webMethods Integration Workshop

Exercise Guide

611-67E

TABLE OF CONTENTS

This page intentionally left blank

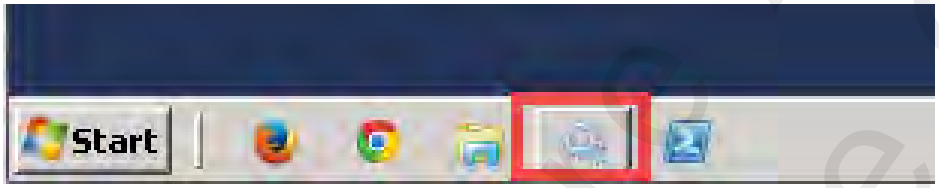
EXERCISE 1:
START UNIVERSAL MESSAGING, BROKER, INTEGRATION SERVER, AND MWS

Objectives

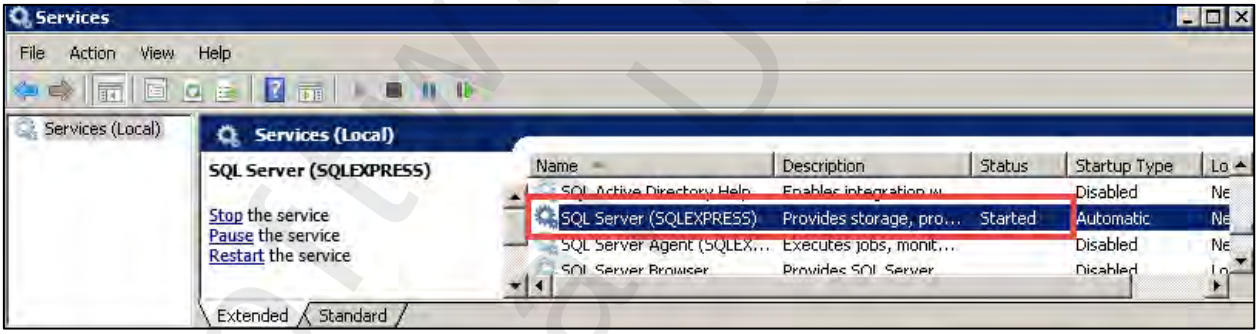
In this exercise, you will start the server components of the suite, and then open the Integration Server (IS) Administrator UI to confirm that everything is correctly set up for your class.

Steps

1. Start the **Services** administrative tool.



2. Make sure that the **SQL Server (MSSQLSERVER)** service is started. If it is not started, then start it from the Services administrative tool.



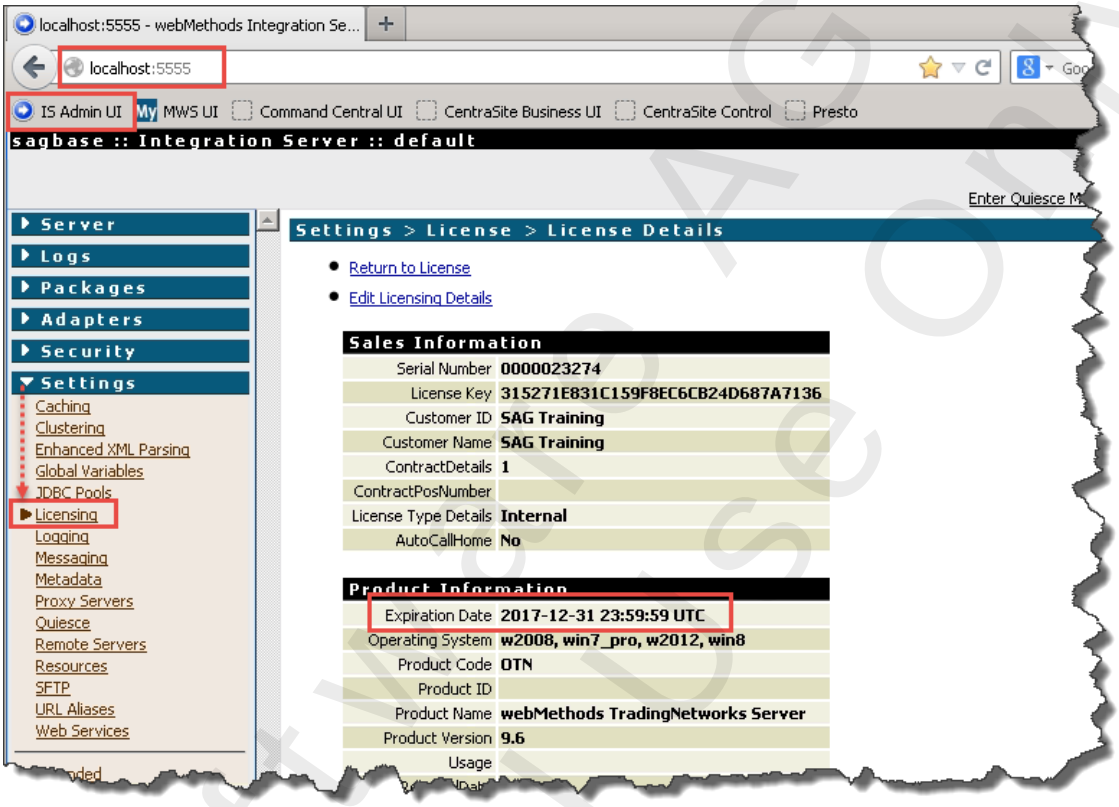
3. Start the following webMethods products in the order listed:
   a. **Software AG Broker Monitor 9.6 (6850)** - it will also start the Software AG Broker Server service. Wait for both services to show Started in the status column. Press F5 to refresh the view.
   b. **Software AG Universal Messaging 9.7.x.x.xxxxx (umserver) 1**
   c. **Software AG Integration Server 9.7 (default)**
   d. **Software AG My webMethods Server 9.7 (default)**

   Note: we are starting both Broker and Universal Messaging (UM) because we have exercises that use both. In the webMethods environment in your workplace you will typically use UM **OR** Broker, not both (unless you are an existing customer already using Broker).
   If you have to restart your training VM, you will have to start the above services again.

4. After a couple minutes, verify that Integration Server (IS) has started by using a browser to access the IS Administrator UI (http://localhost:5555):

   a. You can enter the URL or use the **IS Admin UI** bookmark. Login as **Administrator** | **manage**.

   b. In the **IS Administrator UI's Settings** area, check the Integration Server license key by selecting the **Licensing --> Licensing Details** 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!*



If Integration Server fails to start, this can be caused by lock files named .lock and wrapper.anchor in C:\SoftwareAG\profiles\IS_<*instance name*>\bin. These lock files are used to prevent more than one instance of the same Integration Server instance from running. If the Integration Server is not shutdown gracefully these files may not be cleaned up, and will prevent the IS from starting.

To fix this problem, verify that the Integration Server instance is not running then delete the files .lock and wrapper.anchor in folder C:\SoftwareAG\profiles\IS_default\bin and try to start the service again.

5. Now enable the IS packages we will need in the exercises.

   In the IS Administrator UI, choose **Packages --> Management**. Enable the following packages:

a. AcmeSupport
b. CommonSupport
c. WmFlatfile

Note: we will also use the WmJDBCAdapter package, however this is enabled already.

6. The My webMethods Server (MWS) will take awhile to start since it is sharing resources on the training VM with other products and the O/S. We do not need the MWS server until the Monitoring exercise so you can consider this exercise complete now.

   Note: Later you can verify that MWS has started by using a browser to access the MWS UI at **http://localhost:8585**. You can enter the URL or use the **MWS UI** bookmark. Login as **Administrator | manage**.

Check Your Understanding

1. What is the URL to access the Integration Server?
2. What is the URL for My webMethods?
3. Why should you set the Broker Monitor service to Automatic start, but not the Broker Server?
4. Do you have to start Broker Monitor/Server and Universal Messaging in your webMethods environment?

This Page intentionally left blank

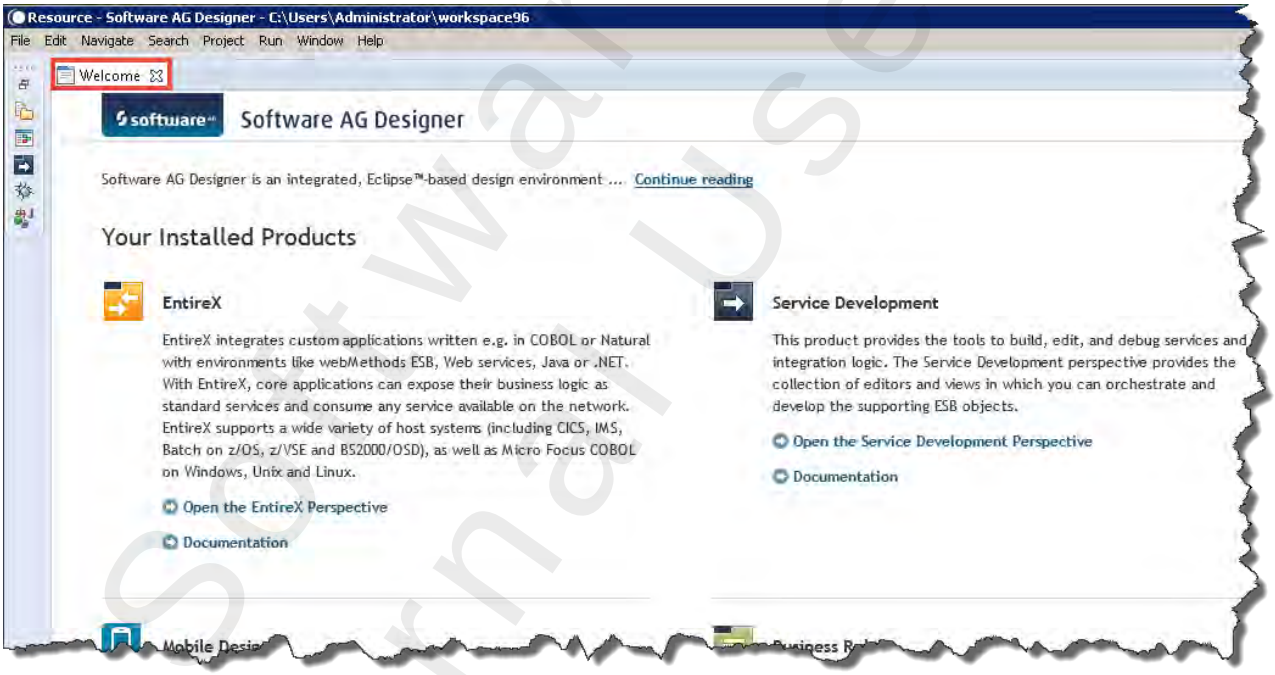
EXERCISE 2:
PACKAGES AND FOLDERS

Overview

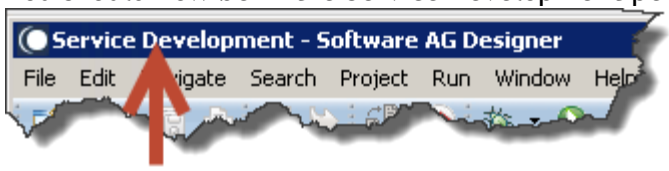
In this exercise, you will create a package and folders to store the Integration Server development work you will perform in future exercises.

Steps

1. Open Software AG Designer,
   a. Choose **Start --> Software AG Designer 9.7**
   b. At startup if asked to select a workspace, keep the default workspace, and then check the box beside **Use this as the default and do not ask again**.
2. If Designer shows its default welcome screen, then on the Welcome tab, click the "X" to close the welcome screen.



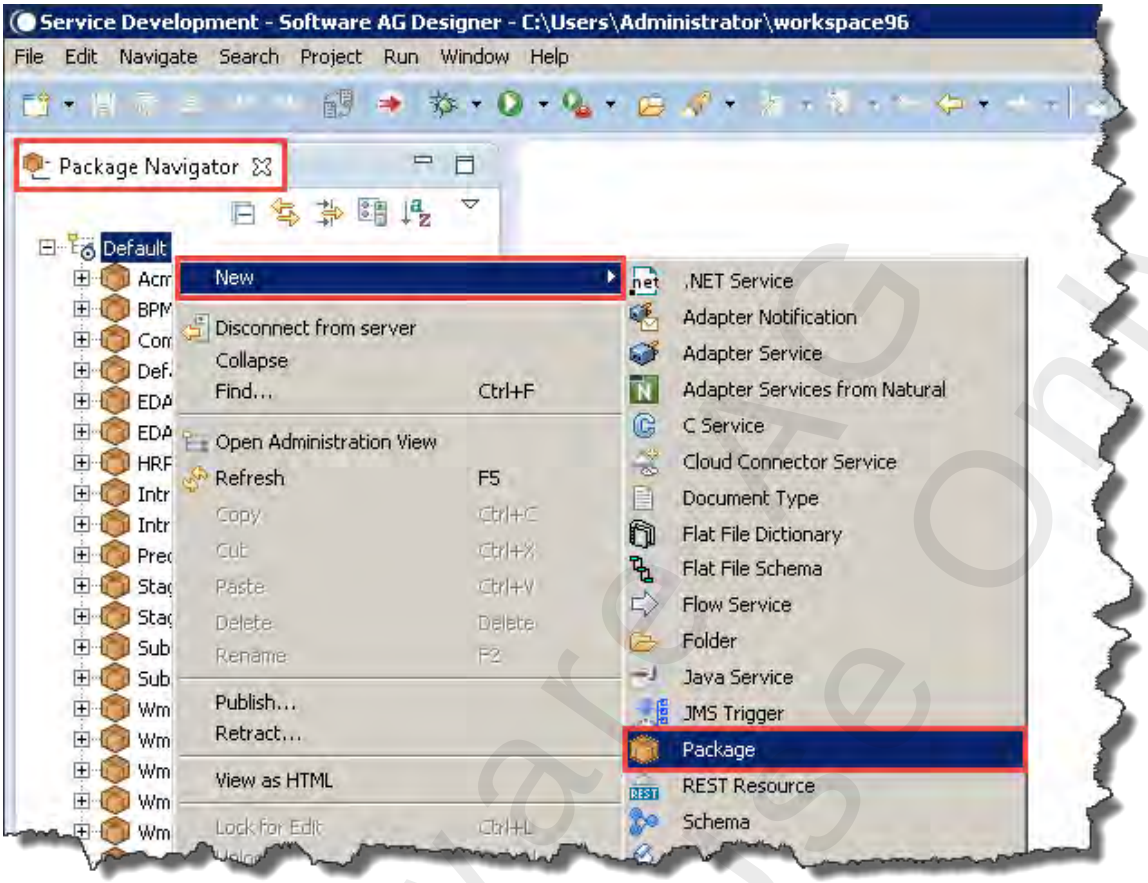
3. You should now be in the Service Development perspective.

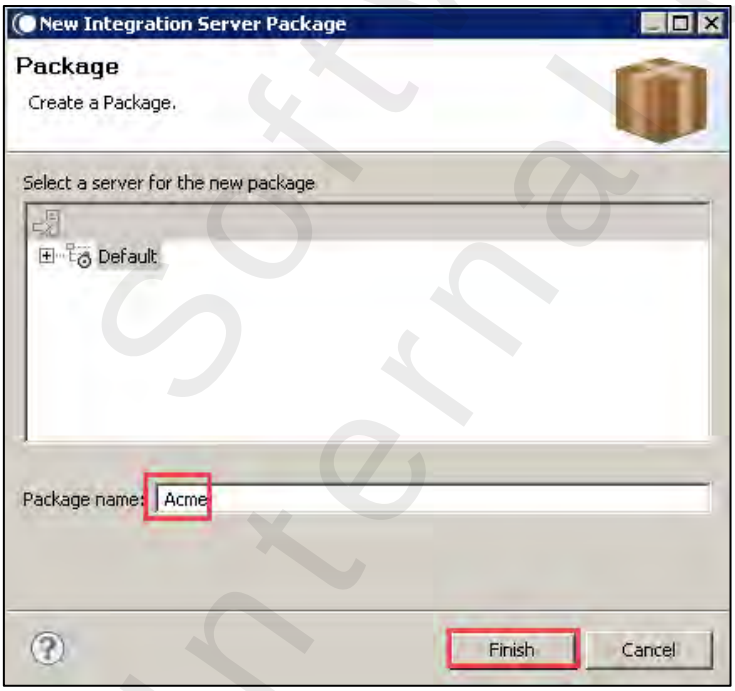


If you are not then choose **Window --> Open Perspective --> Service Development**.

