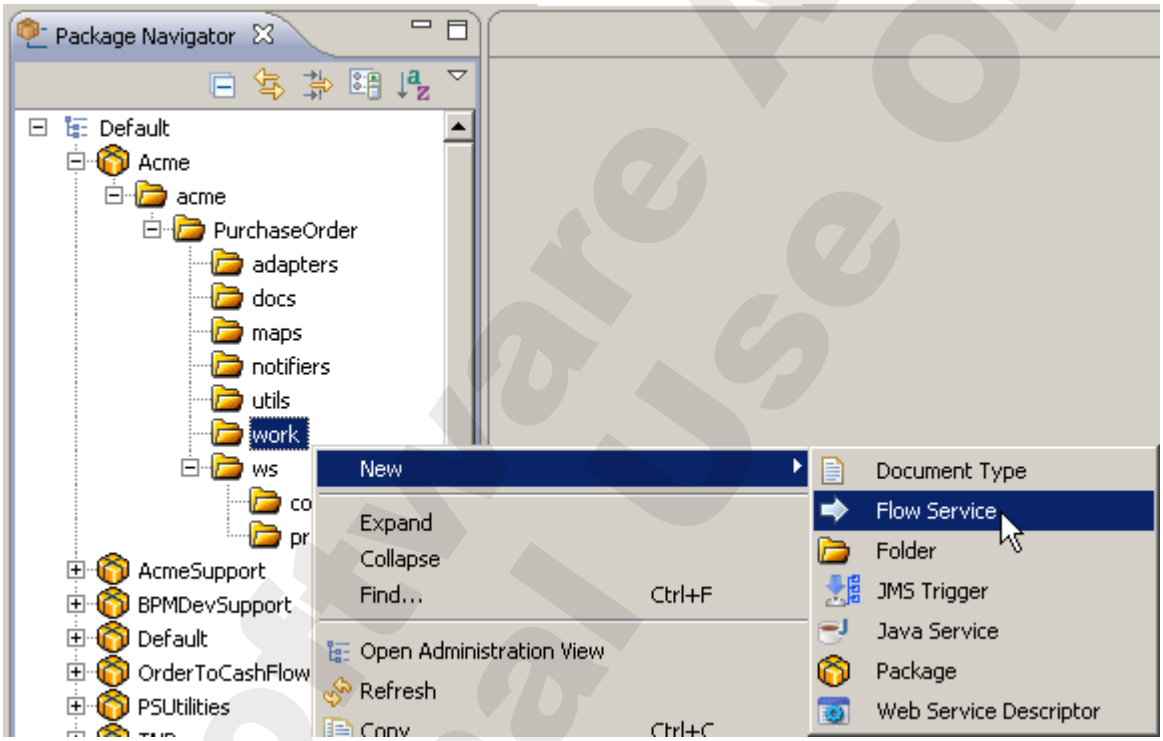
Exercise 3:
Create a Service

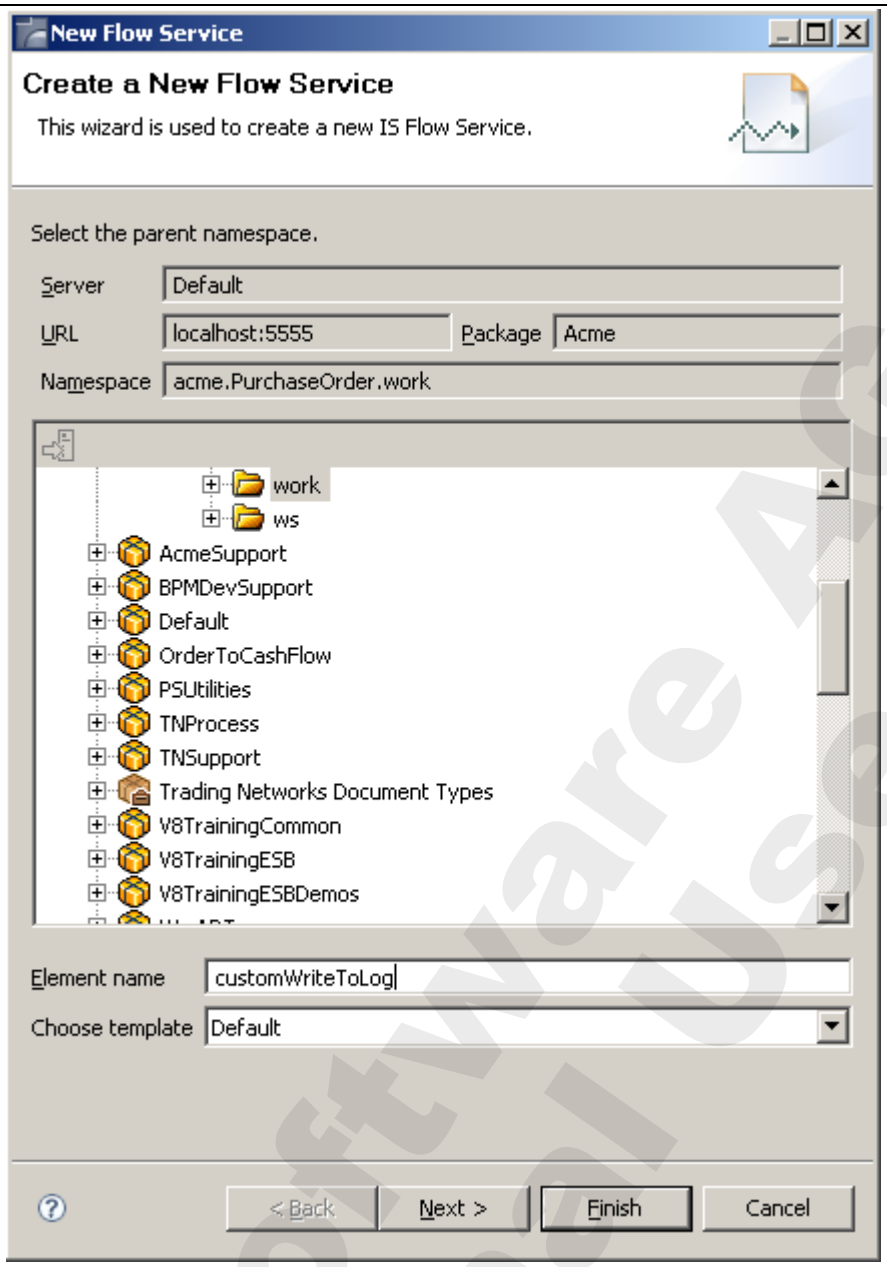
Overview

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

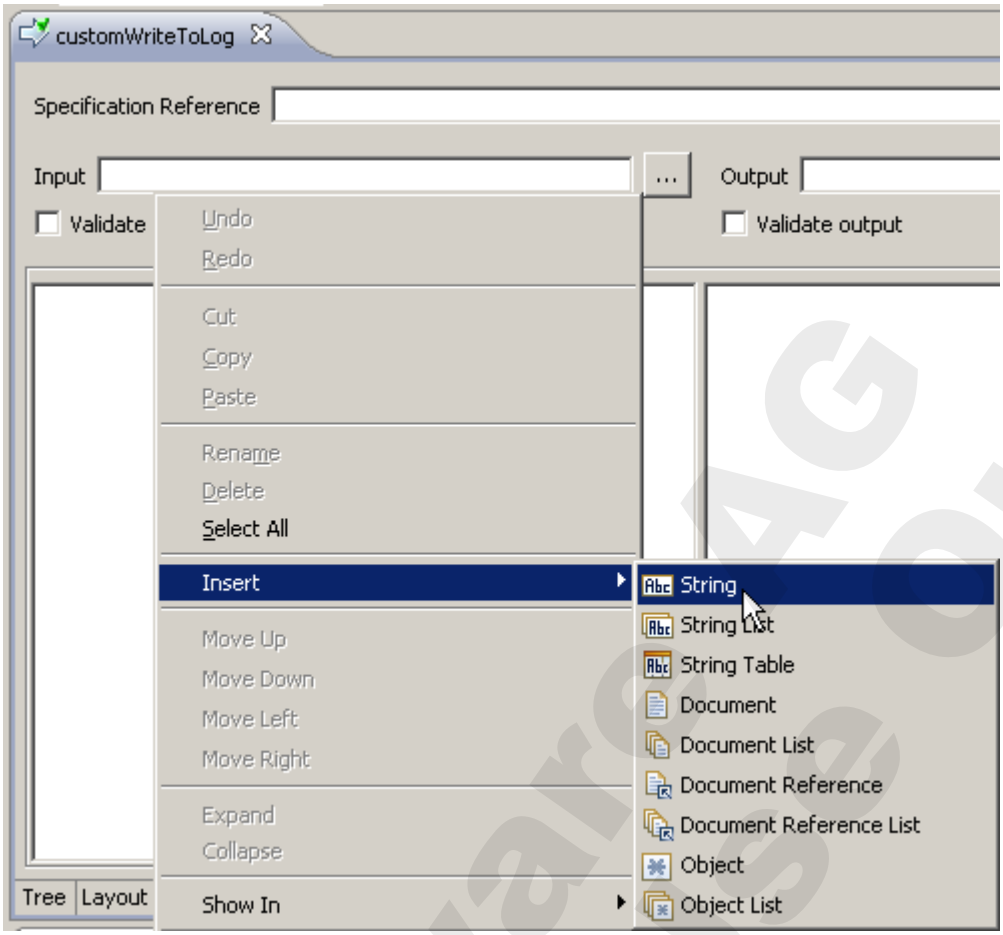
Steps

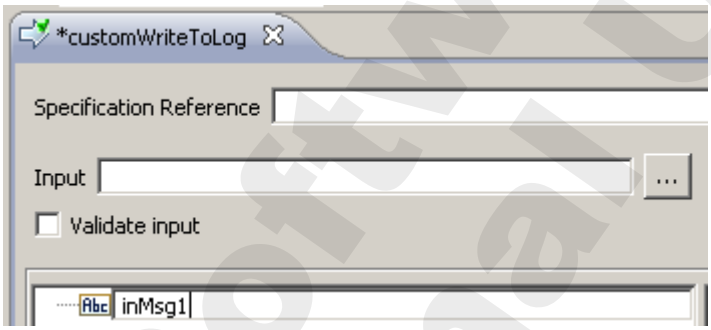
1. In the **Acme** package, in the **acme.PurchaseOrder.work** folder, create a flow service called **customWriteToLog**.

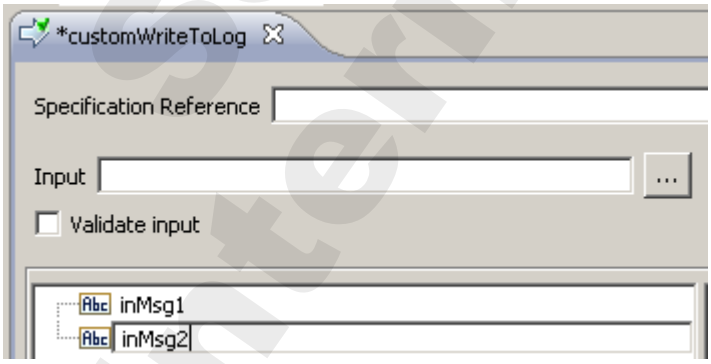




2. Add two String inputs to the new service, called **inMsg1** and **inMsg2**. To bing up the menu below, do a right click in the Input panel.







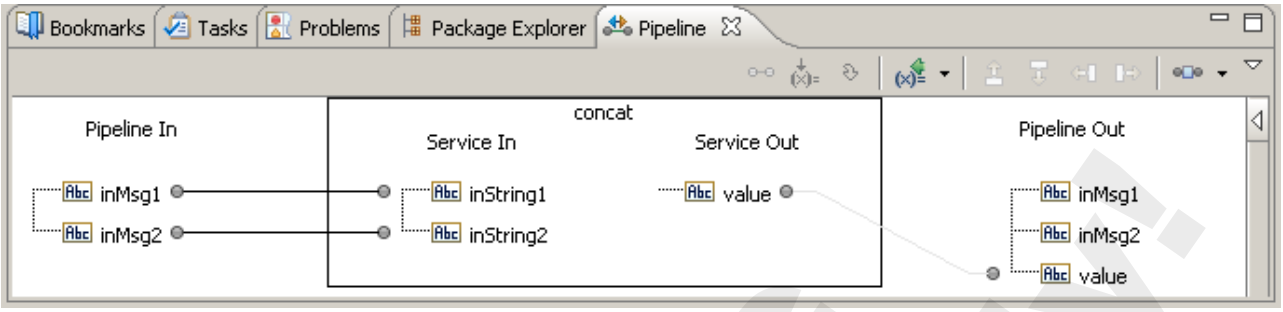
3. Add three service steps to the new service, as follows:
   a. **pub.flow:debugLog**
   b. **pub.string:toUpper**
   c. **pub.string:concat**

   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.

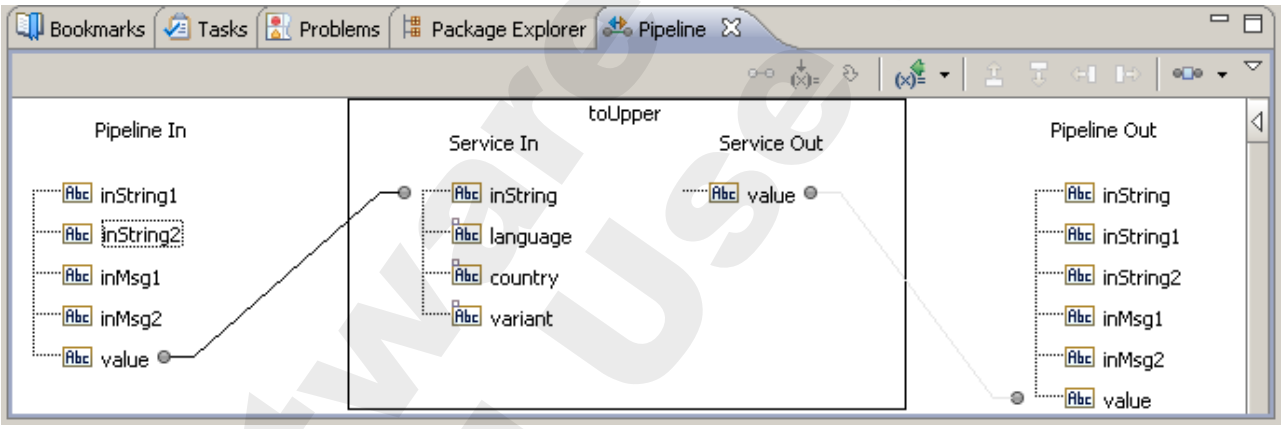
5. Map the data flow of your input through the services, as follows:
   a. pub.string:concat
      i. **inMsg1** should map to **inString1**
      ii. **inMsg2** should map to **inString2**



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.

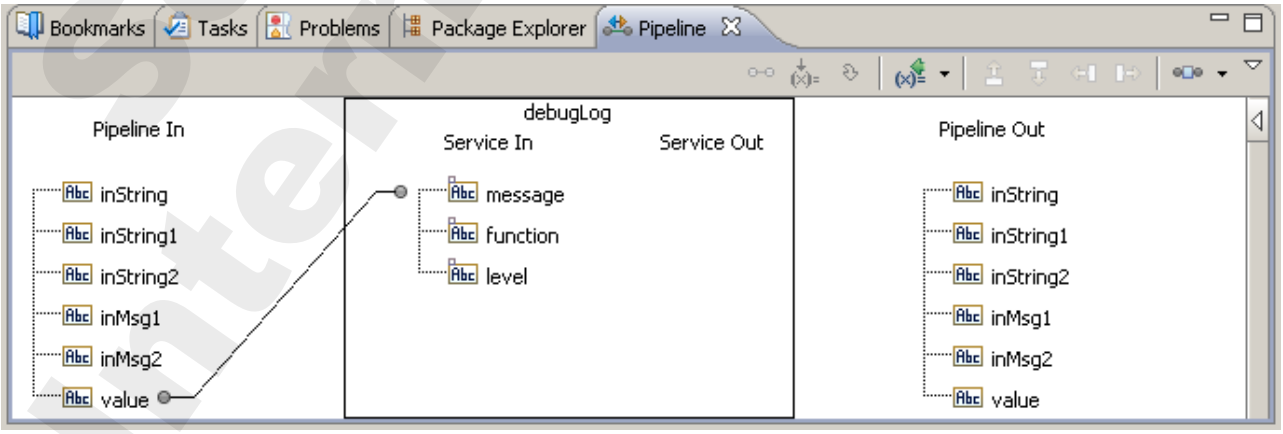
b. pub.string.toUpper

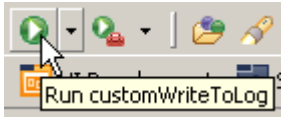
i. **value** should map to **inString**



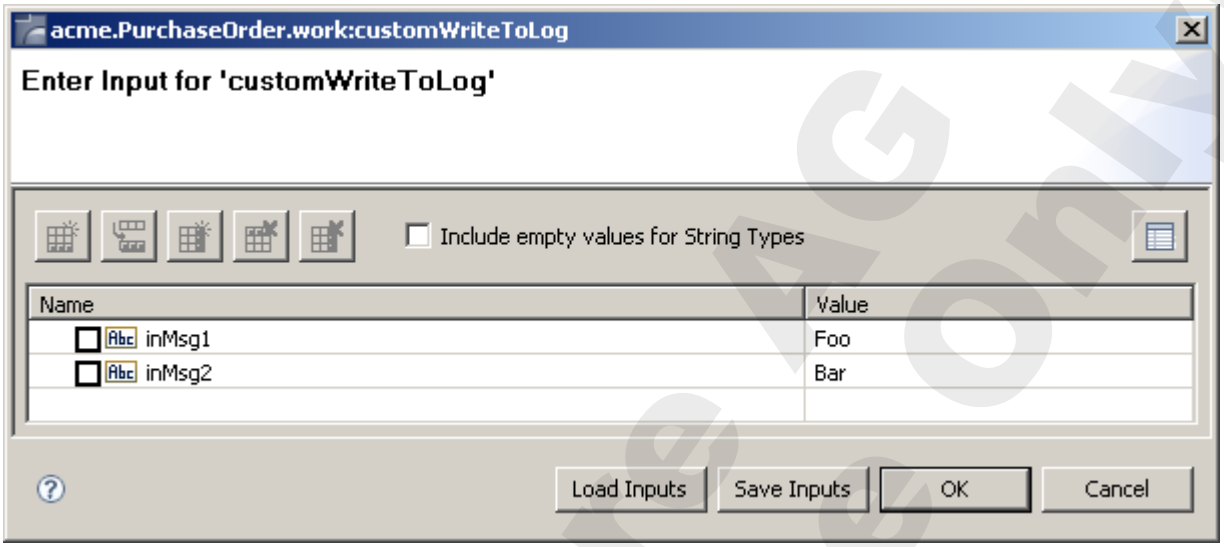
c. pub.flow:debugLog

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

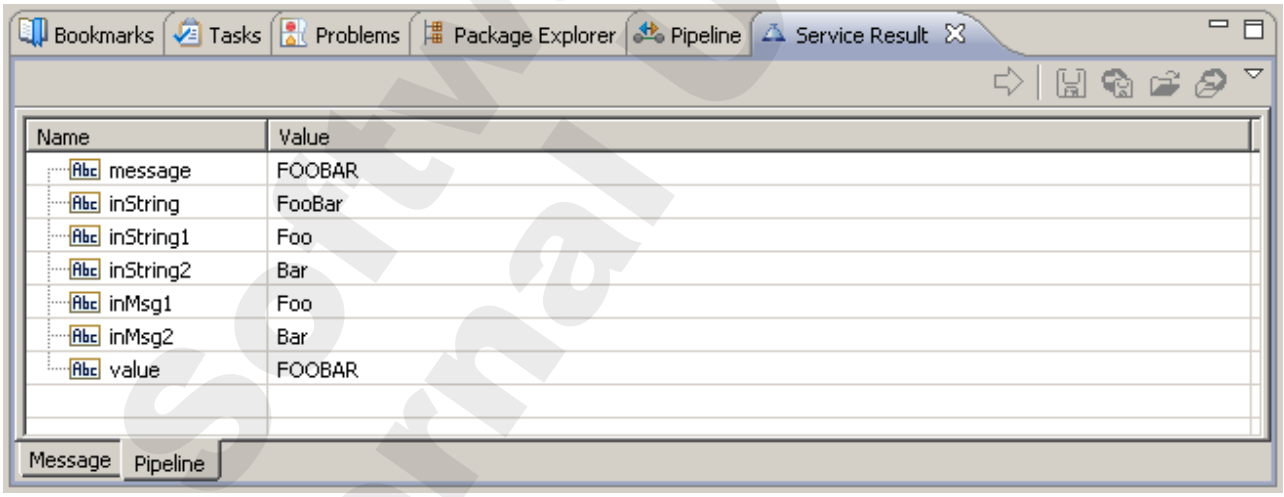


6. Save and run the service. For the first run of your service you must choose the "Run As" ➡ "Run Flow Service" from the context menu of the service. For further invokes you can simply hit the green arrow button in the upper toolbar: 

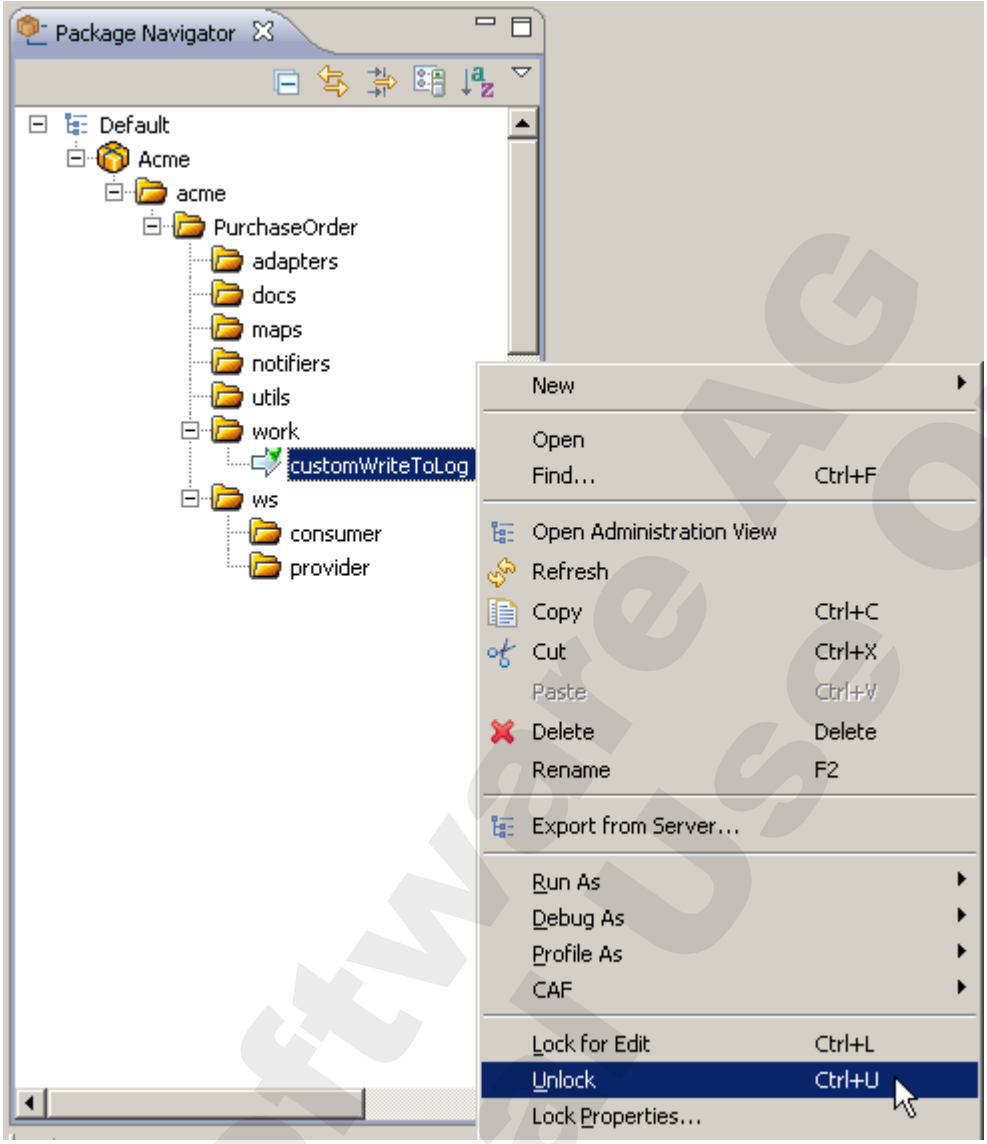
Then provide string inputs of your choice.



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?
2. How many inputs can the pub.string:toUpper service accept?
3. Where would the server log appear if the server is not running through the Command Prompt?

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.

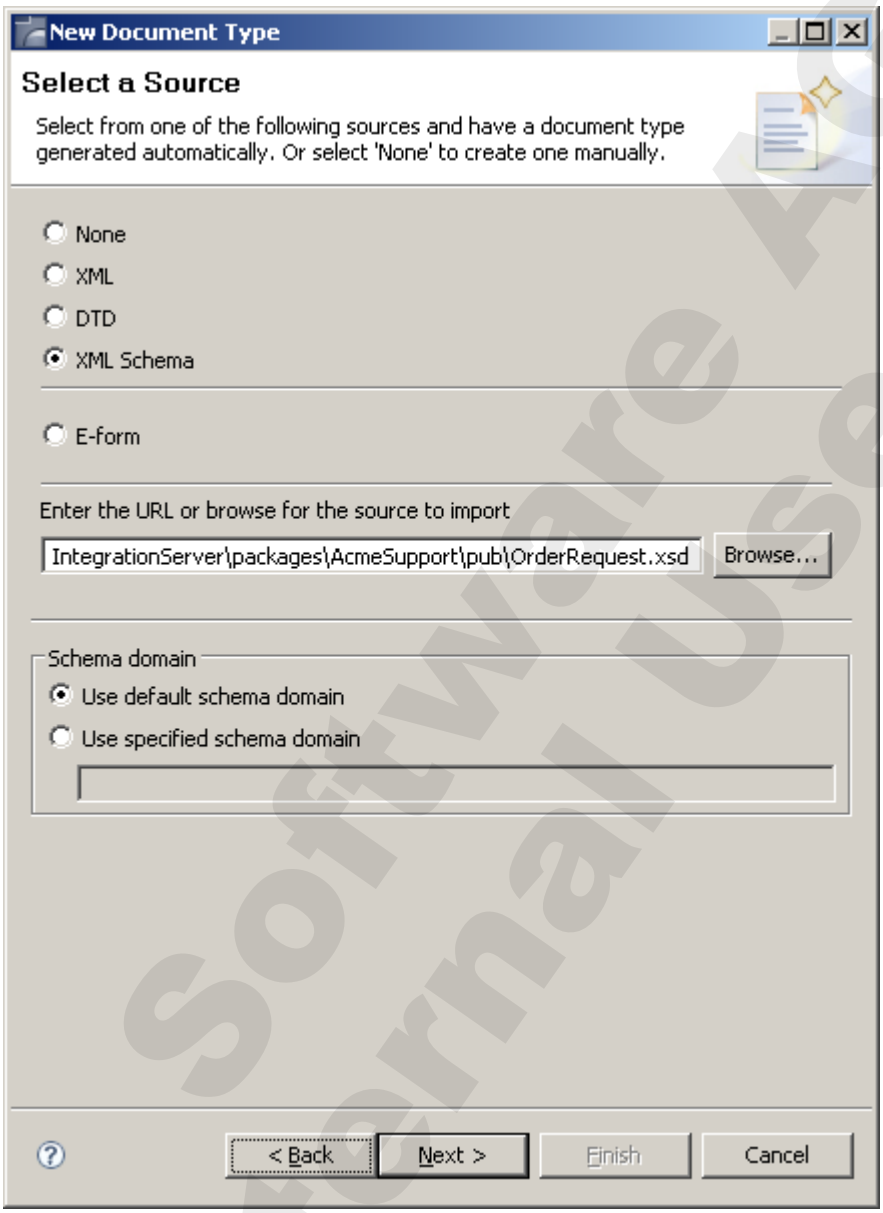
Steps

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

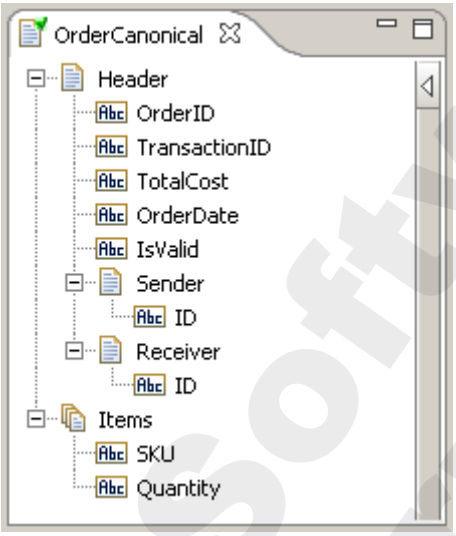
```
<?xml version="1.0" ?>
<!-- webMethods Integration Platform Overview Training Course
Sample XML Message Purchase Order Request
  -->
- <PurchaseOrderRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
  xsi:noNamespaceSchemaLocation="OrderRequest.xsd">
- <PurchaseOrder>
  - <deliverTo>
```

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?
2. What is the benefit of using a schema for import over using a DTD?
3. What are the two ways to indent a document type element under another?

This page intentionally left blank

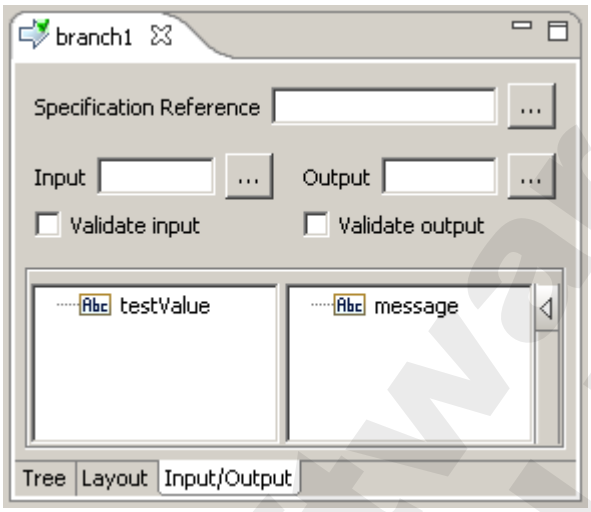
Exercise 5:
Flow Services - BRANCH

Overview

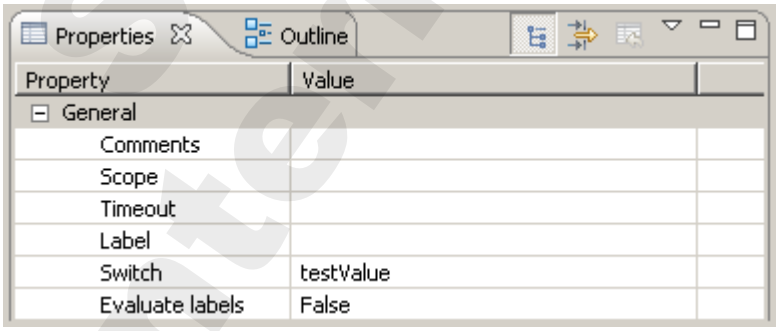
In this exercise, you will create business logic using two different Branch steps: one to test for contents of a variable, and one to evaluate labels associated with logic.

Steps

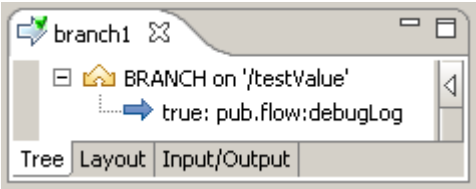
1. In the **acme.PurchaseOrder.work** subfolder**,** create a new Flow service called **branch1.**
2. Define the inputs and outputs of branch1 as input String called **testValue** and output String called **message**.



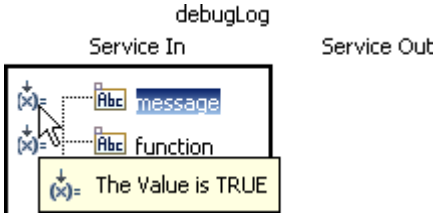
3. In the next steps you will add a **Branch** statement to the flow service that conditionally writes a message to the Server Log, based on the contents of the testValue variable. For example, if testValue = true, write a message saying "The value is TRUE" to the server log.
   a. Add a **Branch** statement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

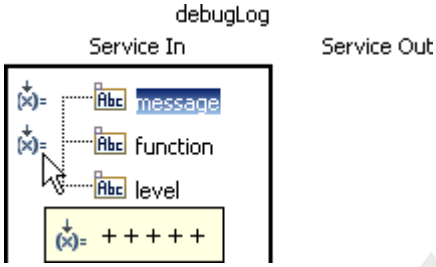


   b. Add a **pub.flow:debugLog** statement below the Branch (be sure to indent it under the Branch so that it becomes part of the Branch logic). In the debugLog properties, set the Label for the service to be **true** (all lower-case).

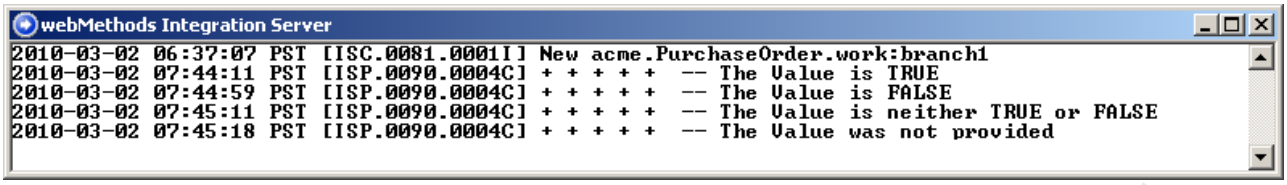


c. On the Pipeline tab for the debugLog statement, set the value of the variable **message** to be "The value is TRUE". As an eyecatcher set the value of **function** to "+ + + + + ". Of course, do not enter the Quotes.





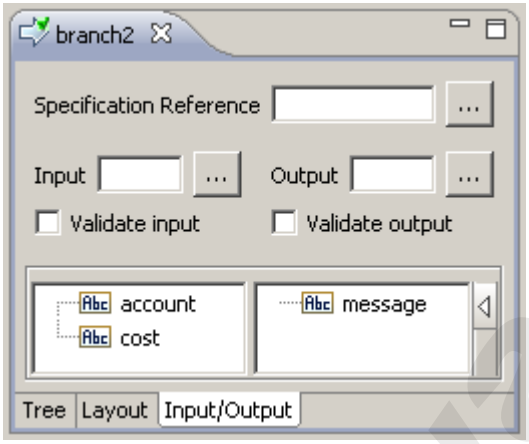
4. In the next section you add three more pub.flow:debugLog statements to your Branch (be sure to indent all of them under the Branch so that they become part of the Branch logic). Set the Label property and the variable message for each of them as follows:
   a. pub.flow:debugLog (already completed 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**.

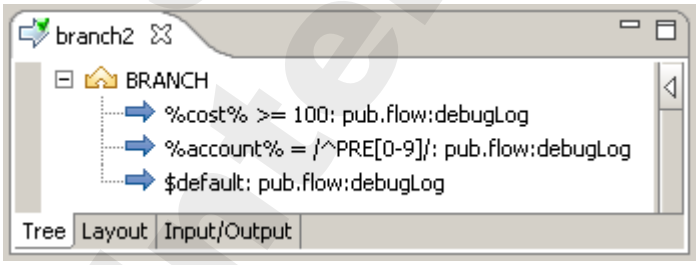


7. In the next section you write **Branch** and **pub.flow:debugLog** code to write a message (based on the value of the input fields) to the Server Log. In this service, we want to evaluate labels, so be sure to leave the Branch **switch** parameter empty and set **Evaluate Labels** to True. The structure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

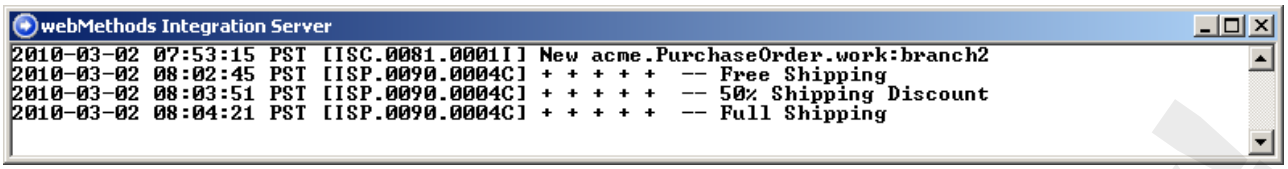
   a. If the contents of variable **cost** are >= **100** then write **Free Shipping** to the server log.
   Note: %cost% >= 100 evaluates the contents of cost at run-time.

   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

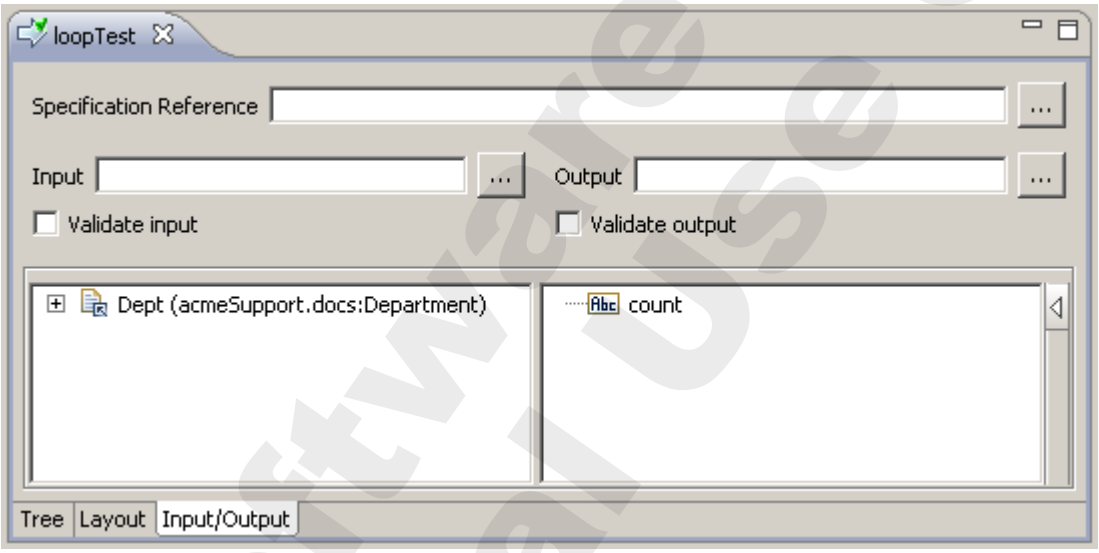
Overview

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

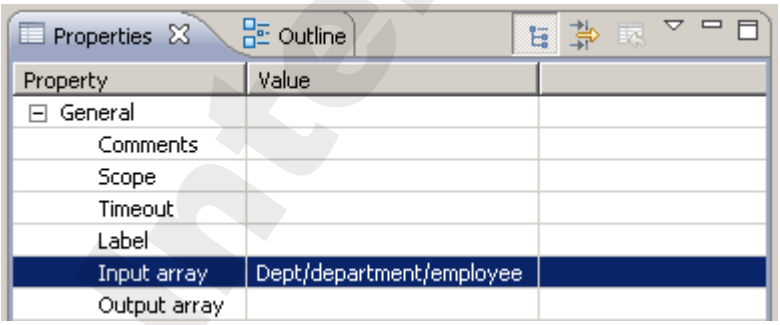
Steps

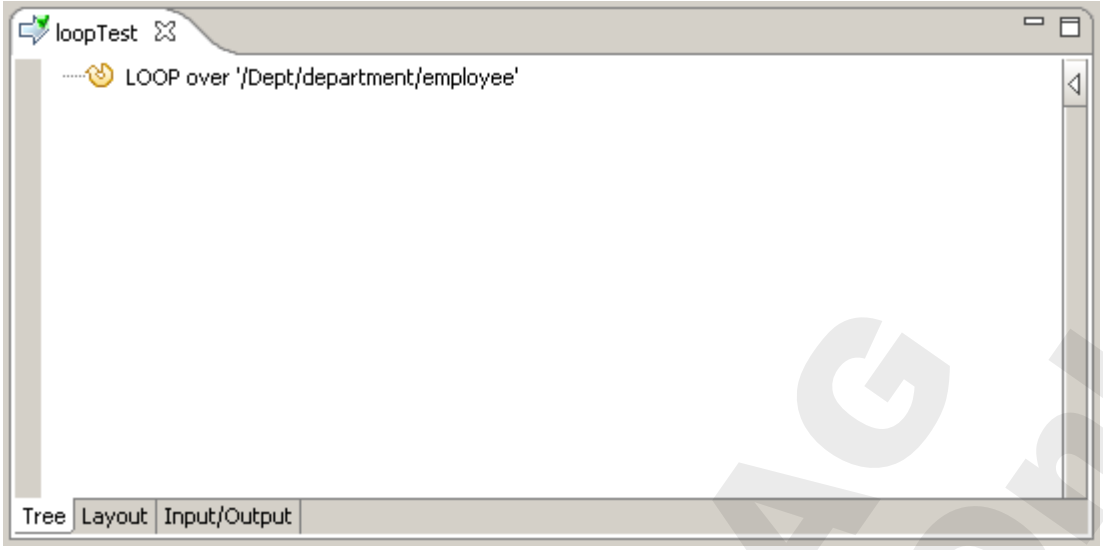
1. In the **acme.PurchaseOrder.work** folder, create a new flow service called **loopTest**. Set the input to be a Document Reference to **acmeSupport.docs:Department** called **Dept**, and set the output to be a single **String** field called **count**.

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

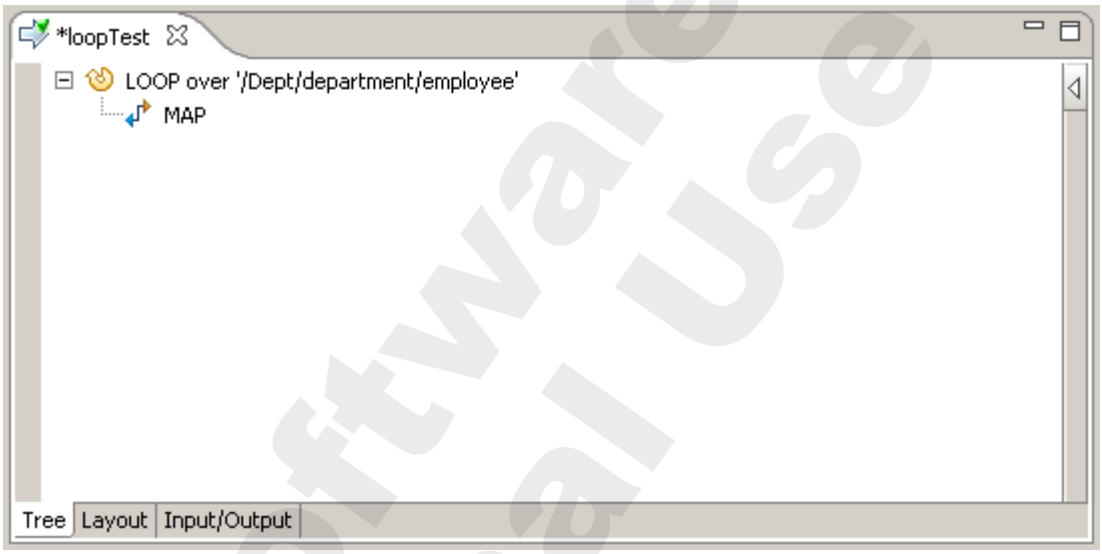


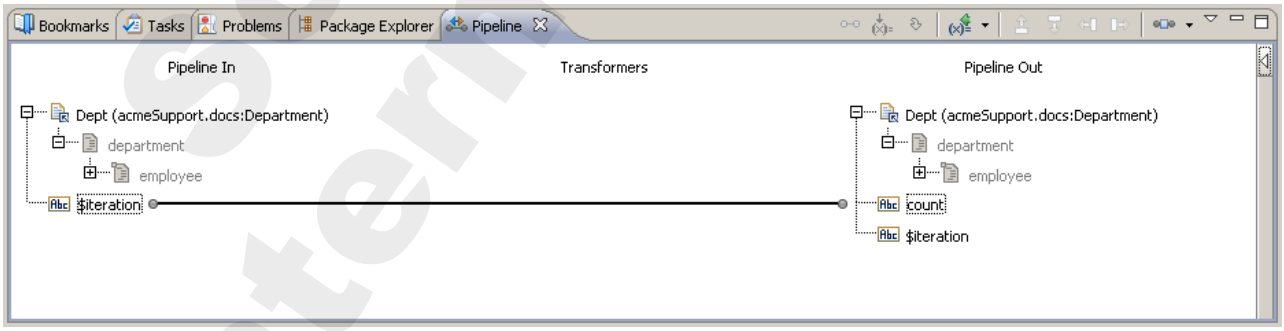
2. On the **Input/Output** tab of your service, expand the **Dept** variable. Find Dept/department/employee and right-click to **Copy**.
3. In your service, add a **LOOP** statement and in the **Input Array** property, paste in Dept/department/employee. Press enter to see the Input array displayed in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



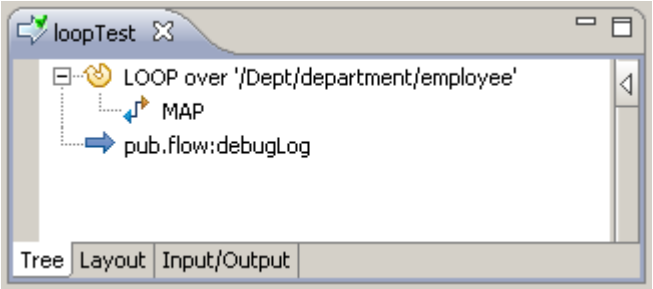


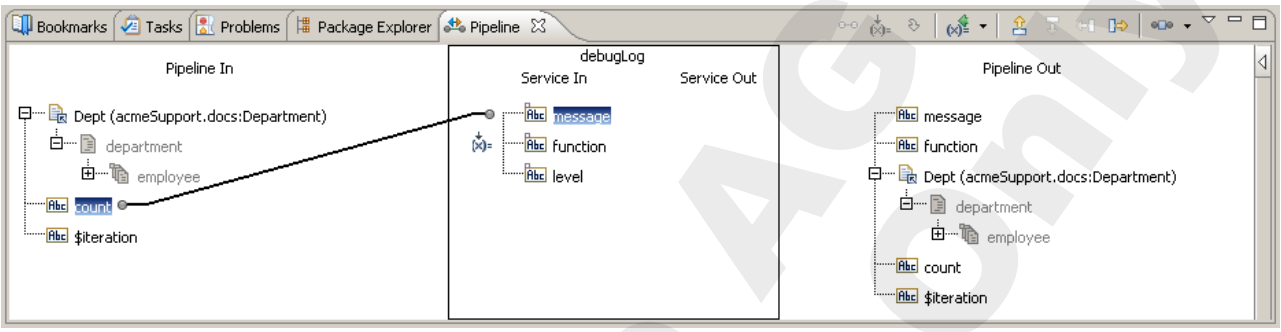
4. Add a **MAP** statement under the Loop step (be sure it is indented under the Loop). Map **$iteration** to **count**.





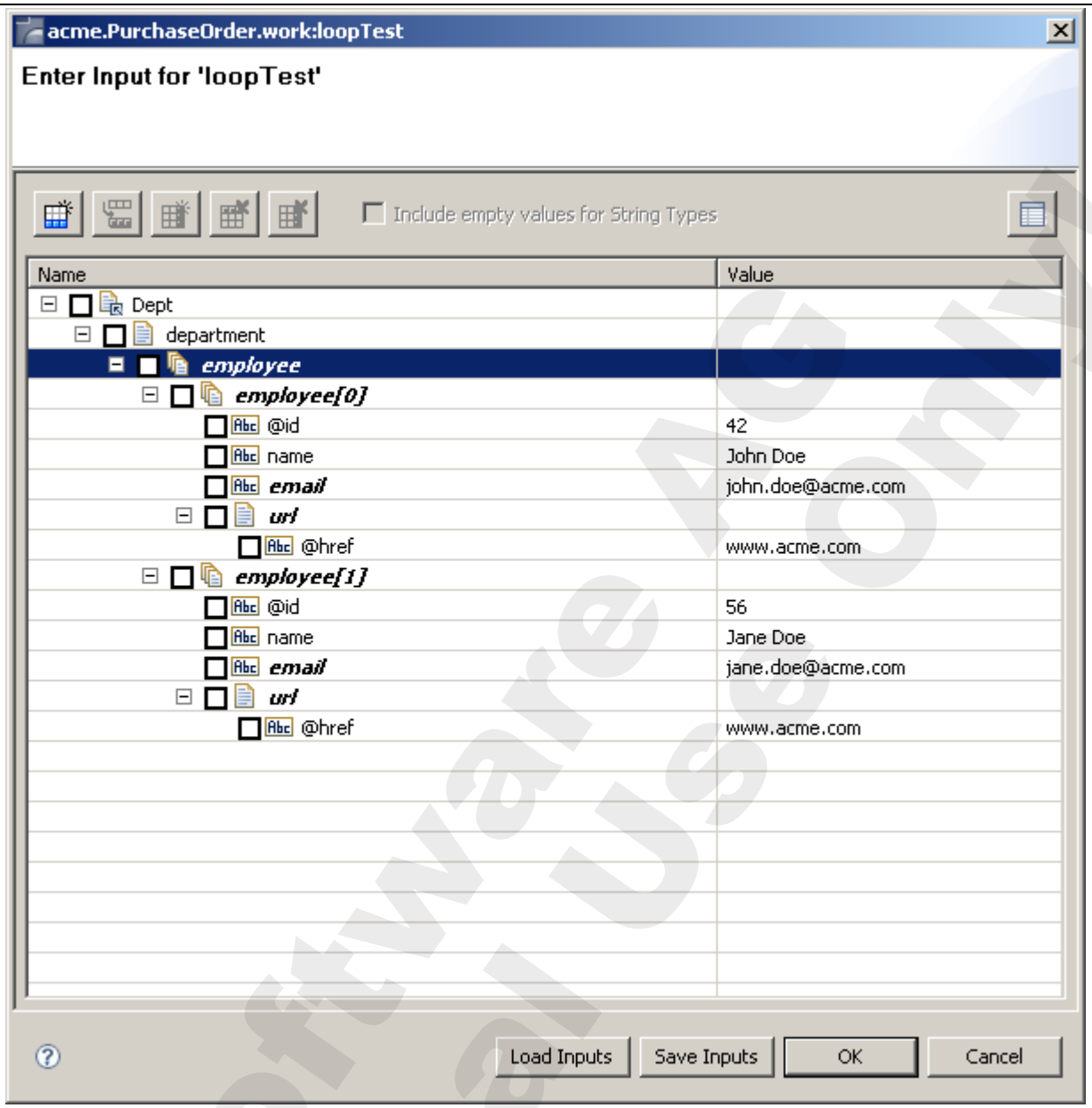
5. Add a **pub.flow:debugLog** below the MAP step (make sure it is NOT indented under the Loop). Map **count** to **message** and set **function** to an eyecatcher like **"+ + + + + ".**





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?

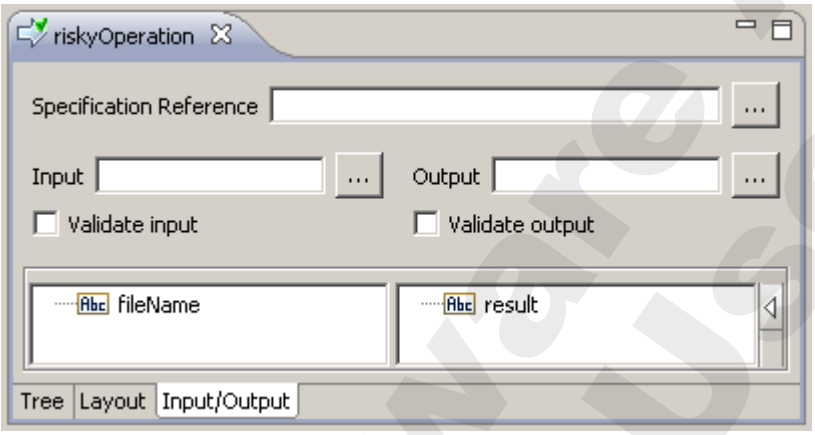
Exercise 7:
Building Flow Services – SEQUENCE

Overview

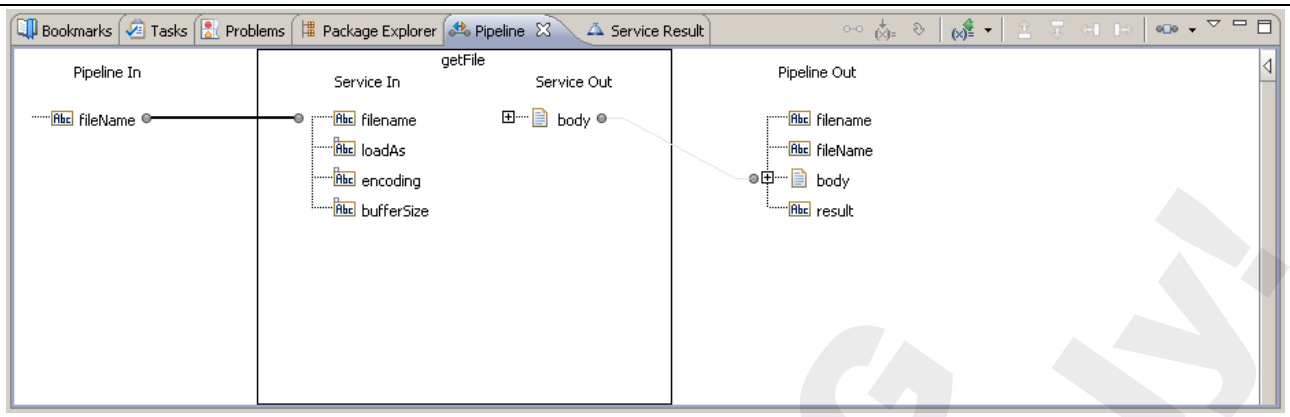
In this exercise, you will create business logic for a Try/Catch using sequence in a Flow service. We will create a service that will return an exception, and then embed that service in a Try/Catch sequence.

Steps

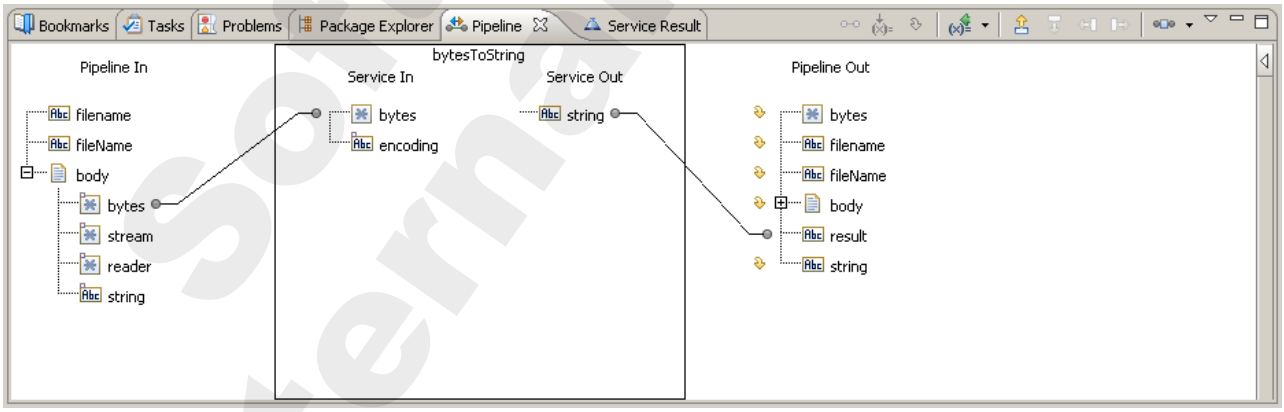
1. In the **acme.PurchaseOrder.work** folder, create a new Flow service called **riskyOperation**. Set the input to be a String called **fileName** and the output to be a String called **result**.



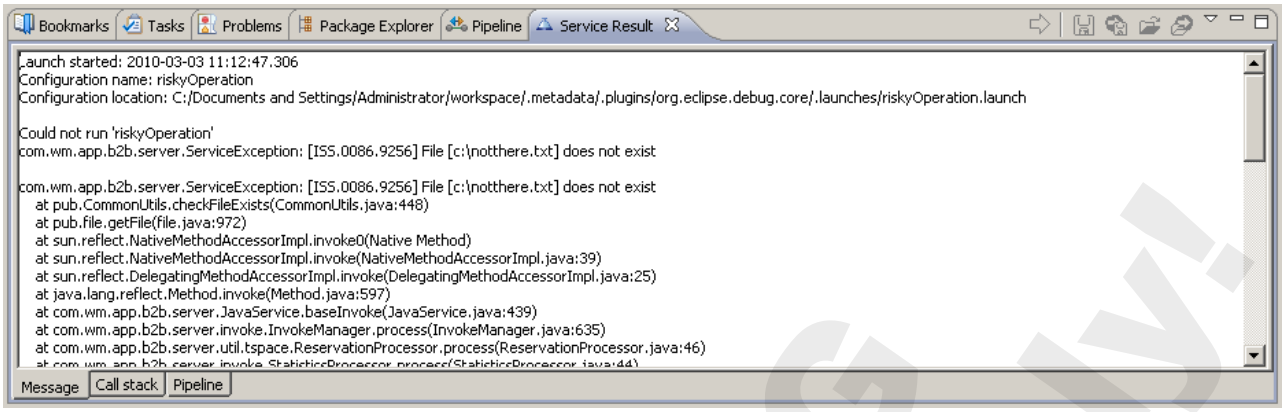
2. Add an invocation of the service **pub.file:getFile** to your service. Map **fileName** to **filename**.

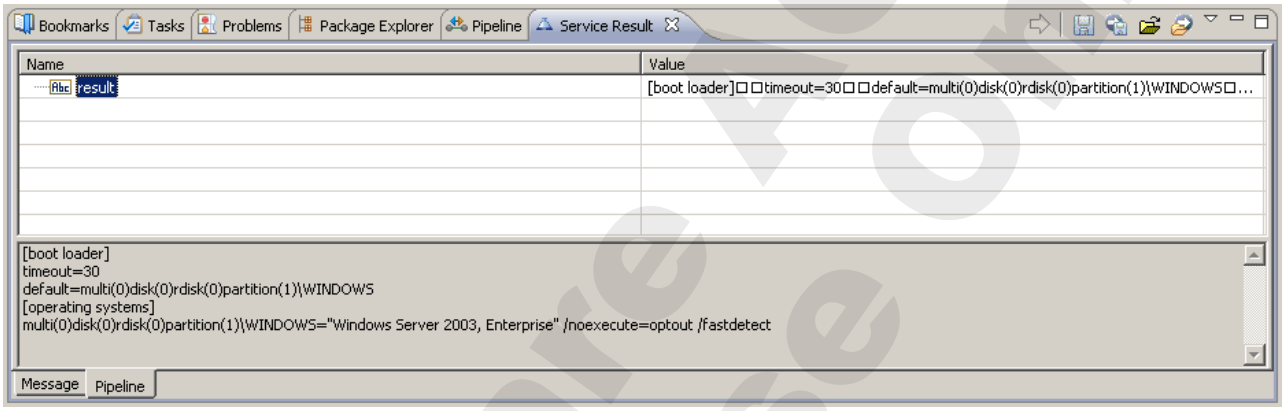


3. Add an invocation of the service **pub.string:bytesToString** to your service.
   a. Map **body/bytes** to **bytes** and **string** to **result**.
   b. Drop all other variables from the Pipeline Out.

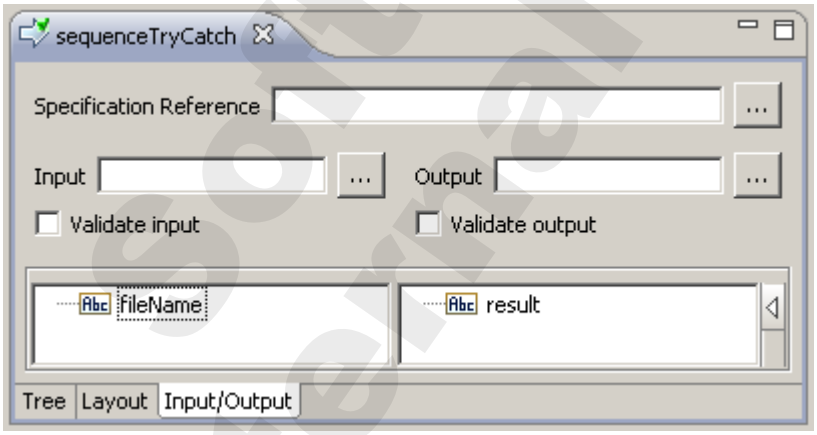


4. Save and run the service. Provide it with a **fileName** of **c:\notthere.txt** (or any other file that does not exist!) What result do you receive? Also try the service with an existing file like **c:\boot.ini.** What result do you receive then?



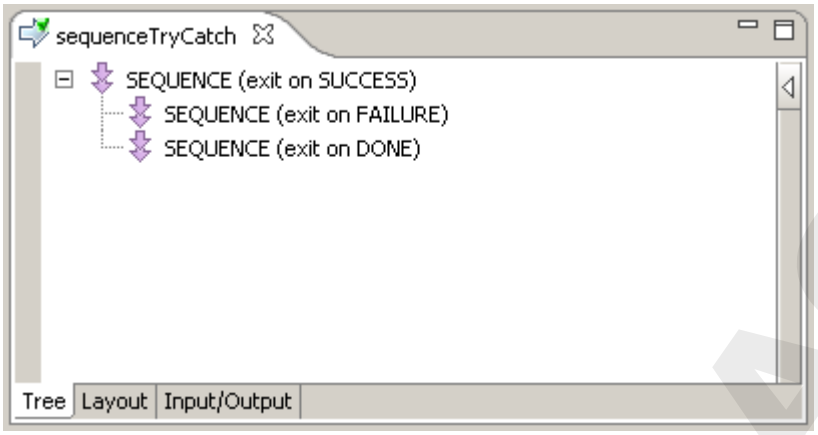


5. Now create another service in the **acme.PurchaseOrder.work** folder called **sequenceTryCatch.** We will use this service to allow us to catch the exception that **riskyOperation** raises if the file name is not correct. Define a single input String for the service called **fileName** and a single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

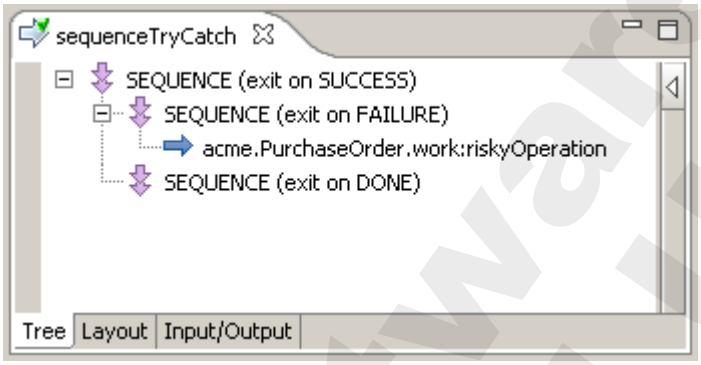


6. In the sequenceTryCatch service, add three **SEQUENCE** steps to create a Flow try/catch block, as follows:
   a. SEQUENCE set to Exit on Success
      i. SEQUENCE set to Exit on Failure (be sure it is indented under the first SEQUENCE)
      ii. SEQUENCE set to Exit on Done (be sure it is indented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

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



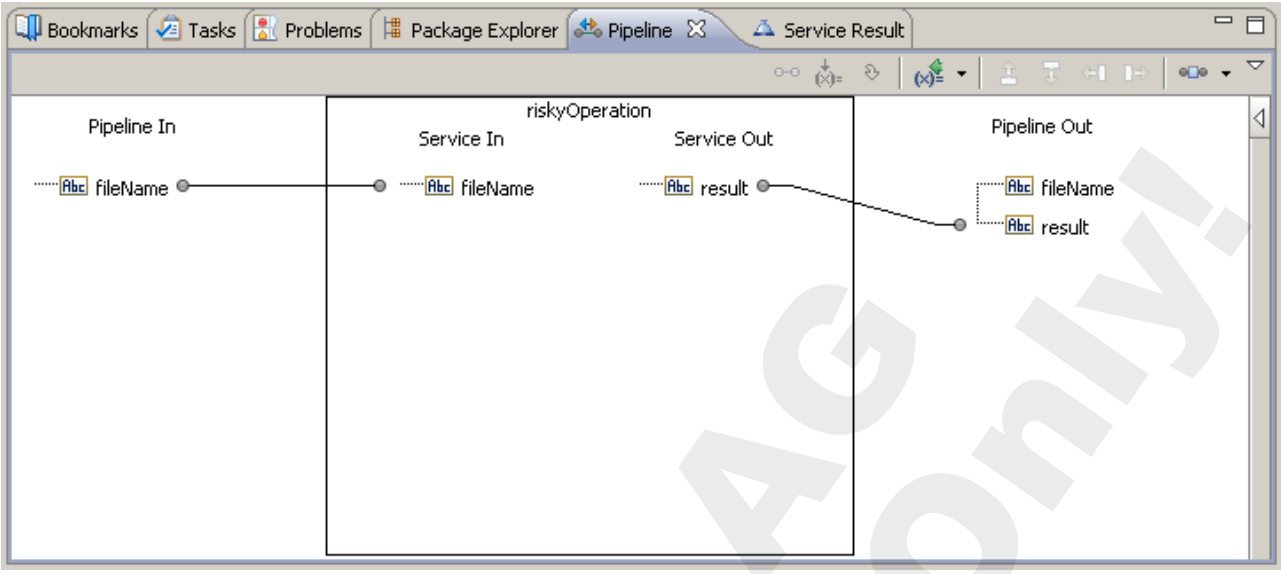
7. In the SEQUENCE Exit on Failure, add the service **acme.PurchaseOrder.work:riskyOperation** (be sure it is indented under the SEQUENCE Exit on Failure).



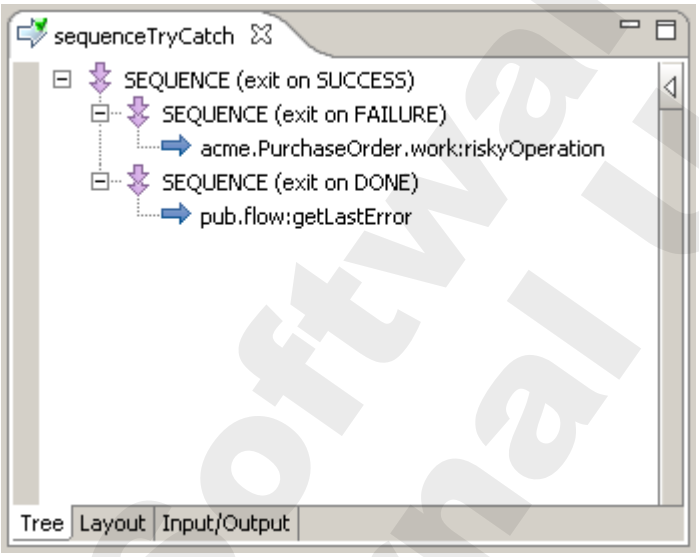
On the Pipeline tab, map as follows:

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

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

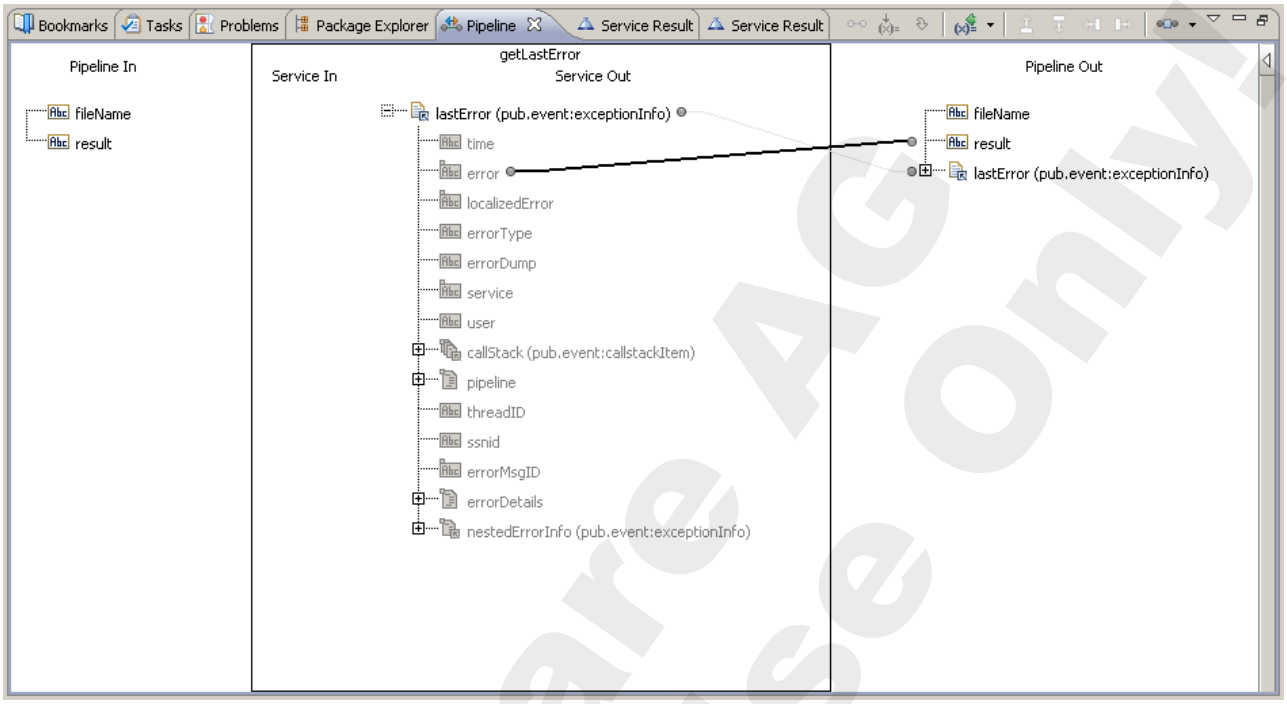


8. In the SEQUENCE Exit on Done, add the service **pub.flow:getLastError** (be sure it is indented under the SEQUENCE Exit on Done).

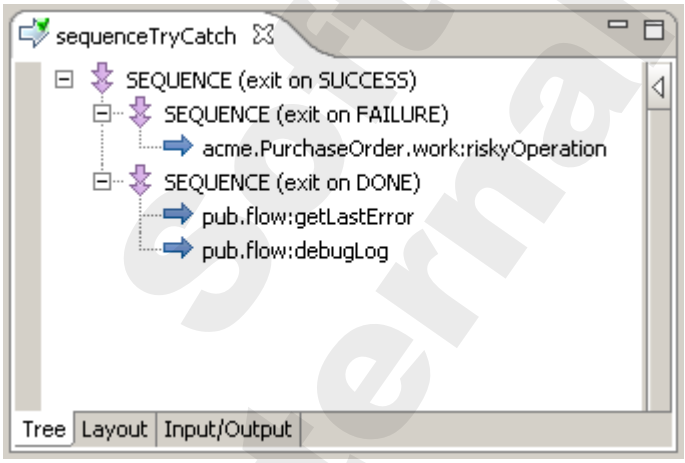


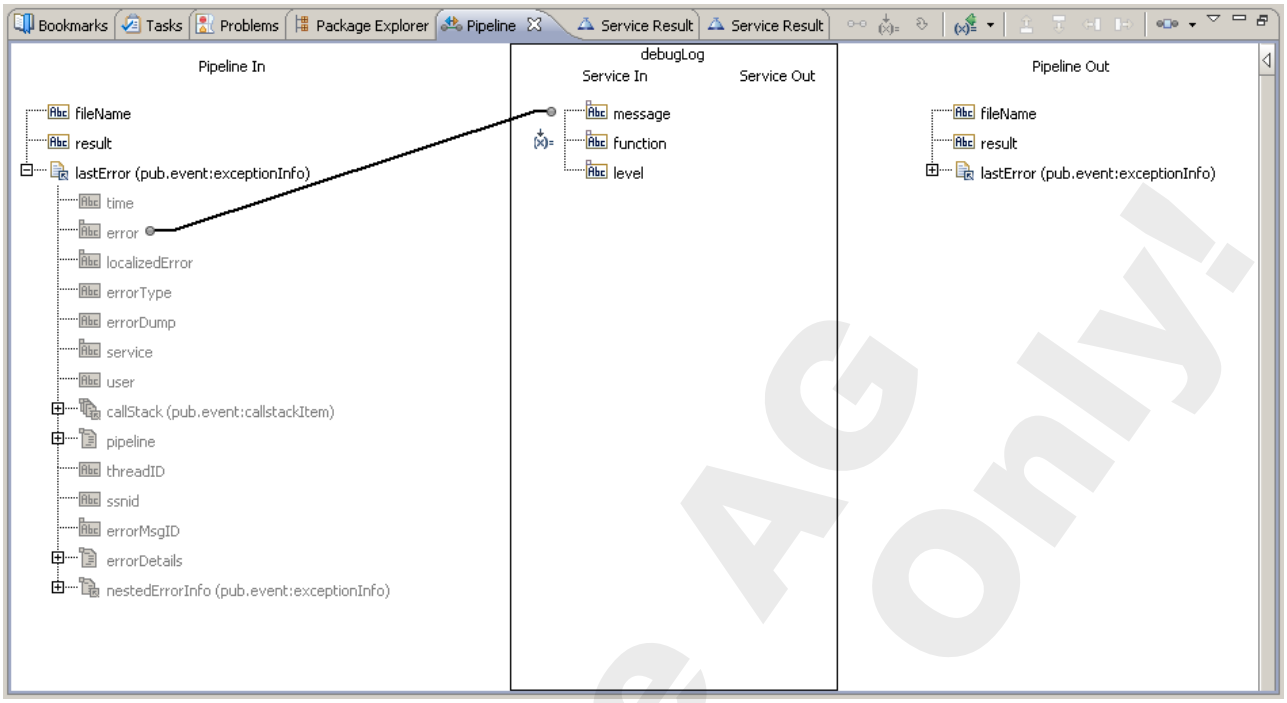
On the Pipeline tab, map as follows:

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

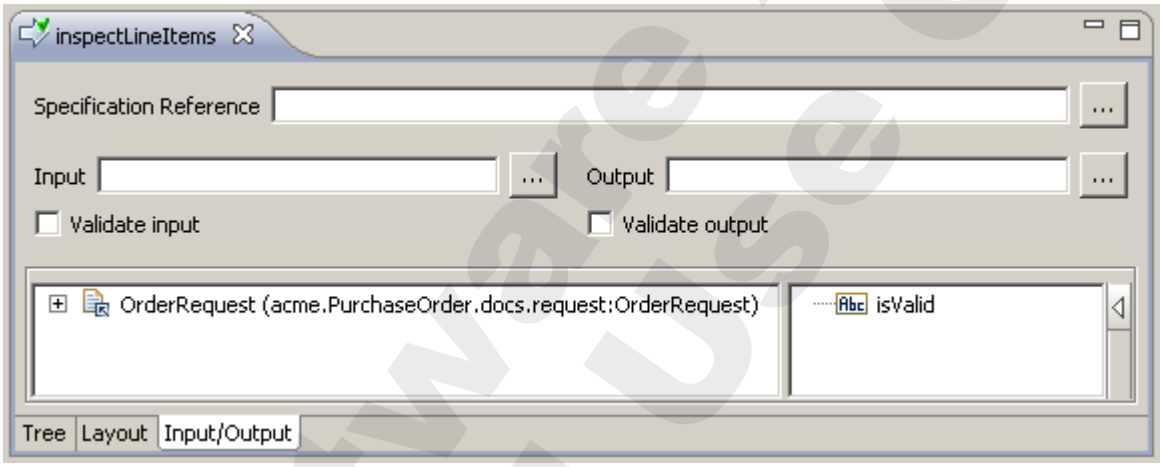
Exercise 8:
Validation Service

Overview

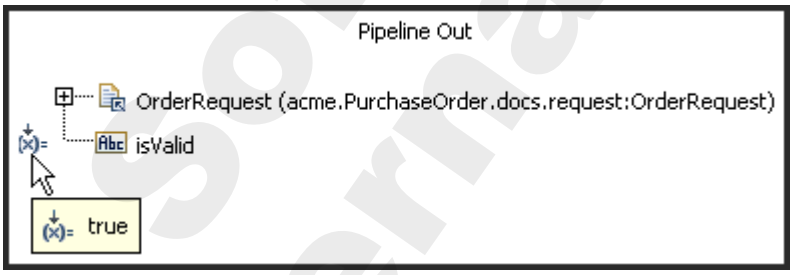
In this exercise, you will create business logic to validate an inbound Purchase Order. For now, this means to verify that the line items in the Purchase Order have a valid quantity. If this is not the case, you will flag the order as invalid for follow up by a customer service representative.

Steps

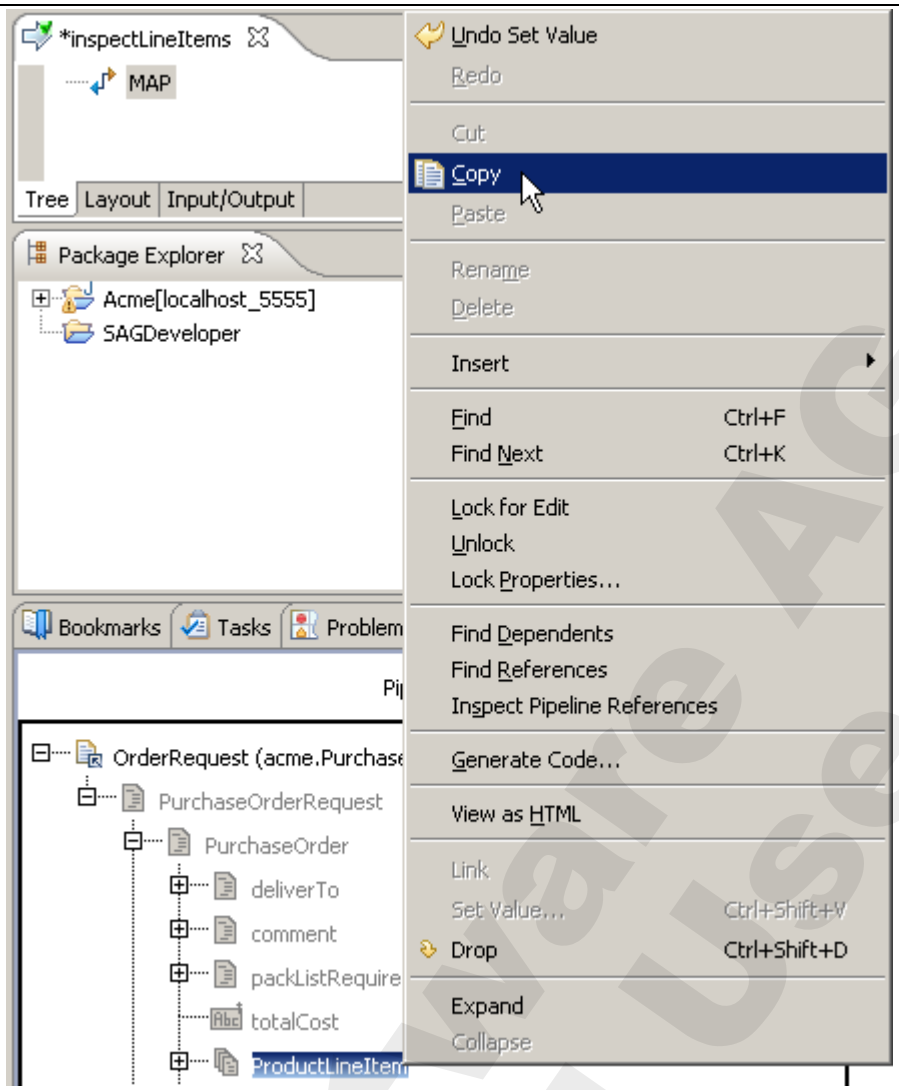
1. In the **acme.PurchaseOrder.utils** folder, create a service called **inspectLineItems.** For input, specify a document reference to **acme.PurchaseOrder.docs.request:OrderRequest.** Call this input Variable **OrderRequest**. As output variable, create a String called **isValid**.

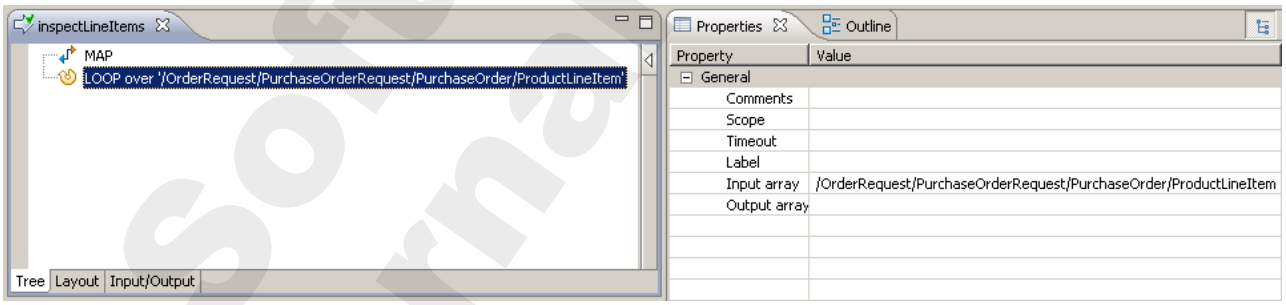


2. In the service, add a MAP step to initialize the **isValid** variable to **true**.



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.