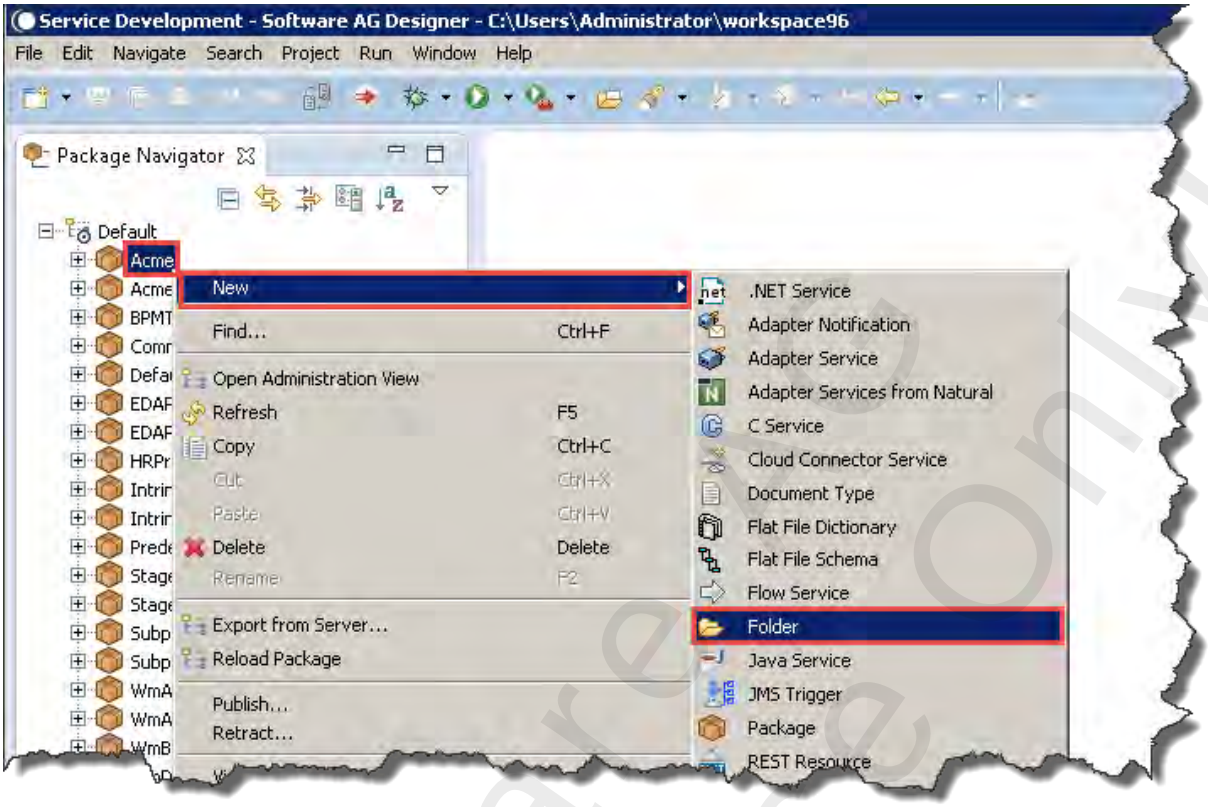
Note: You can also select the perspective from the Designer's shortcut bar.

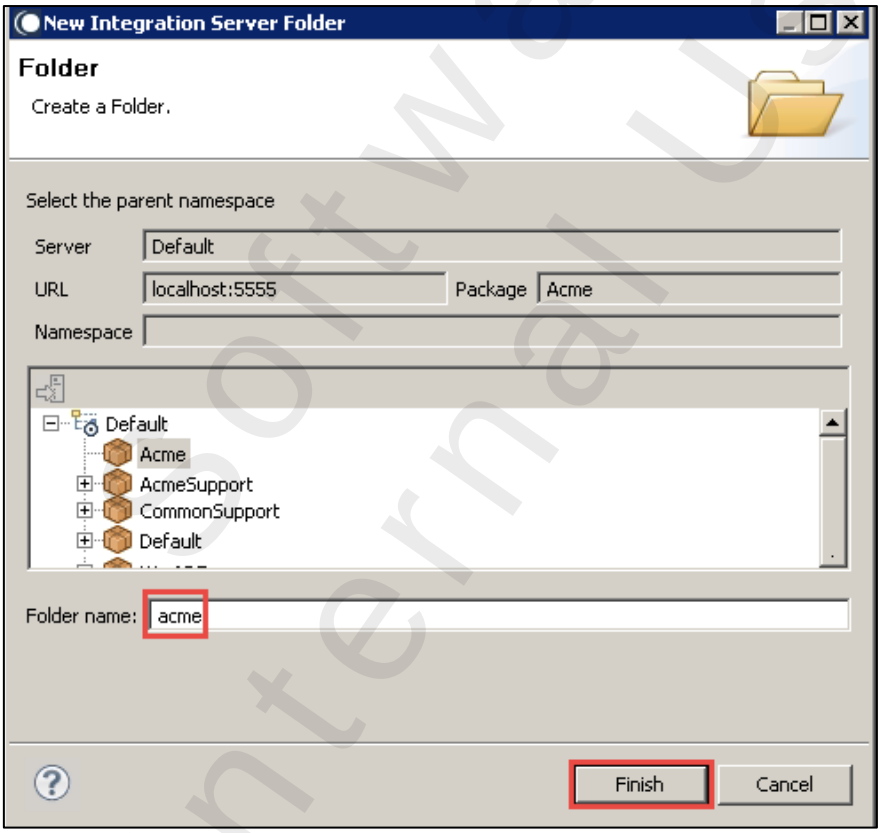
4. Create a new package by right clicking on **Default** and choosing **New --> Package.** Name the package **Acme** and click **Finish.**





5. In the **Acme** package, create a folder called **acme**. Click **Finish**.

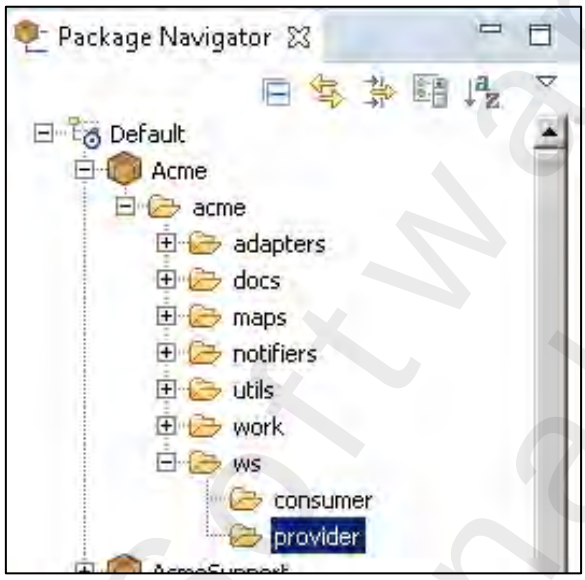




6. In the **acme** folder, create seven new folders, named as follows:
   - **adapters**
   - **docs**
   - **maps**
   - **notifiers**
   - **utils**
   - **work**
   - **ws**

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

7. In the **acme.ws** folder, create two new folders, named as follows:
   - **consumer**
   - **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?

This Page intentionally left blank

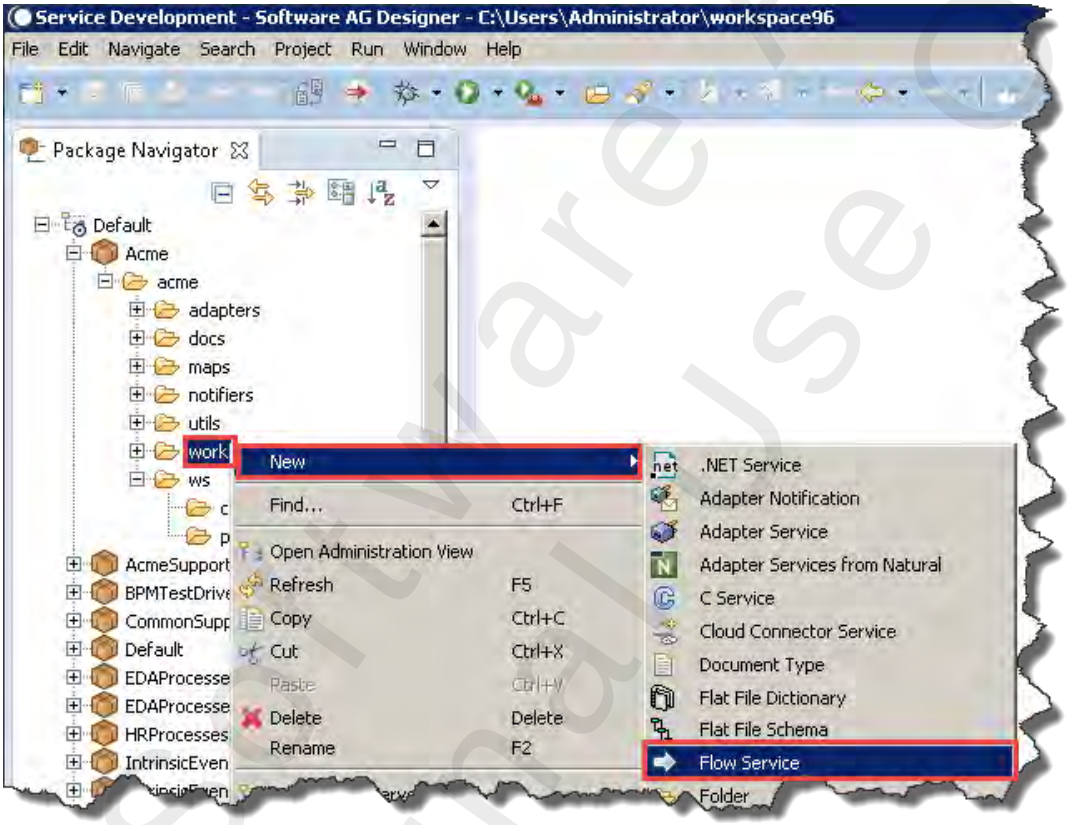
EXERCISE 3:
CREATE AND TEST A FLOW SERVICE

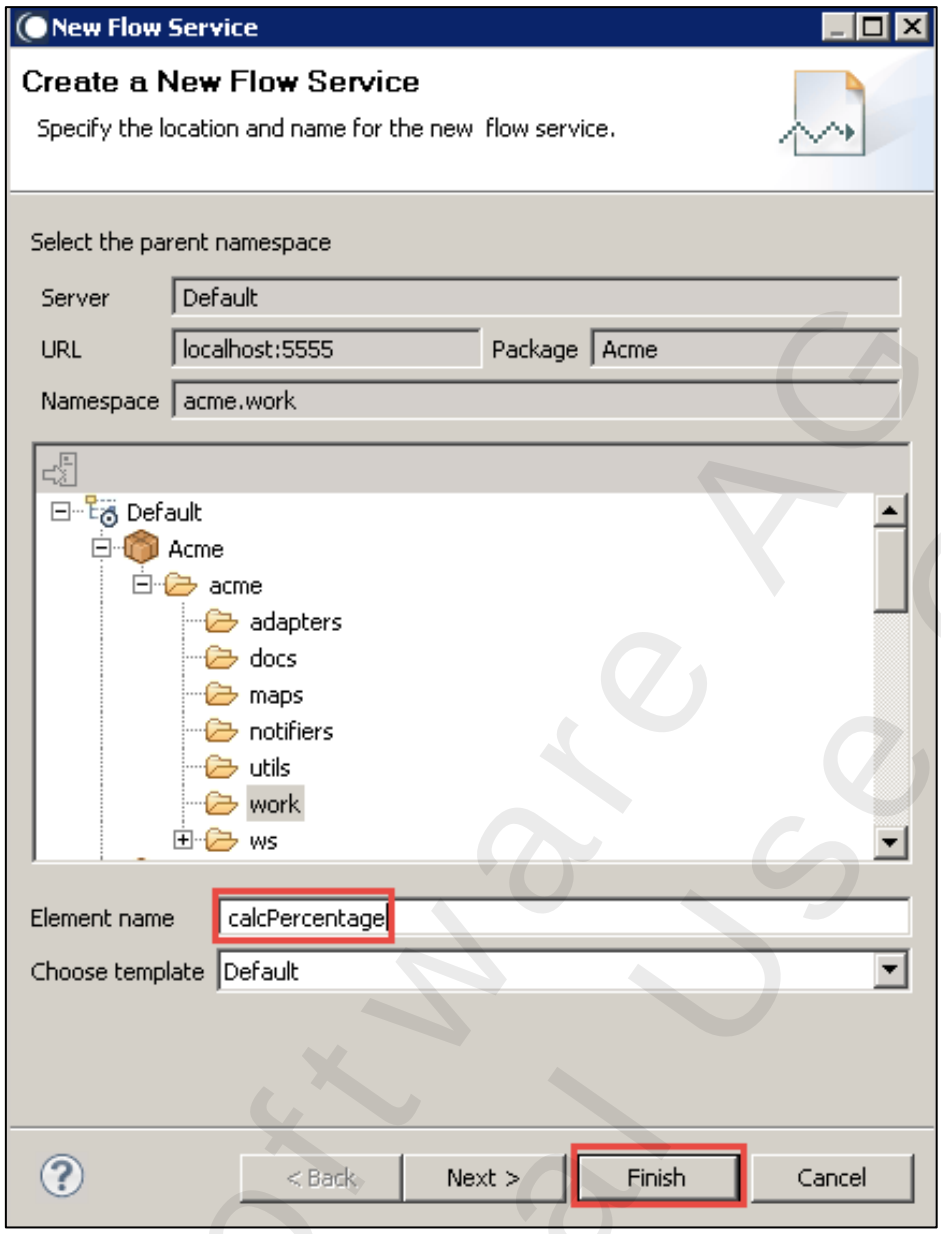
Overview

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

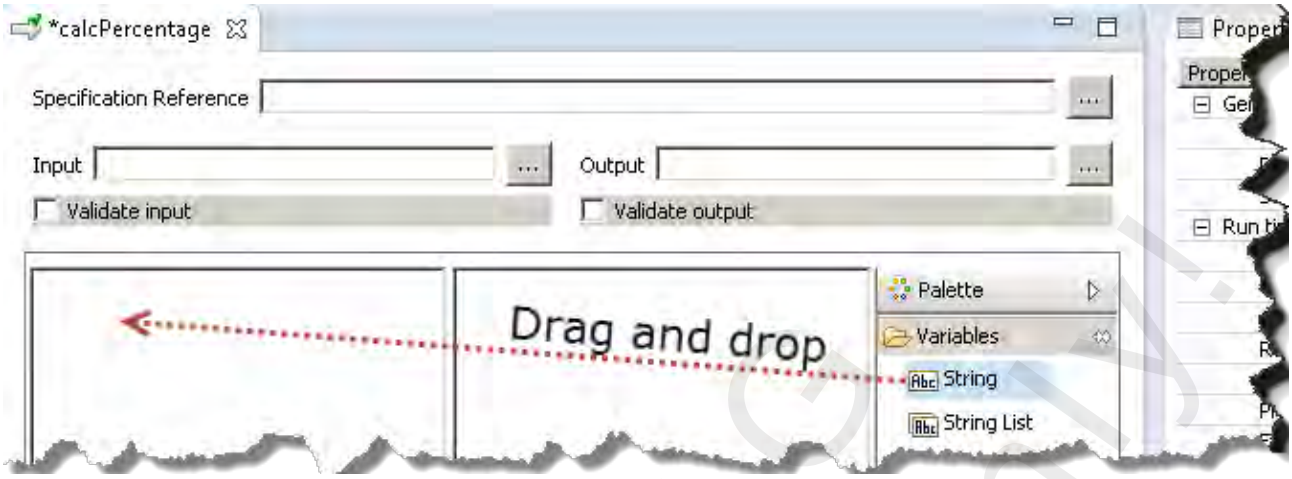
Steps

1. In the **Acme** package, in the **acme.work** folder, right click **work --> New --> Flow Service** to create an empty Flow service. Name the service **calcPercentage**. Click the **Finish** button.