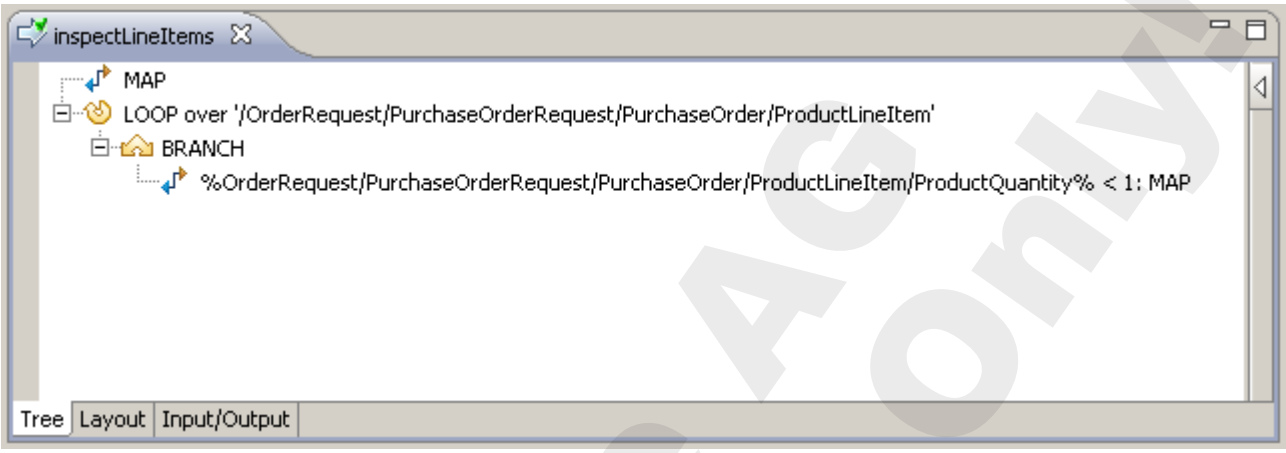
4. Under the LOOP step in your service, add a **BRANCH** step (be sure it is indented under the LOOP). Leave the **Switch** property empty and set **Evaluate labels** = **True**.

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

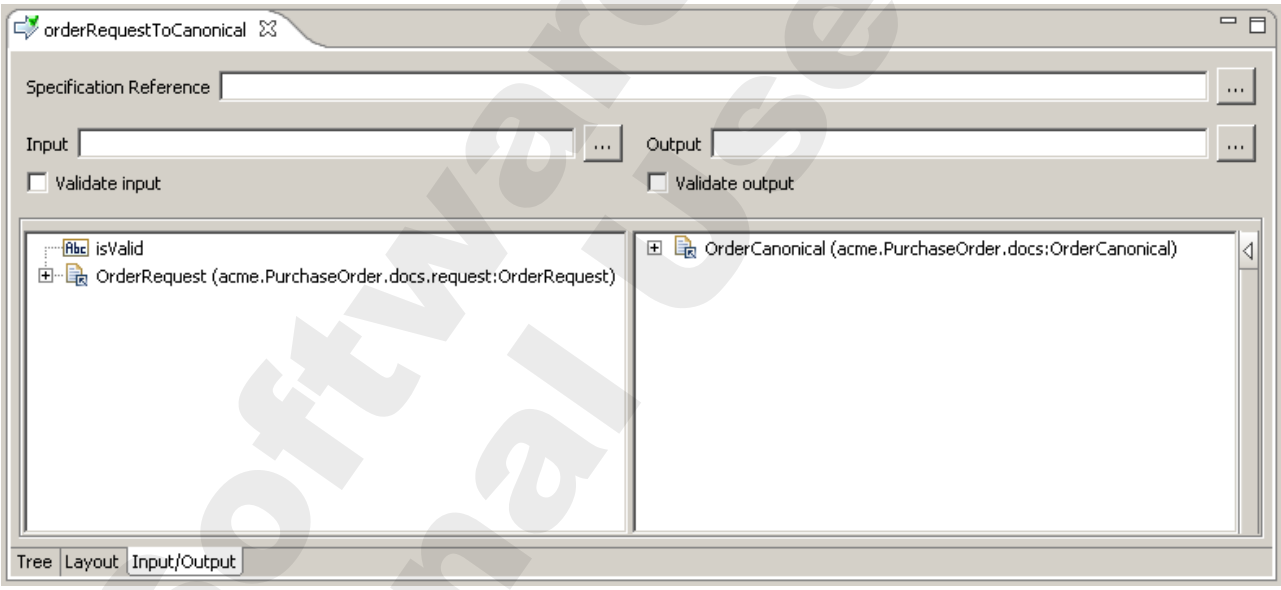
Exercise 9:
Mapping Service

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**

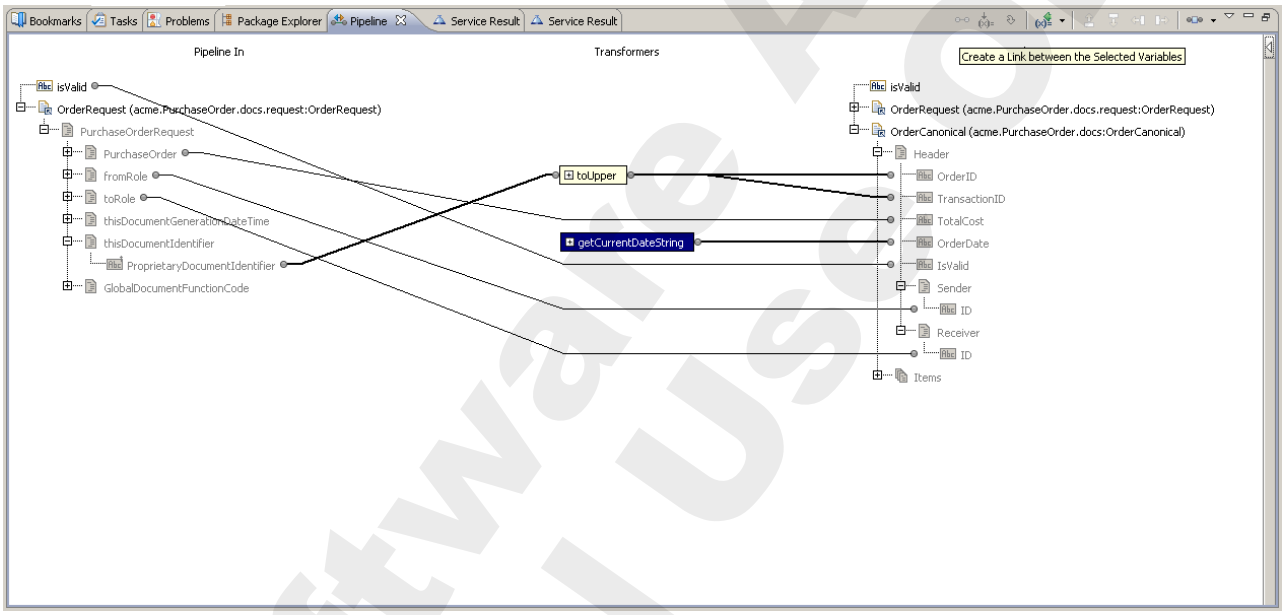
   b. The transformer value to **OrderCanonical/Header/OrderID**

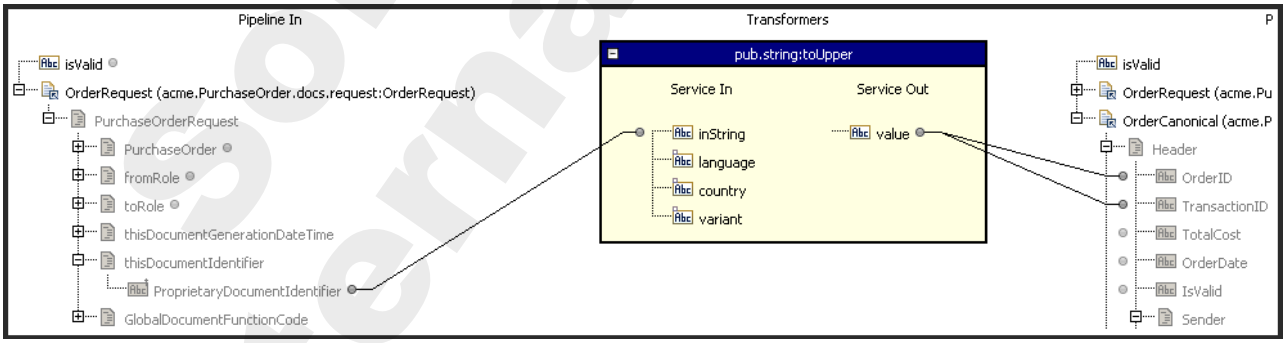
   c. The transformer value to **OrderCanonical/Header/TransactionID**

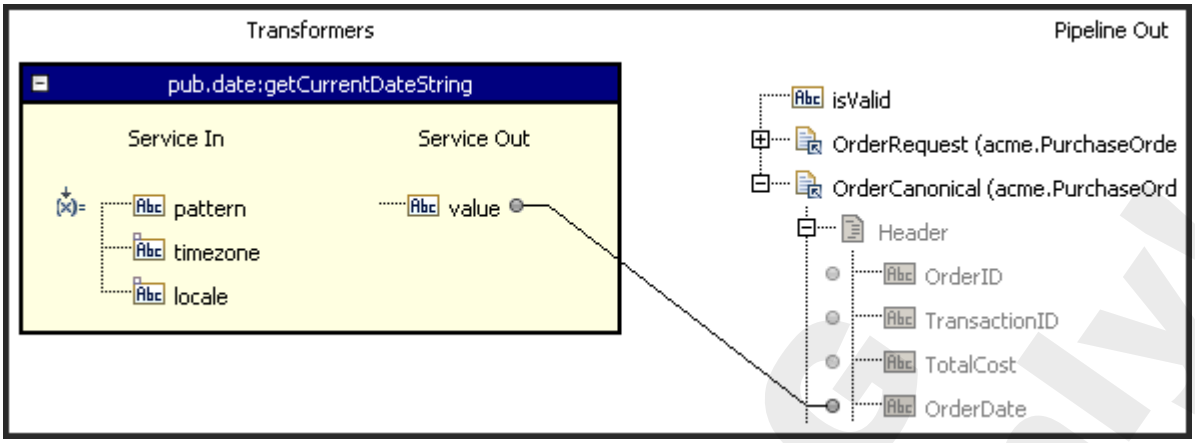
4. Add the service **pub.date:getCurrentDateString** as a transformer in your MAP step. Map the following variables in the transformer and the OrderCanonical document.

   a. Set the transformer **pattern** to **MMMM dd, yyyy**

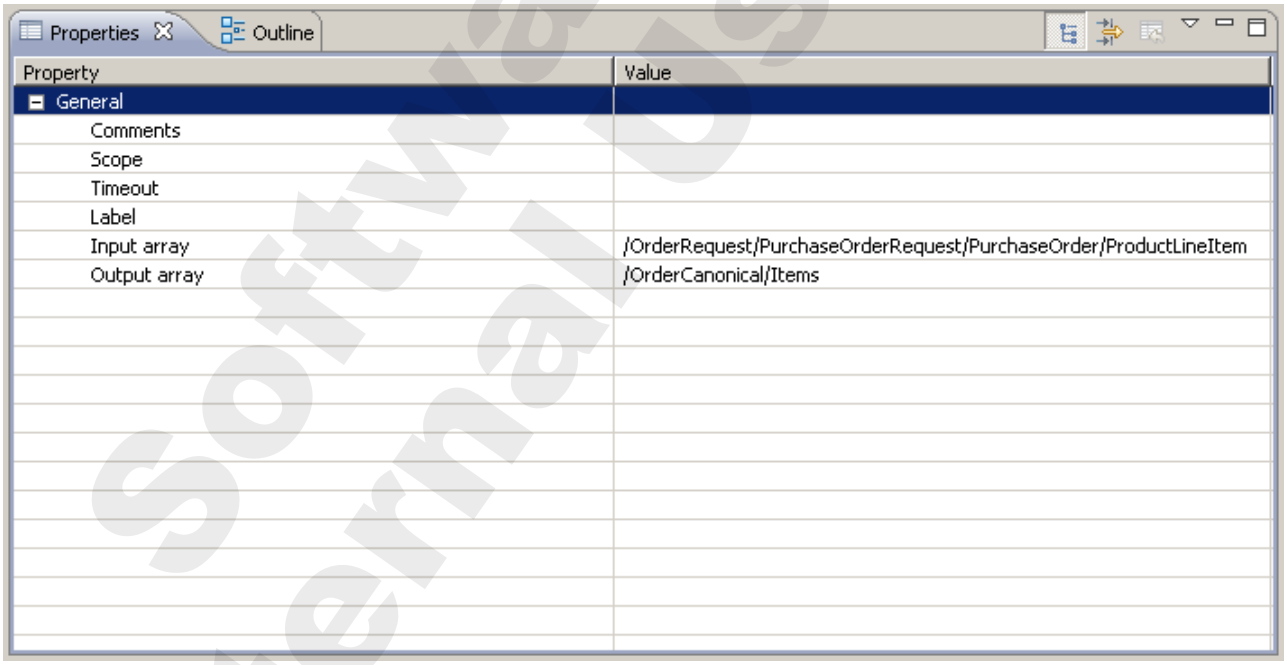
   b. Map the transformer **value** to **OrderCanonical/Header/OrderDate**



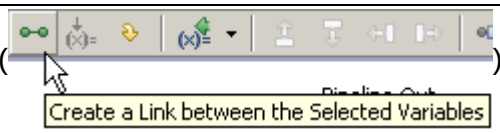




5. Add a **LOOP** step to the service. Set the input array to be **/OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem**. Set the output array to be **/OrderCanonical/Items.**

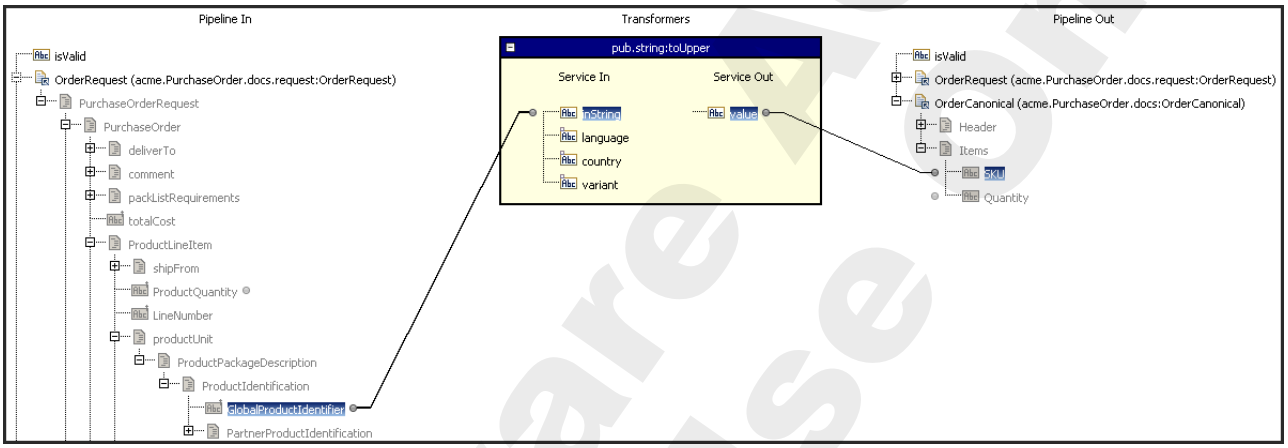


6. Add a **MAP** step in the LOOP (be sure it is indented under the LOOP statement). Map **/OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/ProductQuantity** from the **OrderRequest** variable to the **OrderCanonical**'s variable **/OrderCanonical/Items/Quantity** member.
   Hint: When 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..."

(

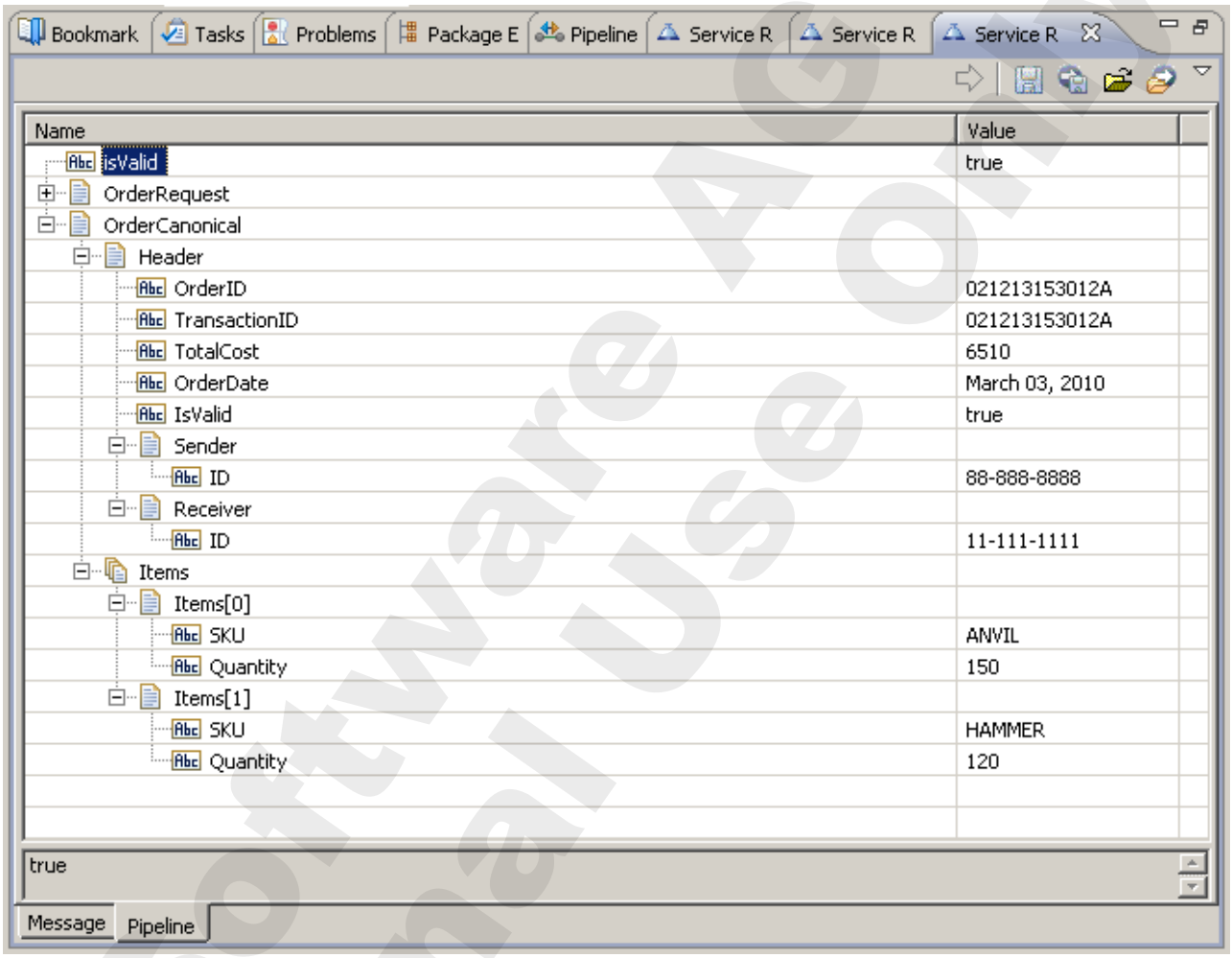
) button. This button is located in the titlebar of the pipeline window.

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?

This page intentionally left blank

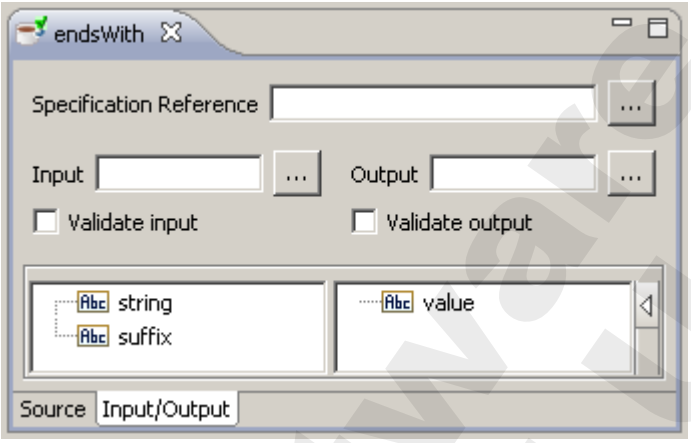
Exercise 10:
Create a Java Service

Overview

In this exercise, you will create, compile, and run a Java Service using Designer. Imagine that you require a special service that tests if a string ends in a second string. There is no such service in the pub.string folder, but you want to use the String.endsWith() method in the Java runtime environment.

Steps

1. Create a new Java service called **endsWith** in the **acme.PurchaseOrder.work** folder. This service has two string inputs, called **string** and **suffix**.



2. Enter the following code for your service:

```
IDataCursor cursor = pipeline.getCursor();

String string = IDataUtil.getString(cursor, "string");
String suffix = IDataUtil.getString(cursor, "suffix");

String value = string.endsWith(suffix) ? "true" : "false";

IDataUtil.put(cursor, "value", value);

cursor.destroy();
```

   Note: All Java development features, like code completion, that you are used to from the Eclipse IDE are available.

3. Run your service with some sample input values and verify that the returned values are correct.

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?

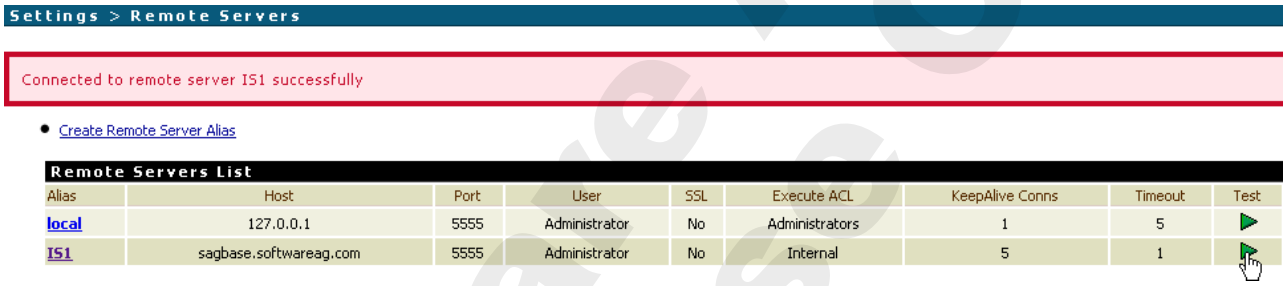
Exercise 11: Monitoring Services

Overview

In this Exercise, you will use My webMethods to track the execution of services.

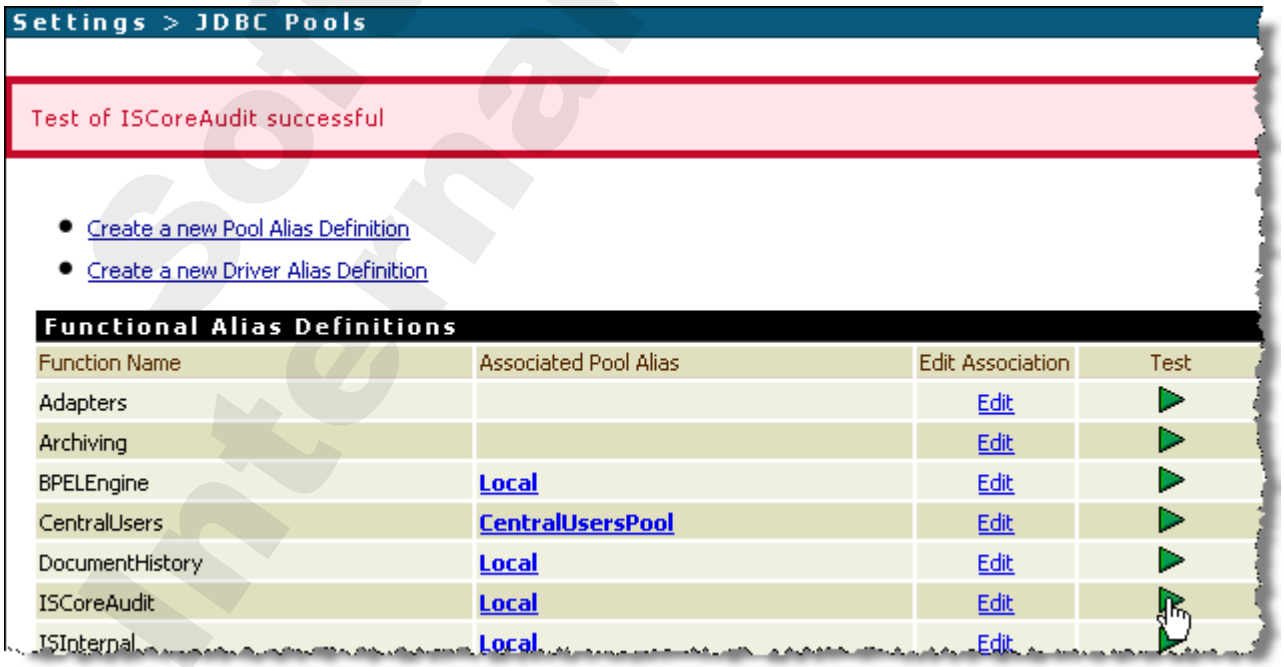
Steps

1. Open the IS Administrator console (*http://localhost:5555*) and log in as "Administrator" using a password of "manage". In the "**Settings**" menu select **Remote Servers** and verify that there is an entry for IS1 in the **Remote Servers List**. Test the IS1 alias by clicking on its Test icon (▶).



Alias	Host	Port	User	SSL	Execute ACL	KeepAlive Conns	Timeout	Test
local	127.0.0.1	5555	Administrator	No	Administrators	1	5	▶
IS1	sagbase.softwareag.com	5555	Administrator	No	Internal	5	1	▶

2. Go to the Settings ➡ JDBC Pools entry area in the Administrator console and confirm that a **JDBC Pool Alias** named **Local** is **Associated** with **ISCoreAudit, ISInternal, ProcessAudit** and **ProcessEngine** Functional Aliases. Click on the **Test** button (▶) to the right of the **ISCoreAudit** Functional Alias in order to confirm your connection to the database is working.



Function Name	Associated Pool Alias	Edit Association	Test
Adapters		Edit	▶
Archiving		Edit	▶
BPELEngine	Local	Edit	▶
CentralUsers	CentralUsersPool	Edit	▶
DocumentHistory	Local	Edit	▶
ISCoreAudit	Local	Edit	▶
ISInternal	Local	Edit	▶

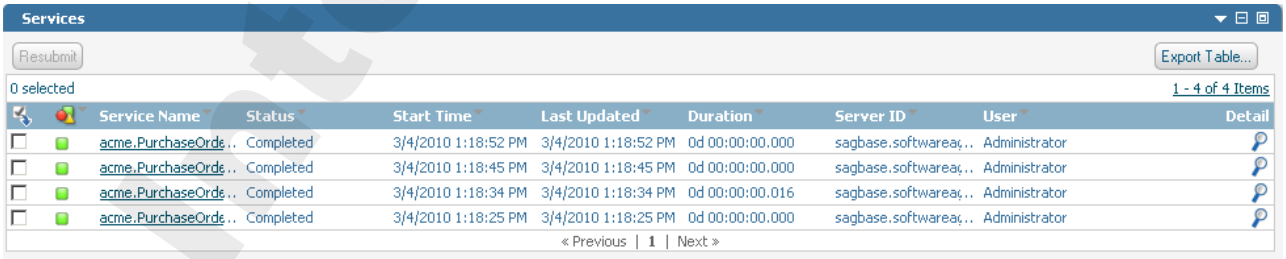
3. If you changed any values in the previous 2 steps, then restart the Integration Server. If you did not change any values, continue without restarting.
4. In Designer, open the service **acme.PurchaseOrder.work:branch1**. In the Service Properties, enable Audit Logging. Set the Audit properties as follows:
   a. Enable Auditing = Always
   b. Log on = Error and Success
   c. Include Pipeline = Always

Audit	
Enable auditing	Always
Log on	Error and success
Include pipeline	Always

5. Save the **branch1** service and run it several times. Provide different input strings during each run.
6. Open the My webMethods console (http://localhost:8585) in a browser and log in as **Administrator | manage**.
7. Note: When using the My webMethods console for the first time on a freshly started MWS, the response times can be inappropriately long. This is caused by the fact that MWS has to load a lot of Java classes when they are referenced for the first time. Please be patient and do not start to click on arbitrary buttons to get some response from the system.

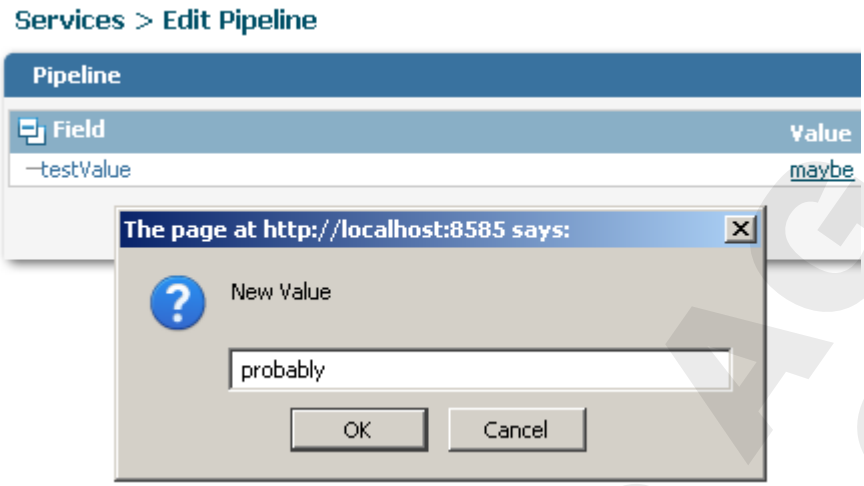
   In the navigation bar on the left side select Applications ➡ Monitoring ➡ Integration ➡ Services.

   Find the acme.PurchaseOrder.work:branch1 service in the **Services** result list.

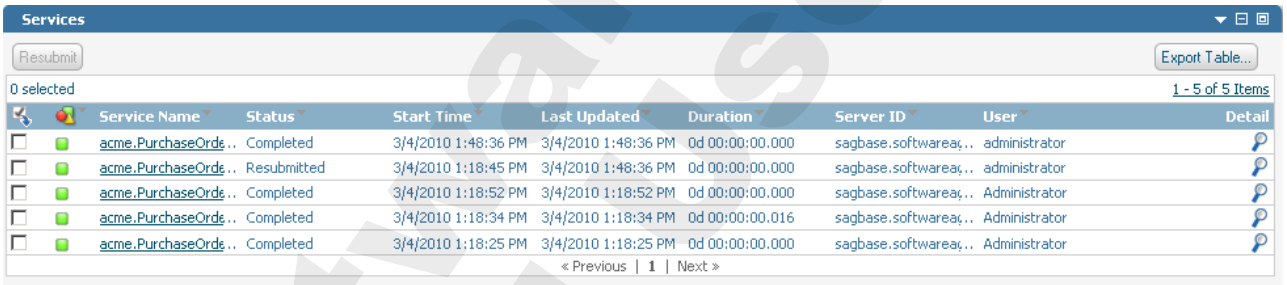


Click on the **View Details** button ( ). The statistics about this individual service execution will be displayed.

8. Use the "**Edit Pipeline"** button and change the field **testValue**.



Click the OK and **Save** buttons. Finally click on the **Resubmit** button and you will see a "Resubmitted" entry in the **Service Information** tab.



Verify that the service resubmission is also shown in the Server Log file:

Check Your Understanding

1. Why is it necessary to create remote server aliases?
2. Under what circumstances would it be acceptable to resubmit a service? Why?

This page intentionally left blank

Exercise 12:
Invoking Services

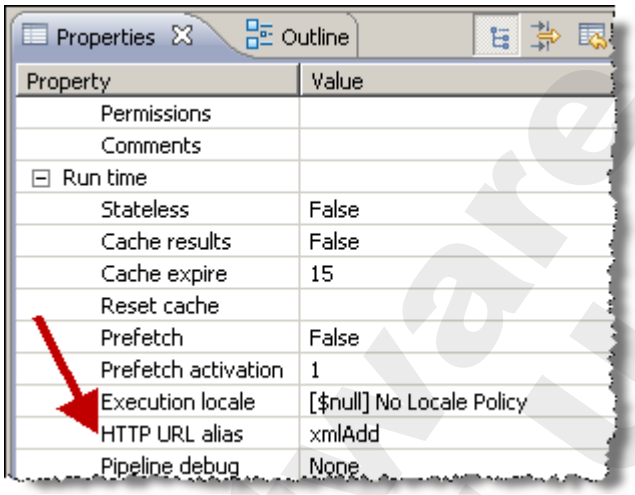
Overview

In this Exercise, you will use different ways to invoke a service.

Steps

1. Invoke a service using HTTP.

   a. To invoke a service using HTTP open designer and find the xmlAdd service in the acmeSupport.xml package. Find it's "**HTTP URL alias"** property and set it to **xmlAdd**.



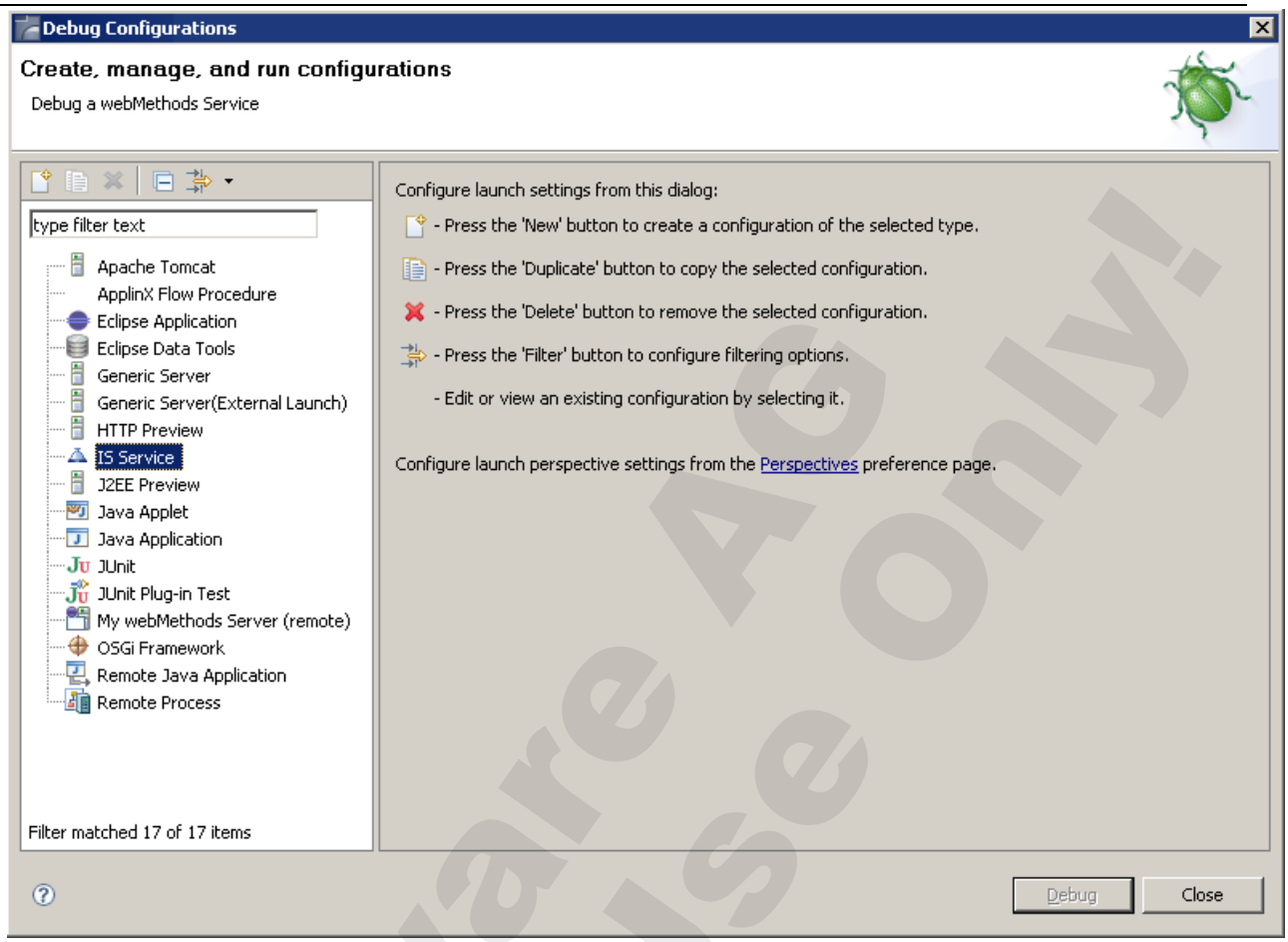
   b. Save the service.

   c. Open a browser and visit the URL "**http://localhost:5555/xmlAdd? a=12&b=23"**. Now open the alternative URL "**http://localhost:5555/invoke/acmeSupport.xml/xmlAdd? a=12&b=23"**. Compare the two results for differences.

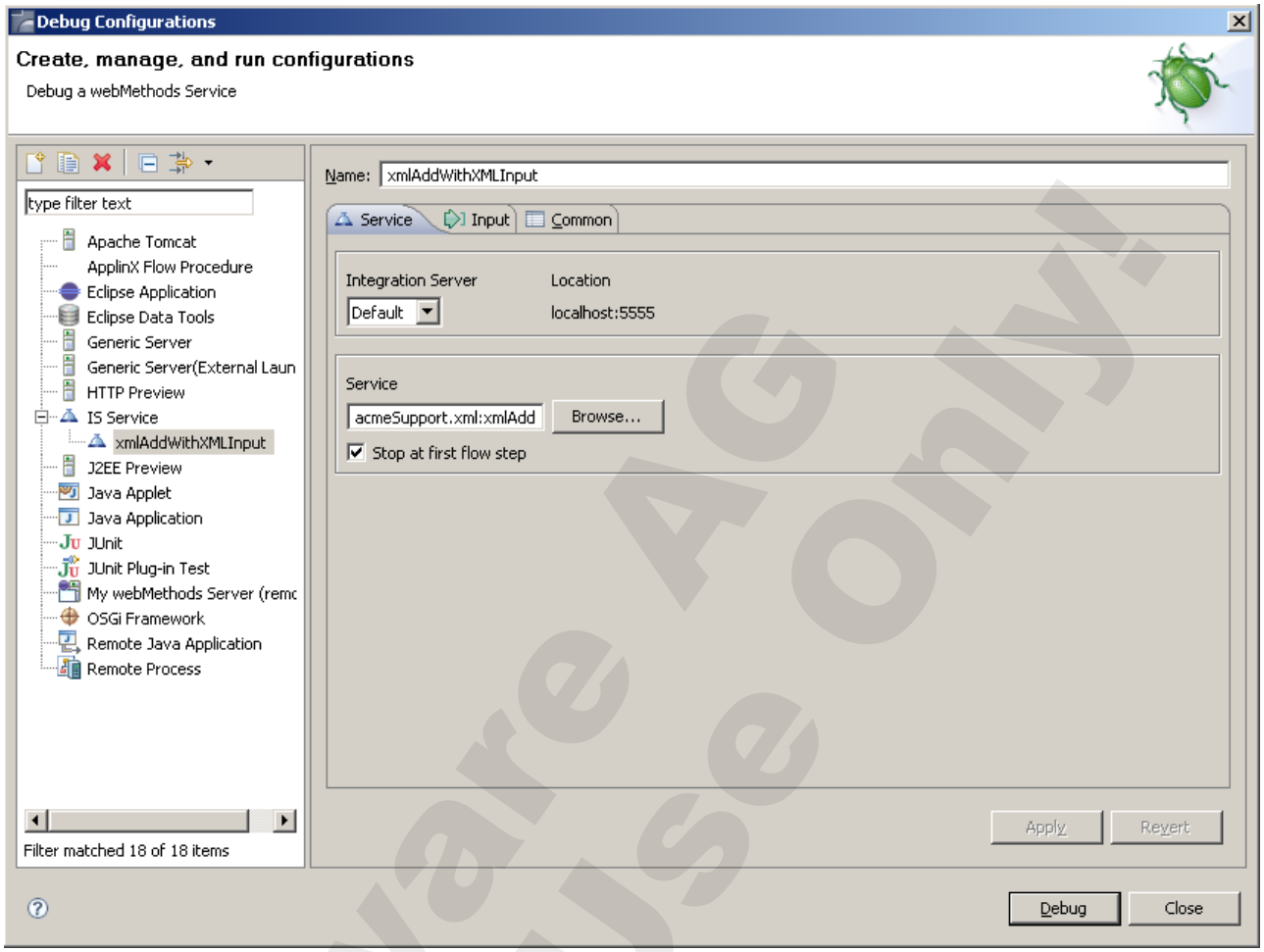
aList	12
bList	23
value	35

2. Invoke a service with XML input

   a. Right click the **acmeSupport.xml:xmlAdd service** and select **Debug as ➡ Debug Configurations**. In the upcoming Dialogue double click on IS Service:

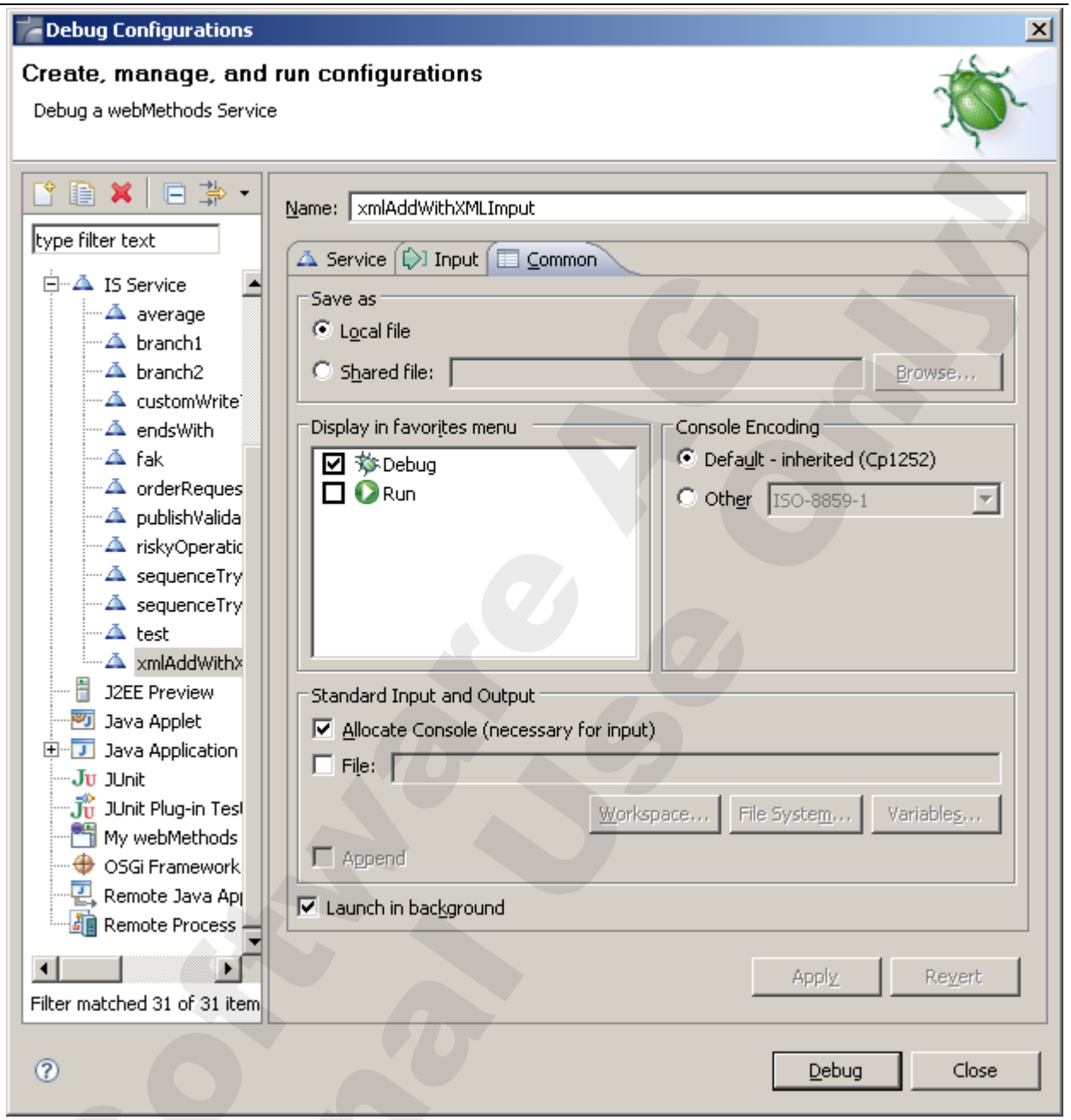


b. Click the browse Button and select the **acmeSupport.xml:xmlAdd** service. Hit OK. Then change the name of your launch configuration from New_configuration to a meaningful name like "**xmlAddWithXMLInput**" and hit Apply.

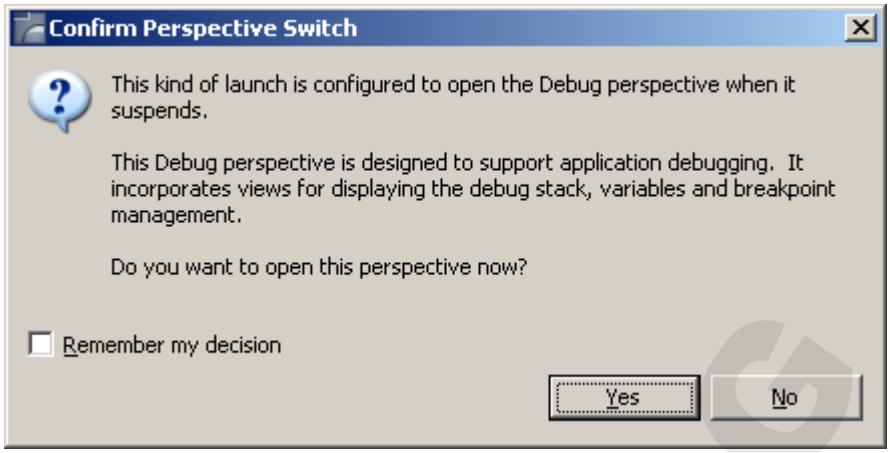


c. Now choose the Input tab and select the "**Use XML**" radio button. Click browse and navigate to the XML File **...\IntegrationServer\packages\AcmeSupport\pub\addInput.xml and hit** Open**.** Then click **Apply** again.

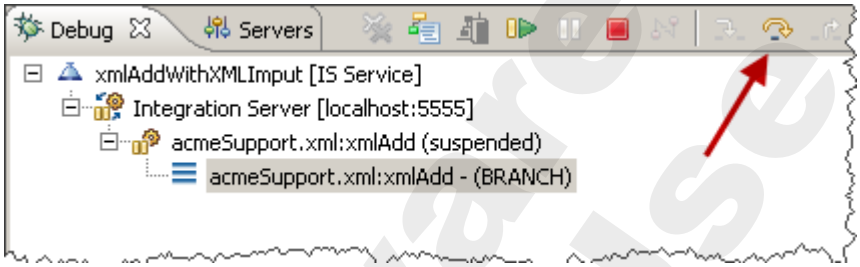
d. Now select the common tab and check the checkbox "Debug" in the "Display in favorites menu". Hit apply again and then click on Debug.



Confirm the dialog asking about a Perspective switch by clicking **Yes**, where you optionally can suppress further appearances of this dioalogue by checking the "Remember my decision" checkbox.



e. Now the debugger comes up and you are debugging the Service with a single input variable node of type Object already in the pipeline. Step through the service to see how this gets converted and added.

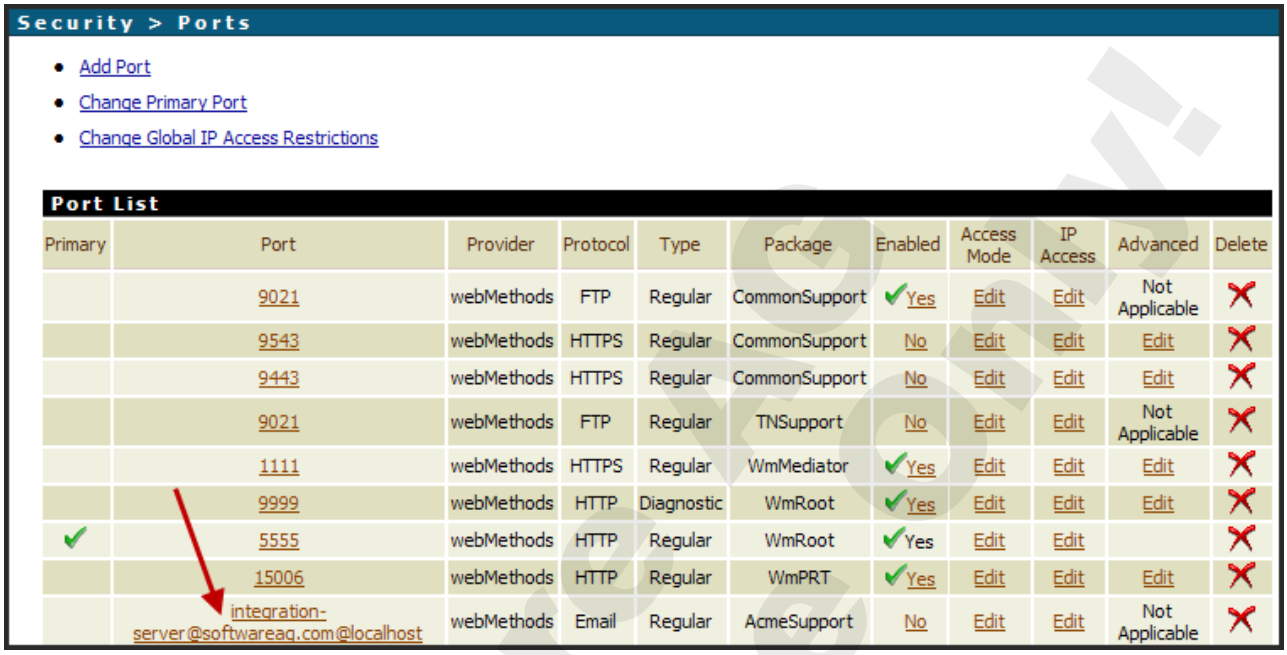


3. Invoke a service using SMTP (mail)

a. Open designer and inspect the document **acmeSupport.xml:addDocument**. You should find that it contains a simple root node containing two variables called a and b.

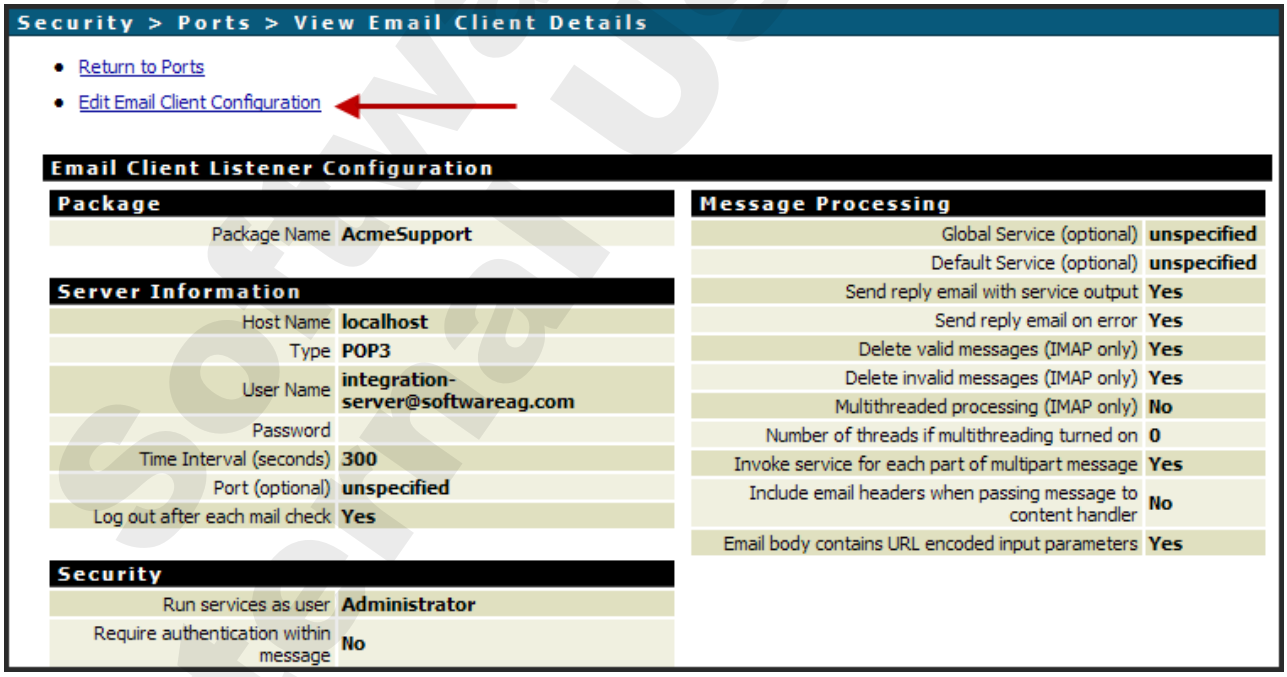
b. Now inspect the xmlAdd service in the same folder and find out what it is doing. Also have a look at the content of the file **...\IntegrationServer\packages\AcmeSupport\pub\addInput.xml**. Given this file as input to the xmlAdd service, what result would you expect?

c. Enable the Email port in Integration server, which is turned off by default. To do so, open the Integration server administration console and got to the Security ➡ Ports menu. Click on the integration-server@softwareag.com@localhost entry

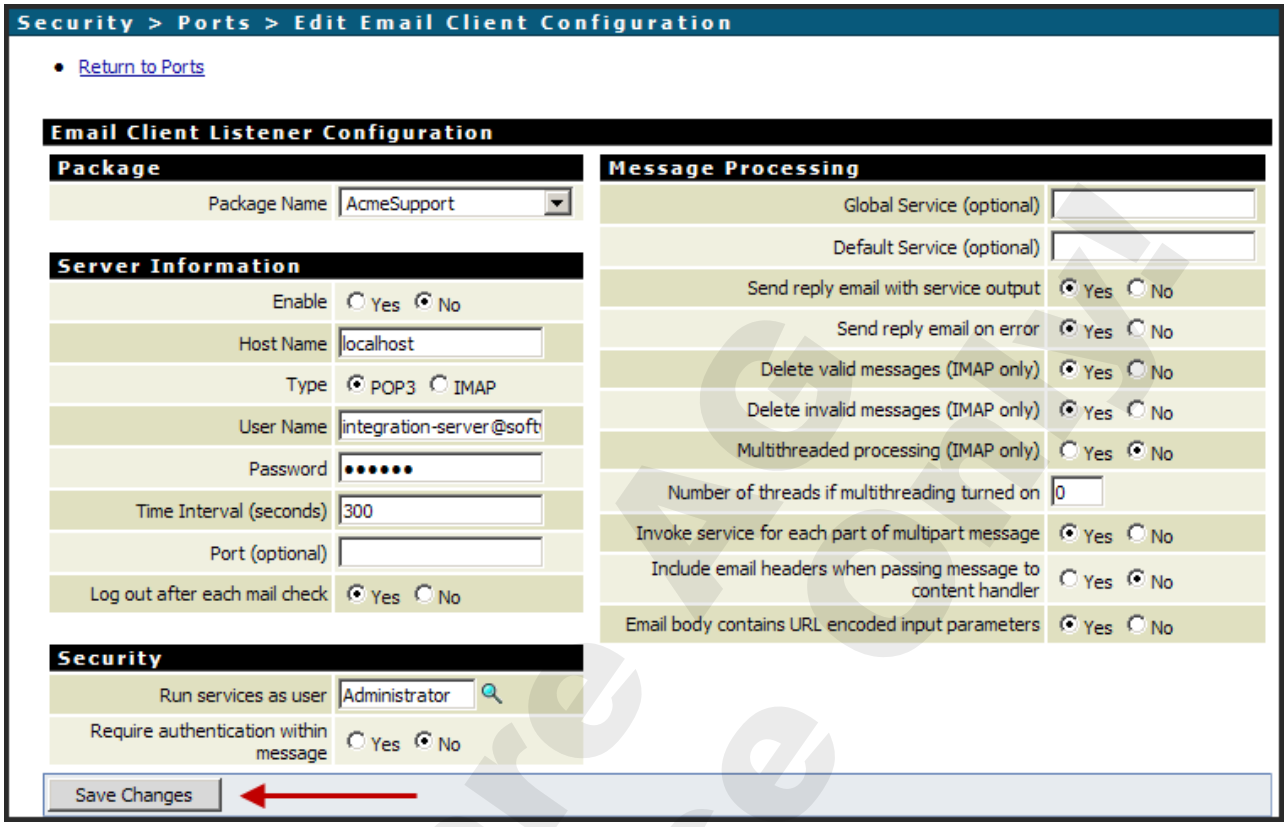


Primary	Port	Provider	Protocol	Type	Package	Enabled	Access Mode	IP Access	Advanced	Delete
	9021	webMethods	FTP	Regular	CommonSupport	✔ Yes	Edit	Edit	Not Applicable	✘
	9543	webMethods	HTTPS	Regular	CommonSupport	No	Edit	Edit	Edit	✘
	9443	webMethods	HTTPS	Regular	CommonSupport	No	Edit	Edit	Edit	✘
	9021	webMethods	FTP	Regular	TNSupport	No	Edit	Edit	Not Applicable	✘
	1111	webMethods	HTTPS	Regular	WmMediator	✔ Yes	Edit	Edit	Edit	✘
	9999	webMethods	HTTP	Diagnostic	WmRoot	✔ Yes	Edit	Edit	Edit	✘
✔	5555	webMethods	HTTP	Regular	WmRoot	✔ Yes	Edit	Edit		✘
	15006	webMethods	HTTP	Regular	WmPRT	✔ Yes	Edit	Edit	Edit	✘
	integration-server@softwareag.com@localhost	webMethods	Email	Regular	AcmeSupport	No	Edit	Edit	Not Applicable	✘

and choose edit email client configuration

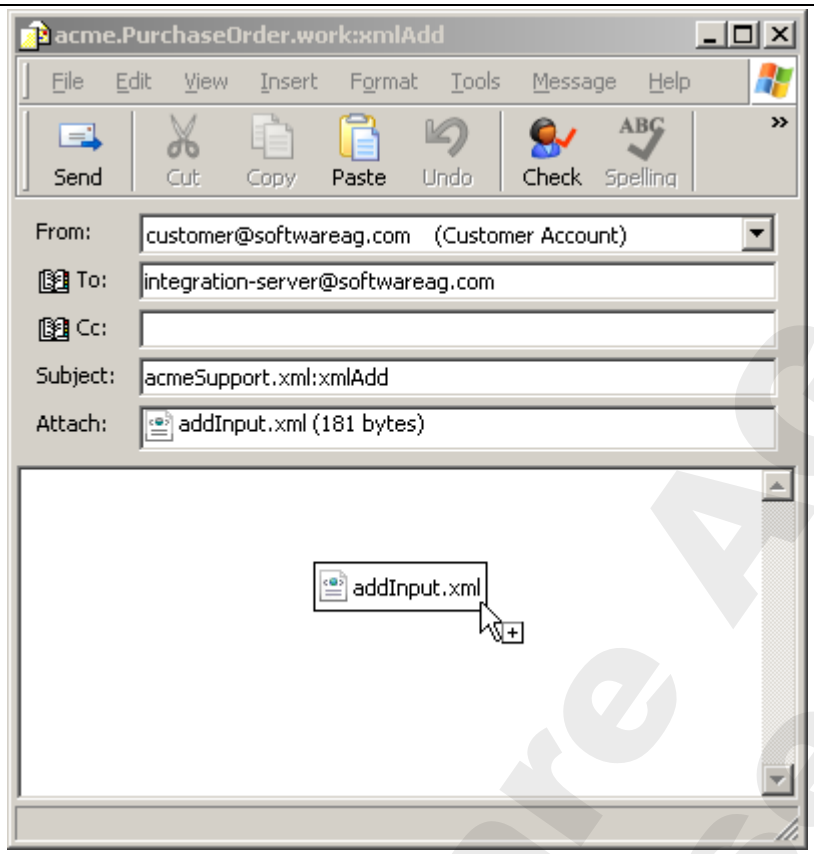


enter the password of **manage** and click save changes

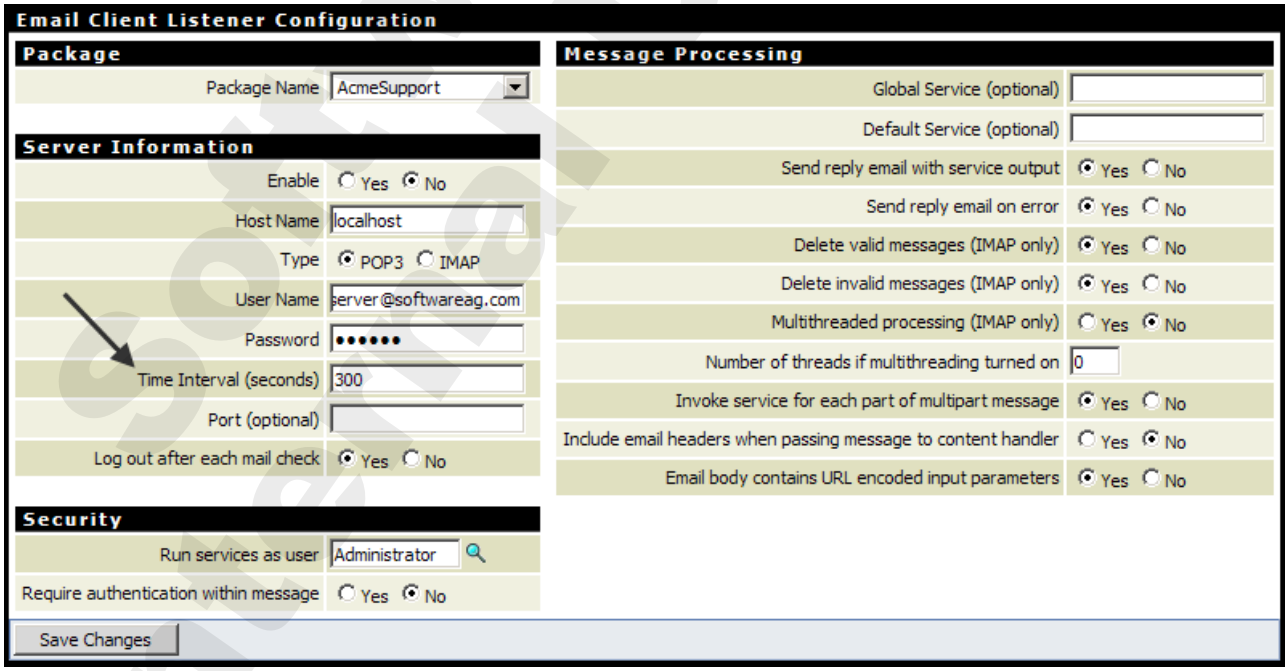


click on the word “No” in the “Enabled” column to activiate this port.

d. Start Microsoft Outlook Express and send a mail from **customer@softwareag.com** to **integration-server@softwareag.com** with an empty message body and the subject set to **acmeSupport.xml:xmlAdd.** As attachment drag the **addInput.xml** document from an explorer window into the mail message.
Once you completed you mail, press the send button.



Note1: In order speed up processing, you may want to change the parameter "Time Interval (seconds) from the configured value of 300 to something less like 10 seconds.



Do not forget to enable the Port after changing this value. Remember to change this value back to 300 after the exercise.

Note2: The mail service and the outlook express program on your virtual machine are set up to handle all mail locally. There is no connectivity to any outside mail system.

Outlook Express and the hmail server are set up to serve the softwareag.com and the v8training.net domains. Please do not change any of the configuration settings unless otherwise noted.

After you sent your mail, press the Send/Recv button in Outlook Express and you will receive a reply from integration server with the result of the message processing. To see the content of this message, simply drag it into the window of a running onstance of the Notepad++ editor.

4. Invoke a service using FTP.

a. Before using ftp, make sure there is an enabled FTP port in integration server available. Open the Integration Server administration tool and go into the security ➡ ports submenu. Make sure an FTP port exists for port 9021 and make sure its access mode setting allows every service to be executed:

Port List

Primary	Port	Provider	Protocol	Type	Package	Enabled	Access Mode	IP Access	Advanced	Delete
	9021	webMethods	FTP	Regular	TNSupport	✔ Yes	Edit	Edit	Not Applicable	✘
	9999	webMethods	HTTP	Diagnostic	WmRoot	✔ Yes	Edit	Edit	Edit	✘
✔	5555	webMethods	HTTP	Regular	WmRoot	✔ Yes	Edit	Edit		✘
	15006	webMethods	HTTP	Regular	WmPRT	✔ Yes	Edit	Edit	Edit	✘
	integration-server@softwareag.com@localhost	webMethods	Email	Regular	AcmeSupport	✔ Yes	Edit	Edit	Not Applicable	✘

Port Service Access Settings

Access Mode	**Allow by Default**

Deny List

Folders and Services	Remove

b. Now open a windows command promt window and execute the command script as shown below. Your input is shown **in bold font**. Make sure you understand what each command is doing before typing it in.

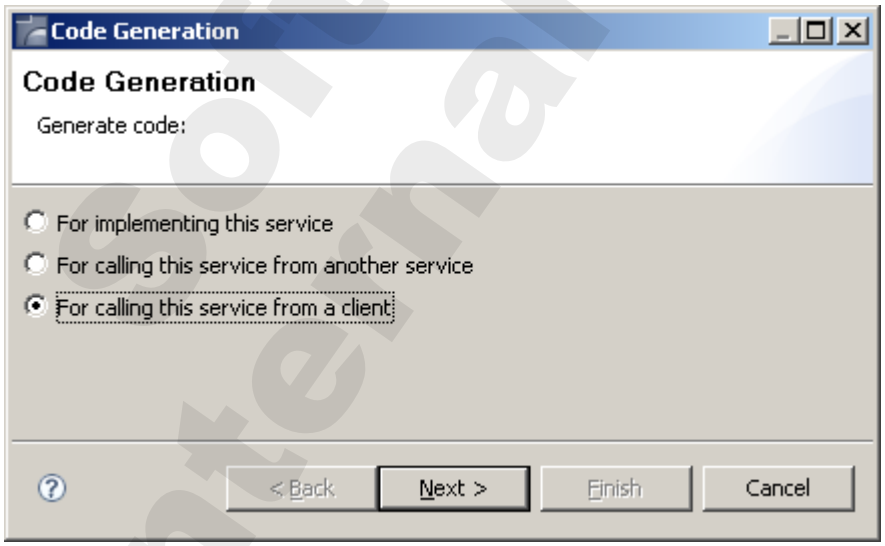
```
C:\>cd /d C:\SoftwareAG\IntegrationServer\packages\AcmeSupport\pub

C:\SoftwareAG\IntegrationServer\packages\AcmeSupport\pub>dir addInput.xml
 Volume in drive C has no label.
 Volume Serial Number is 9C80-4210
 Directory of C:\SoftwareAG\IntegrationServer\packages\AcmeSupport\pub

03/09/2010  03:54 PM               181 addInput.xml
               1 File(s)            181 bytes
               0 Dir(s)  42,108,518,400 bytes free

C:\SoftwareAG\IntegrationServer\packages\AcmeSupport\pub>ftp
ftp> open localhost 9021
Connected to sagbase.softwareag.com.
220 sagbase:9021 FTP server (webMethods Integration Server version 8.0.1.0)
ready.
User (sagbase.softwareag.com:(none)): Administrator
331 Password required for Administrator.
Password: manage
230 User Administrator logged in.
ftp> cd ns
250 CWD command successful.
ftp> cd acmeSupport
```

```
250 CWD command successful.
ftp> cd xml
250 CWD command successful.
ftp> cd xmlAdd
250 CWD command successful.
ftp> send addInput.xml
200 PORT command successful.
150 ASCII mode  data connection for addInput.xml (127.0.0.1,2366).
226 ASCII transfer complete.
ftp: 181 bytes sent in 0.00Seconds 181000.00Kbytes/sec.
ftp> dir
200 PORT command successful.
150 ASCII mode  data connection for /bin/ls (127.0.0.1,2368).
total 1
dr-xr-xr-x  3 root      root                        1 Mar 09 16:44 .
dr-xr-xr-x  3 root      root                        1 Mar 09 16:44 ..
-r--r--r--  1 tx        tx                        106 Mar 09 16:44
addInput.xml.out
226 ASCII transfer complete.
ftp: 232 bytes received in 0.02Seconds 14.50Kbytes/sec.
ftp> get addInput.xml.out
200 PORT command successful.
150 ASCII mode  data connection for addInput.xml.out (127.0.0.1,2370) (106
bytes).
226 ASCII transfer complete.
ftp: 106 bytes received in 0.00Seconds 106000.00Kbytes/sec.
ftp>!type addInput.xml.out
<?xml version="1.0" encoding="UTF-8"?>

<Values version="2.0">
  <value name="value">42</value>
</Values>
ftp> quit
221 Goodbye.
```

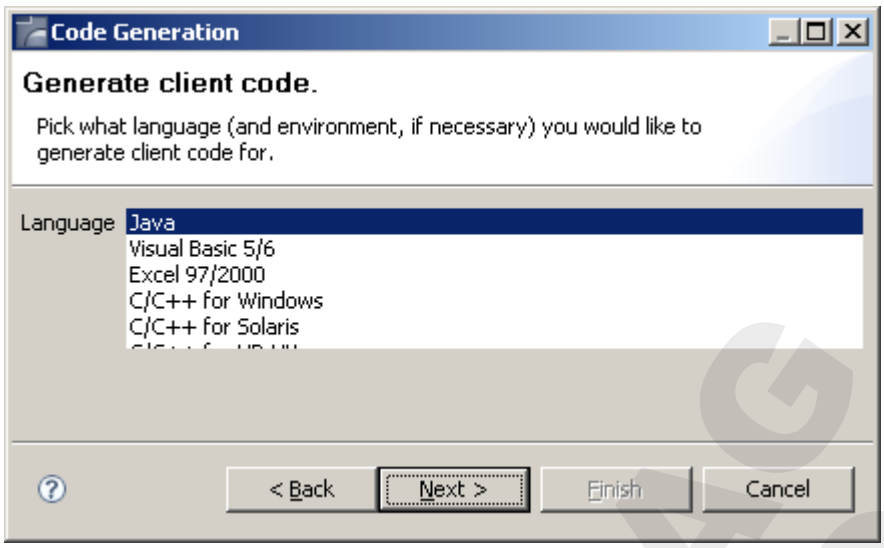
5. Invoke a service using Java

   a. Find the xmlAdd service above in Designer.

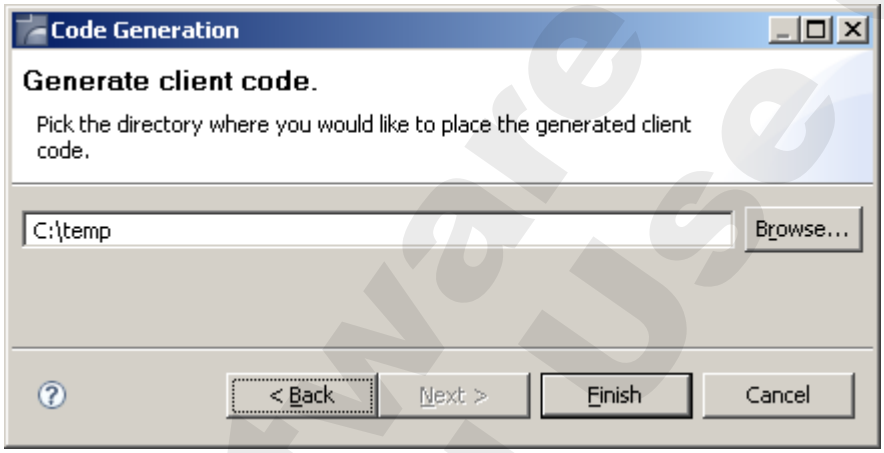
   b. Right click the service and choose the “Generate Code” entry. In the dialog box that opens select “For calling this service from a client”.



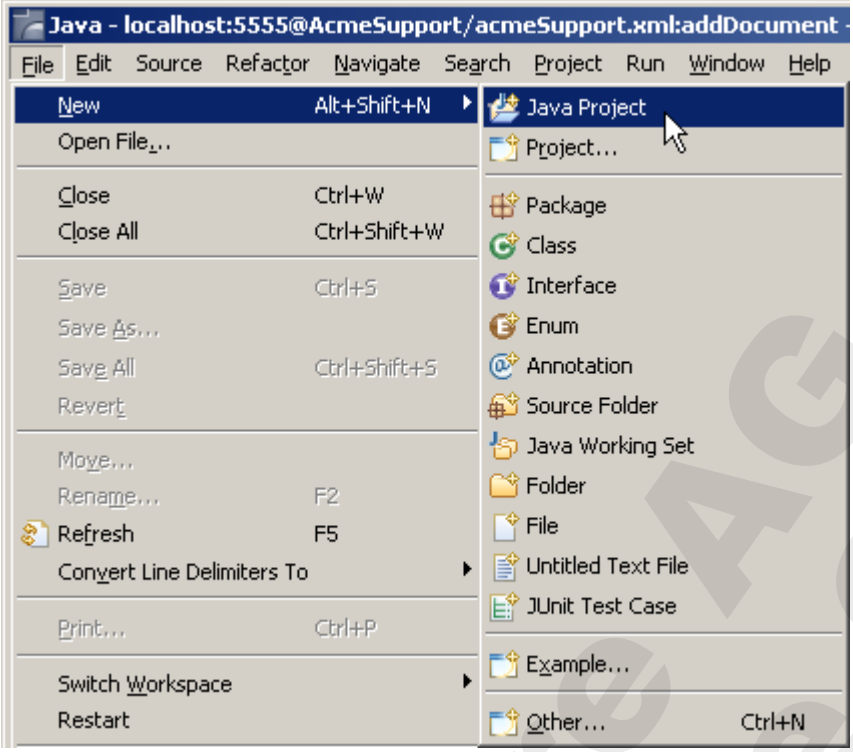
   c. Then choose **Java** as language



and use C:\TEMP as directory for code generation.

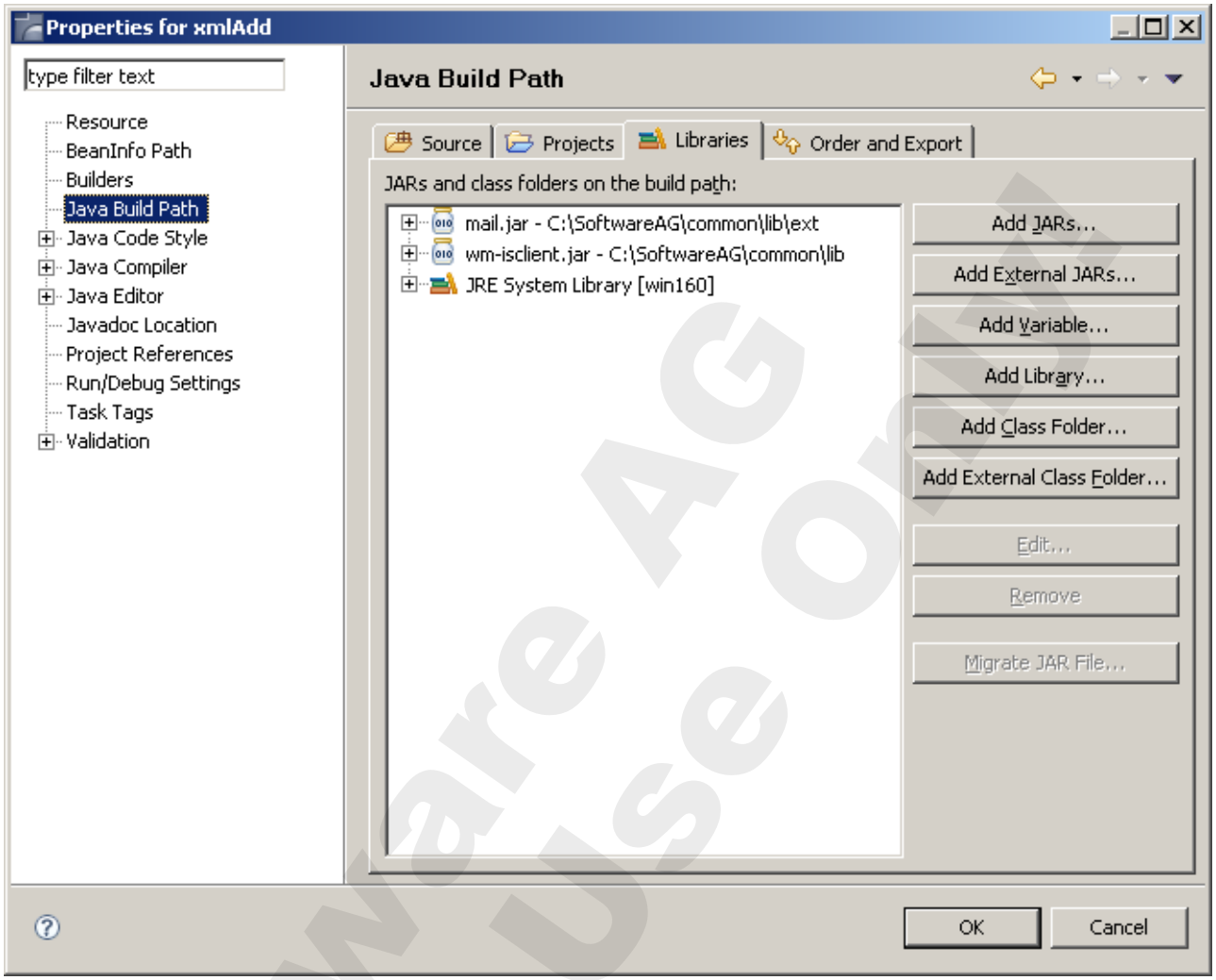


d. Now create an Eclipse Java project.



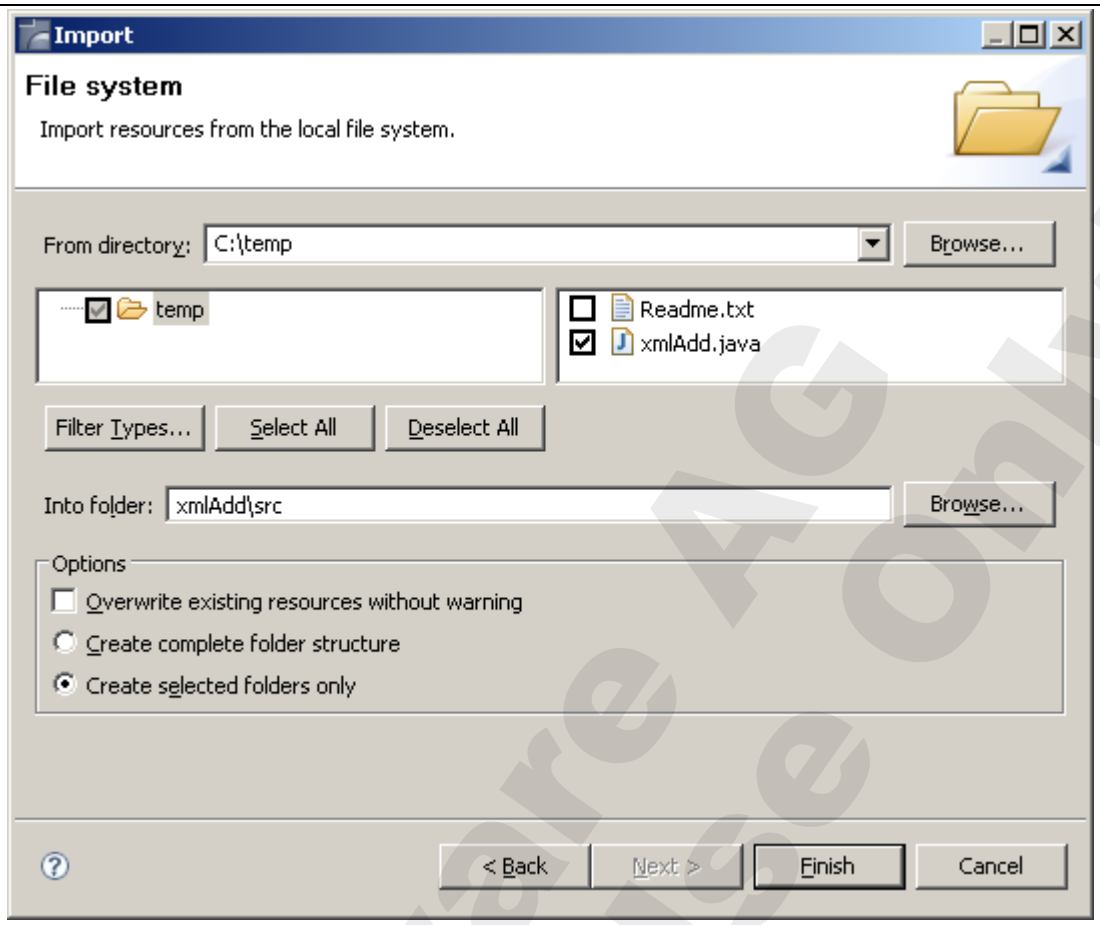
In the upcoming Dialog enter **xmlAdd** as project name. Do not change any of the defaults and hit the "Finish" button.

e. This Project requires two additional external Jar files. To add them, Right click on the project node (xmlAdd) and select the **Properties** entry at the very bottom of the pop up. In the appearing dialog select "**Java Build Path**" and click on the "**Libraries**" selector. In this window choose "**Add external Jars**" and add (in 2 steps) the Libraries **...\common\lib\wm-isclient.jar** and **...\common\lib\ext\mail.jar**. When finished, your window should look like this one:



Close the dialog by clicking the OK button.

f. Now import the Java source generated in the first step. To do so, right click the xmlAdd node once more and select the "**Import**" option from the menue. Choose "**General**" ➡ "**File System**" and click "**Next**". In the upcoming window enter C:\temp as directory and make sure the xmlAdd.java file is selected. In to "**Into Folder**" field enter "**xmlAdd\src"**. Your dialog should look like this:



Click the "**Finish**" button.

g. Open the xmlAdd.java file and change the lines

```
// Set user name and password for protected services
String username = null;
String password = null;
```

to correct credentials like the following:

```
// Set user name and password for protected services
String username = "Administrator";
String password = "manage";
```

h. Now you can run your program by right clicking the xmladd.java file and selecting "**Run As" ➡ "Java Application"**. You may have to confirm saving your sources. Look for the console view. Enter two small numbers for the "a =" and "b =" prompts and verify the result.

```
Console
<terminated> xmlAdd (1) [Java Application] C:\SoftwareAG\jvm\win160\bin\javaw.exe
a =12
b =34

********* Successful invoke **********

************* Inputs *****************
DOCUMENT "Input"
a = 12
b = 34

************* Outputs *****************
DOCUMENT "Output"
value = 46
```

Check Your Understanding

1. Why and when would you use an HTTP URL alias for your services?
2. How do the services find their input data?
3. How do the services return their result?