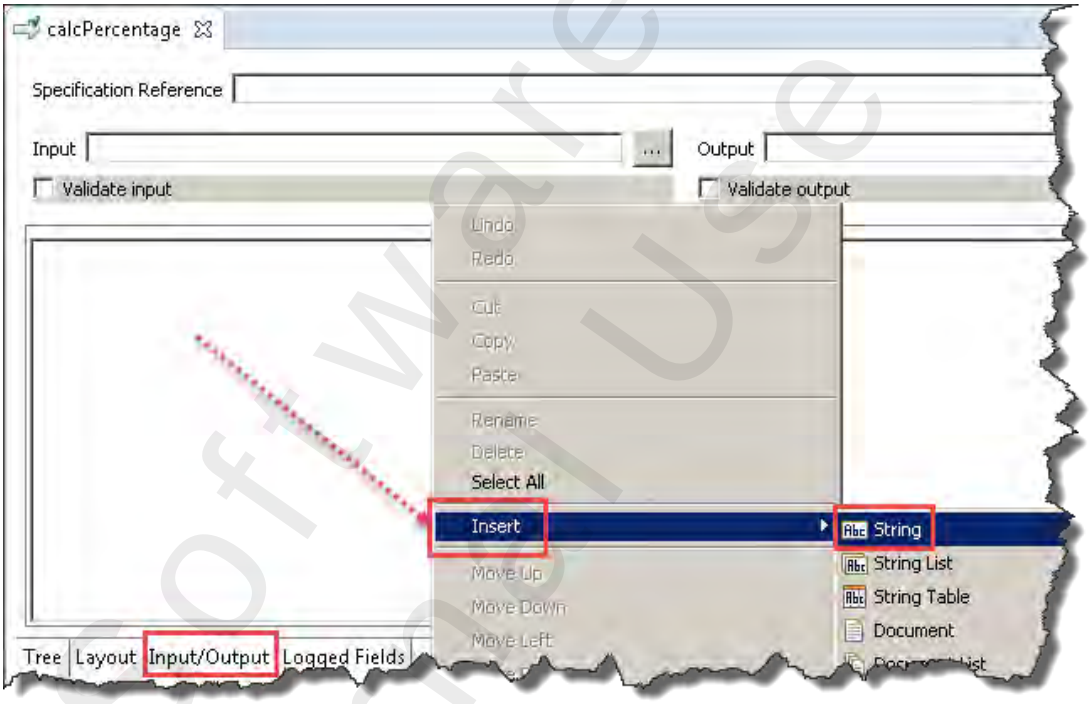


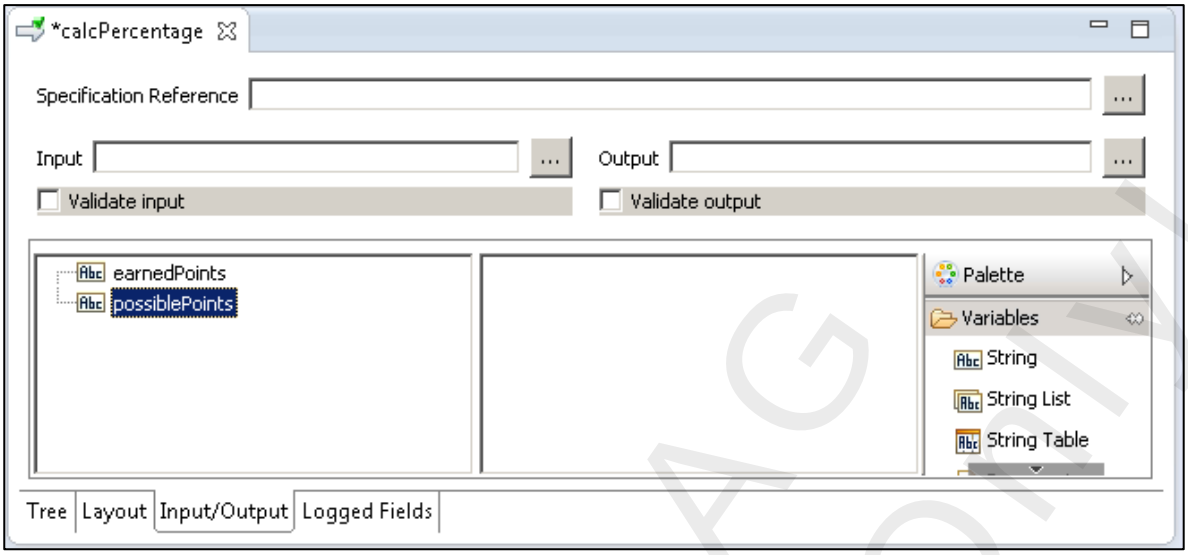
2. Add two String inputs to the new service, called **earnedPoints** and **possiblePoints**.
   a. Choose the **Input/Output** tab.
   b. From the Palette on the right side, drag-and-drop the field type (String) into the input panel (on the left side)
   Note : If the palette is not visible click on the arrow pointing left, next to the output panel.



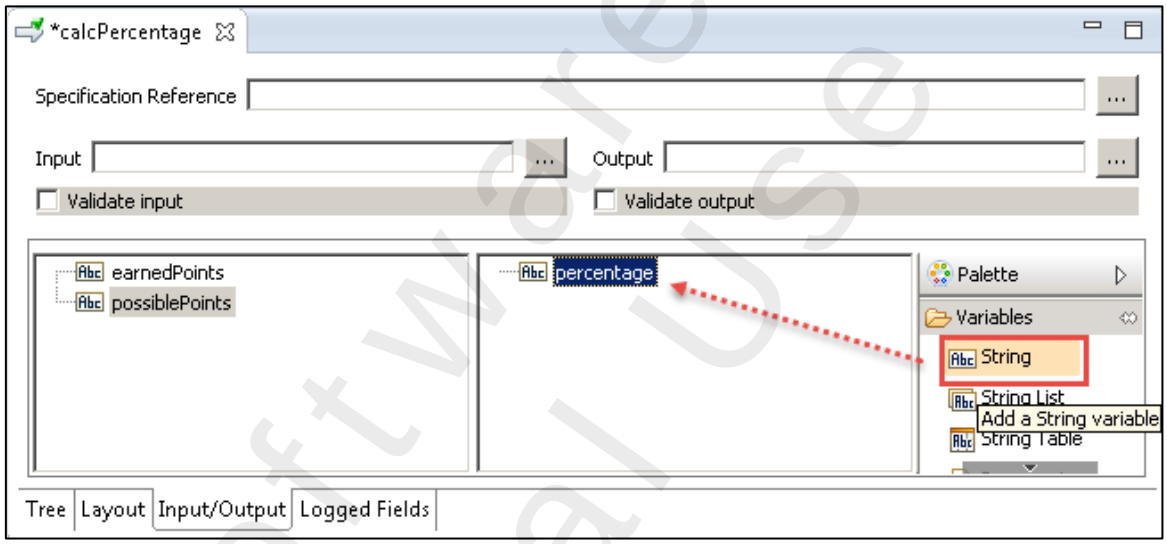
Name the input field **earnedPoints**. Repeat this process for the input **possiblePoints**.

Note: you can also add input / output fields by right-clicking in the panel and choosing Insert --> String.



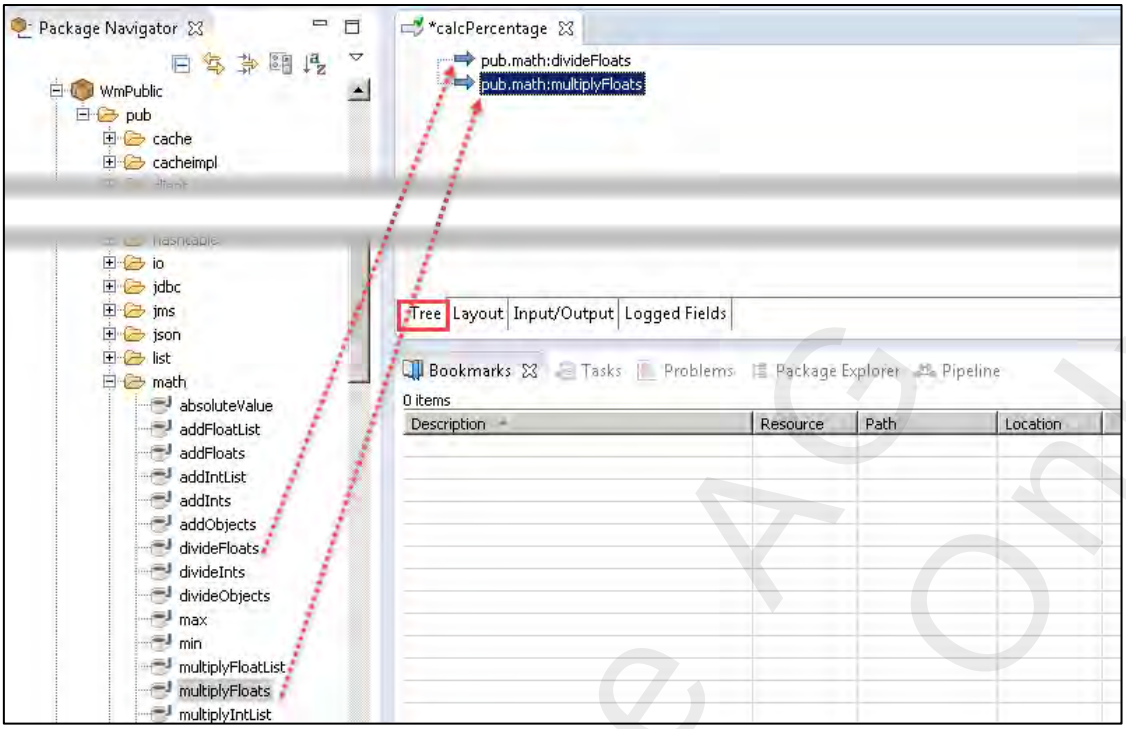


3. Add a String output to the new service, called **percentage**. Drag **String** to the Output pane and name the output **percentage.**



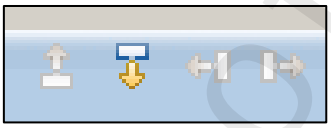
4. Add the following two service steps using the method described below:
   a. **pub.math:divideFloats**
   b. **pub.math:multiplyFloats**

   The first way you will insert the service invocations is to switch to the **Tree** tab on the bottom of the service editor and drag in the two services from the **WmPublic** package.

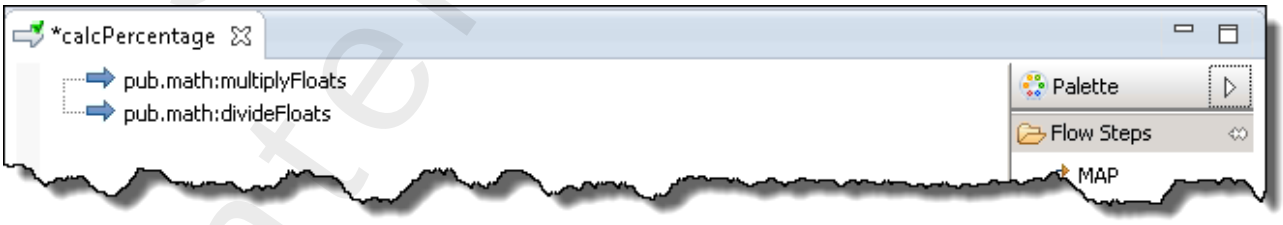


Note: even though we tell you to use the Tree tab in the exercises, you can also develop Flow services using the Layout tab. The Tree tab gives you a hierarchical (tree) view of your Flow code. The Layout tab gives you more of a flowchart (diagram) view. Feel free to try both while you develop/edit the Flow services. The use of one over the other becomes a personal preference with each developer :)

5. Using the Move Up and Move Down arrow icons, re-order the services to the following:
   a. **pub.math:multipleFloats**
   b. **pub.math:divideFloats**



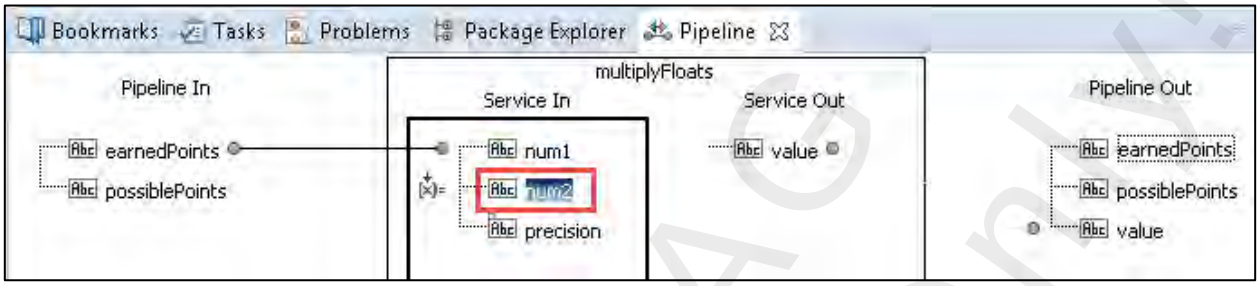
Please note the (currently disabled) **Move Left** and **Move Right** Icons next to the **Move Up** and **Move Down** icons. You will need them in later exercises as well.

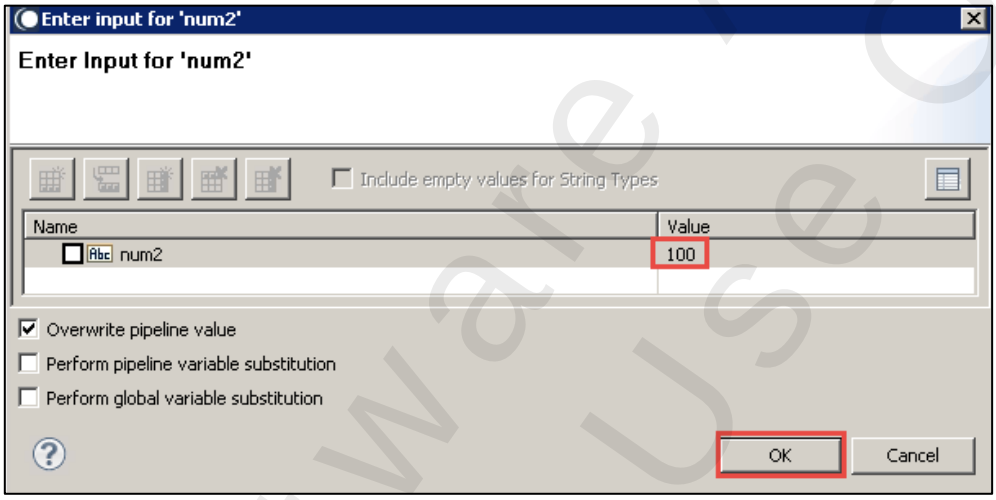


6. Map the data flow using the method described below:
   a. **pub.math:multiplyFloats** (from **Pipeline In** to **Service In**)
      i. **earnedPoints** should link to **num1**

ii. **num2** should have its value set to **100**

In order to complete this step, select the **pub.math:multiplyFloats** service in the Tree pane of your service. Then, select the **Pipeline** view at the bottom of the IDE. Then drag the **earnedPoints** argument to the **num1** parameter of the **multiplyFloats** service. Double click the **num2** parameter, and in the pop-up window enter **100** for the value.