This page intentionally left blank

Exercise 13:
Create a Flat File Schema

Overview

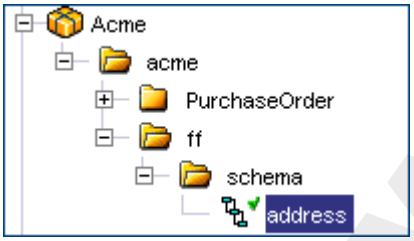
In this exercise, you will create, configure and test a Flat File Schema object to parse flat files like the following one:

```
ADDRESS,Acme Hammer Company,123 Wilson St.,Sacramento+CA+95833
ADDRESS,Johnson Supply Co.,456 Nadia Ave.,Seattle+WA+98188
```

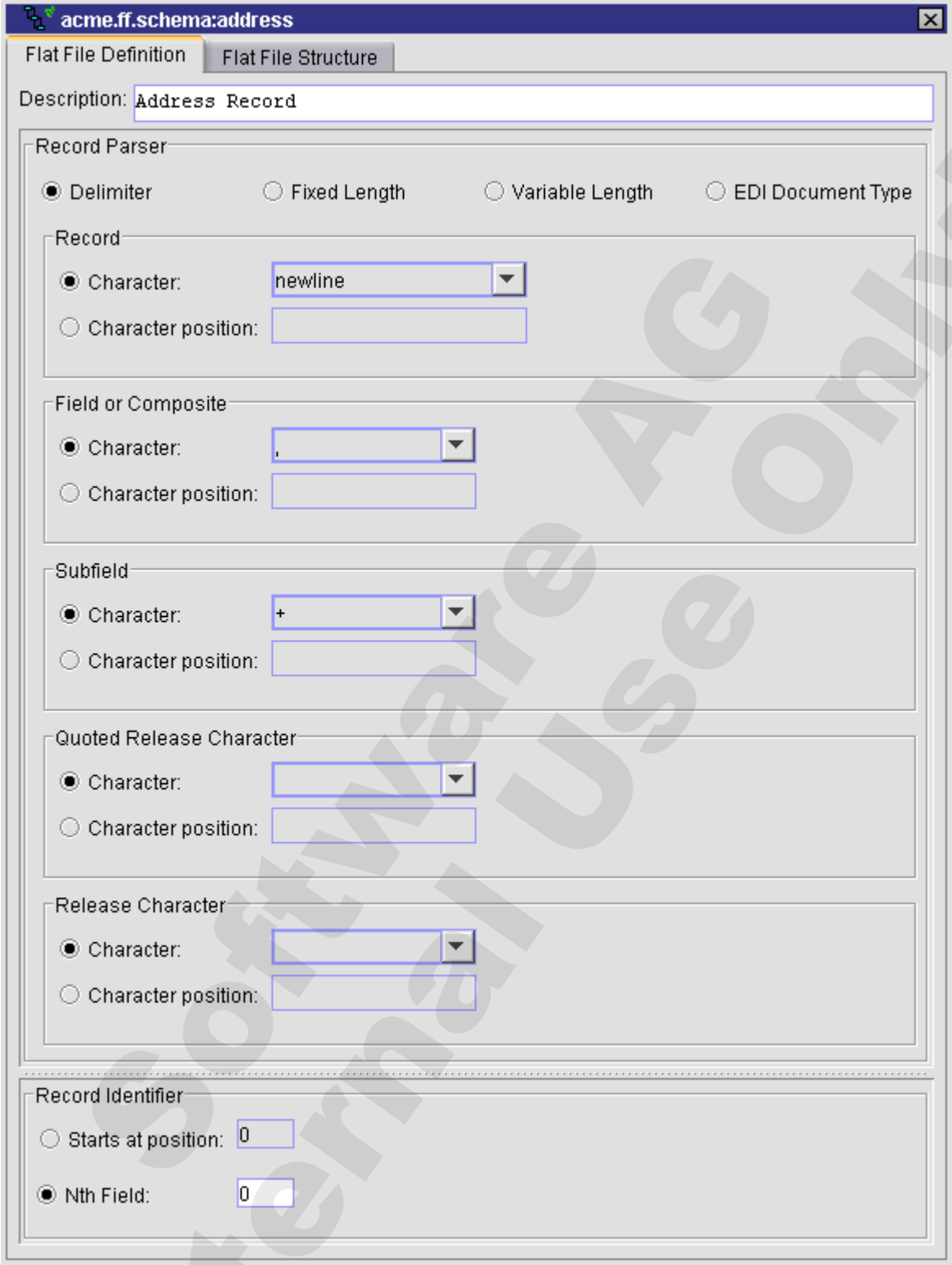
However, since the flat file processing is not jet part of designer, you have to use the developer tool to perform this exercise.

Steps

1. Start Developer
2. In the **Acme** package's **acme** folder, create a new folder called **ff** and a folder inside of **ff** called **schema**. Create a new **Flat File Schema** object called **acme.ff.schema**:**address.**

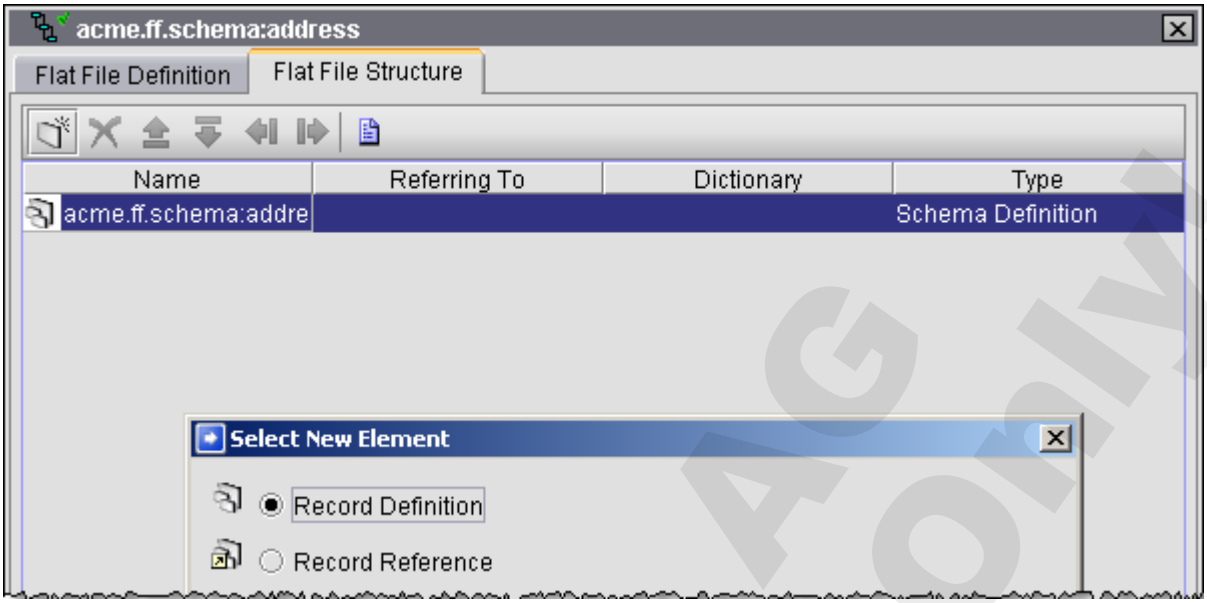


3. Configure **acme.ff.schema:address** to have the following information:
   a. Description = Address Record
   b. Record Parser = Delimiter
   c. Record Character = newline
   d. Field or Composite Character = ,
   e. Subfield Character = +
   f. Quoted Release Character = *leave blank*
   g. Release Character = *leave blank*
   h. Record Identifier = Nth Field: 0

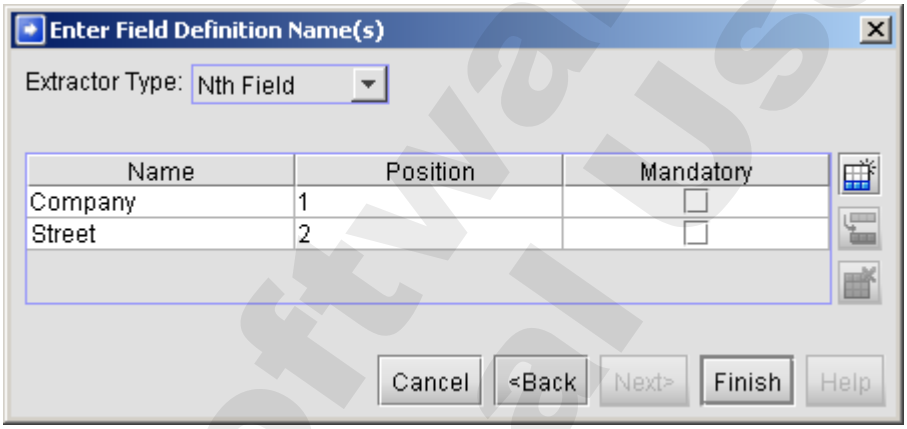


Note: due top limited screen space, you may have to move some of the sliders and scrollbars to be able to see all the input fields.

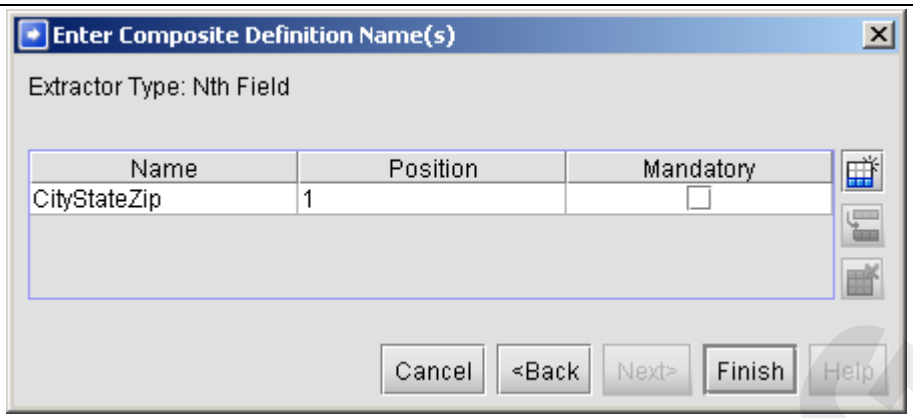
4. Change to the **Flat File Structure** tab. Select the **address** schema with the mouse and create a record definition (by clicking on the  button) called **ADDRESS**.



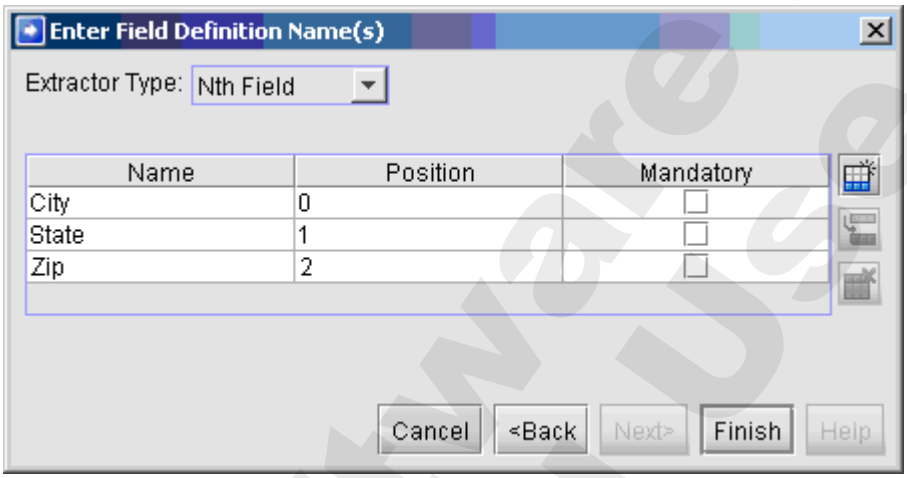
5. Right click the **ADDRESS** record and select **New ➡ Field Definition**. In the upcoming dialogue select an **Extractor Type** of **Nth Field**. Then create 2 **Field Definitions**, called **Company** and **Street** at positions 1 and 2, respectively. Note: make sure you select the extractor type before entering the field names.



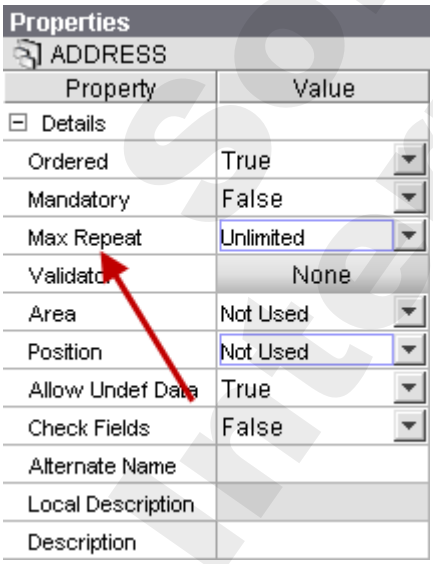
6. Now create a Composite Definition called **CityStateZip** at position 3. This is done by closing the above dialogue and right clicking the **ADDRESS** record definition. Choose **New ➡ Composite Definition** and fill in the Name Field with **CityStateZip.** The Position Field gets the number 1 assigned.



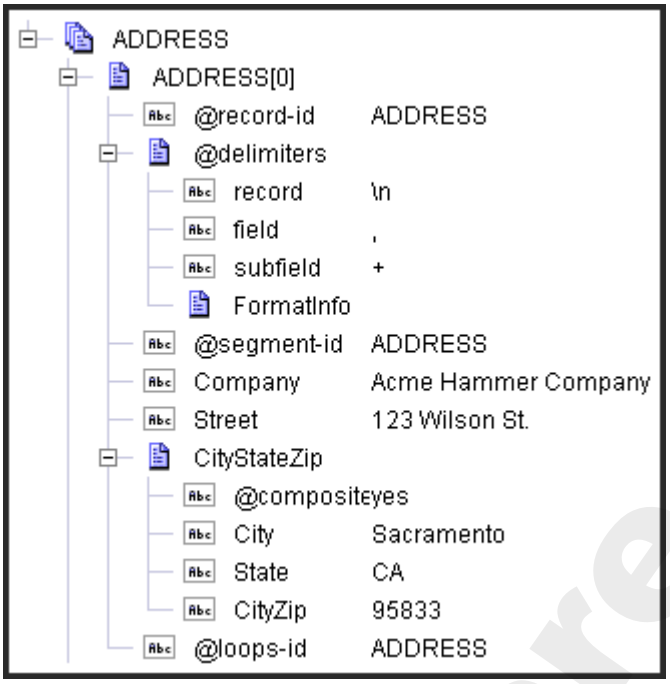
7. In the **CityStateZip** field composite create 3 subfields. They are created by right clicking the composite field and choosing **new ➡ Field Definition**. Make sure an extractor of type Nth Field is used and call your new fields **City**, **State**, and **Zip** and assign positions 0, 1, and 2, respectively.



8. Set the **ADDRESS** record definition to have an **Unlimited** value for the **Max Repeat** property.



9. Save and test the new **Flat File Schema** called **address** by running it in Developer. When asked for an input file, use **address.txt** in the directory **...\IntegrationServer\packages\AcmeSupport\pub\FlatFile.** Make sure that the first ADDRESS record looks like the following:



Check Your Understanding

1. Why can't flat files be imported like XML documents?
2. What is the meaning of Nth field?

This page intentionally left blank

Exercise 14:
Create a Flat File Dictionary

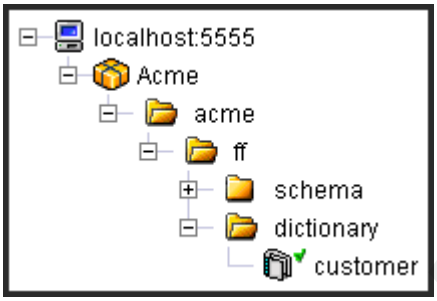
Overview

In this exercise, you will create a new Flat File Dictionary, create a reusable record definition in the Dictionary, reference this record definition in a new Flat File Schema, and test the Flat File Schema.

Just like the last exercise, you have to do this exercise using the developer tool.

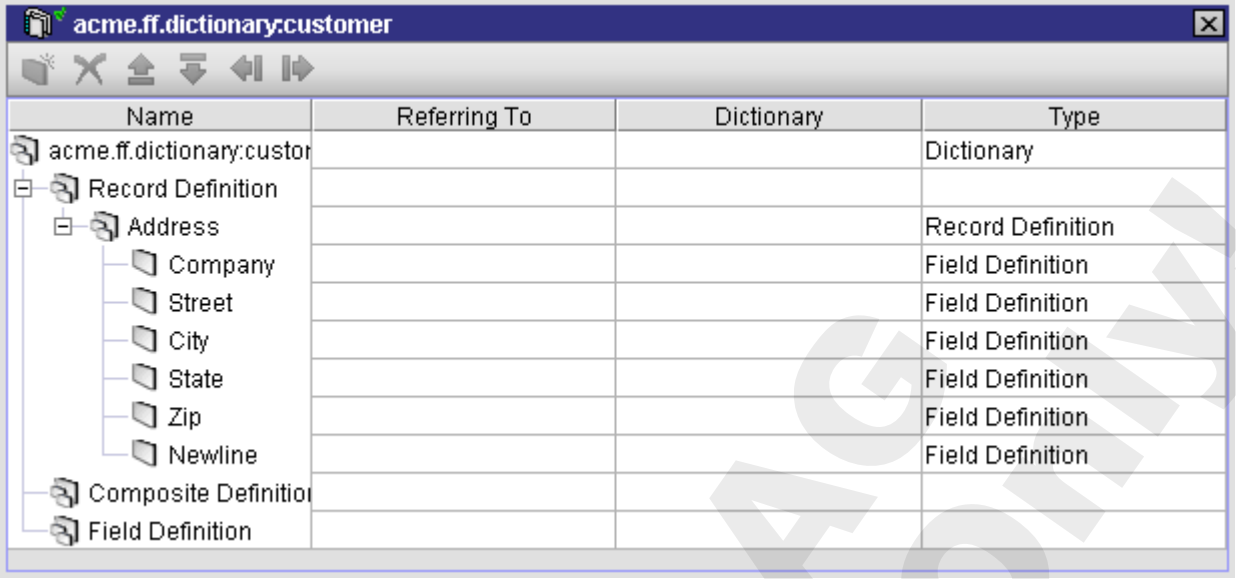
Steps

1. In the Acme package's **acme.ff** folder, create a new folder called **dictionary**. Create a **Flat File Dictionary** called **acme.ff.dictionary:customer.**



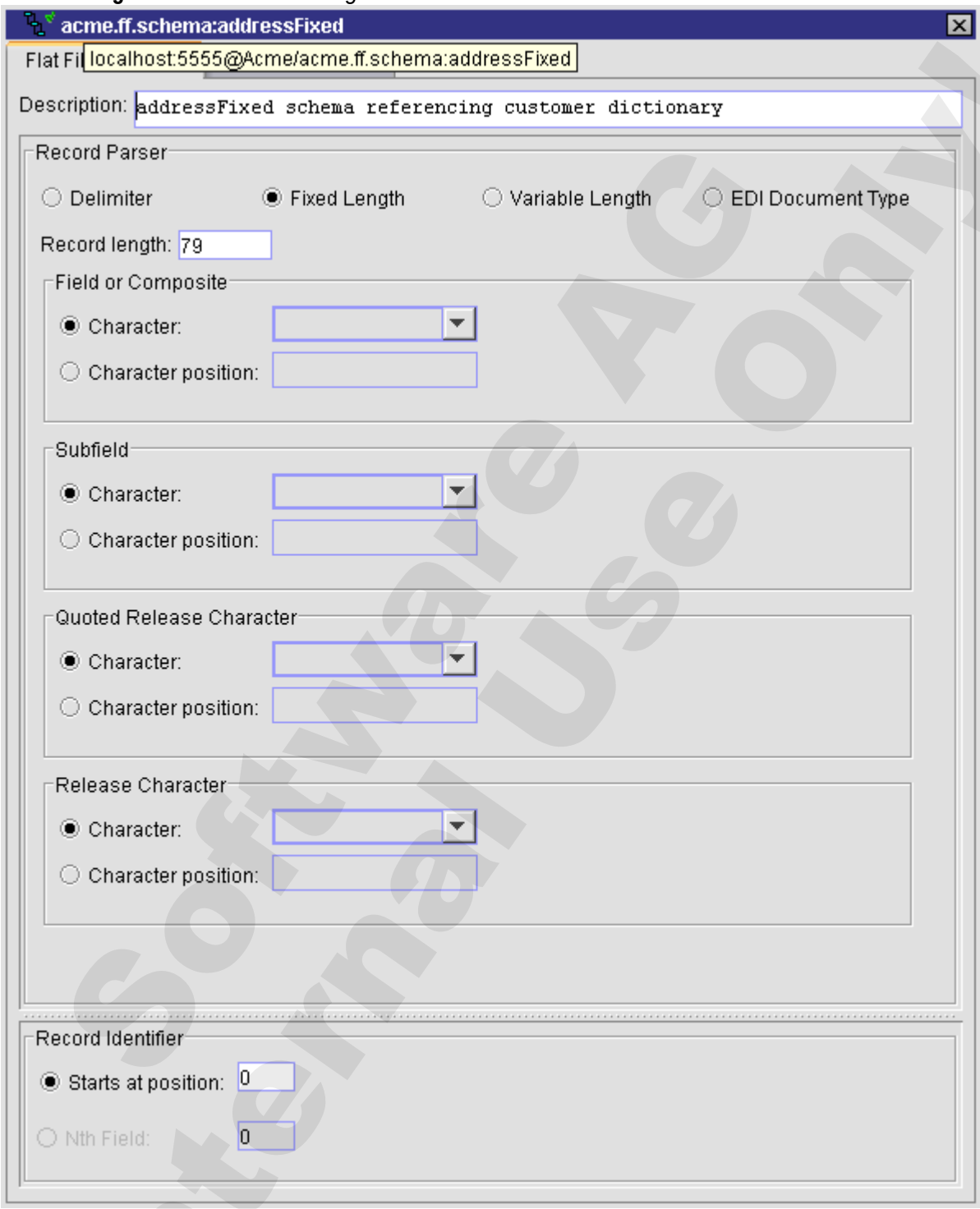
2. Add a record definition called **Address** (case sensitive) to the new **Flat File Dictionary**. Using an **Extractor Type** of **Fixed Position**, add 6 new field definitions to the **Address** record as shown in the table below:

Name	Start	End
Company	0	30
Street	30	55
City	55	70
State	70	72
Zip	72	77
Newline	77	79



Name	Referring To	Dictionary	Type
acme.ff.dictionary:custor			Dictionary
Record Definition			
Address			Record Definition
Company			Field Definition
Street			Field Definition
City			Field Definition
State			Field Definition
Zip			Field Definition
Newline			Field Definition
Composite Definitio			
Field Definition			

3. Create a new **Flat File Schema** called **acme.ff.schema:addressFixed**. Specify that it is **Fixed Length** with a *Record length* of **79** characters.



4. In the "**Default Record**" property of the **addressFixed** schema, add a reference to the **customer** Flat File Dictionary's **Address** record definition**.**

Properties

acme.ff.schema:addressFixed

Property	Value
Default Record	
Set	Set...
Delete	Delete
Dictionary	acme.ff.dictionary:customer
Name	Address

5. Save your work and test the **addressFixed** Flat File Schema with the file **...\IntegrationServer\packages\AcmeSupport\pub\FlatFile\addressFixed.txt**
   Once the **addressFixed** schema functions correctly, click on the **Flat File Structure** tab and select the **Create Document Type** icon ( ) to create the **addressFixedDT** IS document type.

Check Your Understanding

1. What is the difference between a dictionary and a schema?
2. Why should you create the IS document type when the schema is complete?

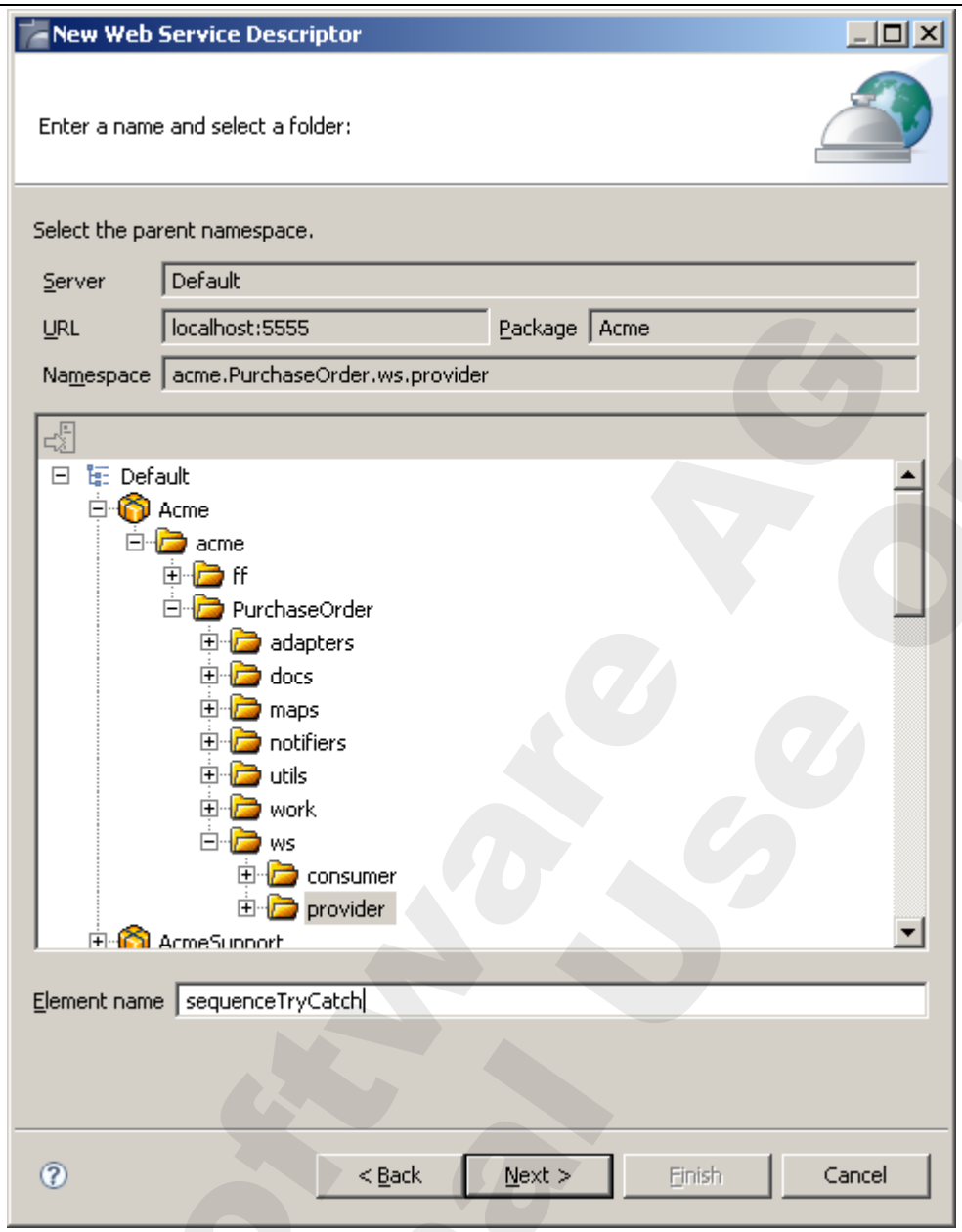
Exercise 15:
Web Service Descriptors and Custom Faults

Overview

In this exercise, you will take the flow service you already created called **sequenceTryCatch** and make it callable via a web service by creating a **Provider Web Service Descriptor (WSD)**. To prove that anyone (including the IS itself) can call **sequenceTryCatch** as a web service, you will create a **Consumer WSD** based on the **WSDL** created from the **Provider WSD** and invoke **sequenceTryCatch** using the auto-generated **Web Service Connector**. Finally, you will create a generic **Error** document. You will specify that it can serve as a custom SOAP Fault. Then you test to see if the custom SOAP fault gets returned as expected.

Steps

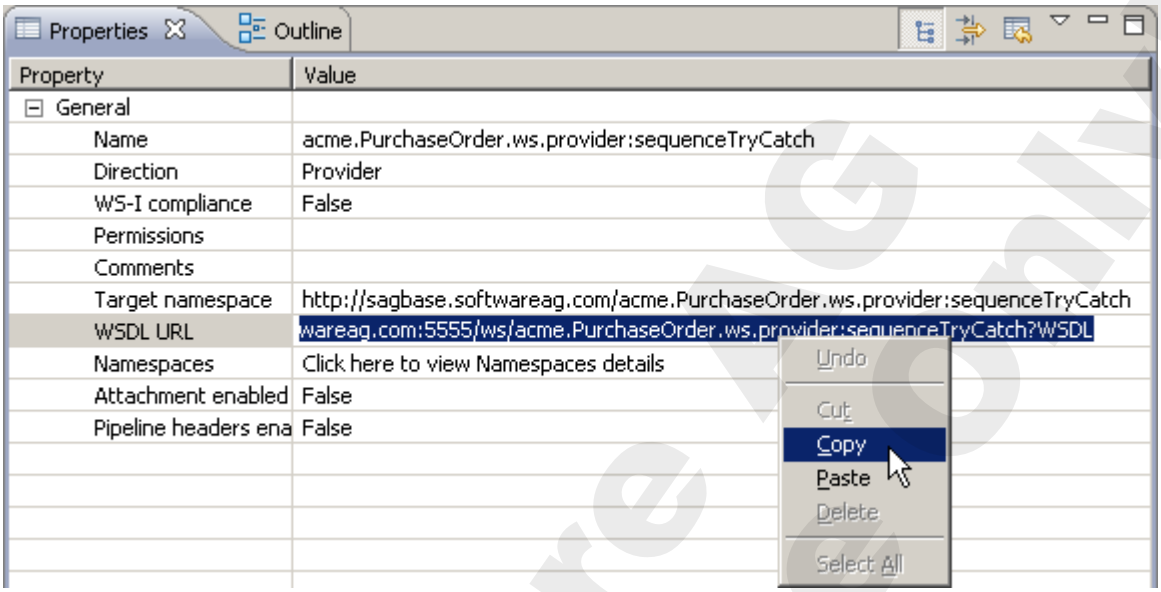
1. In Designer, create a new **Web Service Descriptor** in the **acme.PurchaseOrder.ws.provider** folder.
   a. Accept all of the defaults (**Provider**, **Existing IS service(s)**, and **No** for WS-I compliance) and click the **Next>** button.
   b. Type the name **sequenceTryCatch** and specify the **acme.PurchaseOrder.ws.provider** folder as the location to create the **Provider WSD,** then click the **Next>** button.



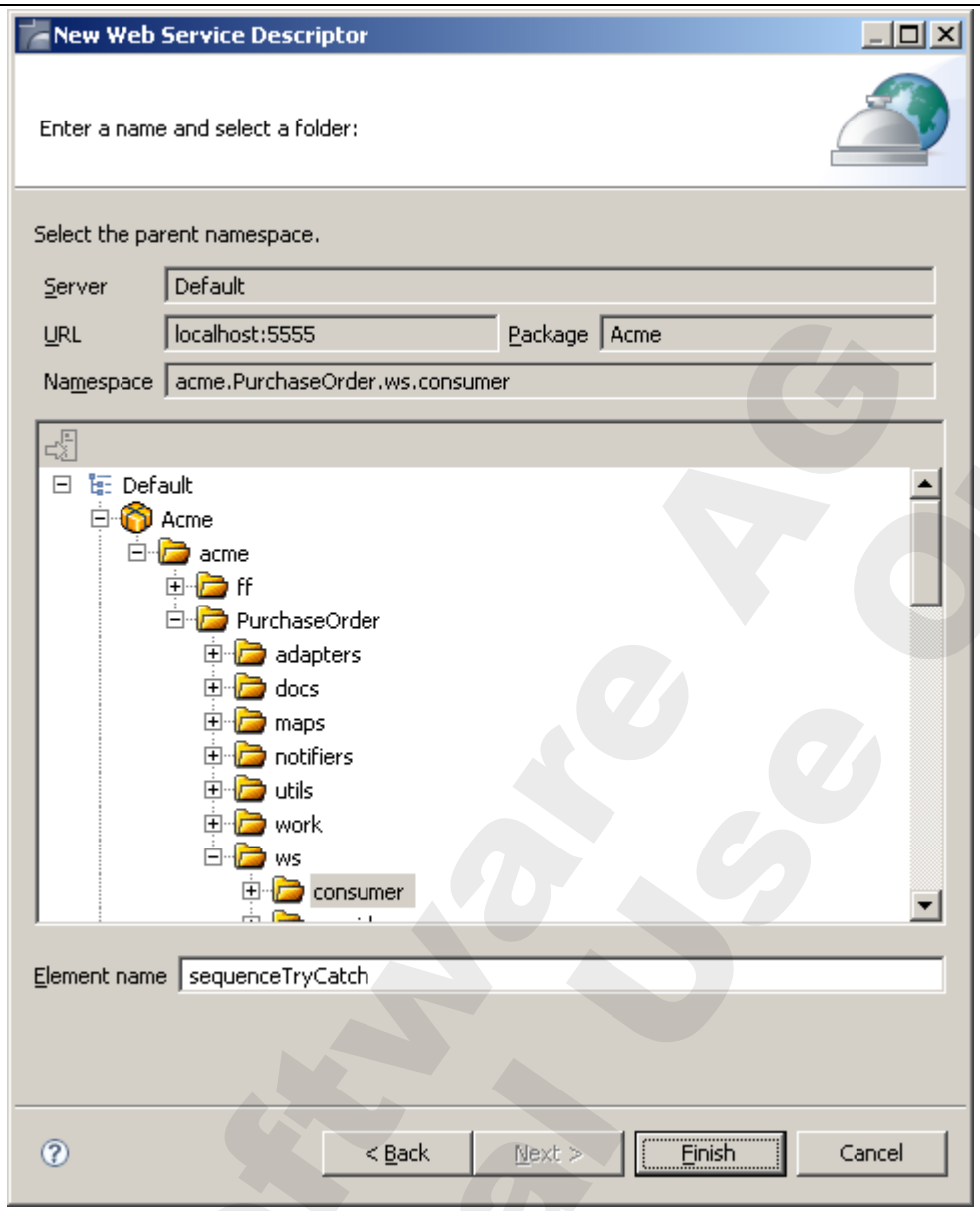
c. In the dialog that appears next, navigate to and select the **acme.PurchaseOrder.work:sequenceTryCatch** flow service, then click the **Next>** button.

d. In the next screen, leave all the defaults in place and click the **Finish** button.

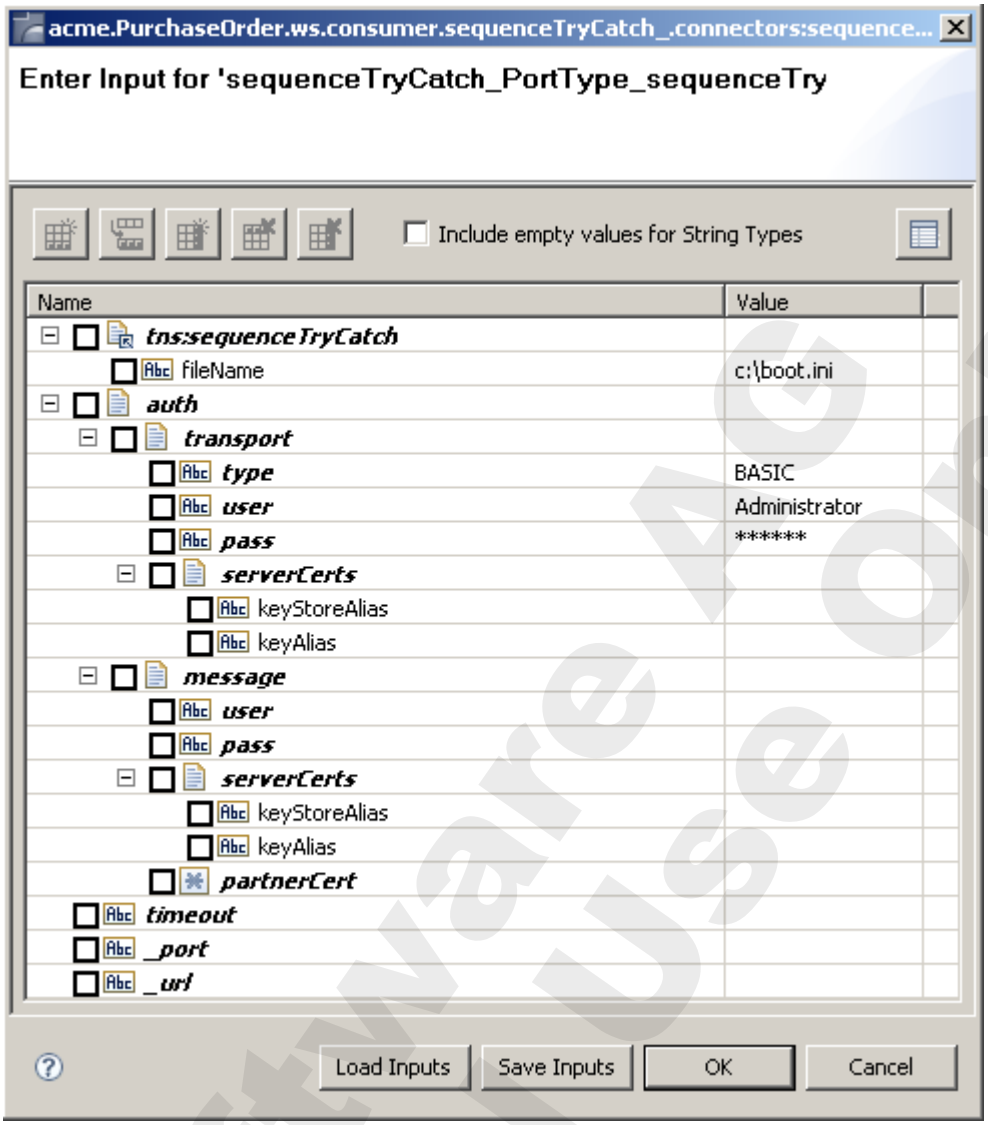
2. When the new **Provider WSD** appears in the editor, in the **Properties** panel, highlight the **WSDL URL** property value and copy it to the clipboard – you will need this value in one of the next steps.



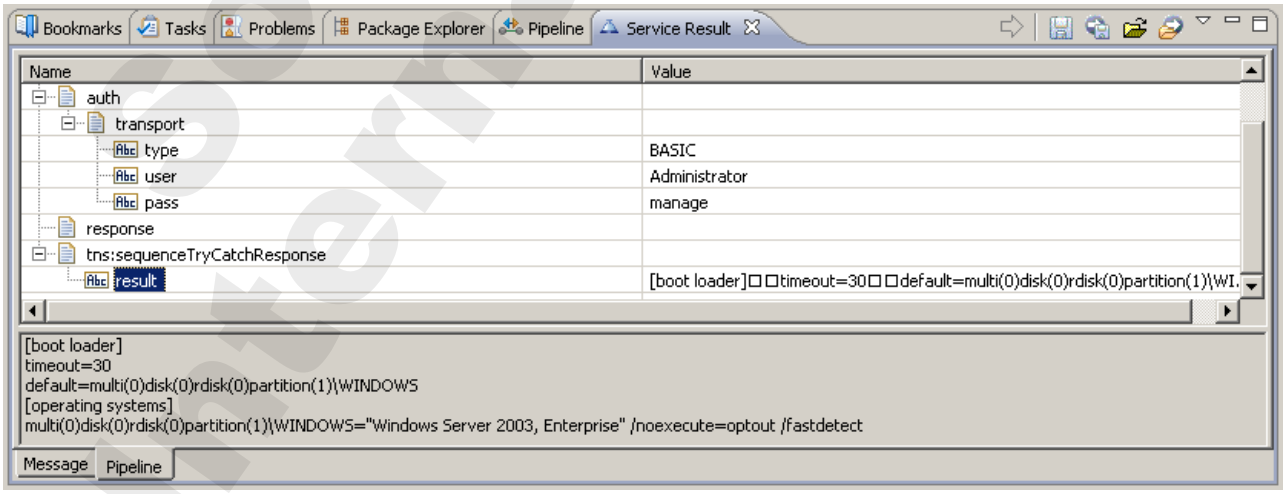
3. Open Internet Explorer, paste the **WSDL URL** in the IE's address field and hit return. The WSDL for the **sequenceTryCatch** web service should appear. This is the URL that consumers of the web service must use to access its WSDL.
4. Now create another Web Service Descriptor in the . **acme.PurchaseOrder.ws.consumer** folder.
   a. This time select the **Consumer** radio button, accept the default for all other fields and paste the just copied WSDL URL into the **WSDL URL** text field. Then click the **Next>** button.
   b. Enter the Name **sequenceTryCatch** and select the **acme.PurchaseOrder.ws.consumer** folder and click the **Finish** button.



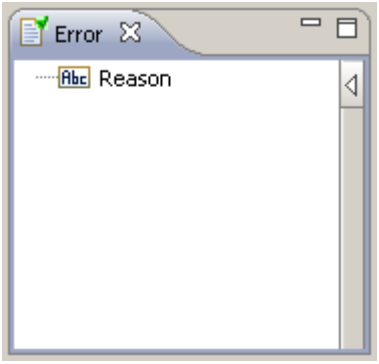
5. After Designer finishes generating the **sequenceTryCatch Consumer WSC**, navigate to and open the **Web Service Connector acme.PurchaseOrder.ws.consumer.sequenceTryCatch_.connectors:sequenceTryCatch_PortType_sequenceTryCatch.**
6. Right-Click to run the **sequenceTryCatch_PortType_SequenceTryCatch** and select "Run As" ➡ "1 Run Flow Service". provide the following values as input:
   a. fileName = c:\boot.ini
   b. auth/transport/type = BASIC
   c. auth/transport/user = Administrator
   d. auth/transport/pass = manage



7. Review the results in Service Result view.

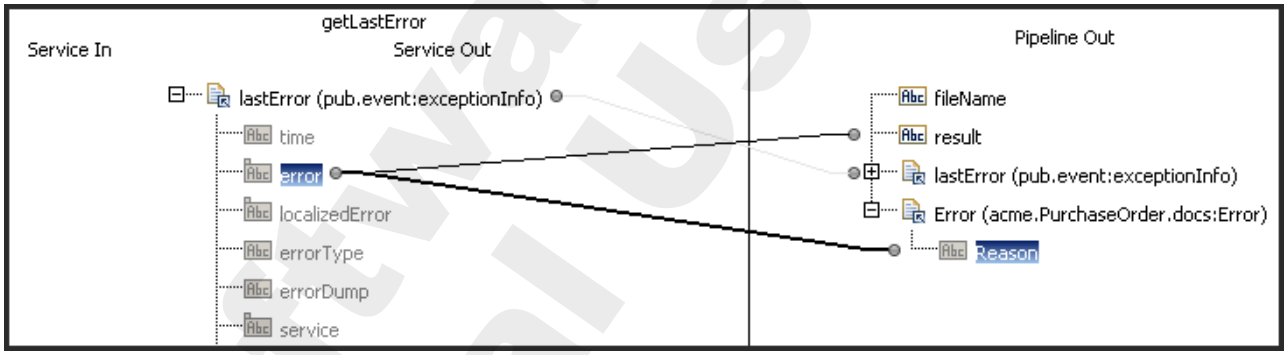


8. Now create a new Document Type called **acme.PurchaseOrder.docs:Error**. Add one string field to the **Error** document type and name it **Reason.**



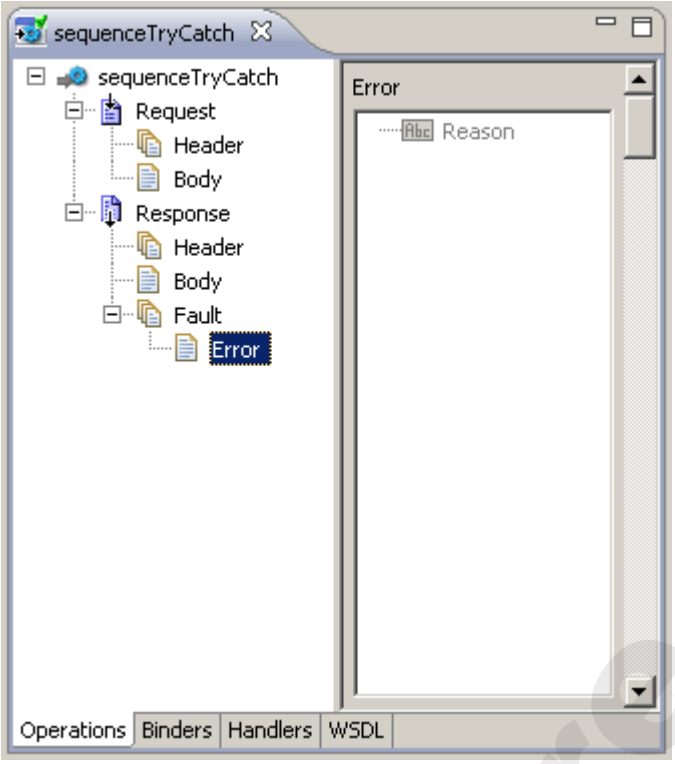
9. Open the original service **acme.PurchaseOrder.work.sequenceTryCatch** flow service and modify it to use the new **Error** doc.

   a. In the service, open the **Tree** tab. Expand the service and select the **pub.flow:getLastError** step. Click on the **Pipeline** tab, select the **Pipeline Out** section, right click in it and select **Insert ➡ Document Reference**, then navigate to the **acme.PurchaseOrder.docs:Error** document type. Name the reference **Error** and then map the **lastError/error** variable from the **Service Out** to the **Reason** variable in the **Error** document reference.



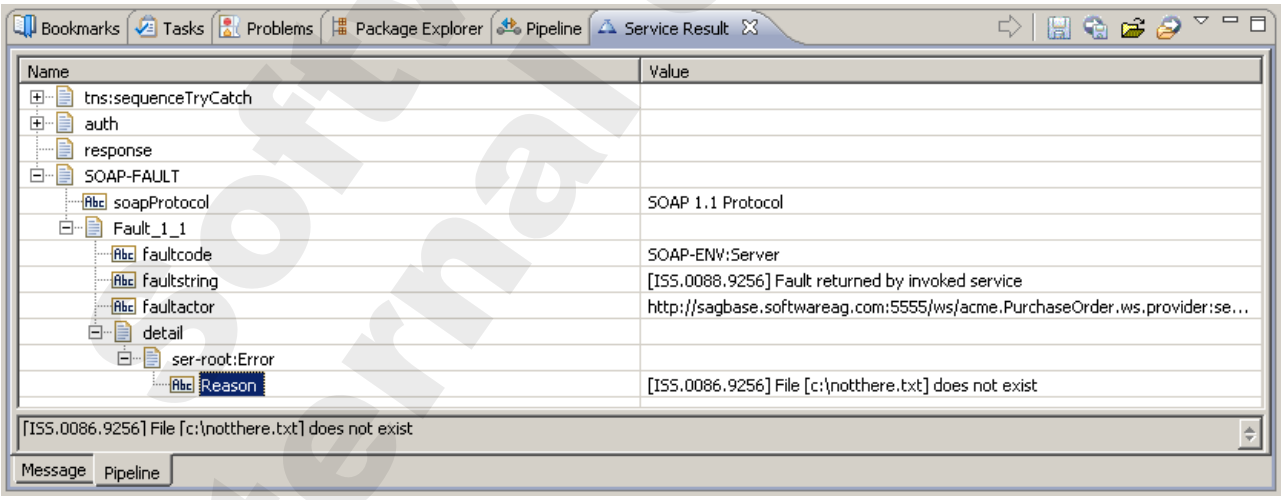
10. Go back to the Provider WSD **acme.PurchaseOrder.ws.provider.sequenceTryCatch** and expand the **sequenceTryCatch** operation, then expand the **Response** document. Click on the **Fault** document list and select the **Add Header or Fault** button.

Note: This button is in the top icon bar of designer. In the popup dialog, navigate to the **acme.PurchaseOrder.docs:Error** document type, select it, and click the **OK** button. Your sequenceTryCatch provider should look like the following:



11. Now you must regenerate the consumer **WSD** connector Flow services so that they capture the change to the Provider WSD. Right-click on the Consumer WSD and select "Refresh Web Service Connectors".

12. Follow steps 5 - 7 above to run the recreated consumer **WSD**, but enter **c:\notthere.txt** for **fileName**. Note: you should see your custom fault document in the Service Result view.



Check Your Understanding

1. When would you create a Provider WSD when a Consumer WSD?
2. How and when are WSC's created?
3. Can you have more than one custom SOAP Fault Document?

This page intentionally left blank

Exercise 16:
Broker Pub/Sub

Overview

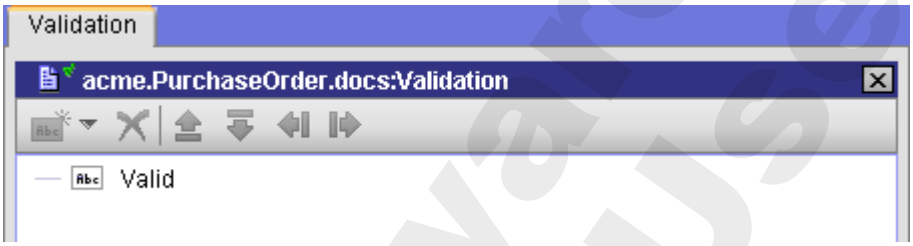
In this exercise, you will create a document type, a handling service and a subscribing Broker trigger. Then you create a service to publish the document.

Since Broker triggers are not yet implemented in Designer, you will use Developer for this exercise. Even so, you could do some of the work in designer; we use developer throughout this exercise so we don't have to swap IDE's continuously.

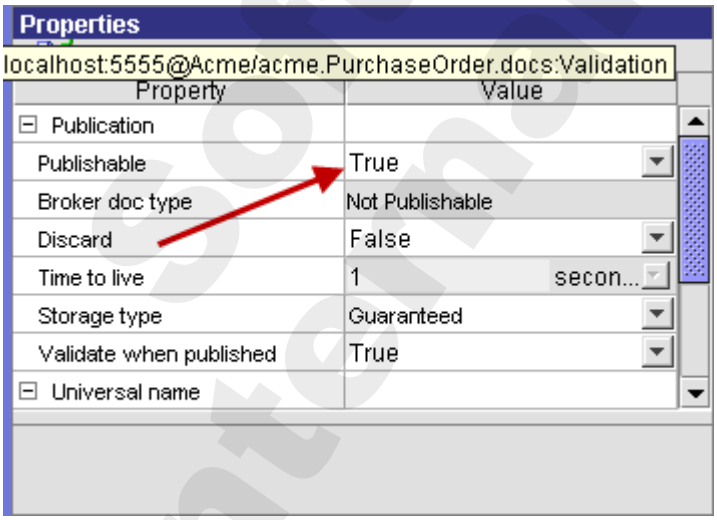
Steps

First, create the subscribing components: document type, handling service, and subscriber.

1. In the Acme package, create a **acme.PurchaseOrder.docs:Validation** document type containing one **String** field named **Valid**.

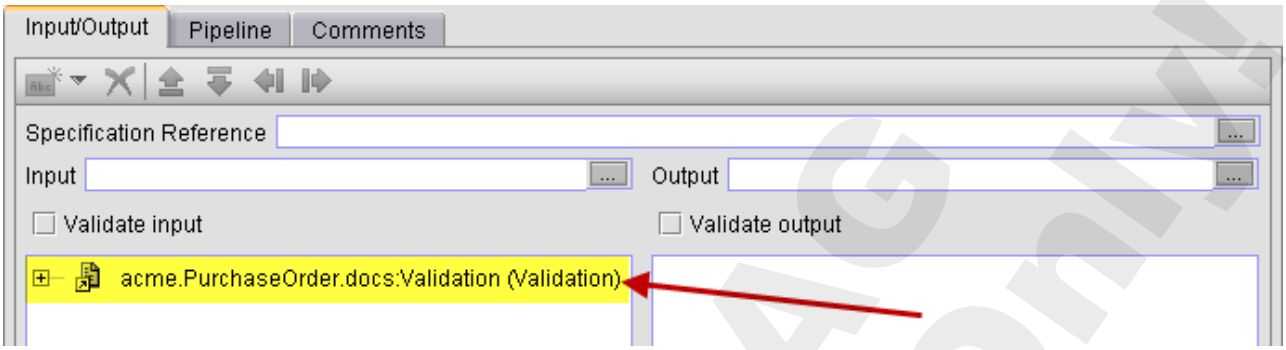


2. Make this document **publishable** to the Broker by setting the **Publishable** property for the document to **true**.

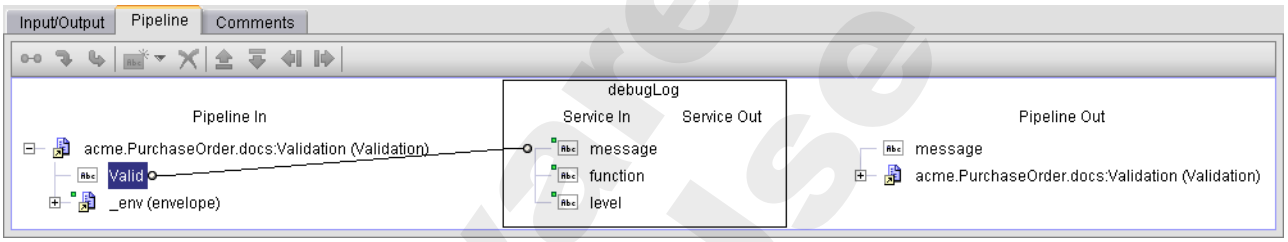


Note: if you do not see the publishable property, make sure you are viewing the properties of the Validation document itself.Open the Documemt Type in the editor and then double-click the editor's title bar of the document type to see its properties.

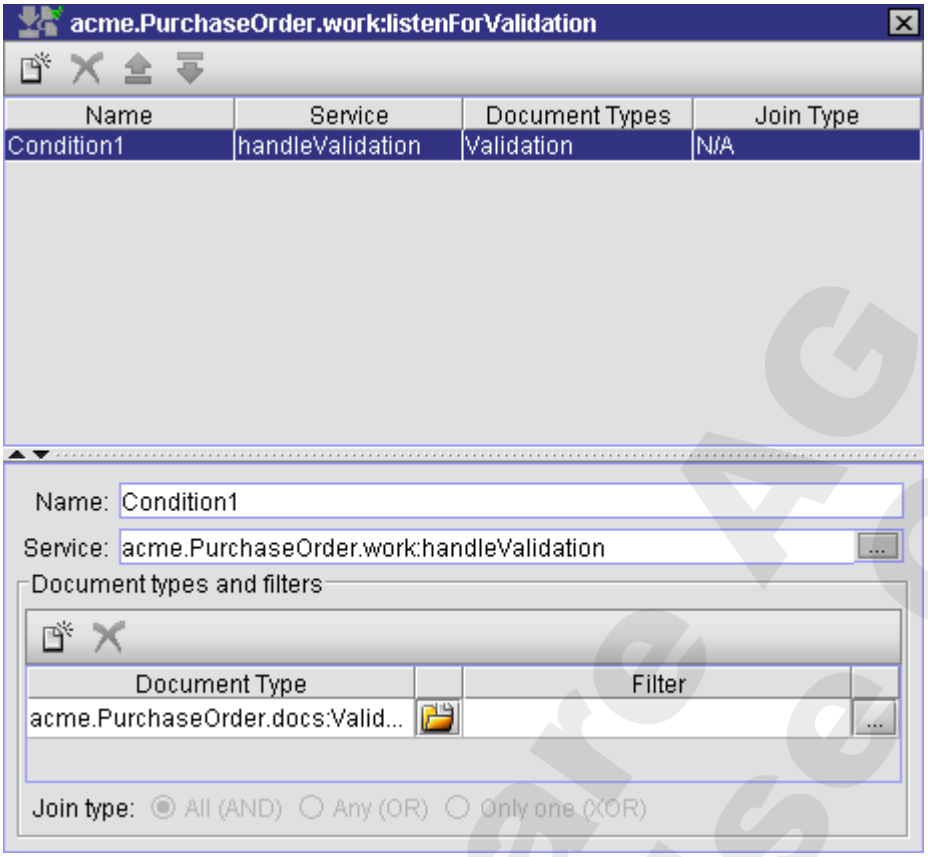
3. Create a new Flow service **acme.PurchaseOrder.work:handleValidation**. Set the input of this service to be a document reference to your document **acme.PurchaseOrder.docs:Validation**. Set the name of the document reference to be the fully-qualified name of the document type: **acme.PurchaseOrder.docs:Validation.** (Copy and paste the document type name rather than type it!)



4. In the service, add a **pub.flow:debugLog** step and map **Valid** to **message**.

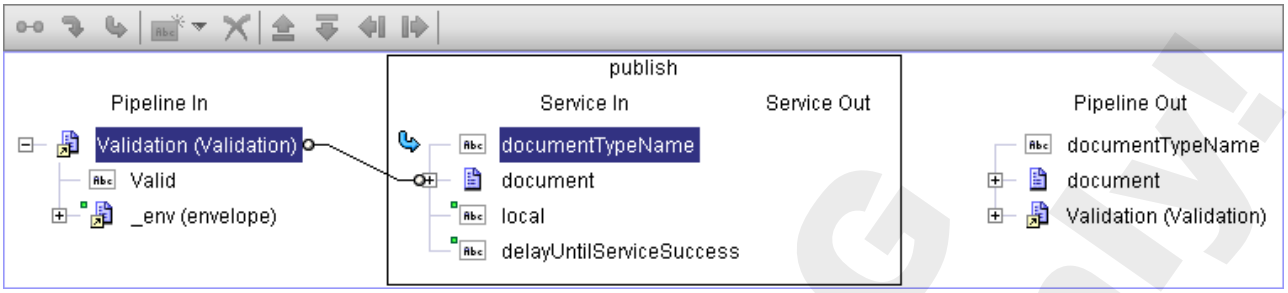


5. In the **acme.PurchaseOrder.work** folder, create a new Broker/Local Trigger, name it **listenForValidation**. Configure the trigger as follows:
   a. Name = Condition1
   b. Service = acme.PurchaseOrder.work:handleValidation (you may use copy and paste here as well)
   c. Document type = acme.PurchaseOrder.docs:Validation
   d. Filter = *leave this empty*



6. Save the trigger. If you want to test you work so far, double-click the **acme.PurchaseOrder.docs:Validation** document in the Navigation view and click the run button. Enter some message int the **Valid** field, leave all other fields unchanged and click next. Then select "Publish to the Broker" and click finish. Your message must appear in the Integration Server log.
7. Next, create the publishing service to publish the publishable document and test your subscription components. Create an **acme.PurchaseOrder.work:publishValidation** Flow service. In the Input of this service add a document reference to your **Acme** package's **acme.PurchaseOrder.docs:Validation.** Name it **Validation.**

8. In the service, add a step to call **pub.publish:publish**. Perform mapping as follows:
   a. **Validation** to **document**
   b. Set documentTypeName = **acme.PurchaseOrder.docs:Validation**



9. Save and run the **publishValidation** service. Enter a value of **true** or **false** for the input of this service. This value should be shown in the Server Log.

10. In your Acme package, open the document type **OrderRequest** imported from XSD in a previous exercise. Make this document **publishable** as well.

Check Your Understanding

1. What happens when a document is made publishable?
2. What would be the appropriate production settings for publishable properties **Discard** and **Time to Live** if the Storage type = Guaranteed?
3. What two objects are required for publishing?
4. What three objects are required for subscribing?
5. Why were you required to use the full document type name as argument name in the handleValidation Service?

Exercise 17:
JMS Pub/Sub

Overview

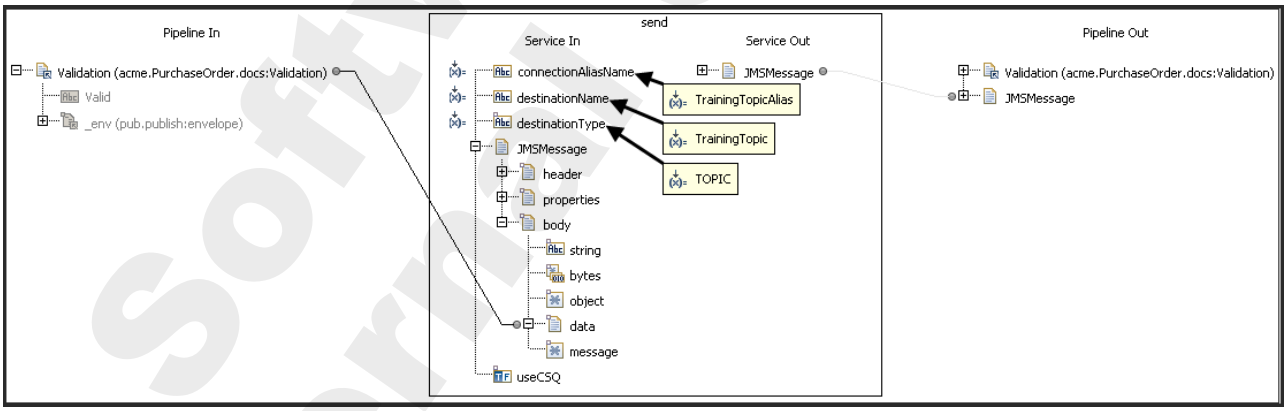
In this exercise, you will create a service to publish an existing document via an existing JMS Topic. Then you create a handling service and a subscribing JMS trigger to receive the document instance.

We will use the existing **acme.PurchaseOrder.docs:Validation** document type and publish it by using a JMS send service. Nothing at the Document needs to be changed in order to use it.

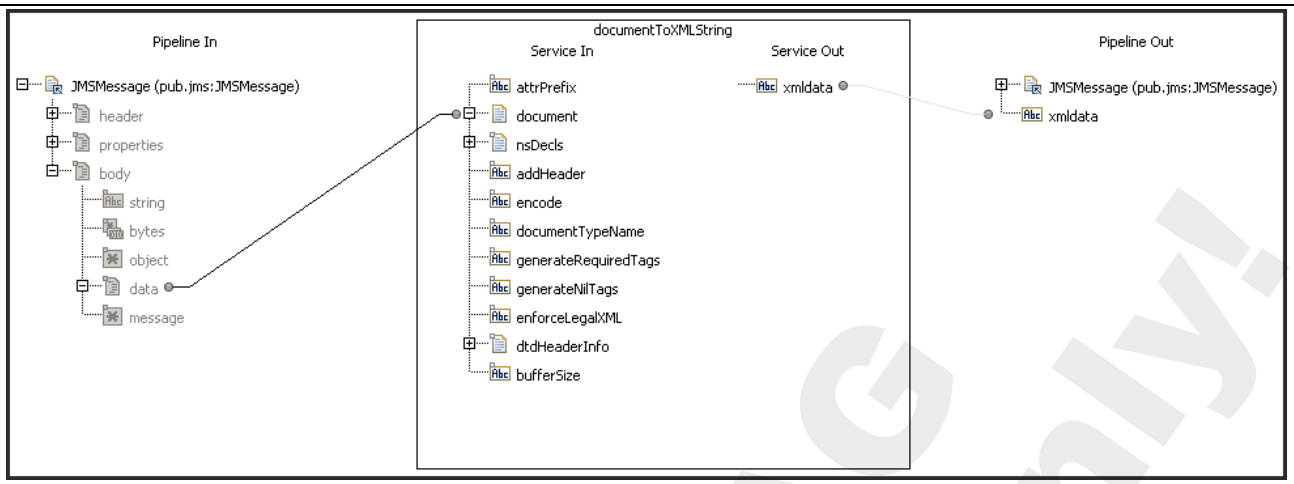
Steps

1. Create a Flow service called **acme.PurchaseOrder.work:publishValidationJMS**. This is used to publish an already created document type. In the Input of this service add a document reference to your **Acme** package's **acme.PurchaseOrder.docs:Validation.** Name it **Validation.**
2. In the new service, add a step to call **pub.jms:send**. Perform mapping as follows:
   a. **Set** connectionAliasName = TrainingTopicAlias
   b. **Set** destinationName = TrainingTopic
   c. **Set** destinationType = TOPIC
   d. **Map** Validation to JMSMessage/body/data

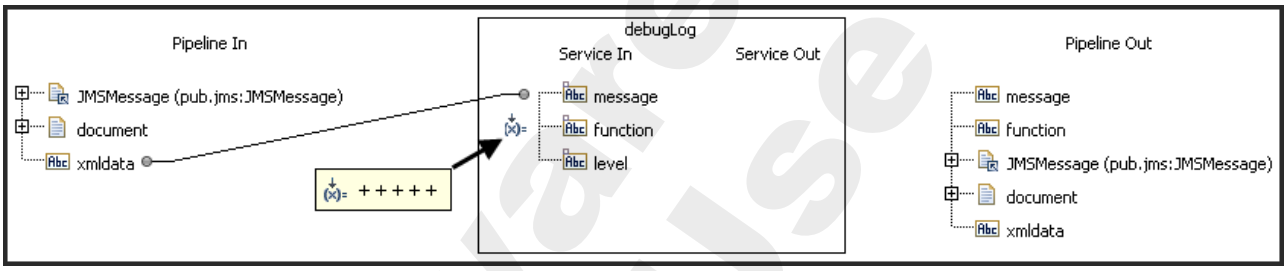
   When you are done editing your service, save the **publishValidationJMS** service.



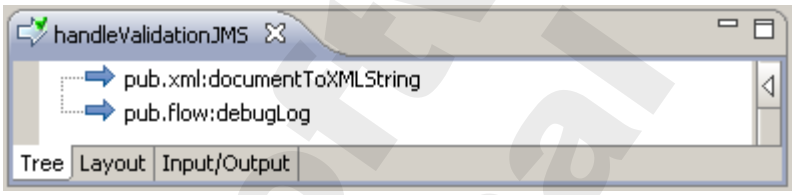
3. Create a new Flow service **acme.PurchaseOrder.work:handleValidationJMS**. This service does the handling service and trigger for subscription. On the Input/Output tab, click the  button to the right of the **Specification Reference** field. Browse for and select **pub.jms:triggerSpec** in the WmPublic package.
4. In the service, add a **pub.xml:documentToXMLString** step and map **JMSMessage/body/data** to **document**.



5. Next add an invocation of **pub.flow:debugLog** to the service and map **xmldata** to **message**. As our eyecatcher set the value of **function** to "+ + + + + ". Of course, you do not enter the Quotes.

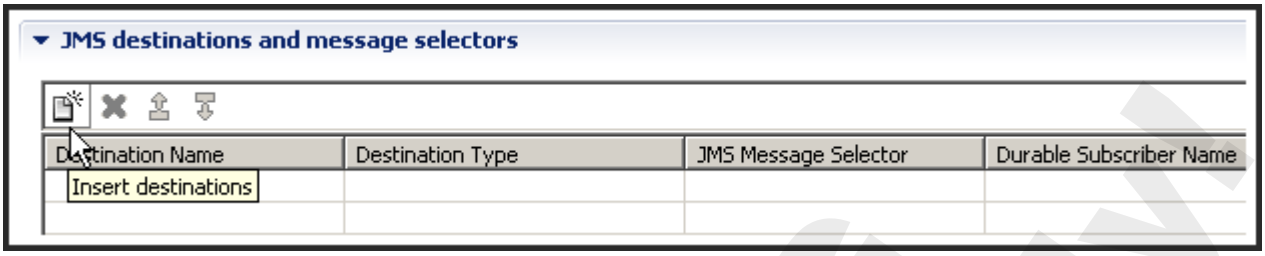


Your finished service should look like this:



6. In the **acme.PurchaseOrder.work** folder, create a new JMS trigger called **listenForValidationJMS.** When prompted, specify that this is a **JMS Trigger**. Configure the **JMS connection alias name** to be **TrainingTopicAlias**.

7. Configure the triggers **JMS destinations and message selectors** by clicking the **Insert destinations** button.

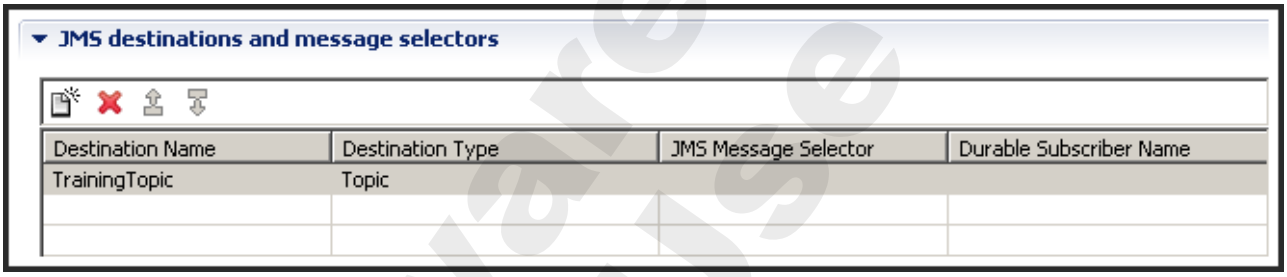


Then enter the following:

a. Destination name = TrainingTopic. If this is not in dropdown, you have to create it by selecting the "create new destination" button.

b. Destination Type = Topic

Leave all other fields empty

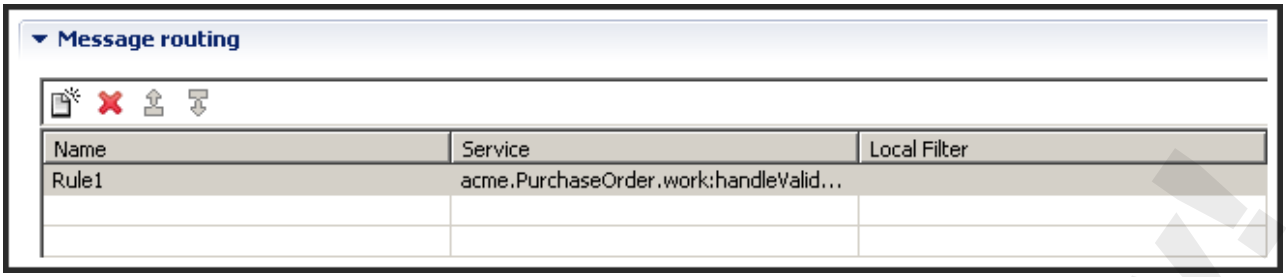


8. Configure the trigger **Message routing** by clicking the **Insert routings** button.

Then enter the following:

a. Name = Rule1

b. Service = acme.PurchaseOrder.work:handleValidationJMS



Then save your trigger.

9. Run the **publishValidationJMS** service. Enter a value of **true** or **false** for the input of this service. This value should be shown, embedded in an XML document, in the Server Log.

Check Your Understanding

1. What is a topic versus a queue?

Exercise 18:
Create Adapter Services

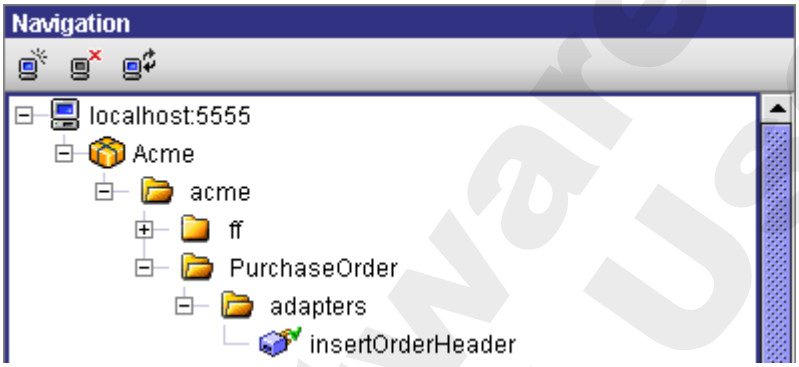
Overview

In this exercise, you will create insert and select adapter services. You combine these with a parent flow service. Take these together and you can easily work with data in a database.

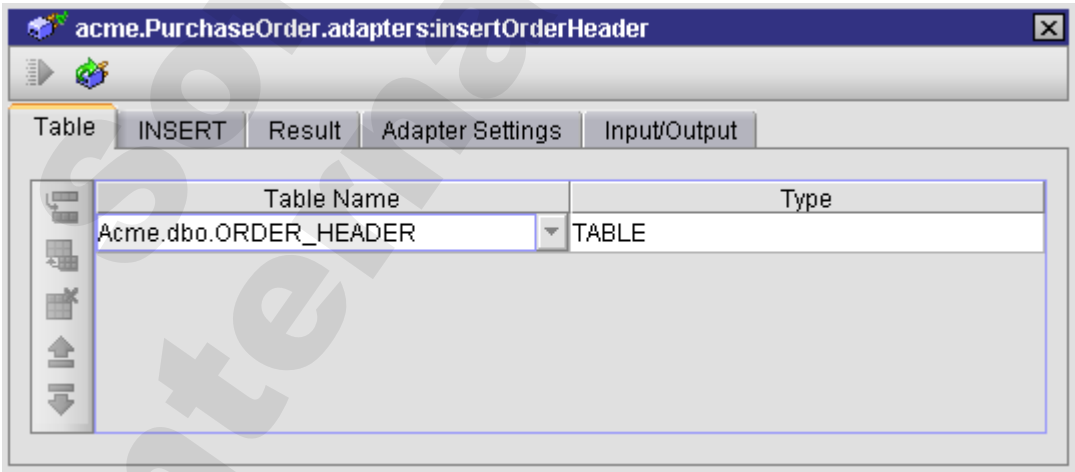
This exercise will be done with Developer.

Steps

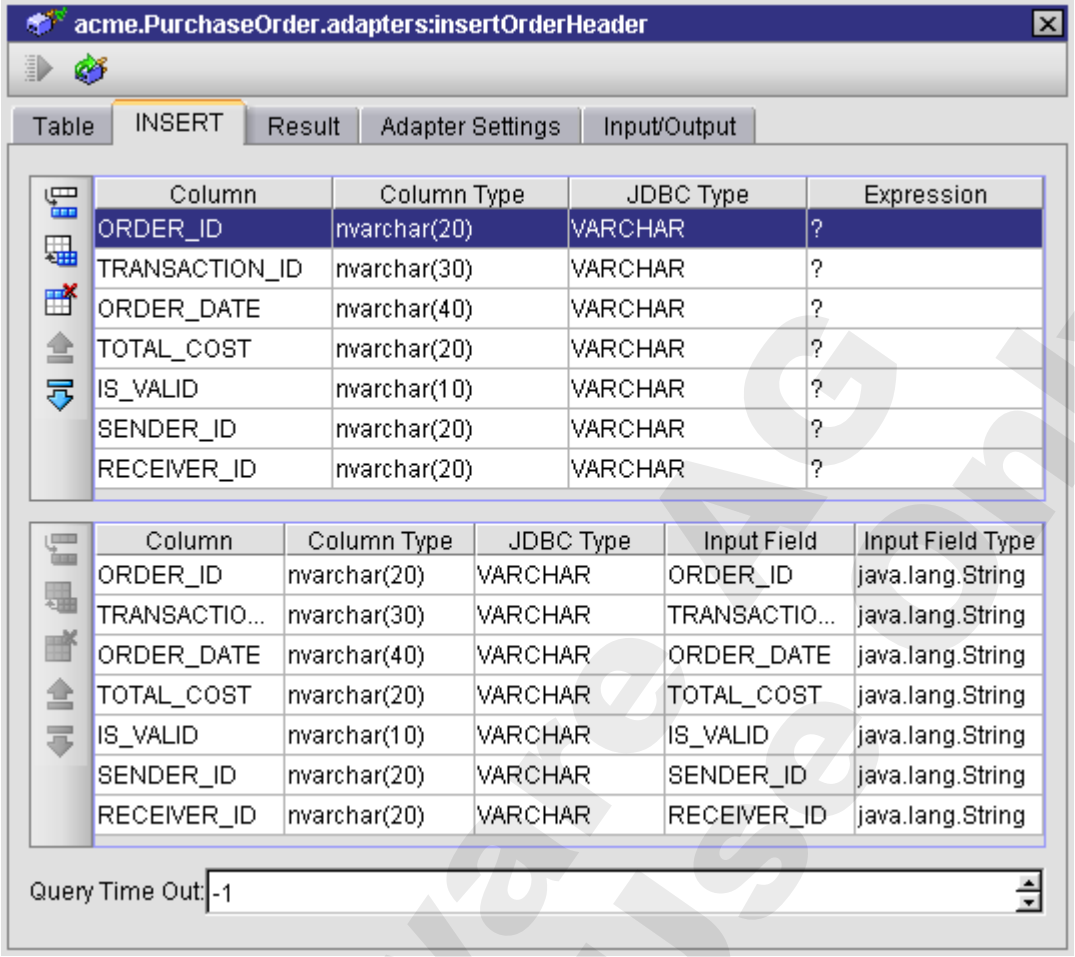
1. Start Developer and create a new Adapter Service in the **acme.PurchaseOrder.adapters** folder. Specify this will be an adapter service, of type **JDBC Adapter**. Then choose the existing **commonSupport.adapters.acmeAdapter** as Adapter Connection Name**.** Finally select the **InsertSQL** as a template, and name your Adapter Service **insertOrderHeader**.



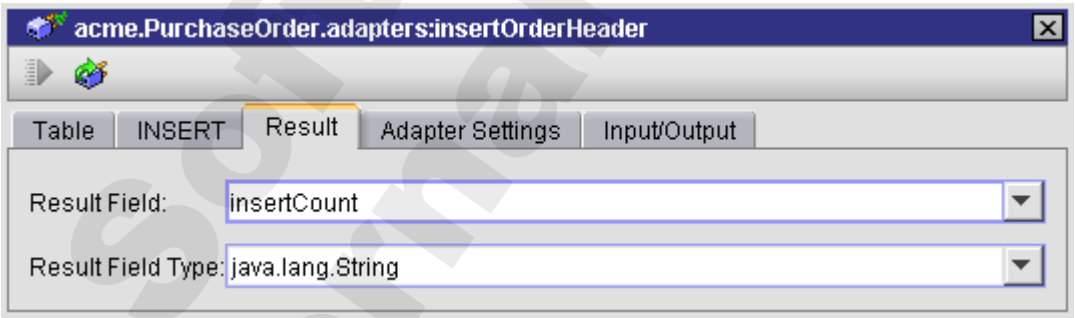
   a. On the Table tab, select **Acme.dbo.ORDER_HEADER**.



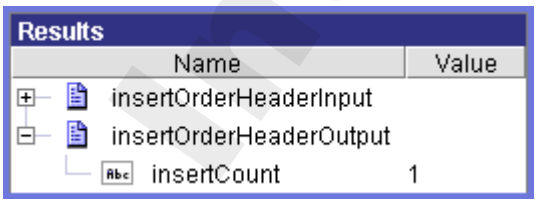
   b. On the Insert tab, click the **Fill in all rows** button. Note that some JDBC Drivers have Problems with this operation. In order to activate the **Fill in all rows** button, you may have to press the **Insert row** button once.



c. On the Result tab, set Result Field to be **insertCount** and Result Field Type to be **java.lang.String**.



2. Save and run the **insertOrderHeader** service. Insert any data (but make sure to insert for every field) and confirm that the **insertCount** returns with a value of 1. Do not provide overrideCredentials or a $connectionName.



3. Like you did in step 1, create another Adapter Service in the **acme.PurchaseOrder.adapters** folder. Specify this will be an **adapter service**, of type **JDBC Adapter**, using **commonSupport.adapters.acmeAdapter,** using the **InsertSQL** template, and name it **insertOrderDetails**.

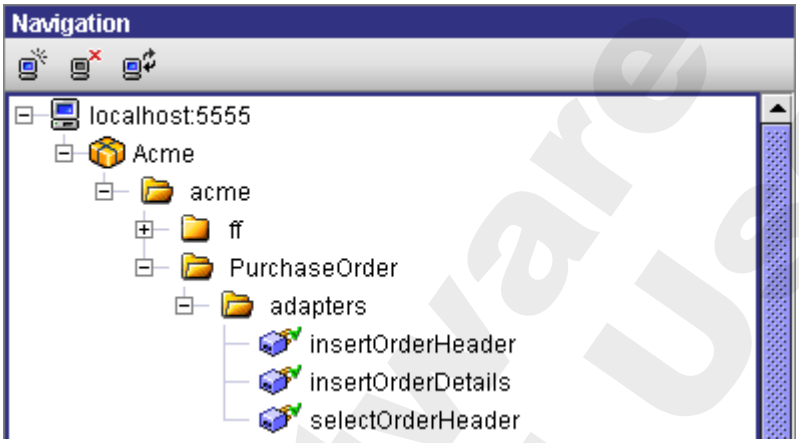
   a. On the Table tab, select **Acme.dbo.ORDER_DETAILS**.

   b. On the Insert tab, click the **Fill in all rows** button.