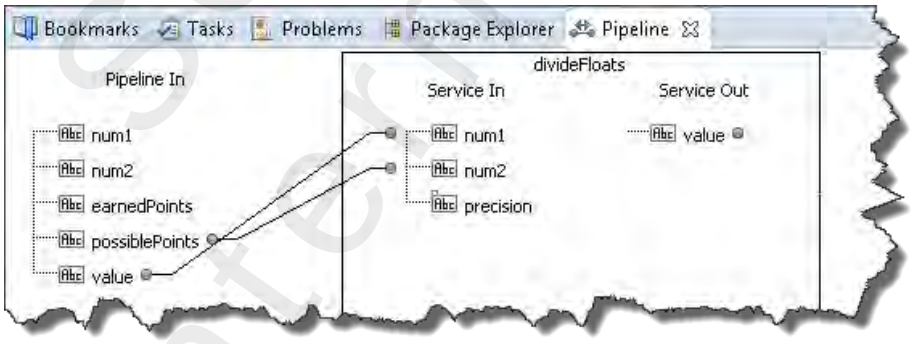




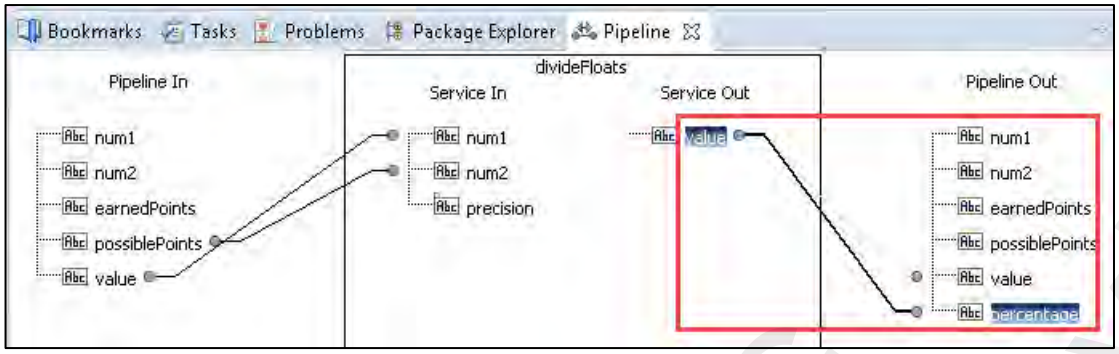
b. Complete the links for the left side of the **pub.math:divideFloats** service step (from **Pipeline In** to **Service In**):

i. **value** should link to **num1.**

ii. **possiblePoints** should link to **num2.**



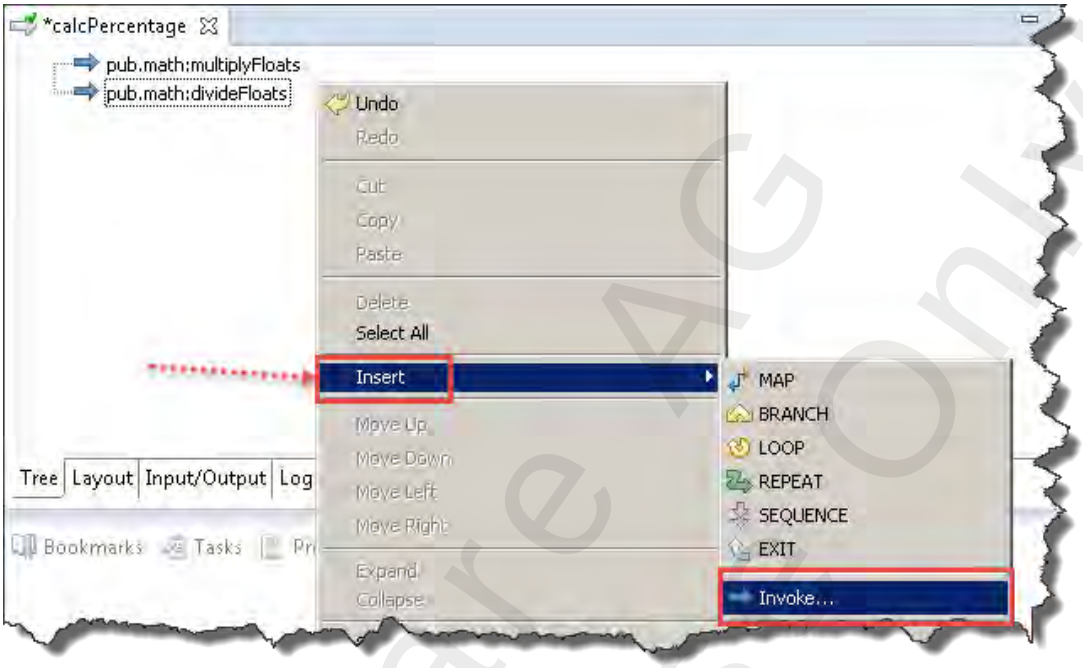
c. Complete the links for the right side of **pub.math:divideFloats** (from **Service Out** to **Pipeline Out).** Link **value** in Service Out to **percentage** in Pipeline Out

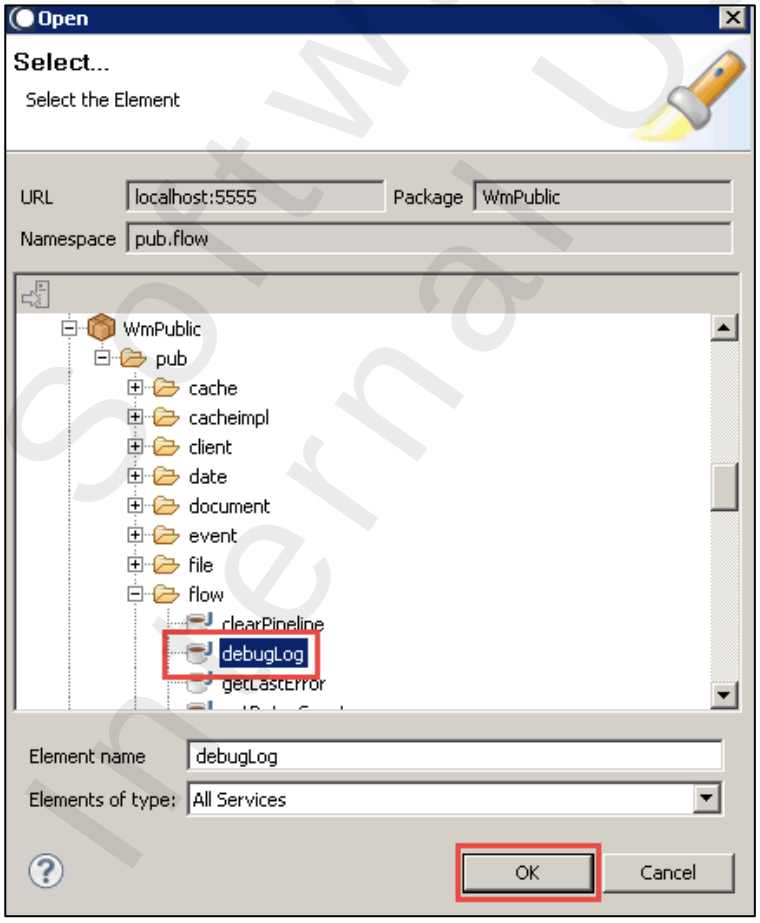


Note:the Palette can be hidden if necessary by clicking the right arrow

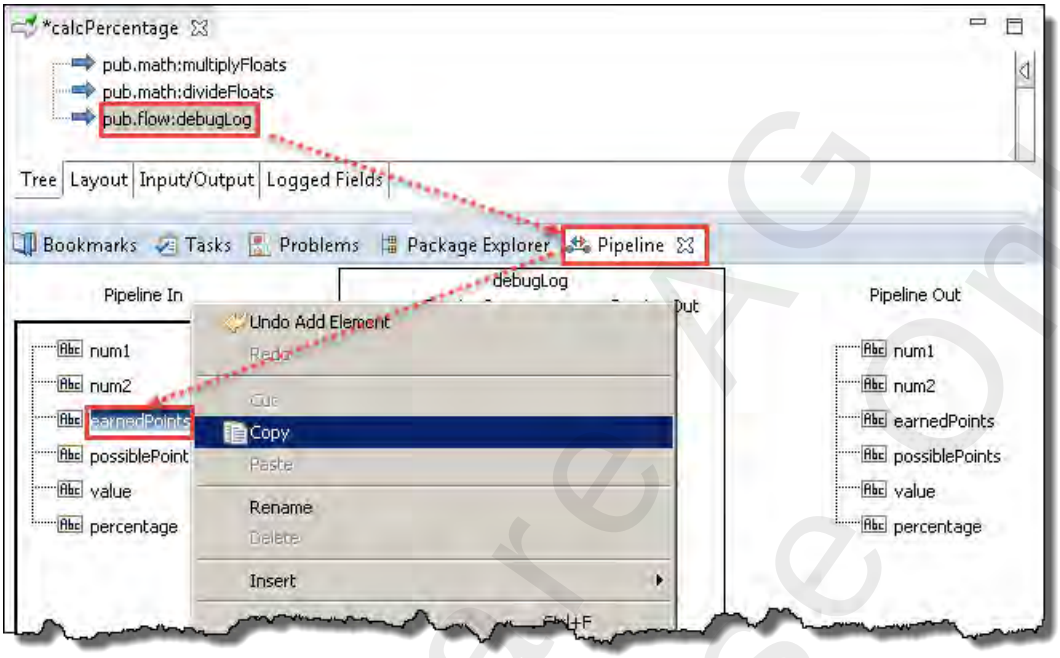
7. Add the **WmPublic** package's **pub.flow:debugLog** service step to your service by doing the following:

    a. In the **Tree** view, right click and choose **Insert --> Invoke.**
    b. In the popup window, choose the **WmPublic** package's **pub.flow:debugLog** service.





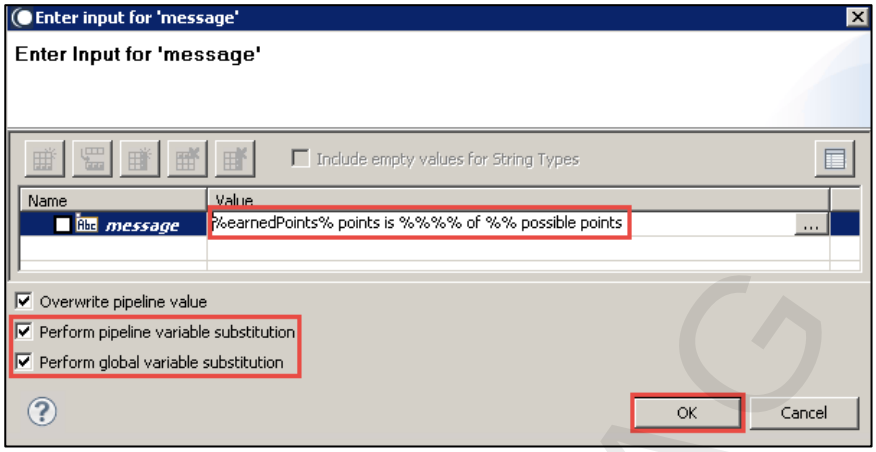
8. Copy the **earnedPoints** variable by doing the following:
   a. In the Tree view, select the **pub.flow:debug** step service.
   b. Click the **Pipeline** tab.
   c. Right click the **earnedPoints** variable in Pipeline In and choose **Copy.**



9. Configure the debugLog service to log a message similar to the following: "15 points is 75% of 20 possible points".
   a. In Service In, double click the **message** parameter and set the Value to the following: **%% points is %%%% of %% possible points**
   b. Put your cursor between the first set of percent signs. Then right click and choose **Paste** to add the **earnedPoints** variable to your message string.

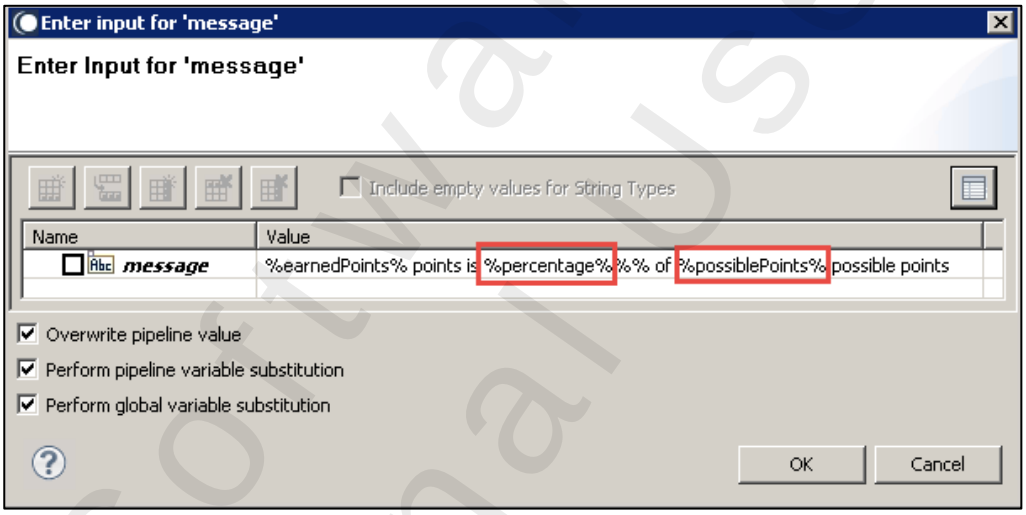
      Note: you can directly type in the variable name but make sure you use the correct spelling and case. Copy/paste is a handy technique to use for some variables.
   c. Check the box to the left of both of the following checkboxes and click **OK** to close the Popup window:
      i. **Perform pipeline variable substitution**

ii. **Perform global variable substitution**



d. Between the second set of percent signs, either type **percentage** OR you can use the copy/paste technique you used above. Sometimes typing a simple variable name is faster :)

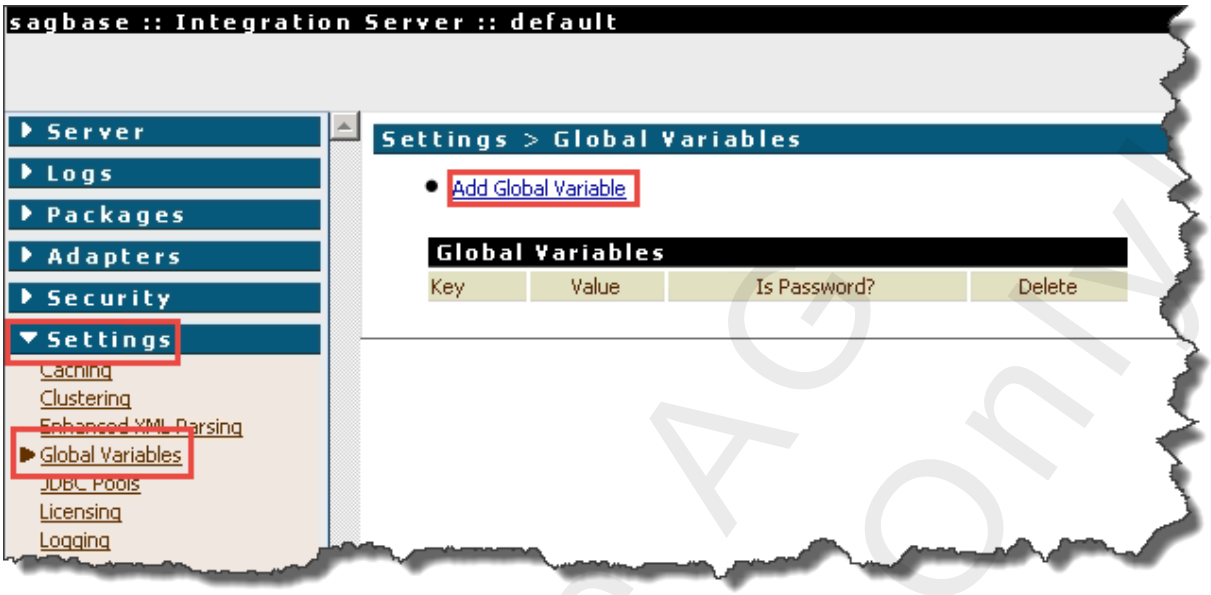
e. Between the last set of percent signs, either type possiblePoints OR you can use the copy/paste technique you used above. Click **OK** to close and save the service.



10. In Mozilla Firefox, use a browser tab to connect to the IS Administrator UI (http://localhost:5555). Use credentials **Administrator / manage** to login.

a. Navigate to **Settings --> Global Variables.**

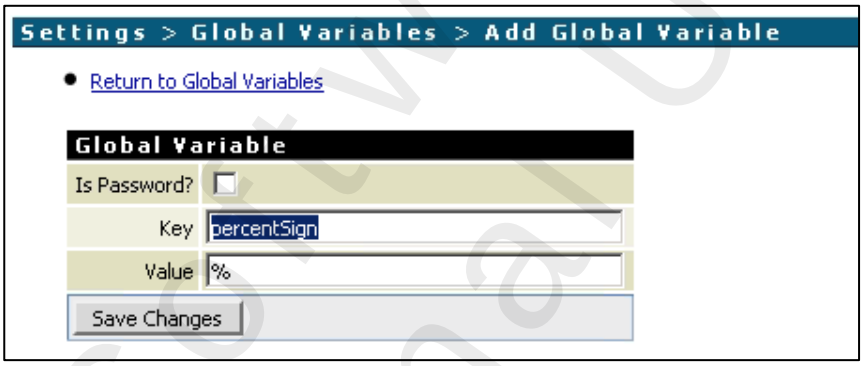
b. Click on the **Add Global Variable** link



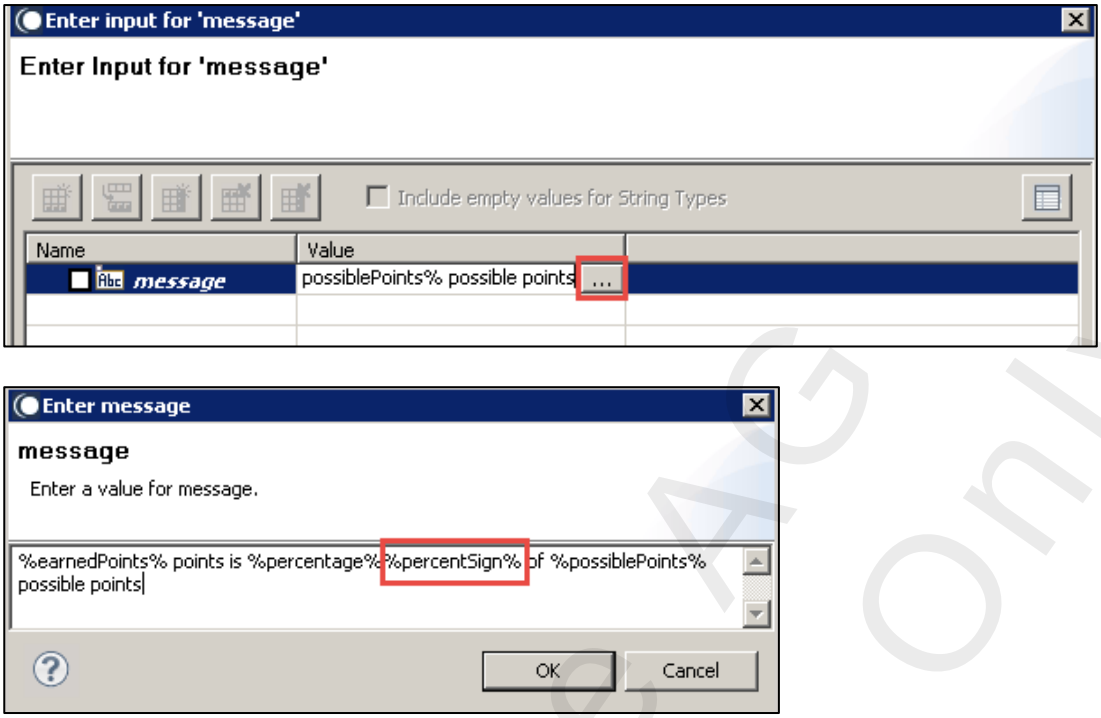
c. Enter the following:

- Key: **percentSign**
- Value: **%**

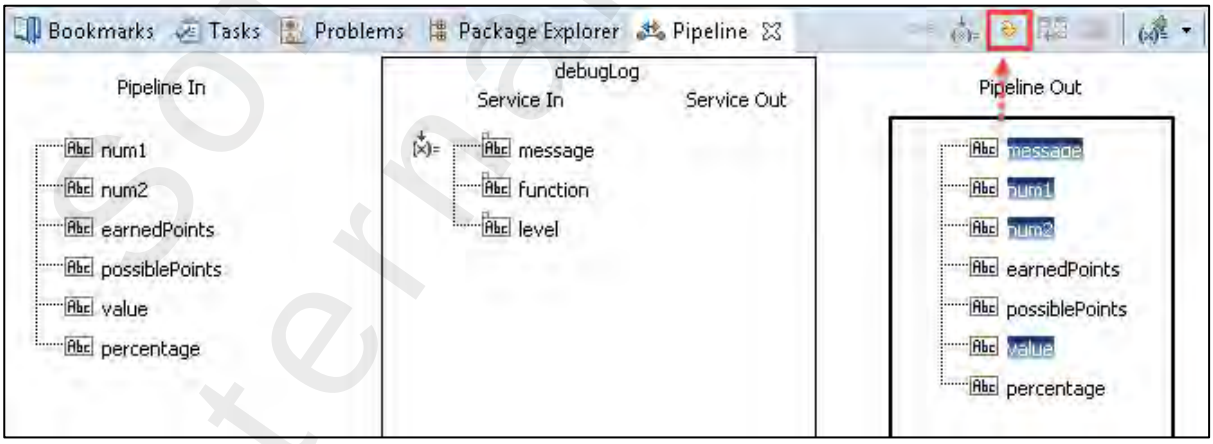
d. Copy the **percentSign** Key value and then click the **Save Changes** button.



11. Switch back to Designer. Again, edit the **message** parameter of the pub.flow:debugLog service step.
    a. Select the Value, and then select the (…) button to use a larger edit window.
    b. Paste the **percentSign** value between the third set of percent signs.
    c. Click the **OK** button twice to finish your edit of the **message** parameter.

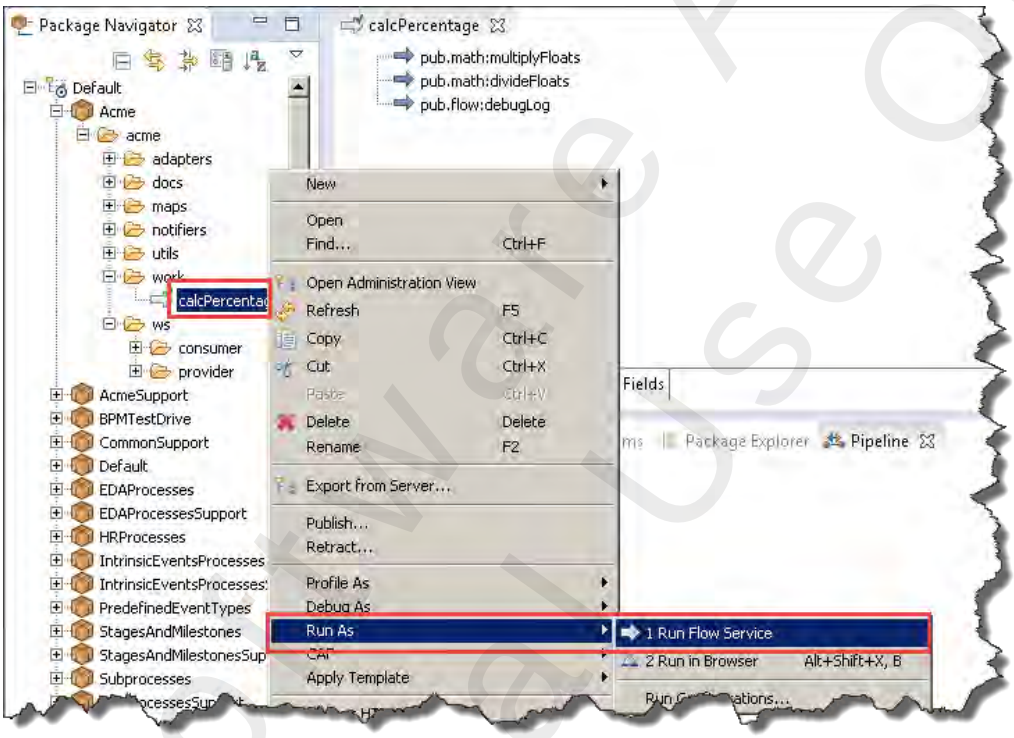


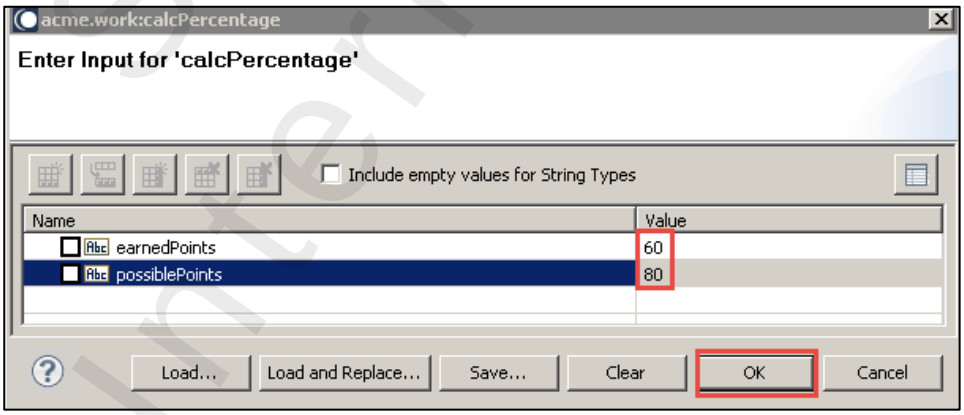
12. Drop all variables which are not in your Input / Output tab by doing the following:
    a. In Pipeline Out, while holding down the **<CTRL>** key select the following variables:
        - **message**
        - **num1**
        - **num2**
        - **value**
    b. Click the **Drop** icon.



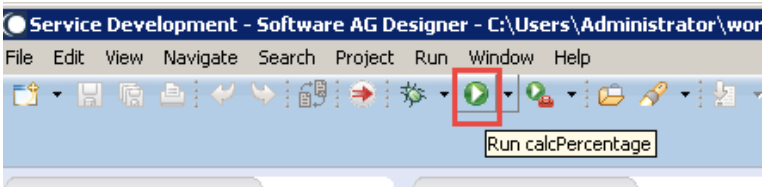
13. Click either the **Save** or **Save All** icons to save your service.

14. Run the service. For the first run of your service, in the Package Navigator view, right click your **calcPercentage** service and choose the **Run As --> Run Flow Service** from the context menu of the service. Enter inputs of the following:
    a. earnedPoints: **60**
    b. possiblePoints: **80**

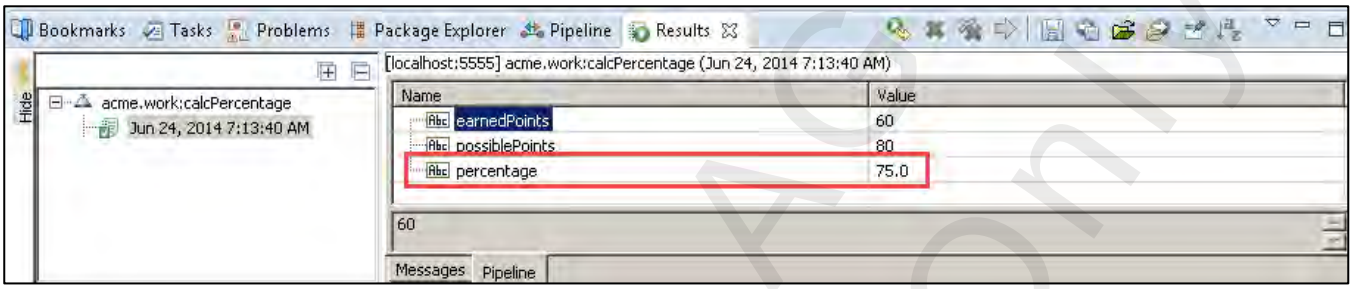




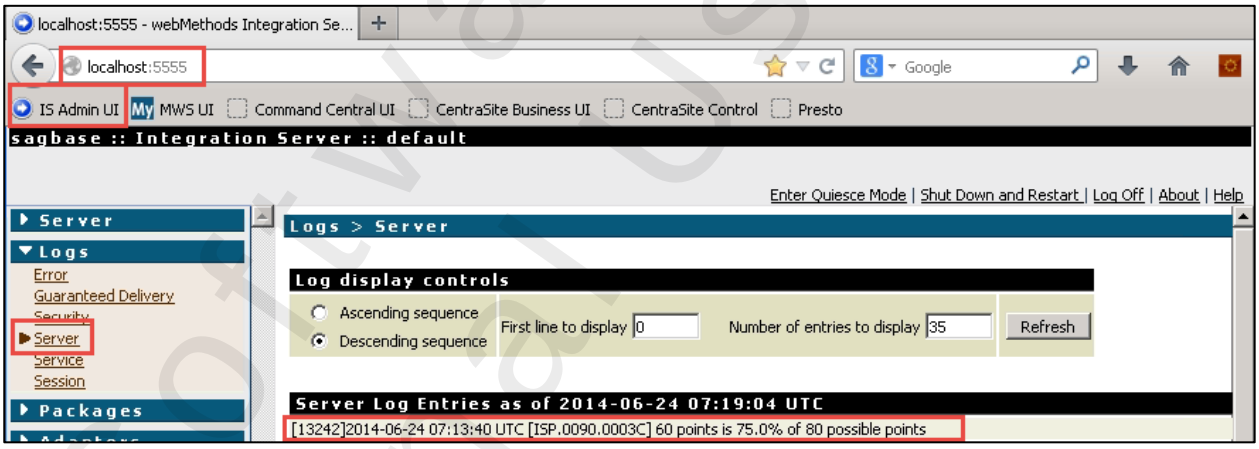
Note: For further invokes you can simply click the green arrow button in the upper toolbar. Then provide string inputs of your choice.



15. Check the service Results view in Designer:



16. Check the Server log by doing the following:
    a. Open up a Browser tab.
    b. Login to IS Administrator UI (http://localhost:5555 or use the appropriate bookmark) and authenticate as **Administrator / manage.**
    c. Navigate to **Logs --> Server**.



Check Your Understanding

1. Why is the order of the services important?
2. What would appear in the Server Log if in the message parameter, you did the following:
    a. Selected “Perform pipeline variable substitution” but NOT “Perform global variable substitution”
    b. Selected “Perform global variable substitution” but NOT “Perform pipeline variable substitution”
    c. Did NOT select either “Perform pipeline variable substitution” or “Perform global variable substitution”

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.