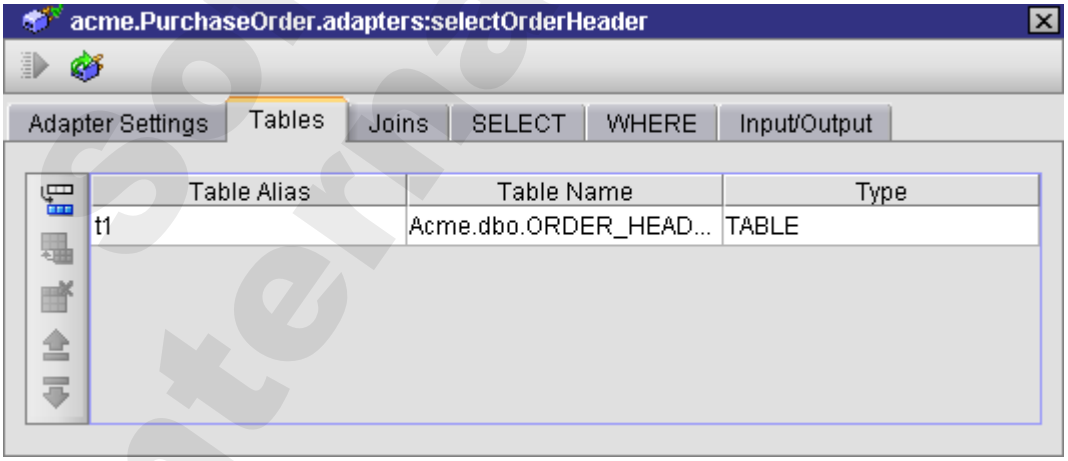
   c. On the Result tab, set Result Field to be **insertCount** and Result Field Type to be **java.lang.String**.

4. Save and run the **insertOrderDetails** service. Insert any data (but make sure to insert for every field) and confirm that the **insertCount** returns with a value of 1. Do not provide overrideCredentials or a $connectionName.

5. Create a new Adapter Service in the **acme.PurchaseOrder.adapters** folder. Specify this will be an **adapter service**, of type **JDBC Adapter**, using **commonSupport.adapters.acmeAdapter,** using the **SelectSQL** template, and name it **selectOrderHeader**.

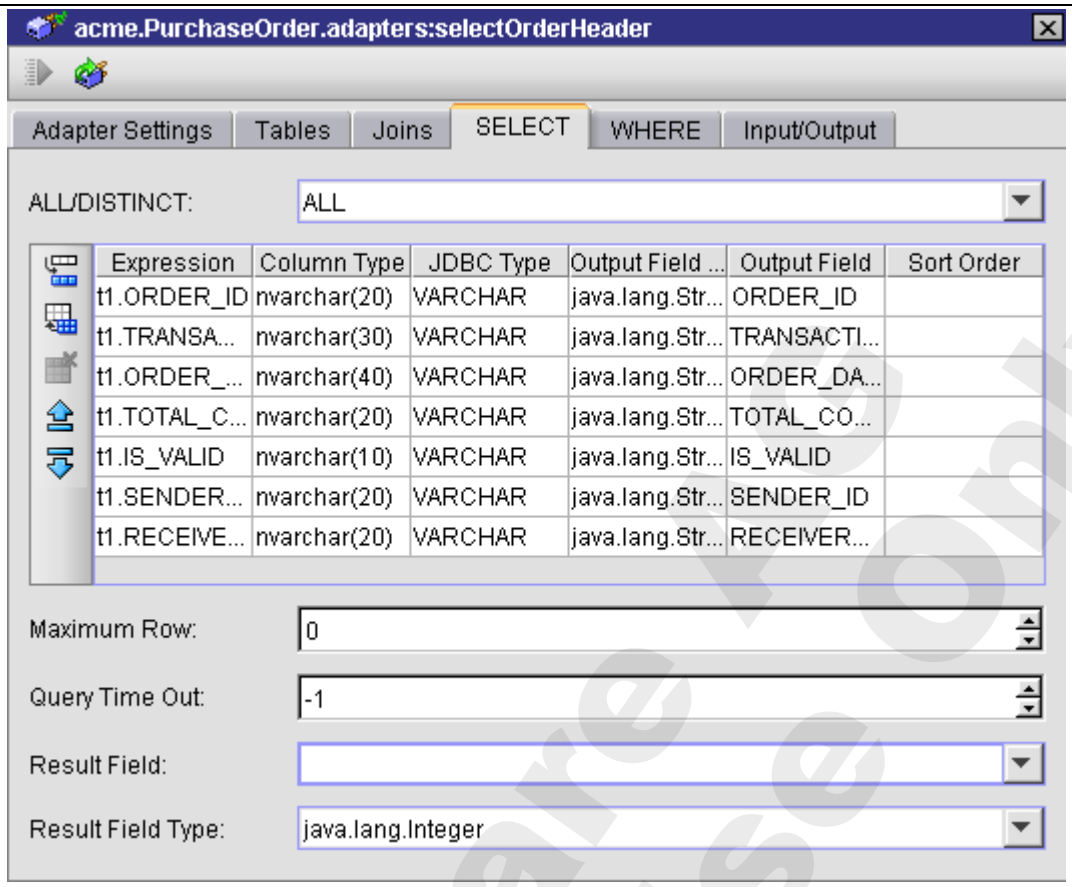


   a. On the Tables tab, in the "**Table Name"** column select **Acme.dbo.ORDER_HEADER**



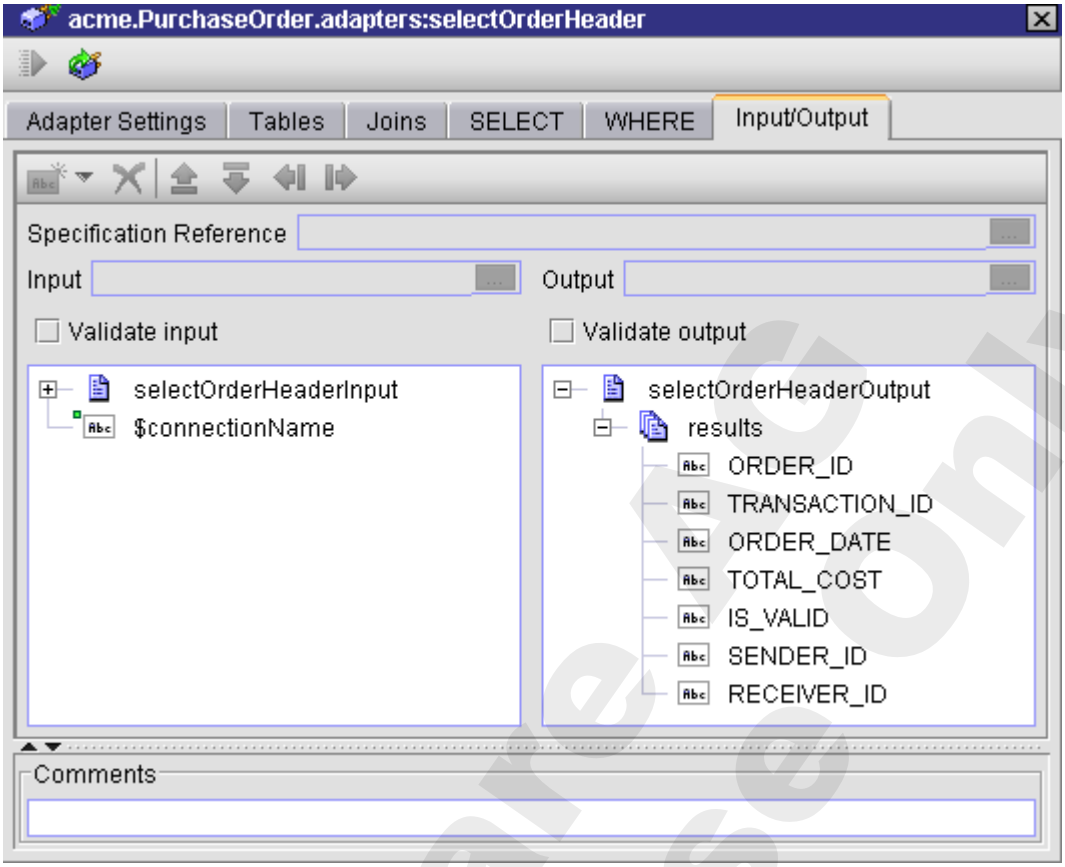
   b. Skip the Joins tab

   c. On the Select tab, specify **ALL**, and click **Fill in all rows**

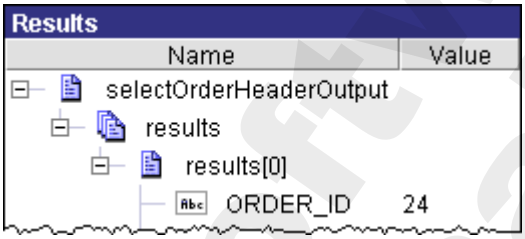


d. Skip the Where tab

e. Review the generated Document type in the Input/Output tab – you should see the database columns that you selected under the Results DocumentList.

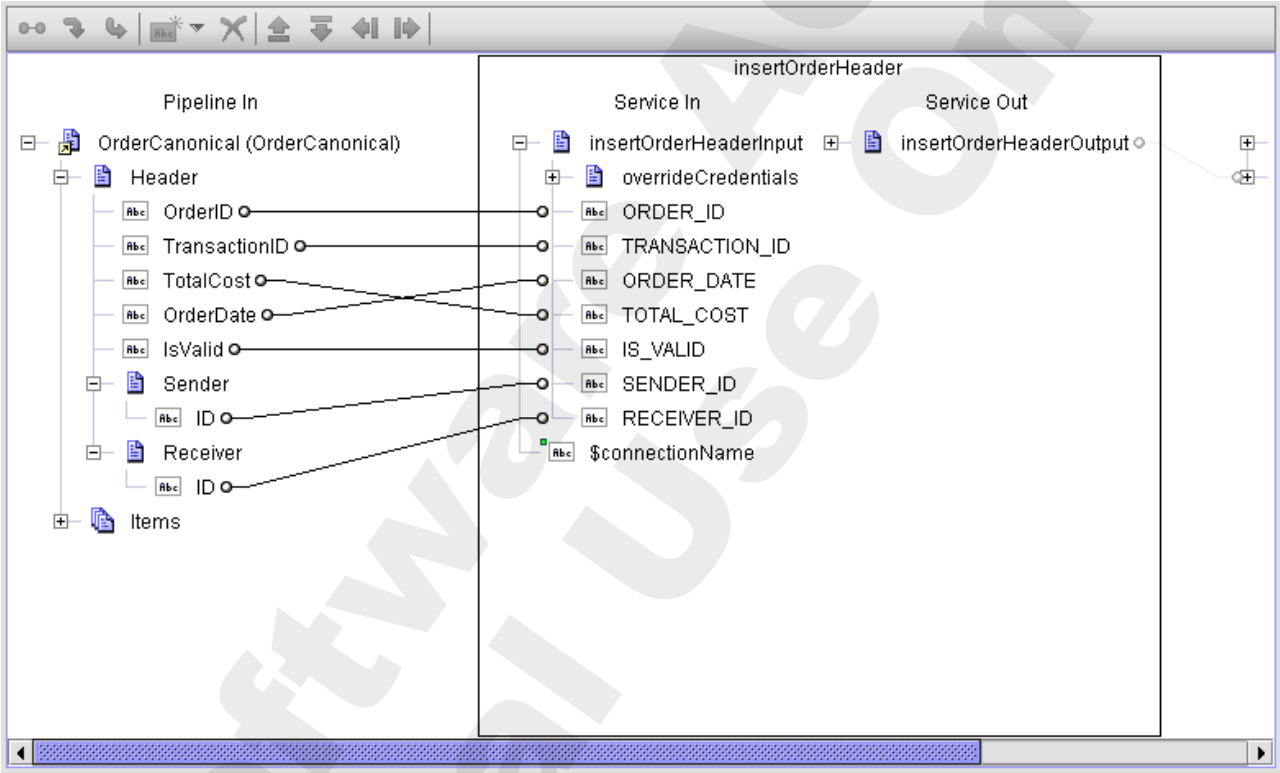


6. Save and run the **selectOrderHeader** service and confirm that the database table was populated with your data from Step 2.



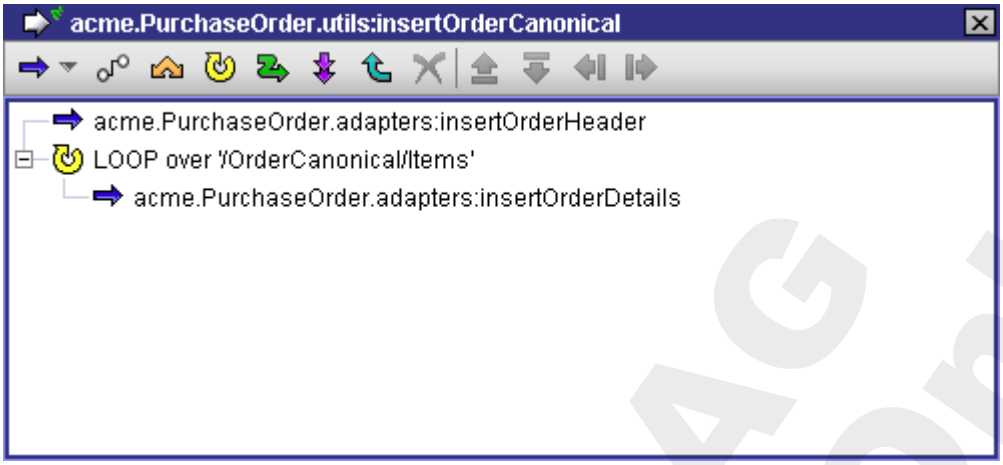
7. Create another Adapter Service in the **acme.PurchaseOrder.adapters** folder. Specify this will be an **adapter service** which is of type **JDBC Adapter** and is using **commonSupport.adapters.acmeAdapter.** Then choose the **SelectSQL** template and name the Adapter Service **selectOrderDetails**.
   a. On the Table tab, select **Acme.dbo.ORDER_DETAILS**
   b. Skip the Joins tab
   c. On the Select tab, specify **ALL**, and click **Fill in all rows**
   d. Skip the Where tab
   e. Review the generated Document type in the Input/Output tab – you should see the database columns that you selected under the Results DocumentList

8. Save and run the **selectOrderDetails** service and confirm that the database table was populated with your data from Step 4.
9. In the **acme.PurchaseOrder.utils** folder, create a new Flow service called **insertOrderCanonical.** Set the input to be a Document Reference to **acme.PurchaseOrder.docs.OrderCanonical** document, named **OrderCanonical**.
   a. Drag **acme.PurchaseOrder.adapters.insertOrderHeader** into your service. Map all the fields from **OrderCanonical/Header** into the similiarly named fields in **insertOrderHeaderInput**. Be sure to include the Sender and Receiver ID fields! Do not map to overrideCredentials or $connectionName.



   b. Add a **LOOP** step and set the Input array property to **OrderCanonical/Items**.

c. Drag **acme.PurchaseOrder.adapters.insertOrderDetails** into your service and indent it under the LOOP step.



In the invocation of the insertOrderDetails make sure that OrderCanonical/Items is no longer an array.

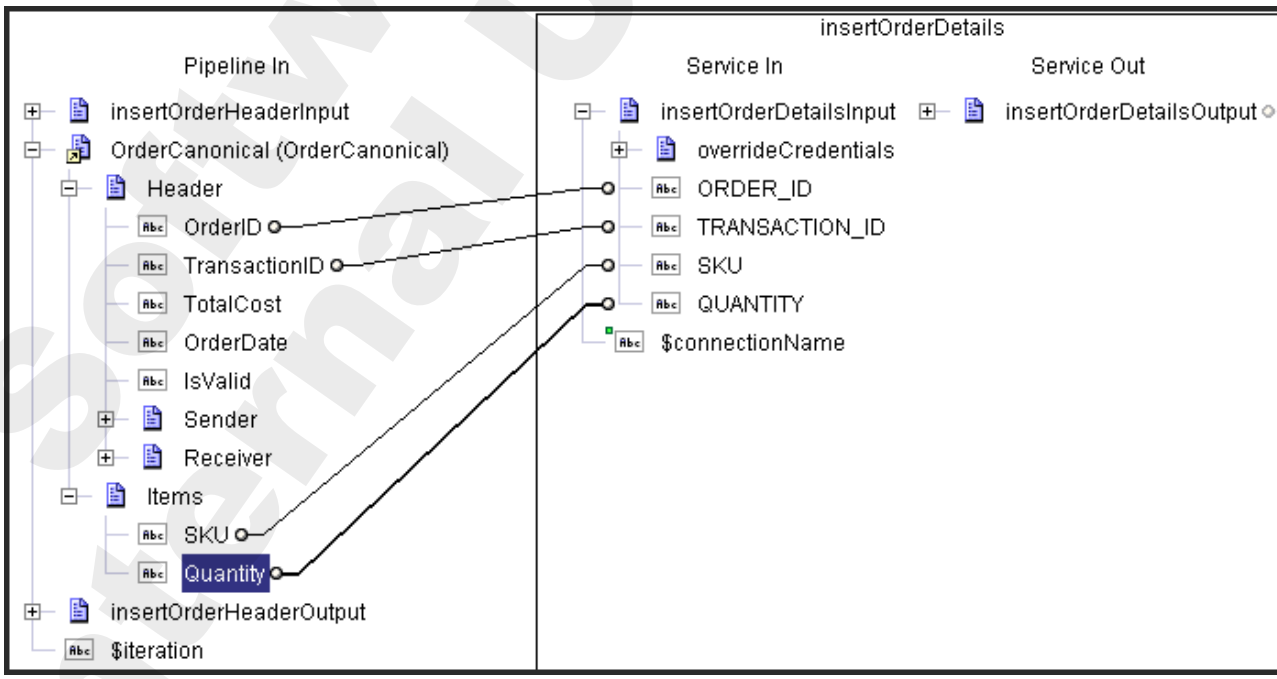
Then map as follows:

i. OrderCanonical/Header/OrderID to insertOrderDetailsInput/ORDER_ID

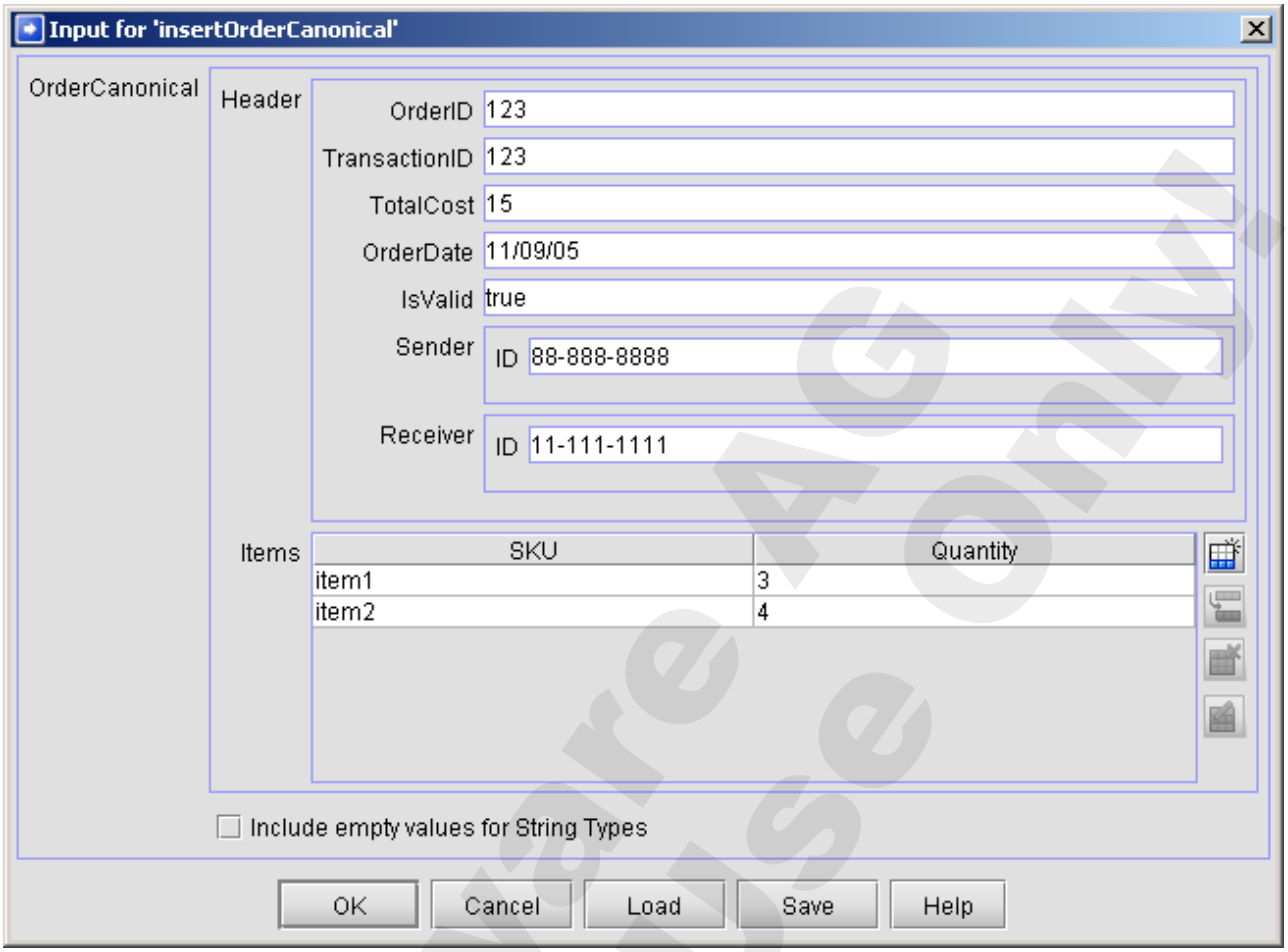
ii. OrderCanonical/Header/TransactionID to insertOrderDetailsInput/TRANSACTION_ID

iii. OrderCanonical/Items/SKU to insertOrderDetailsInput/SKU

iv. OrderCanonical/Items/Quantity to insertOrderDetailsInput/Quantity



10. Save and run the **acme.PurchaseOrder.utils:insertOrderCanonical** service by loading the data from the file **...\IntegrationServer\packages\AcmeSupport\pub\order_canonical_input.txt**.



11. Use the two Select Adapter services (**selectOrderHeader** and **selectOrderDetails**) to check the database tables and verify the **insertOrderCanonical** service worked correctly.

Check Your Understanding

1. What administrative activity must be done prior to using adapter service templates?
2. Why is it important to specify the LOOP input array before mapping the insertOrderDetails fields?

Exercise 19: Adapter Notifications

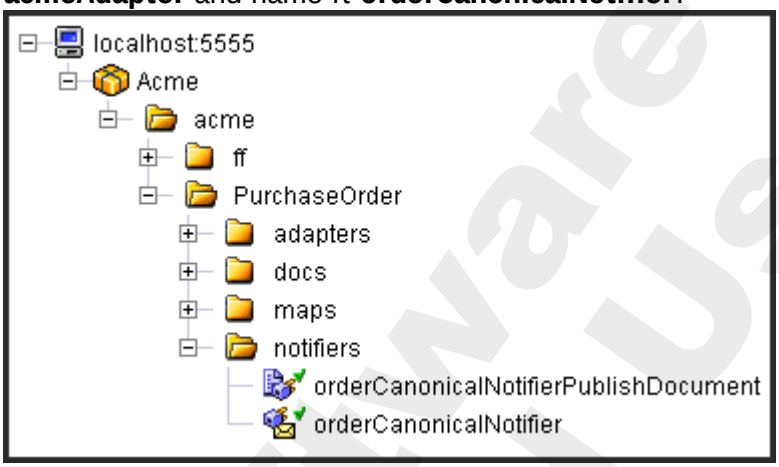
Overview

In this exercise, you will create a notification service to watch for database inserts. As new orders are entered into the database from the ordering interface, a notification document should be published so that interested systems and services can become aware of the insertion.

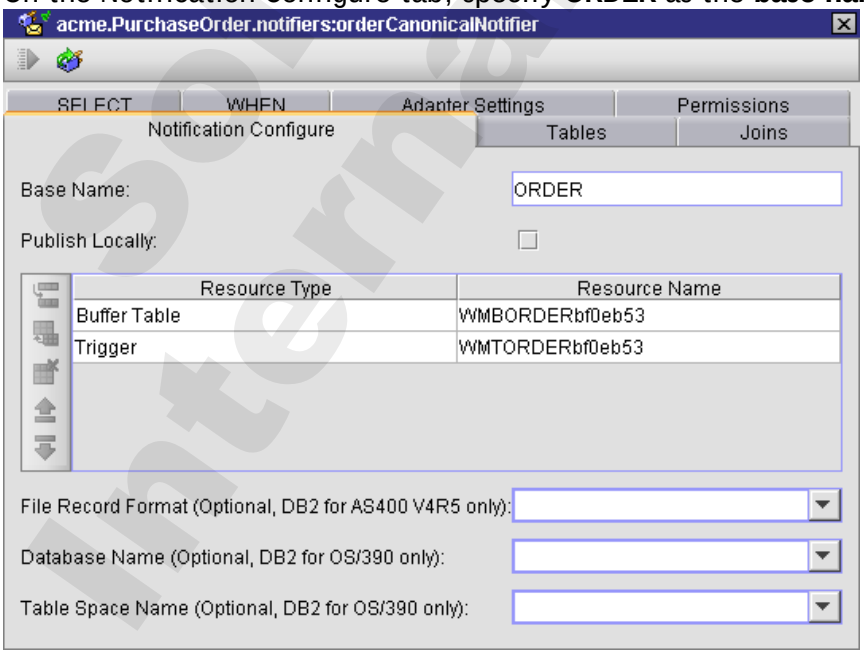
This exercise will also be done using Developer.

Steps

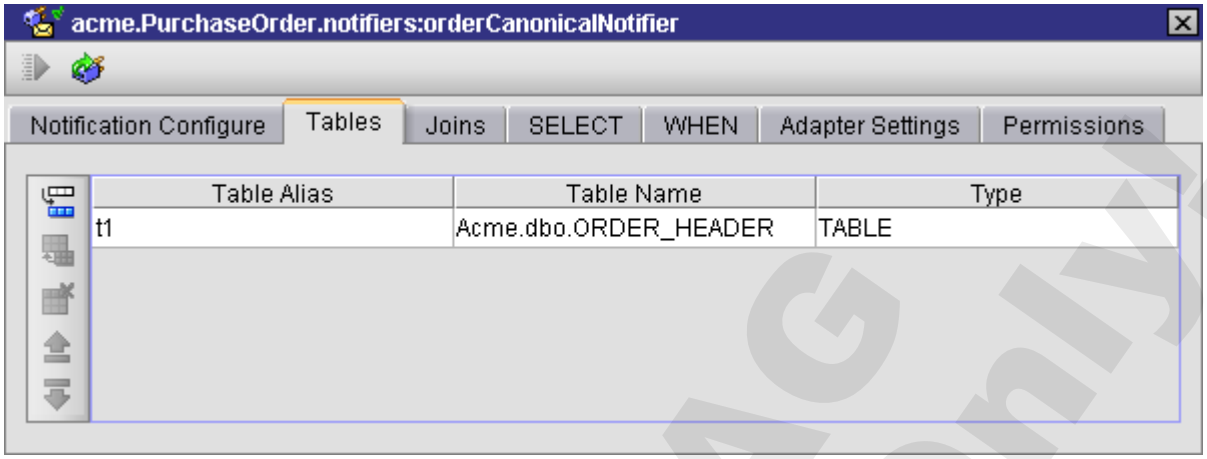
1. Create a new Adapter Notification in the **acme.PurchaseOrder.notifiers** folder. Specify this will be of type **JDBC Adapter**, an **InsertNotification**, using **commonSupport.adapters. acmeAdapter** and name it **orderCanonicalNotifier**.



   a. On the Notification Configure tab, specify **ORDER** as the **base name**.

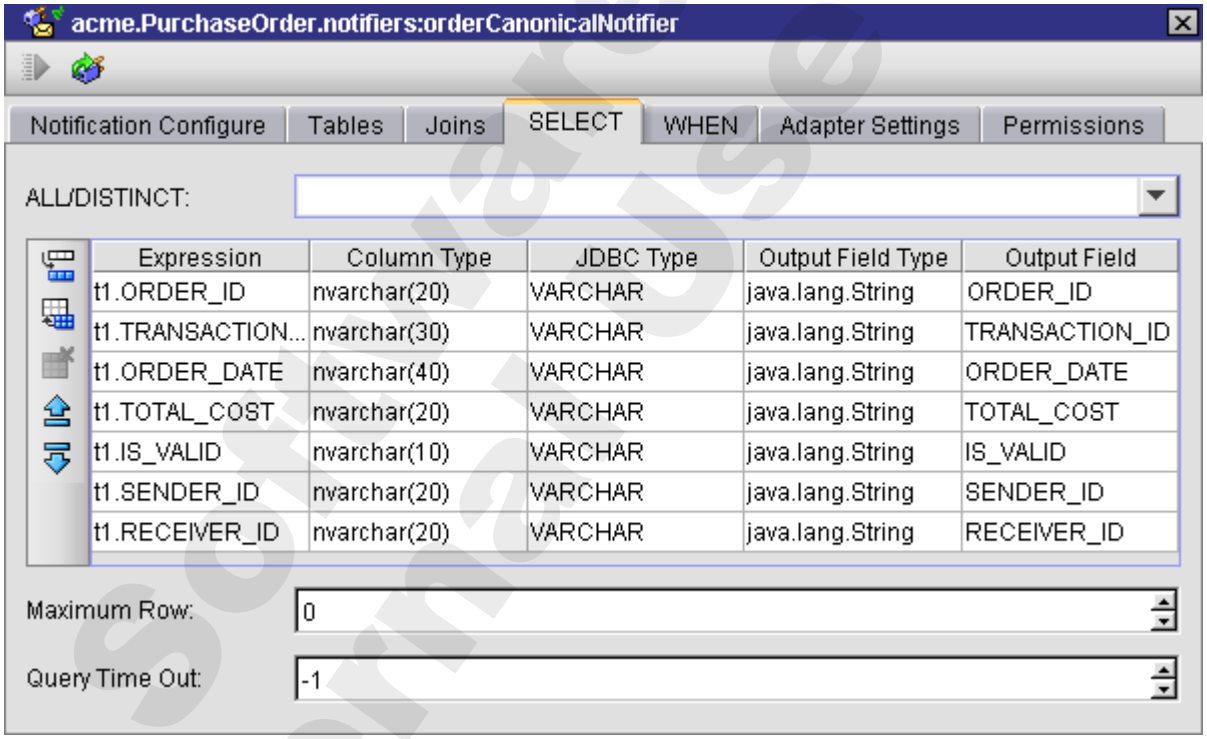


b. On the Tables tab, select the **Acme.dbo.ORDER_HEADER** table.



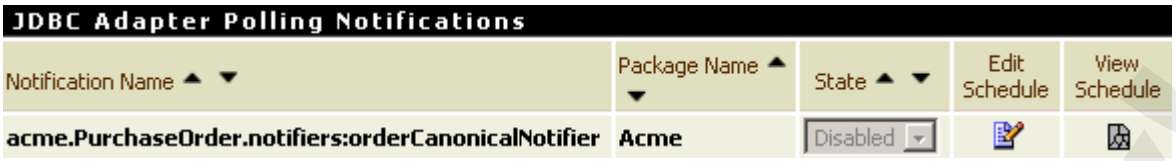
Skip the Joins tab.

c. On the Select tab, click the "**Fill in all rows to the table"** button. Befor this button becomes enabled, you may have to insert the first row by clicking "**Insert Row**".



Skip the When, Adapter Settings, and Permissions tabs and save your service.

2. In the IS Administrator console, select Adapters ➡ **JDBC Adapter** ➡ **Polling Notifications** link. You should see your new notification service listed.

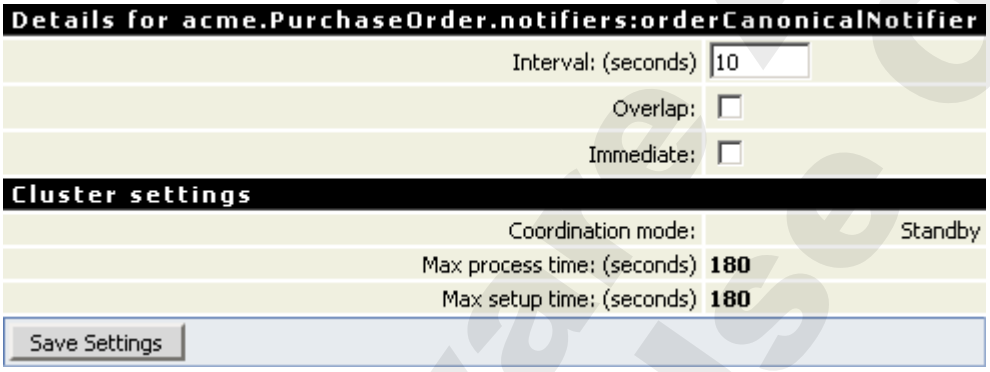


3. Edit your notification schedule by clicking the **Edit Schedule** icon for your notification service. Specify the following parameters and then select **Save Settings**:

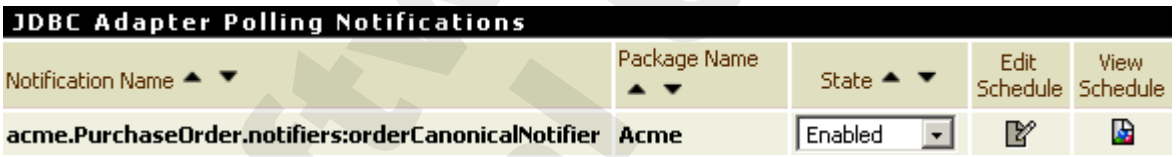
   a. Interval = 10

   b. Overlap = *unchecked*

   c. Immediate = *unchecked*



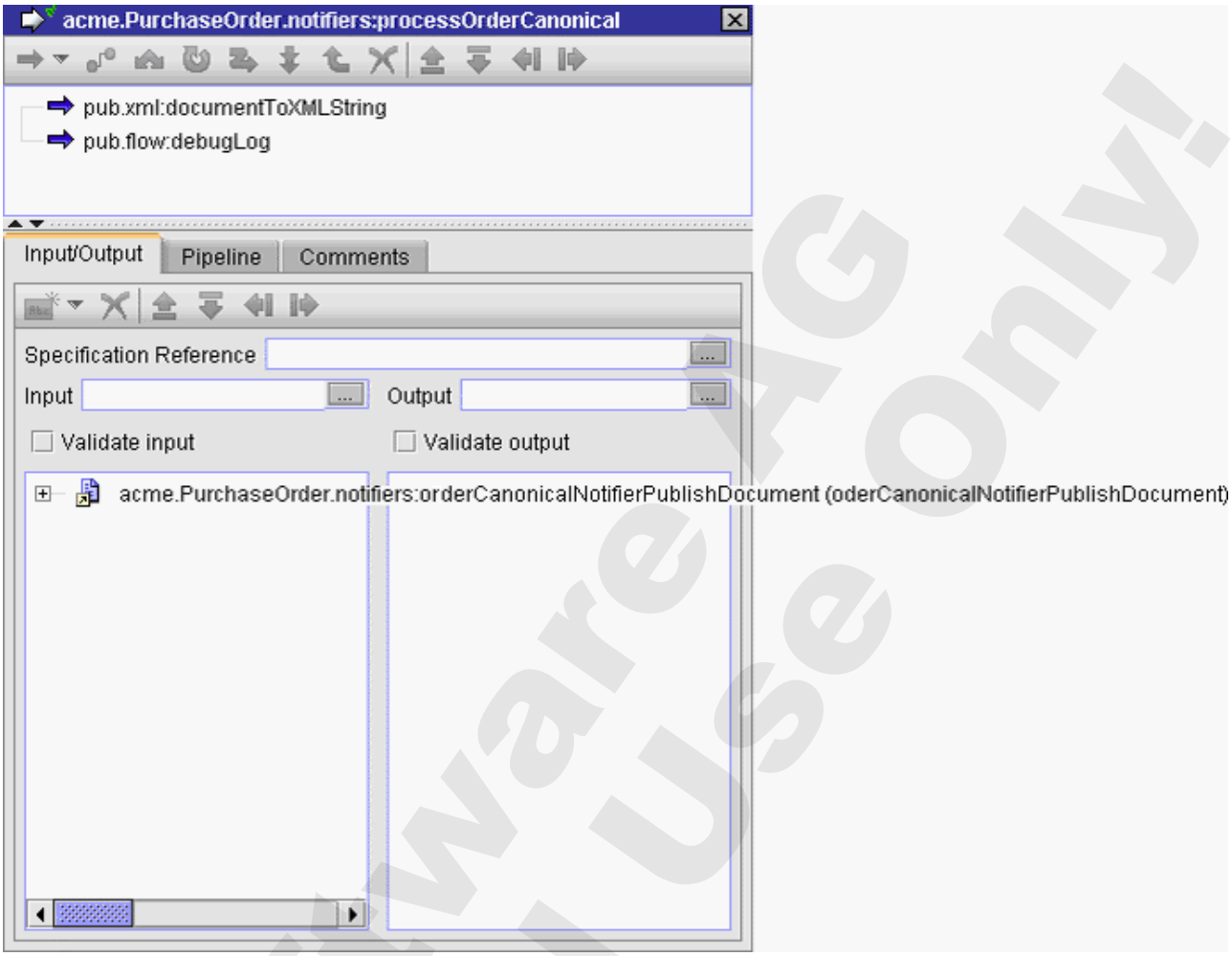
4. Enable the notification schedule.

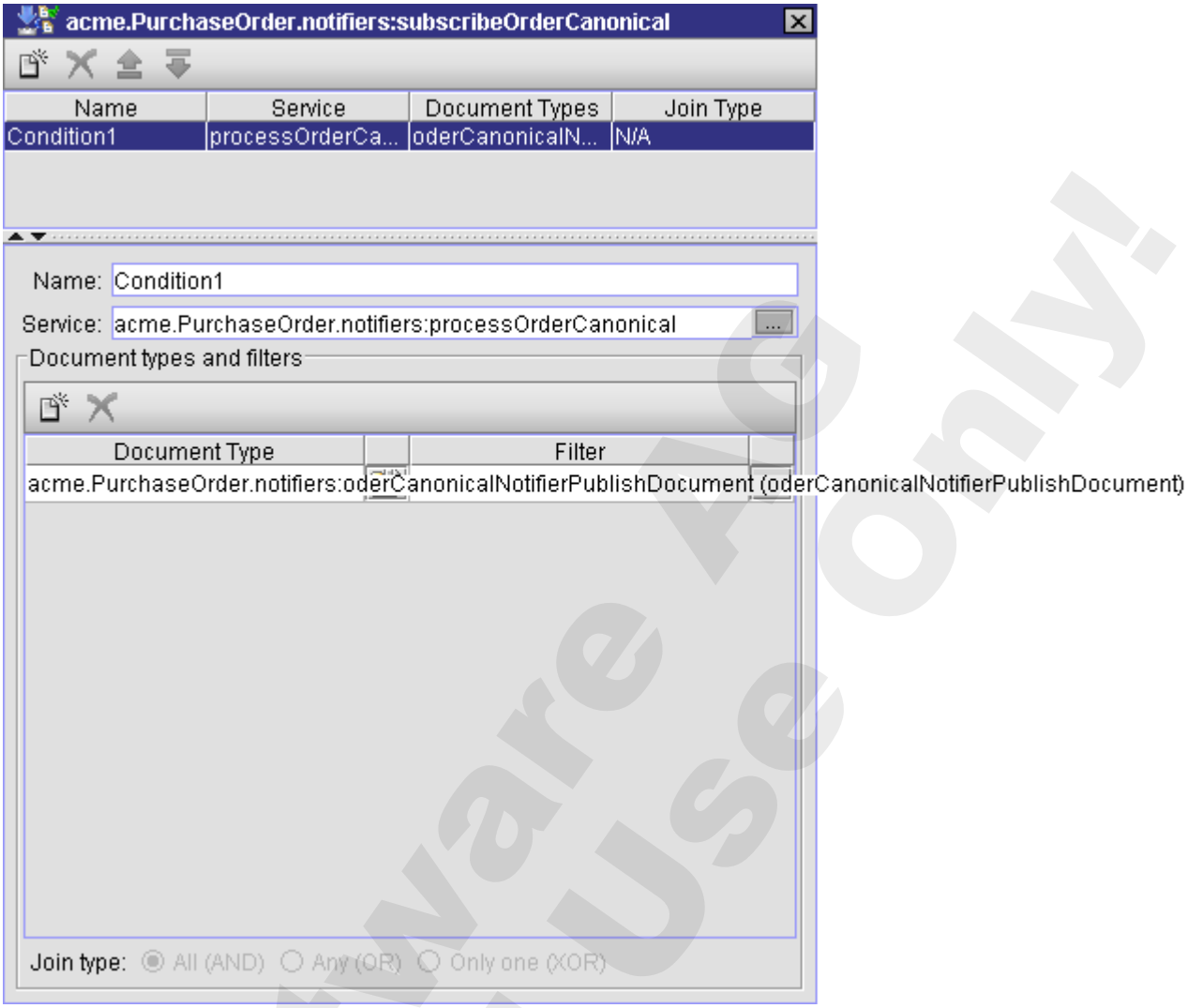


5. As in "**Exercise 16: Broker Pub/Sub**" create a handling Service called "**acme. Purchase Order.notifiers: processOrderCanonical**" and a Trigger called "**acme. PurchaseOrder. notifiers:subscribeOrderCanonical**". Equivalent to Exercise 16 the service receives a document reference to the "**acme. PurchaseOrder. notifiers: orderCanonicalNotifier PublishDocument**" document type. Reminder: Make sure the name of the Document in the handling service input uses the fully qualified name.