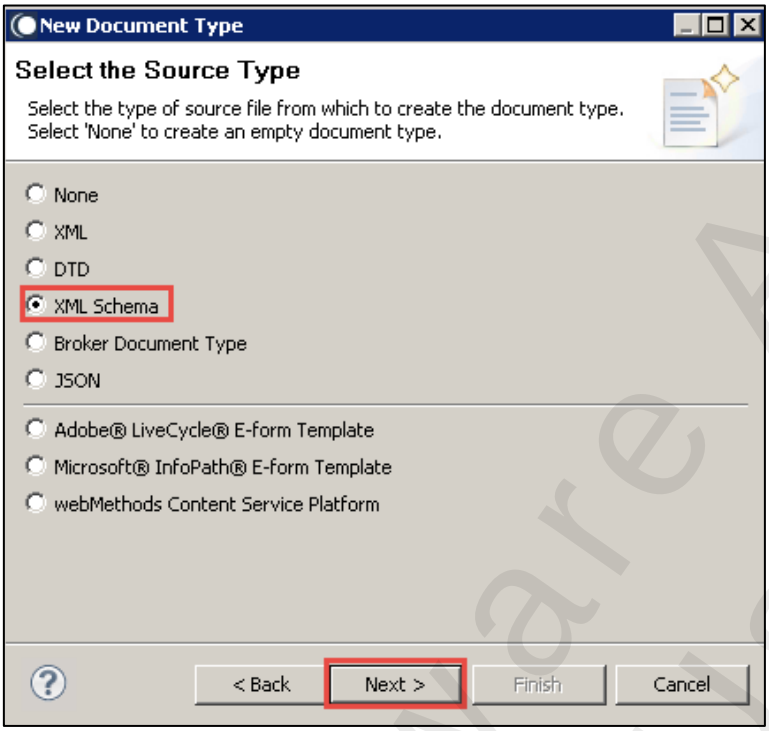
Steps

1. Use Windows Explorer to navigate to the folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub**. Right click the **PORequest.xml** file and use the text editor **Notepad++** to look at the XML document. What is the root node?

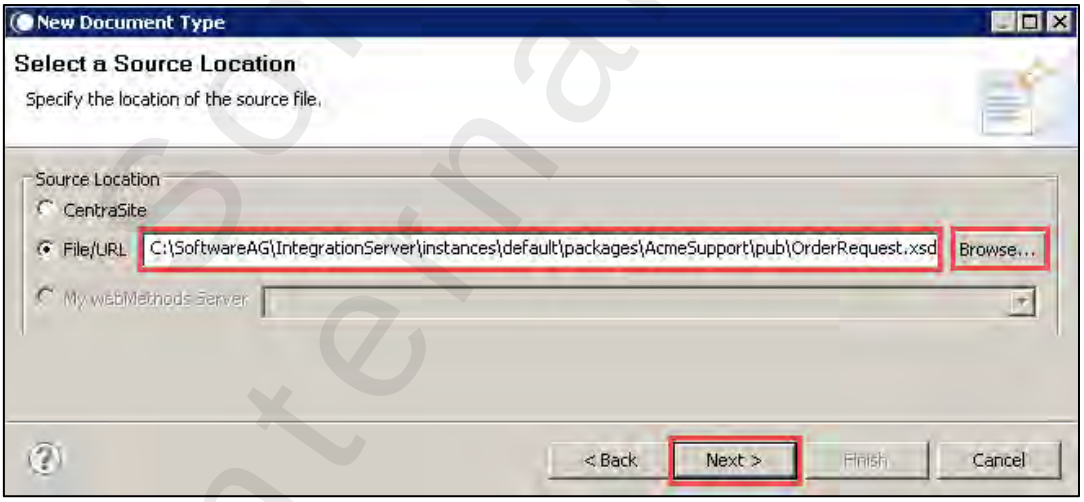
```
<?xml version="1.0" ?>
<!--
webMethods Integration Platform Overview Training Course
Sample XML Message Purchase Order Request
-->
<PurchaseOrderRequest xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="OrderRequest.xsd">
  <PurchaseOrder>
    <deliverTo>
      <PhysicalAddress>
        <cityName>
          <FreeFormText>Fairfax VA</FreeFormText>
        </cityName>
        <addressLine1>
          <FreeFormText>webMethods</FreeFormText>
        </addressLine1>
        <addressLine2>
          <FreeFormText>3877 Fairfax Ridge Road   South Tower</FreeFormText>
        </addressLine2>
        <addressLine3>
          <FreeFormText>Fairfax VA</FreeFormText>
        </addressLine3>
        <NationalPostalCode>22030</NationalPostalCode>
        <regionName>
          <FreeFormText>USA</FreeFormText>
        </regionName>
        <postOfficeBoxIdentifier>
          <FreeFormText/>
        </postOfficeBoxIdentifier>
```

2. Using Designer, in the Package Navigator view, locate the **acme.docs** folder, and create a **request** folder. In the **request** folder, create a new Document Type called **OrderRequest.**

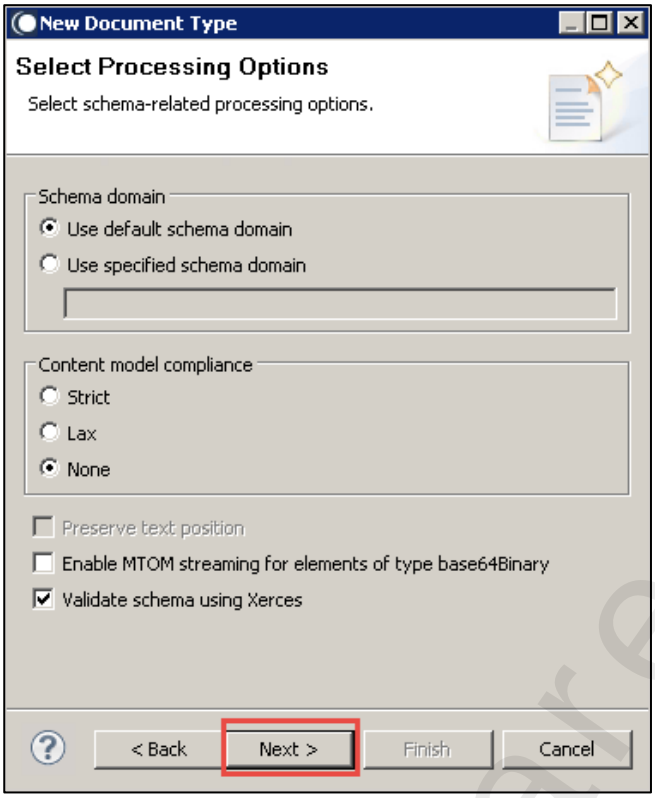
   To create the new Document Type, you right click on the folder and select **New -> Document Type**. In the upcoming dialogue choose **Next**. Select **XML Schema** when asked for a source type and click **Next**.



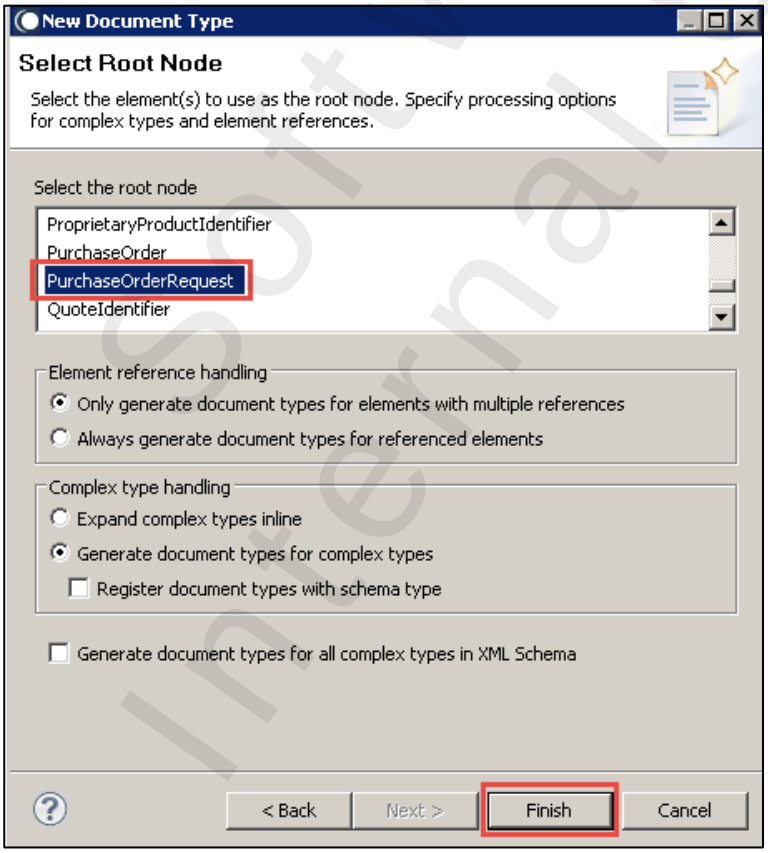
   Import the schema from file **OrderRequest.xsd** contained in folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub.**



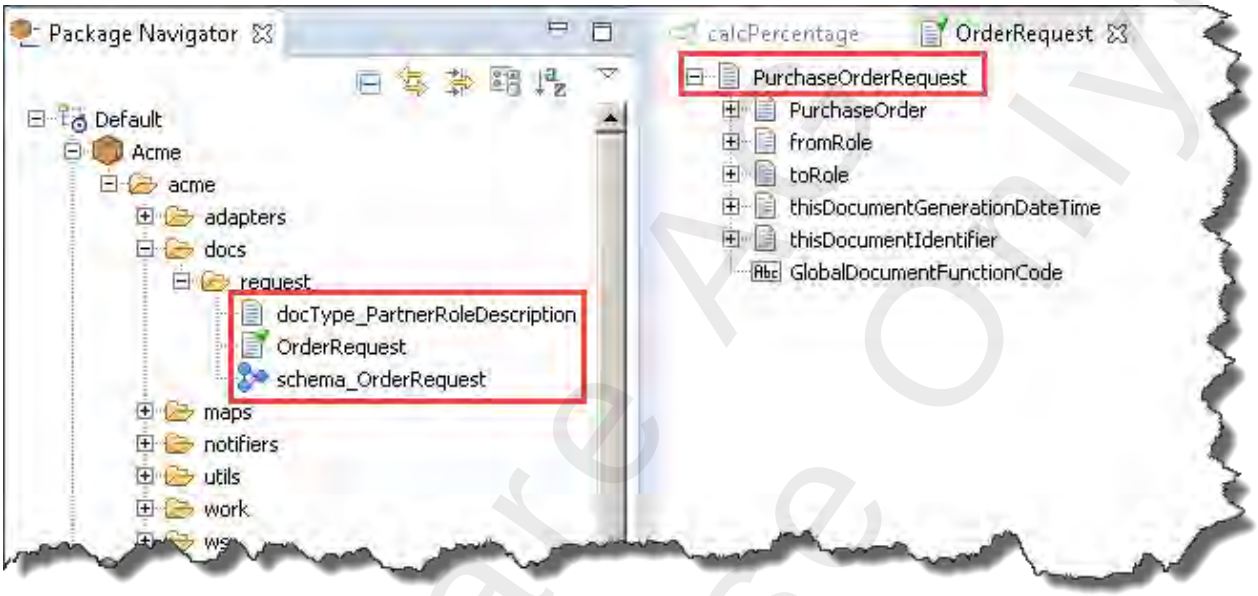
Leave the processing options at their defaults and click **Next**:



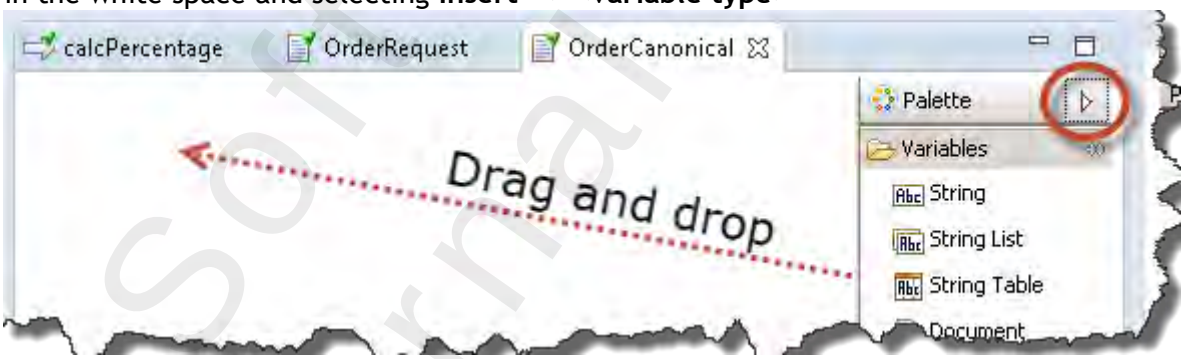
When prompted to select a root node, choose the root node you found in step 1 above. Click **Finish**.



3. Confirm that in your **acme.docs.request** folder, you see the following items:
   a. **docType_PartnerRoleDescription**
   b. **OrderRequest**
   c. **schema_OrderRequest**
   d. In the **OrderRequest** tab, you see that the **PurchaseOrderRequest** is the root node.



4. In the **acme.docs** folder, create a new Document Type called **OrderCanonical**. You will create this document type manually (**None**) with the structure described below.
   You can build the document structure using the Palette on the right side OR by right-clicking in the white space and selecting **Insert --> <variable type>**



   Note: you must be extremely meticulous about using the correct upper and lower case for all the names in the document. Later exercises will not work if you spell something wrong here or specify the wrong data type.

   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)

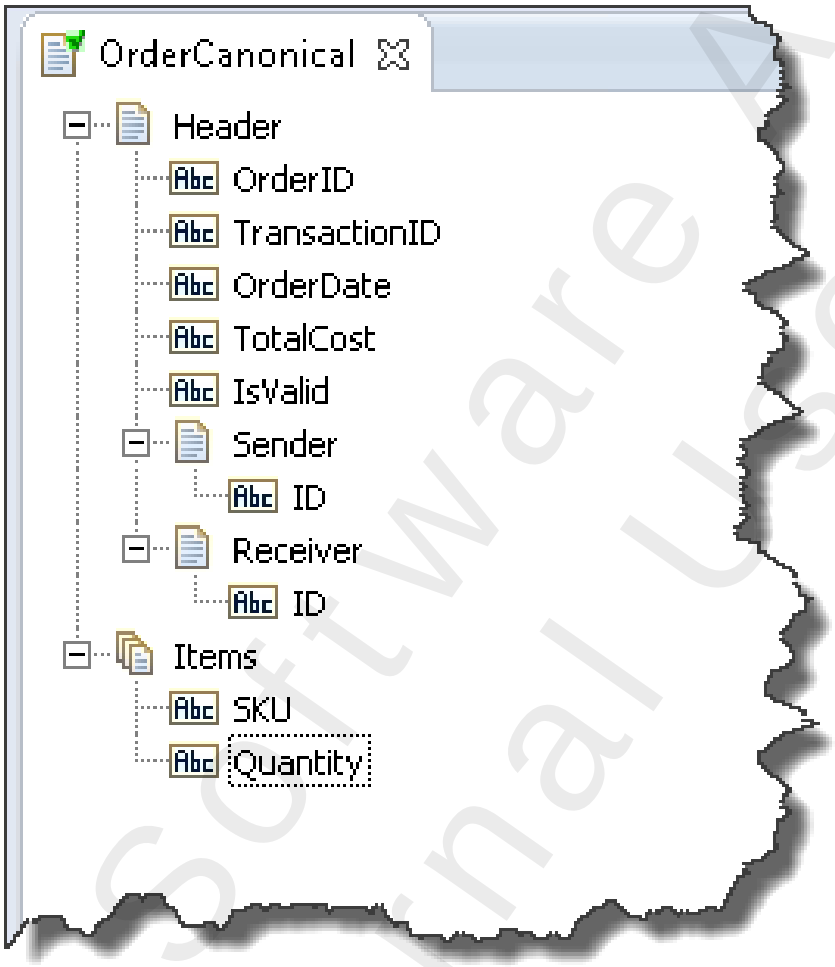
1. **ID** (String – be sure to indent under Receiver)

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)

The document structure should look like the following:



Save your new document types.

Check Your Understanding

1. Why are additional documents created in the acme.docs.request folder when the OrderRequest schema is imported?
2. What is the benefit of using a schema (XSD) for import over using a DTD?
3. What are the two ways to indent a Document Type element under another?