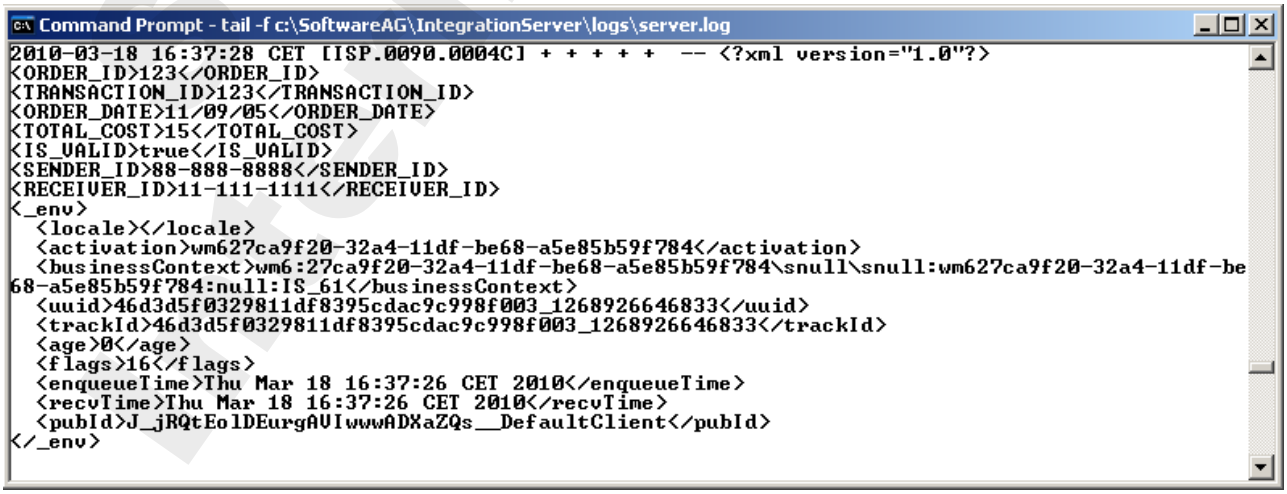
   Use the **documentToXMLString** to convert the document to an XML representation and the **debugLog** service to display the XML document.

6. The trigger subscribes to the abovementioned notification document and invokes the "**processOrderCanonical**" service.





7. Test by running the **insertOrderCanonical** service from the previous exercise. You can load the same file as in the previous exercise. This is **...\IntegrationServer\packages\AcmeSupport\pub\order_canonical_input.txt**. You should see the an XML representation of your orderHeader document within the polling interval (10 Seconds) in the Server Log.



Check Your Understanding

1. What is automatically created and updated when you work with your notification service?
2. What occurs when the notification schedule is enabled?
3. Why is the notification schedule an administration task?

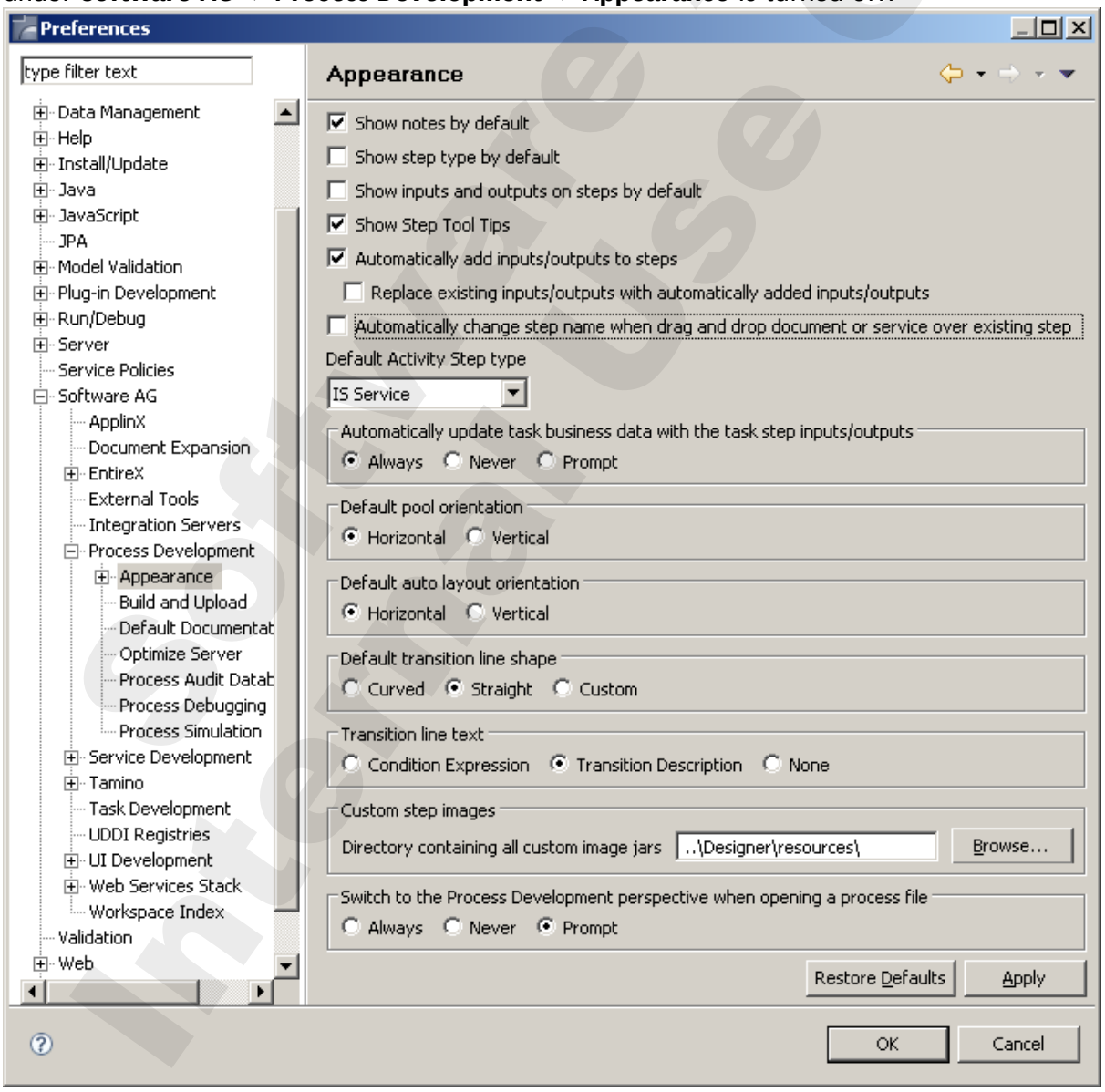
Exercise 20:
Use Services In a Business Process

Overview

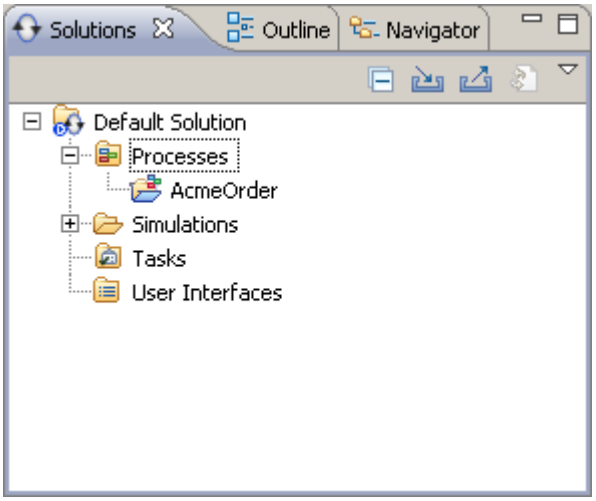
Use Designer to modify an existing business process to subscribe to a starting document, include services from previous exercises, and incorporate an error handling service. Build and update the model, and then test your process in Designer.

Steps

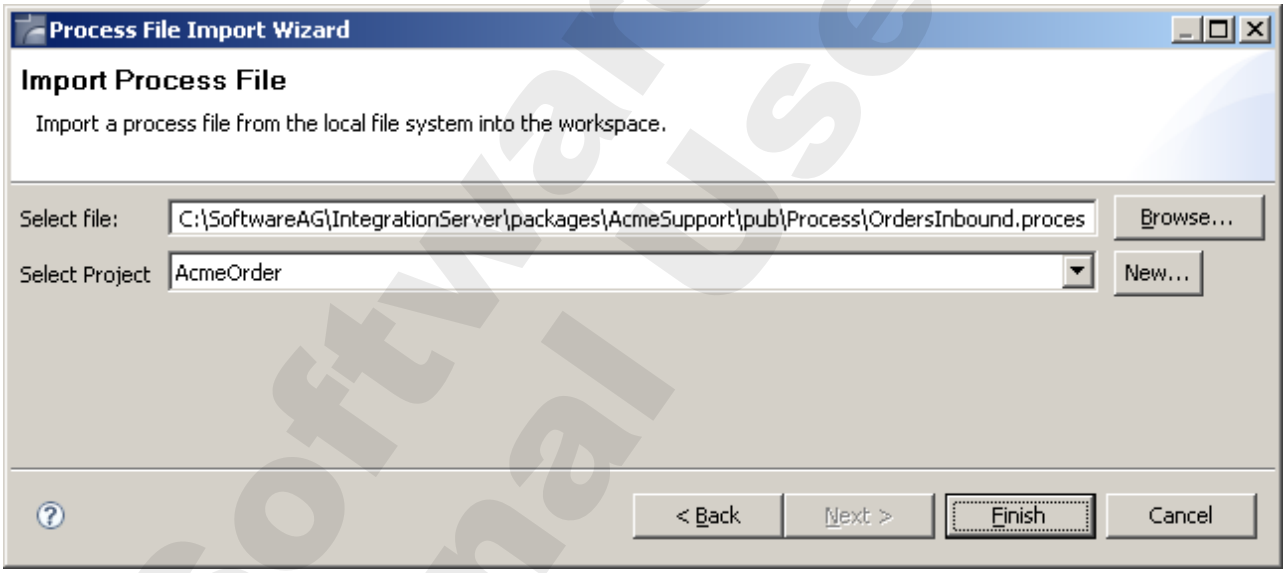
1. Open Designer and select the Process Development Perspective. Open the preferences view under "**Window**" ➡ "**Preferences**" and verify that the Property "**Automatically change step name when drag and drop document or service over existing step**" which can be found under **Software AG ➡ Process Development ➡ Appearance** is turned off.



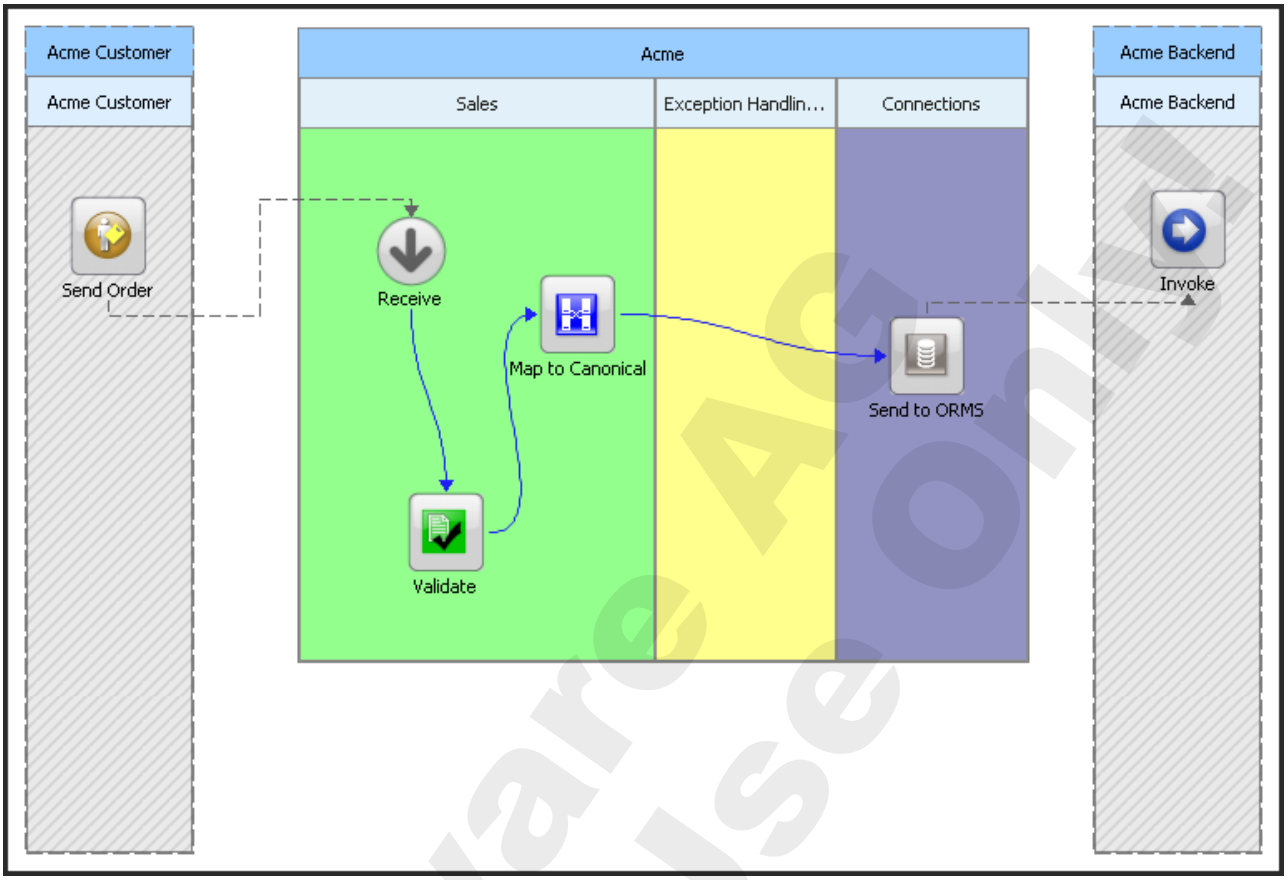
2. Using the File menu **create** a Process Project called AcmeOrder.



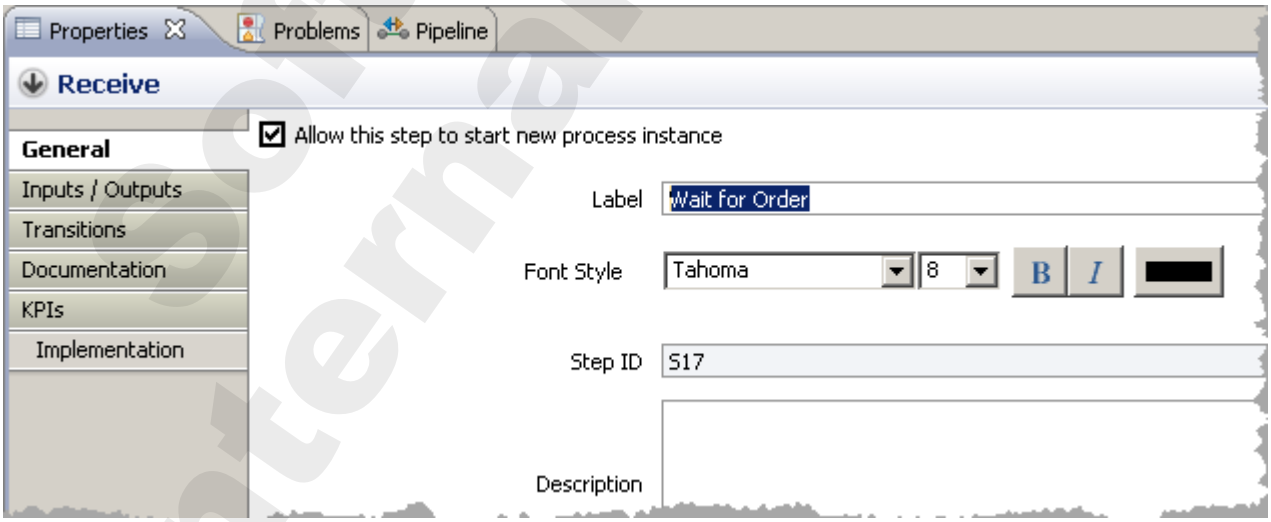
3. From the File Menu select **Import** from an **import source** of **SoftwareAG ➡ Process File. Click Next** and choose the directory **...\IntegrationServer\packages\AcmeSupport\pub\Process**. Select the file **OrdersInbound.process** and the project AcmeOrder.



4. Examine the flow of the process so that you understand what it is intended to do.

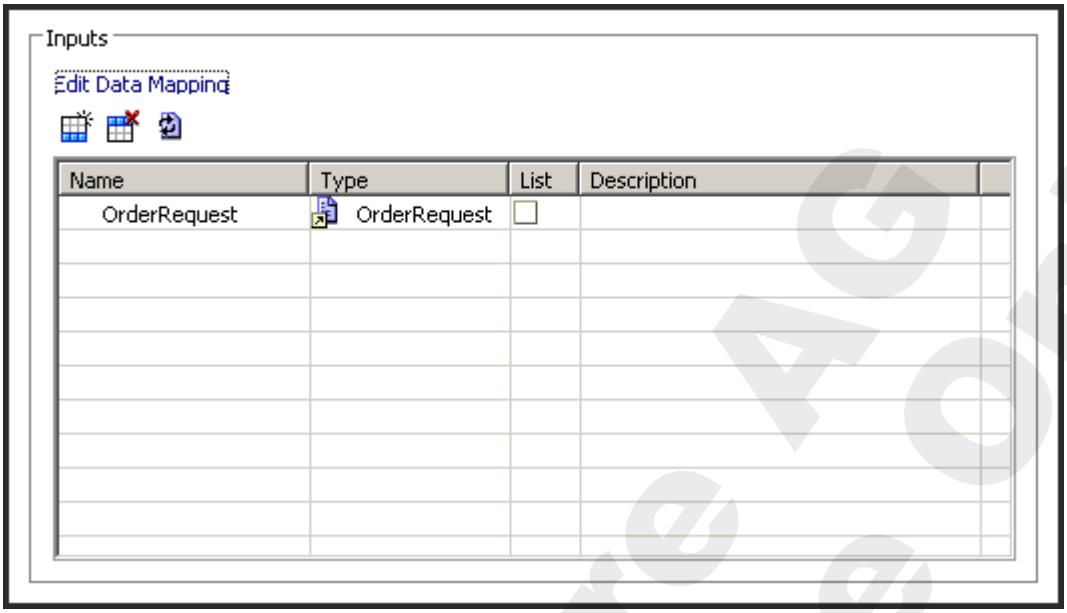


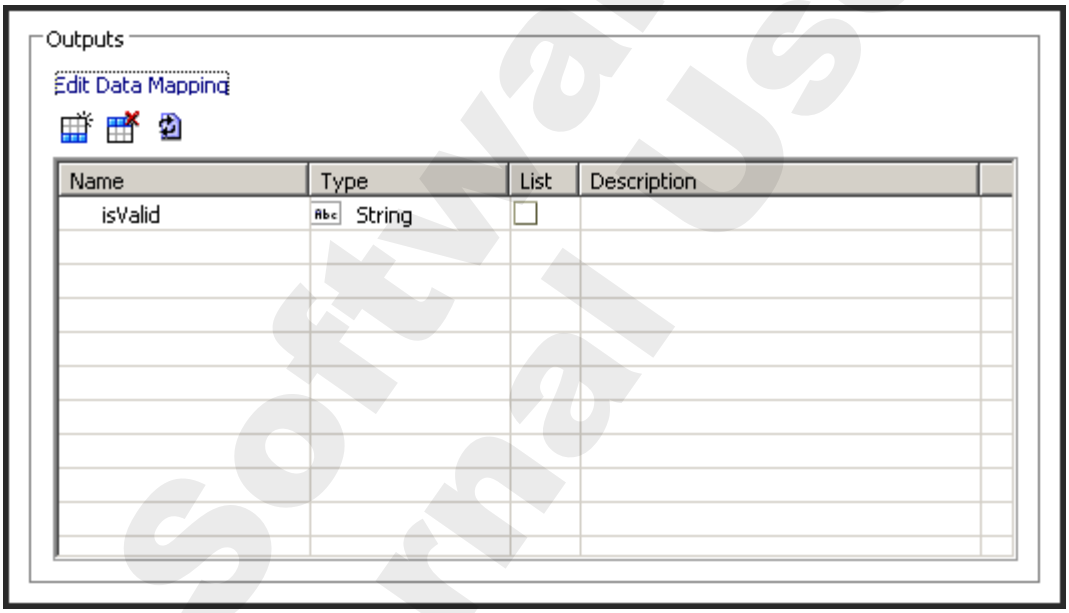
5. Modify the **Receive** step in the process. First, change the name of the step to **Wait for Order**.



In the **Implementations** tab, set the **Receive Document** field to **acme.PurchaseOrder. docs. request: OrderRequest**.

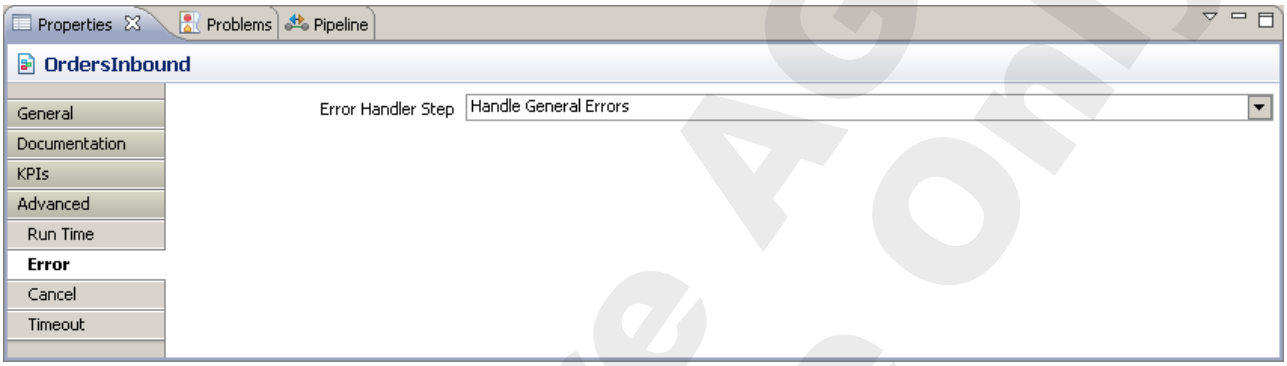
6. Modify the **Validate** step Implementation properties to call the service **acme.PurchaseOrder.utils:inspectLineItems**. Then modify the Inputs/Outputs property and tell Designer to **Add Inputs from Service Signature** (). Repeat this step for the Service Outputs.



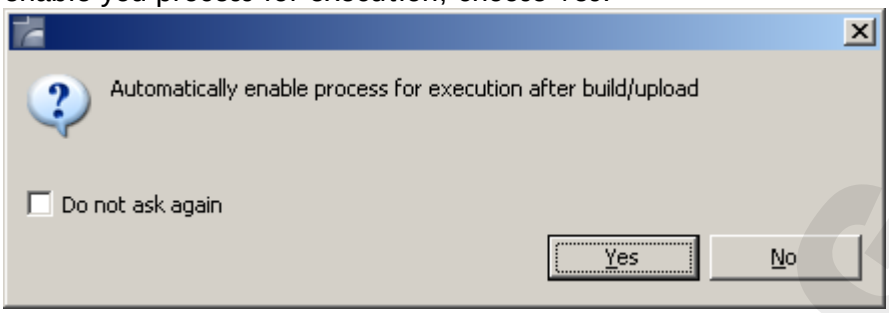


7. Modify the **Map to Canonical** step as you did for the Validate step, but this time call the service **acme.PurchaseOrder.maps:orderRequestToCanonical**. You can set the service also by dragging the Service from the Package explorer view to the "**Map to Canonical**" step. Bring in the Inputs/Outputs from the service signature.
8. Modify the **Send to ORMS step** as you did for the two previous steps, but this time use the service **acme.PurchaseOrder.utils:insertOrderCanonical**. Remember to bring in the Inputs/Outputs from the service signature.

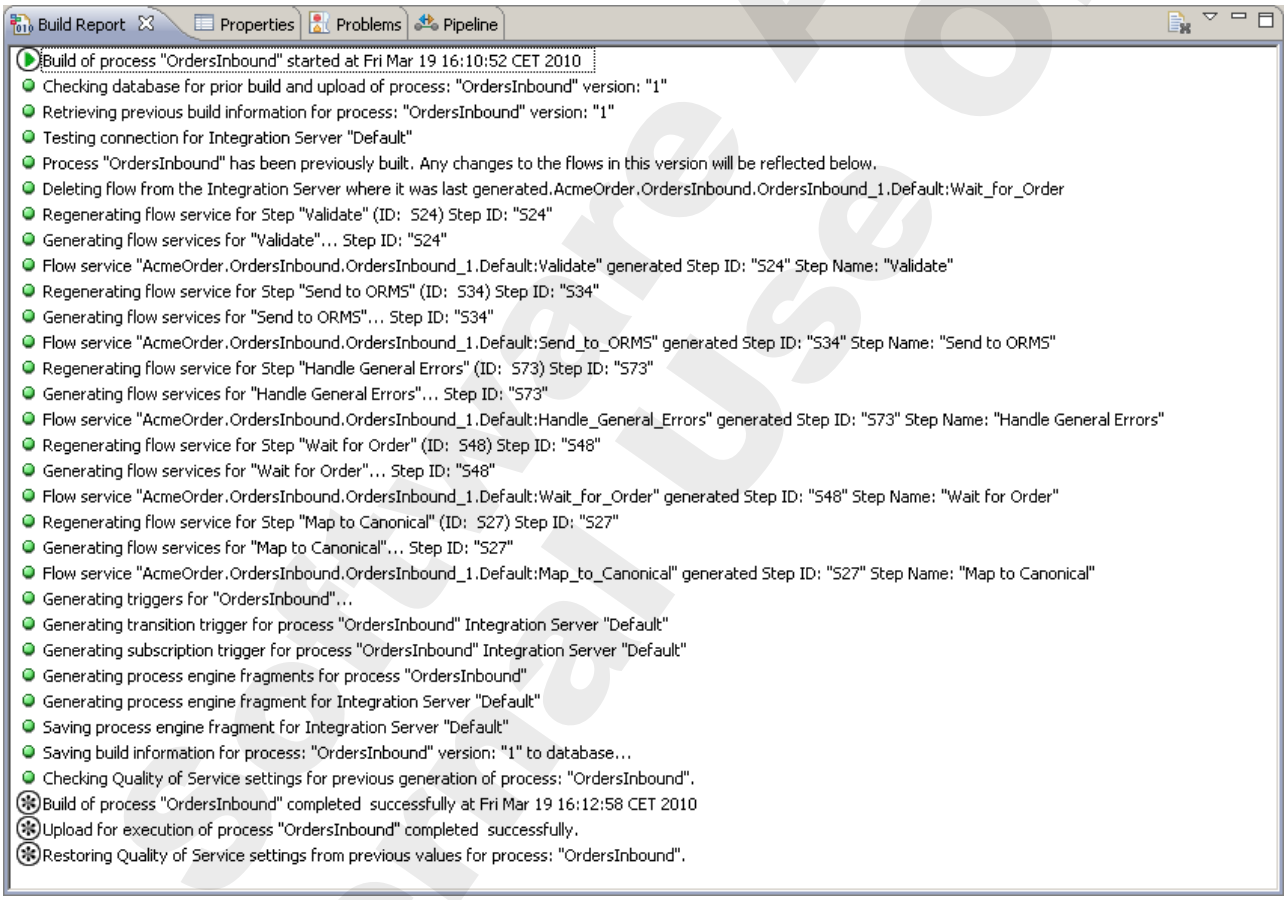
9. Now use the drag and drop development feature to add a error handler step for the process. Locate the Package Navigator view in Designer and expand the **AcmeSupport** package. Find the service **acmeSupport.process:processErrorHandler** and drag it into the Exception Handling swimlane. Do not connect it to any other steps with a transition. Rename the step to **Handle General Errors (Properties ➡ General)**, and change the image (right click on the process step) to the stop sign with the X in the middle. Then select it as the **Error Handler Step** on the process level **Error** properties. To get access to the process properties click somewhere in your process diagram where there is only white background.



10. Save the process. Then **build and upload** the process by clicking on the  Icon. Check for errors in the Build Report, and make corrections as necessary. If asked wether you want to enable you process for execution, choose Yes.



Your build report should look like the following:



11. Open the Service Development Perspective, and review the generated package **AcmeOrder** and generated services. Make sure the mapping for **Handle General Errors** is correct and modify if necessary.
    Note: you may have to refresh the Package Navigator view to see the generated code. To do this right-click on Default in the Package Navigator and select Refresh (this will reload everything under the Integration Server packages folder)
12. Switch back to the Process Development Perspective and debug the process by clicking the debug button ( ). When prompted for inputs to the OrderRequest document, load the file **...\IntegrationServer\packages\AcmeSupport\pub\publish_order_request_input.txt**.

    Continue through the process in debug mode, using the Step Into, Step Over, or Step Out buttons in the Trace window.

Check Your Understanding

1. Why do you create swimlanes within a process? What effect do they have on the execution of the process?
2. What occurs throughout the system when you perform a build and upload from Designer?
3. What are the different start mechanisms for a process?

This page intentionally left blank

Check Your Understanding: Anwers to the Questions

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

1. What is the URL to access the Integration Server? **http://localhost:5555/**
2. What is the URL for MWS? **http://localhost:8585/**
3. Why is the Broker Monitor set to Automatic start, but not the Broker Server? **The Broker Monitor is responsible for starting and monitoring the Broker Server. If the Broker Server terminates for whatever reason, broker monitor tries to restart it again.**

Exercise 2: Packages and Folders

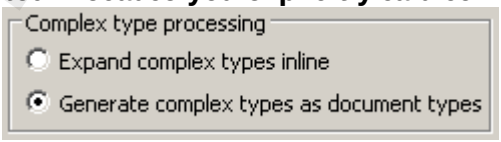
1. If you place the folders in the wrong parent folder, how could you correct it? **You can drag and drop folders with the mouse.**
2. Why is a consistent folder structure in all packages important? **This makes it easier for new people on a project to understand the inner workings of a package.**

Exercise 3: Create a Service

1. Why is the order of the services important? **The order of service invacation determines the runtime behaviour of your own service.**
2. How many inputs can the pub.string:toUpper service accept? **It accepts 4 Inputs. All inputs marked with a small squere like language, country and variant are optional and have a default value that usually does "the right thing" ™. For the default values being used and the possible values that can be set, consult the services documentation in ... \ _documentation\Developer\8-0-SP1_Integration_Server_Built-In_Services_Reference.pdf**
3. Where would the server log appear if the server is not running through the Command Prompt? **Either in the WEBMDB database or in the server.log file, depending on how logging is set up.**

Exercise 4: Document Types

1. Why are additional documents created in the acme.PurchaseOrder.docs.request folder when the OrderRequest schema is imported? **Because you explicitly said so when importing the schema by leaving the checkbox**

   on its default value "Generate ... as document types". The generated Document type docType_PartnerRoleDescription is reused in several places inside the OrderRequest document, making maintenance of it much easier.
2. What is the benefit of using a schema for import over using a DTD? **Both DTD and XML schemas allow the same functionality. So there is not much of a difference. But DTD's are written in their own language, while schemas are written using XML syntax and can be processed with the standard XML toolset. Furthermore DTD's are more oriented towards text based documents (books) while schemas are oriented towards data based documents (orders, invoices).**

3. What are the two ways to indent a document type element under another? **You can use drag and drop with the mouse or the arrow buttonw in the top toolbar**

Exercise 5: Flow Services - BRANCH

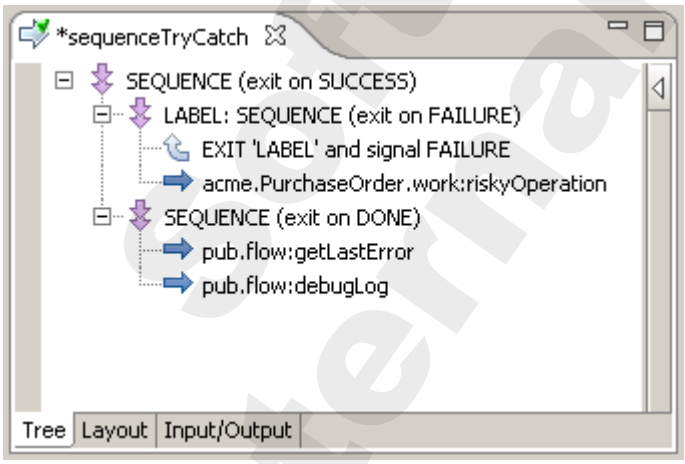
4. When are regular expressions useful in branch? **When you want to perform an operation on a set of similar input values.**
5. Can you combine a switch variable with Evaluate labels=True? **No, this wouldn't make sense.**
6. What are the special test values that can be used as labels in a branch statement? **Those values are "$default", which is used when none of the other input values matches and "$null", which is used when the switch variable is not specified. The usage of "$null" makes no sense when the "Evaluate labels" option is set to true.**

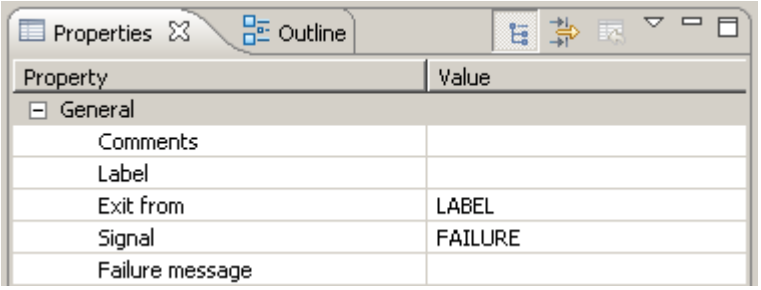
Exercise 6: Building Flow Services - LOOP

1. What would happen if the MAP step is not indented under the Loop? **There MAP step would be executed only once. The value of $iteration outside the loop is not defined.**
2. How many employees could you have added? Does Loop have a limit? **There is no limit set by the LOOP statement itself. It can handle arbitrary large arrays.**
3. Why do you want to use document references rather than creating the document in the service input? **Sending in the document as argument to the service allows for more flexibility when using the service.**

Exercise 7: Building Flow Services – SEQUENCE

1. Rather than using a service you know will fail, how can you throw an Exception in Flow? **You can use an EXIT 'LABEL' statement with signal set to FAILURE like in the screenshots below:**





2. What happens if the riskyOperation service works (doesn't fail) ? **Only the 1st and the 2nd SEQUENCE statements get executed. The content of the file is read into memory and is returned to the caller.**

Exercise 8: Validation Service

1. Why did we set isValid to true at the very beginning? **isValid is an Output variable. As such, it should always been initialized.**
2. Is there another way we could have validated this particular value with writing Flow or Java? **We could have written a Java service, but dealing with such nested structures in Java produces deeply nested code that is hard to maintain. When using Flow, we could have used other loop (eg REPEAT) or mapping (eg using Indexes) functionality.**

Extra credit solution:

Exercise 9: Mapping Service

1. How is a transformer different from a normal service? **It requires explicit assignment of its mappings. It does not do implicit mapping. Transformers do not receive a full copy of the pipeline and all transformers in a MAP step can execute in parallel.**
2. What if the transformer you want to use is not in the transformer drop-down list? **You can use any service as transformer by selecting the browse (Browse...) button at the bottom of the transformer dropdown.**
3. Why did we need to LOOP over ProductLineItems? Why not just map from ProductLineItems to Items? **Because we needed to apply transformers to some of the source values. The structure of the two documents is different as well.**

Exercise 10: Create a Java Service

1. What exactly is each line of the Java code doing in the endsWith service? **See the inline comments below:**

```
// get a cursor to access the pipeline

IDataCursor cursor = pipeline.getCursor();

// retrieve the two input values from the pipeline. NOTE: there is
// no test that these values are actually present!

String string = IDataUtil.getString(cursor, "string");
String suffix = IDataUtil.getString(cursor, "suffix");

// compute the value to be returned

String value = string.endsWith(suffix) ? "true" : "false";

// store the return value in the pipeline

IDataUtil.put(cursor, "value", value);

// destroy (release) the cursor, as we do not need it any longer.

cursor.destroy();
```

2. Is the service thread safe? What would you have to do if not? **Yes, it is. Because it's not using any shared state and it's not calling any method that's not thread safe. Otherwise you would have to add the appropriate synchronization primitives to protect such shared state.**

3. How could the cursor handling be improved? **The cursor should be destroyed before we start the computation of the result (The line calling string.endsWith()). To store the results in the pipeline another cursor should be created using the getCursor() method of the pipeline object. The reason for this is, to have the cursors allocated for as short as possible. You should do this whenever you invoke a method that might require some time to compute its result. In the present example the overhead caused by cursor management outweighs the benefits of a shorter cursor lifetime.**

Exercise 11: Monitoring Services

1. Why is it necessary to create remote server aliases? **By design, MWS will communicate only with one instance of Integration server. When resubmitting a service invocation, MWS tells this Integration server where it wants the service instance to be scheduled for execution. To do so, MWS is sending the name of the remote server alias that should be used to resolve the final execution server.**

2. Under what circumstances would it be acceptable to resubmit a service? Why? **Those circumstances depend only on the service execution to be resubmitted. If a failed service had already executed half of its statements, then those statements may have caused some state changes where it may not be viable to do those changes again**

(Imagine a service giving a 3% raise to all employees, that failed after processing the first 100 employees).

Exercise 12: Invoking Services

1. Why and when would you use an HTTP URL alias for your services? **You can use an HTTP URL alias if the name of the service, to be called by your clients might change from time to time. It also allows invoking services using a much shorter URL.**
2. How do the services find their input data? **They all depend on the presence of the node object in the input pipeline. This is a parsed representation of the XML document that was sent to integration server.**
3. How do the services return their result? **They return their result by XML encoding the content of the output pipeline.**

Exercise 13: Create a Flat File Schema

1. Why can't flat files be imported like XML documents? **Because a flat file contains no metadata like field names. Also it would be pure guesswork to find the correct delimiter characters.**
2. What is the meaning of Nth field? **Nth field is the name of an extractor, that returns a part of the data stored in a record, which is delimited by special delimiter characters.**

Exercise 14: Create a Flat File Dictionary

1. What is the difference between a dictionary and a schema? **A schema describes the records that are contained in a single flat file. A dictionary serves as a repository of record and composite definitions, which can be used across multiple schemas.**
2. Why should you create the IS document type when the schema is complete? **At this time all information is available to create the IS document type.**

Exercise 15: Web Service Descriptors and Custom Faults

1. When would you create a Provider WSD when a Consumer WSD? **You create a provider when you (Integration Server) provides a WEB service. You create a consumer WSD when you want to consume external WEB services.**
2. How and when are WSC's created? **They are implicity created when you create a WEB service consumer or provider.**
3. Can you have more than one custom SOAP Fault Document? **Yes. All you have to do is the addition of more error document types to the Response/Fault document list of the required operations.**

Exercise 16: Broker Pub/Sub

1. What happens when a document is made publishable? **The document is modified to contain a new document reference at the top level called _env. This envelope document contains data that is used internally by the broker to process the document. The second thing that happens is the publication of the new document type to the broker.**

2. What would be the appropriate production settings for publishable properties **Discard** and **Time to Live** if the Storage type = Guaranteed? **Set discard to false, to the broker will never discard instances of this document. Only set discard to true, if your documents become obsolete after a given amount of time. Put this time amount into the "time to live" field.**
3. What two objects are required for publishing? **The Trigger and a document instance.**
4. What three objects are required for subscribing? **The trigger, the document type and the handling service.**
5. Why were you required to use the full document type name as argument name in the handleValidation Service? **Because this is the name, the broker uses to store the received document in the input pipeline of the handling service.**

Exercise 17: JMS Pub/Sub

1. What is a topic versus a queue? **A topic is a many to many communication, while a queue is a many to one communication channel.**

Exercise 18: Create Adapter Services

1. What administrative activity must be done prior to using adapter service templates? **A JDBC Adapter connection must be created. This object contains all administrative data, like connection information and database credentials, to connect to the database.**
2. Why is it important to specify the LOOP input array before mapping the insertOrderDetails fields? **This step is required to be able to access the individual array elements; one per loop iteration.**

Exercise 19: Adapter Notifications

1. What is automatically created and updated when you work with your notification service? **The database buffer tables storing the notification data**
2. What occurs when the notification schedule is enabled? **The Database trigger becomes enabled and the data corresponding to the notification is collected.**
3. Why is the notification schedule an administration task? **Database-Administrators like to know what happens to their databases and especially want to control repeating operations to their database.**
4. What is the database state a DBA prefers the most? **The database is properly backed up and shut down on a disk that is not mounted on an operating system that is not booted on a computer that is turned off and neither connected to a popwer plug or network cable.**

Exercise 20: Use Services In a Business Process

1. Why do you create swimlanes within a process? What effect do they have on the execution of the process? **Swimlanes are used to group steps, they have no meaning for the execution of the underlying implementation.**
2. What occurs throughout the system when you perform a build and upload from Designer? **It generates Code Fragments and glue logic and stores these in integration server. It also stores information about the process build in the database.**

3. What are the different start mechanisms for a process? **You can wait for a given document to be published (subscription), where you have to option to choose between Broker or JMS, or you can expose your process as a service.**