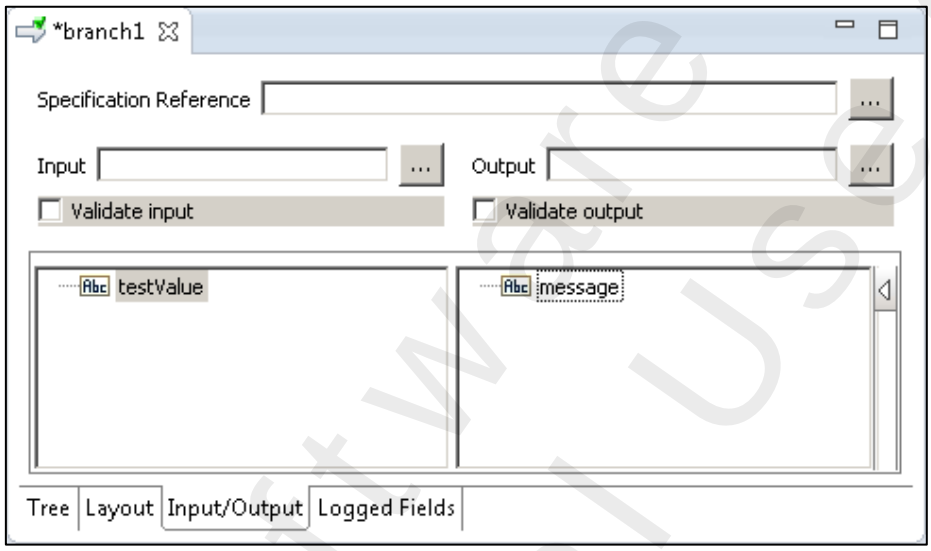
EXERCISE 5:
BUILDING FLOW SERVICES - BRANCH

Overview

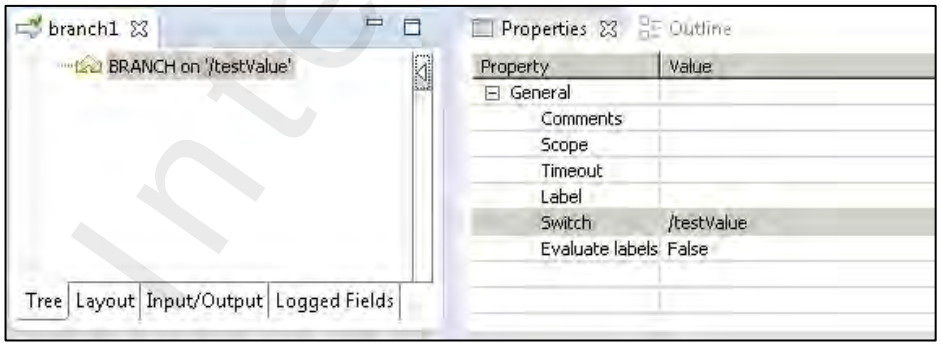
In this exercise, you will create business logic using two different Branch steps: one to test for contents of a variable, and one to evaluate labels associated with logic.

Steps

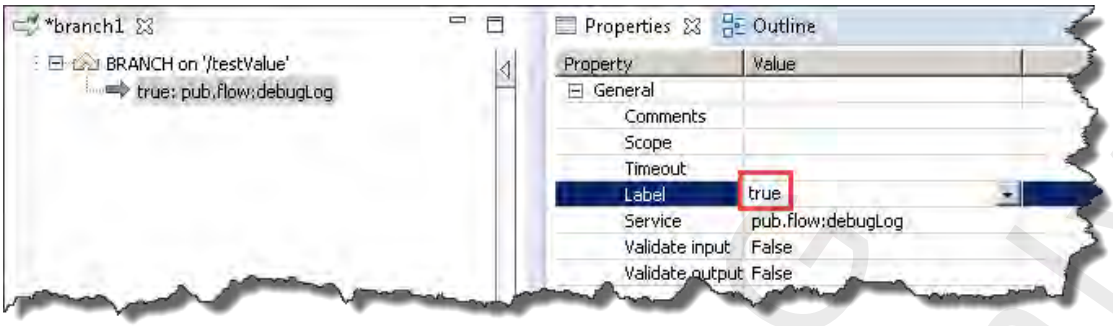
1. In the **acme.work** subfolder**,** create a new empty Flow service called **branch1.**
2. Define the input of branch1 as a single String called **testValue** and the output as a single String called **message**. Copy **testValue** from the Input pane.



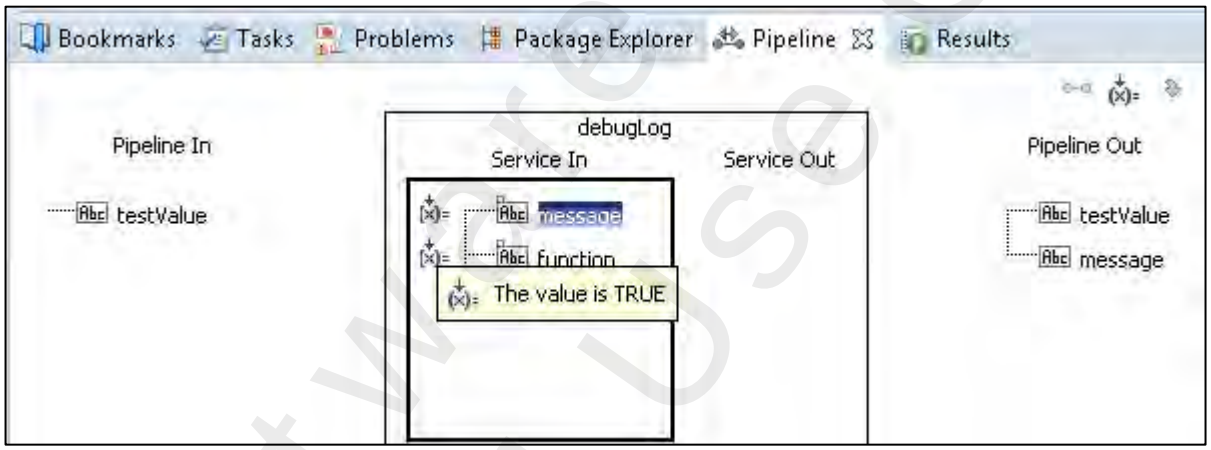
3. In the next steps you will add a BRANCH statement to the Flow service that conditionally writes a message to the Server Log, based on the contents of the testValue variable. For example, if testValue = true, write a message saying: "The value is TRUE" to the IS Server log. To do so:
   a. Select the **Tree** tab. Add a **BRANCH** statement to your service. Paste **testValue** as the **Switch** property. **Save** your work. After you save your work, you should notice that a "/" was added to the front of your **Switch** variable.

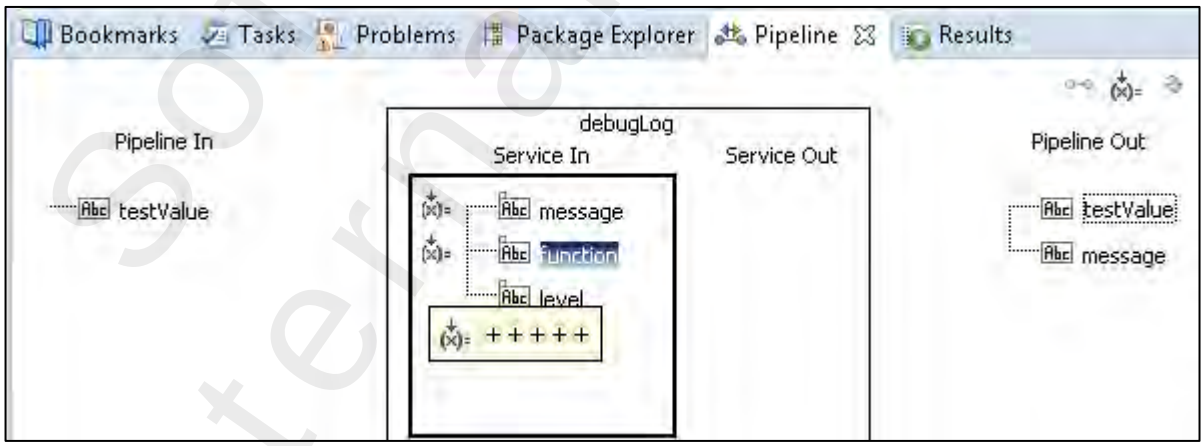


b. Add a **pub.flow:debugLog** statement below the Branch (be sure to indent it under the BRANCH so that it becomes part of the BRANCH logic). In the debugLog's properties, set the **Label** for the service to be **true** (all lower-case).

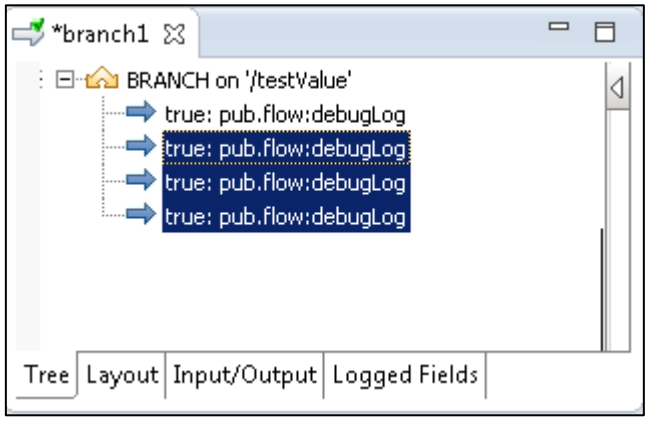


c. On the Pipeline tab for the debugLog statement, set the value of the variable message to The value is TRUE. As an eye-catcher, set the value of function to + + + + + .



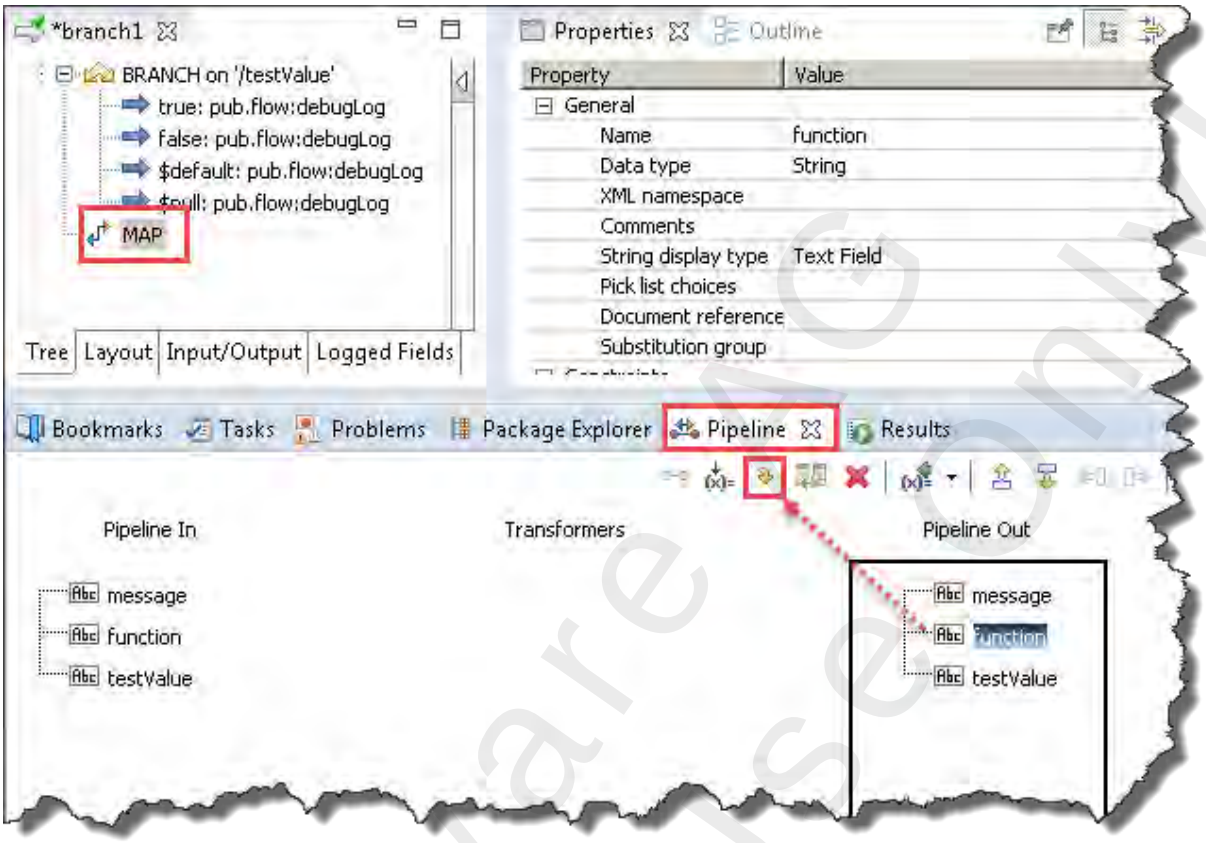


4. **Save** your work.
5. Copy your **pub.flow:debugLog** statement and paste it into your service three more times. Be sure to indent all of the service invokes under the BRANCH so that they become part of the BRANCH logic.

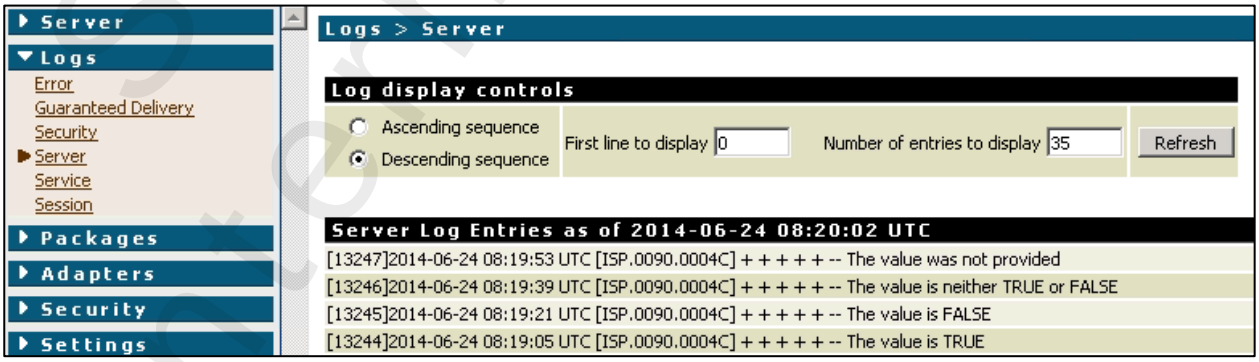


6. Set the **Label** property and the variable **message** for each of the service invokes as follows:
   a. **pub.flow:debugLog** (already completed in step 3, listed here as an example)
      - Label = **true**
      - message = **The value is TRUE**
      - function = **+++++**
   b. **pub.flow:debugLog**
      - Label = **false**
      - message = **The value is FALSE**
      - function = **+++++**
   c. **pub.flow:debugLog**
      - Label = **$default**
      - message = **The value is neither TRUE nor FALSE**
      - function = **+++++**
   d. **pub.flow:debugLog**
      - Label = **$null**
      - Message = **The value was not provided**
      - function = **+++++**

7. Add a **MAP** after your BRANCH. Make sure it is NOT indented under the BRANCH. In the Tree view, select your MAP and then select the Pipeline view. In Pipeline Out, select the **function** variable and click the Drop icon. Save your work.

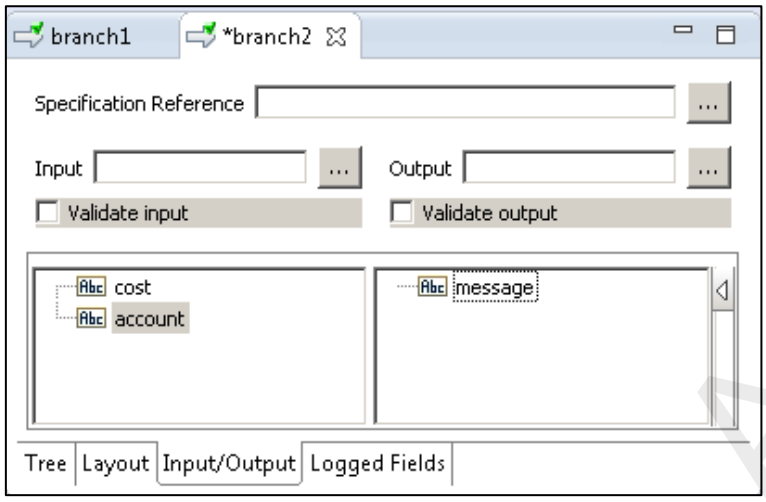


8. Test your service by running it several times with the following values for **testValue**. The output should appear in the Service Result view, and you can also look in the Server Log.
   a. testValue: **true**
   b. testValue: **false**
   c. testValue: **maybe**
   d. Click the **Clear** button to delete all values from testValue



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

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

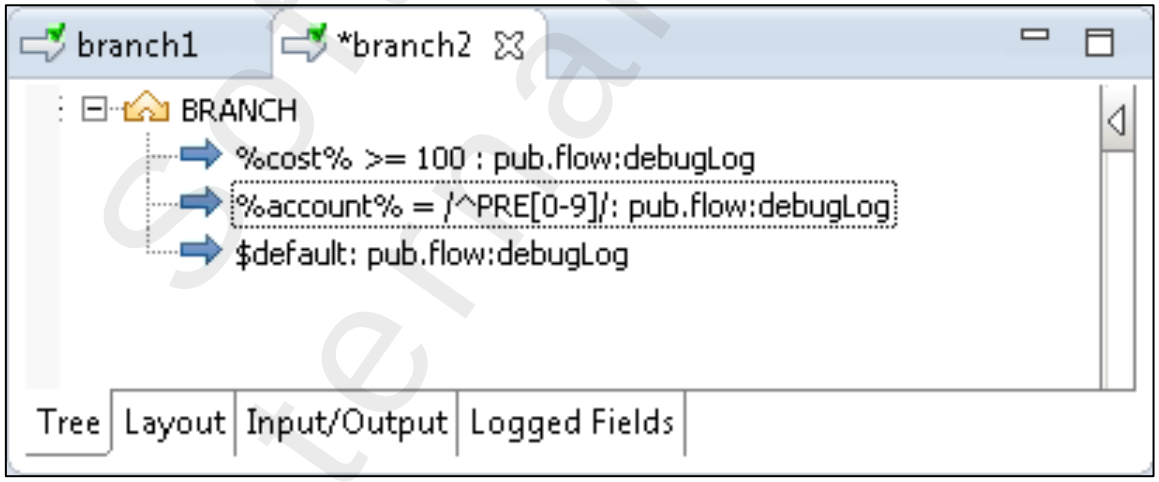


10. In the Tree pane, add a **BRANCH** and three **pub.flow:debugLog** service steps to write a message based on the value of the input fields to the Server Log.
Make sure all Invoke steps are indented under the BRANCH.
In this service, we want to evaluate labels, so be sure to leave the BRANCH **switch** 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 IS Server log.
    Note: **%cost% >= 100** evaluates the contents of cost at run-time.

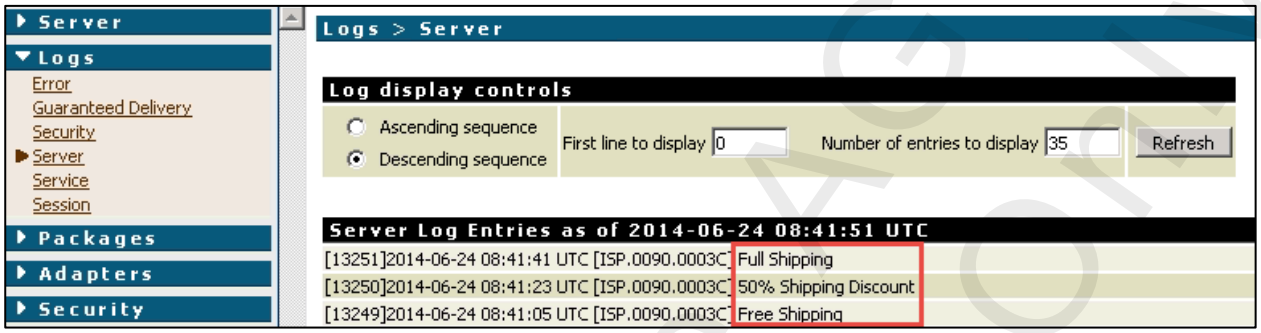
    b. If **account** starts with **PRE0** thru **PRE9** then write, **50% Shipping Discount** to the IS Server log.
    Note: you can test this in one step with a regular expression such as
    %account% = /^PRE[0-9]/

    c. Otherwise, write **Full Shipping** to the IS Server log.



11. Save and test the service. Run your service three times using the following inputs:

run	cost	account	In the IS log you should see
1	100	PRE42 Inc.	Free Shipping
2	42	PRE42 Inc.	50% Shipping Discount
3	42	Bar Inc.	Full Shipping



Check Your Understanding

1. When are regular expressions useful in a 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

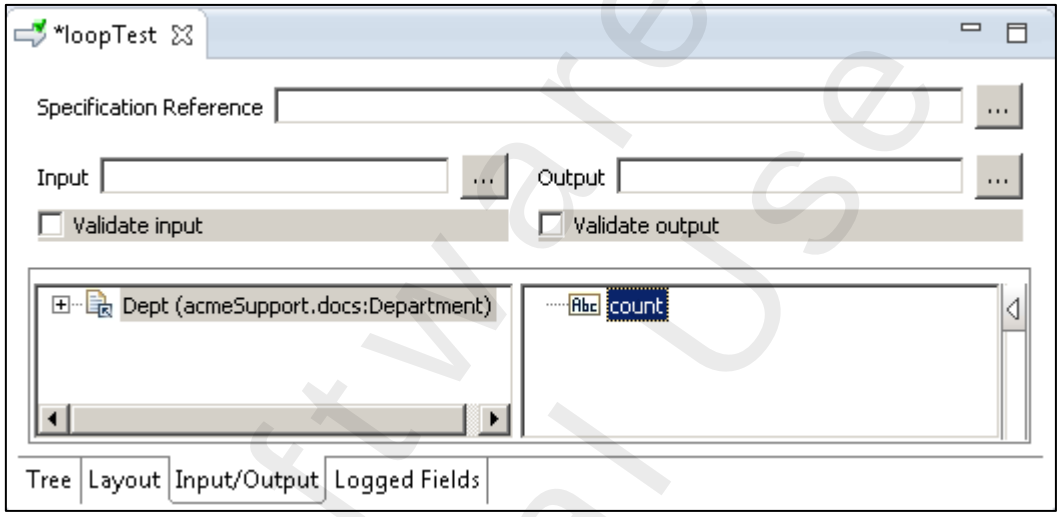
Overview

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

Steps

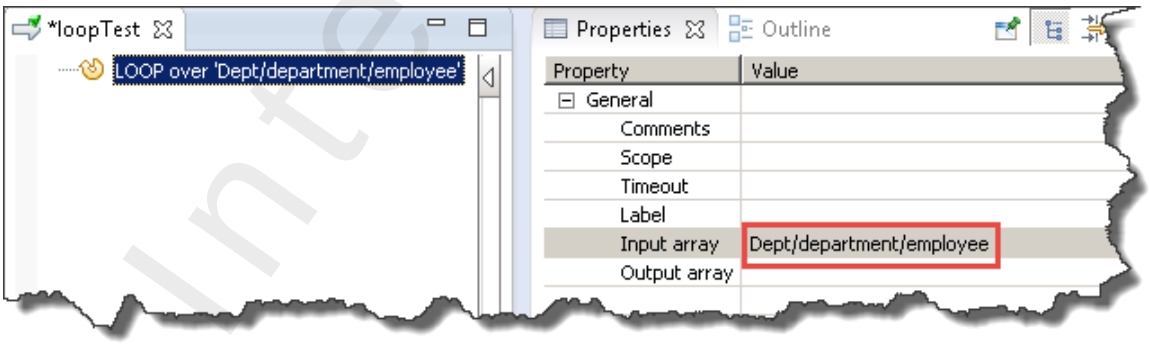
1. In the **acme.work** folder, create a new empty Flow service called **loopTest**. Set the input to be a Document Reference to **acmeSupport.docs:Department** called **Dept**, and set the output to be a single String field called **count**.

   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**. **Save** your work to see the Input array displayed in the LOOP.

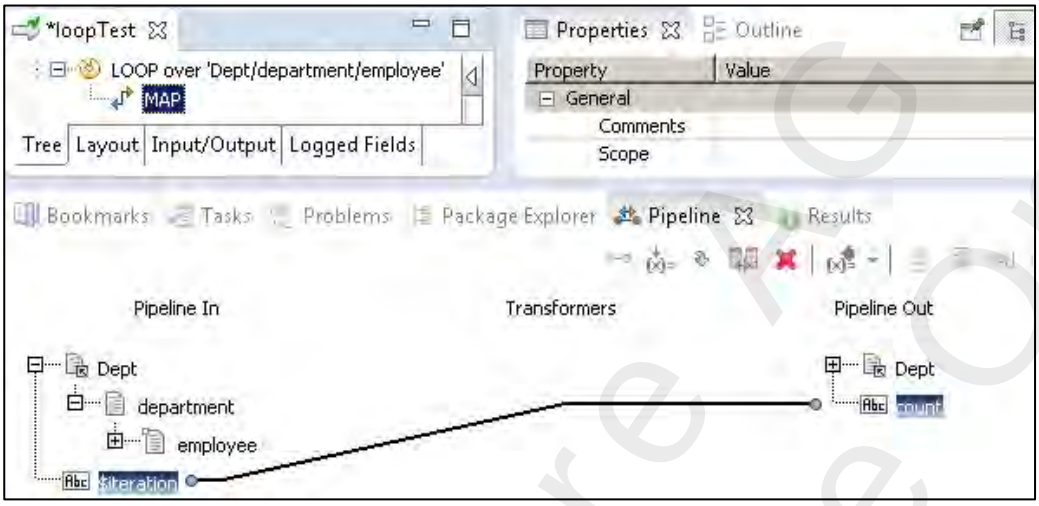


4. Add a **MAP** statement under the LOOP step (be sure it is indented under the LOOP). Select the MAP step in the Tree tab and use the Pipeline view to:

   a. confirm that **Dept/department/employee** is now a **Document** type.

   b. link **$iteration** to **count**.

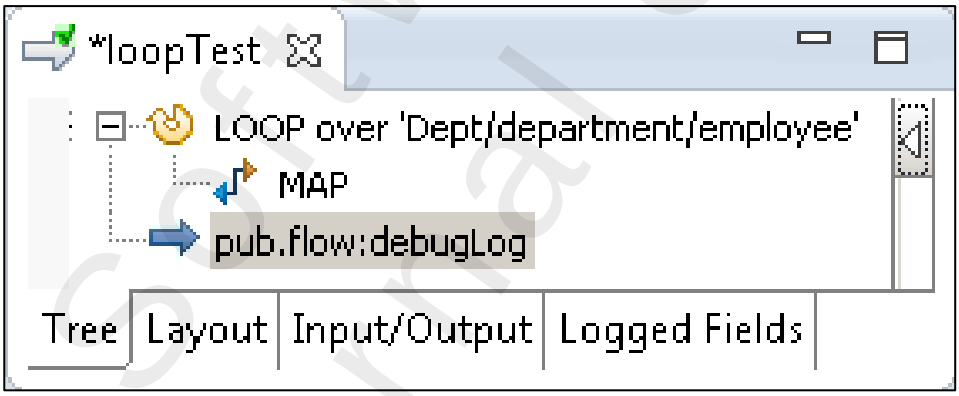
   Note: When a LOOP statement is added, a new variable called **$iteration** appears in the Pipeline. This variable keeps track of the number of iterations of the loop.

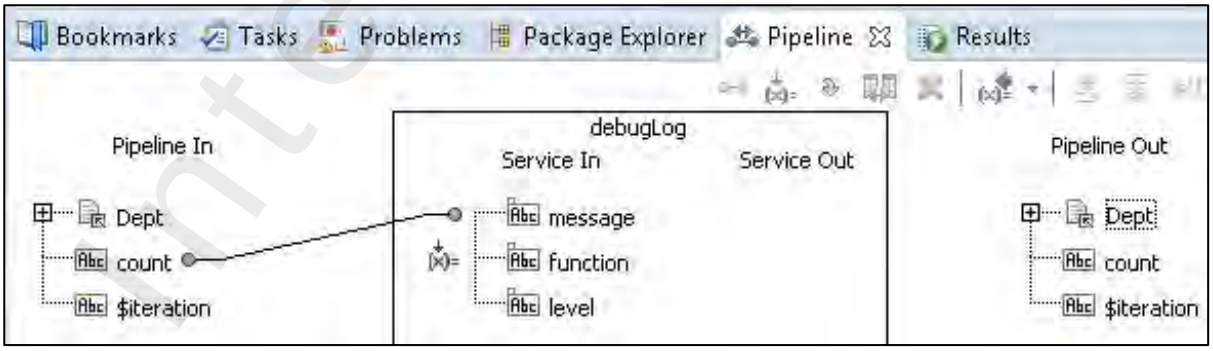


5. Add a **pub.flow:debugLog** below the MAP step (make sure it is NOT indented under the LOOP). In the Pipeline view for the Invoke step:

   a. link **count** to **message**

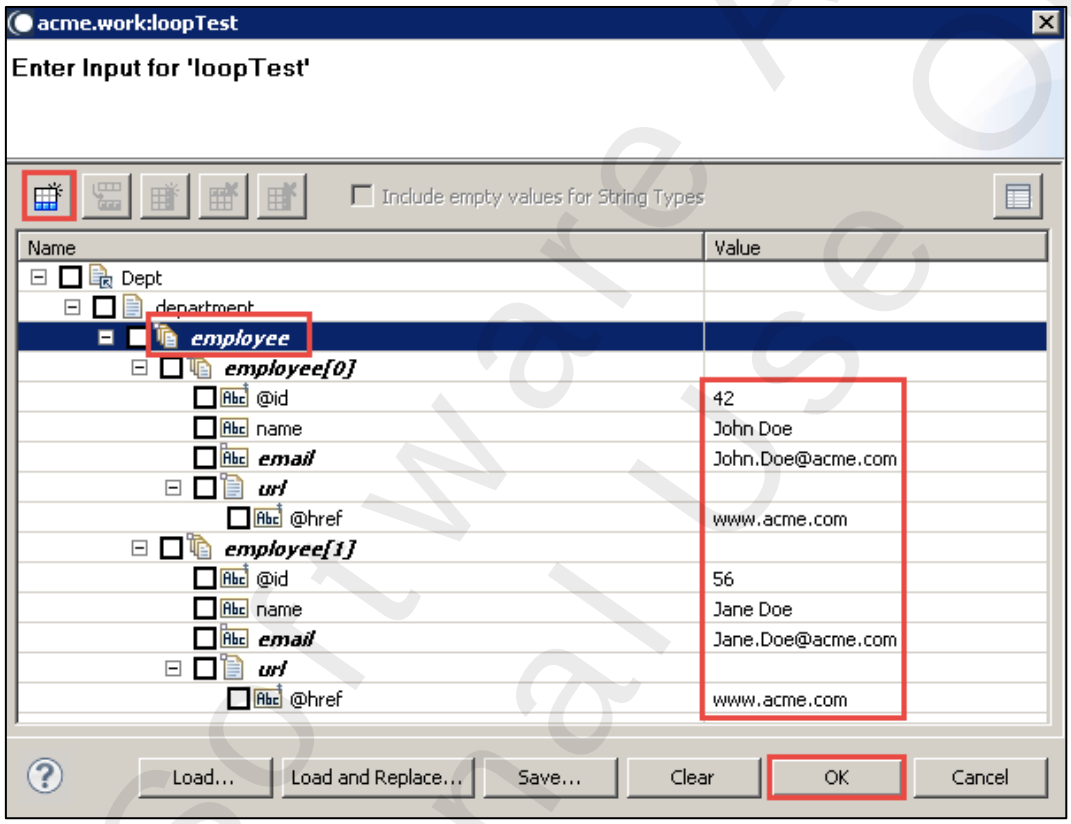
   b. set **function** to **+ + + + +**

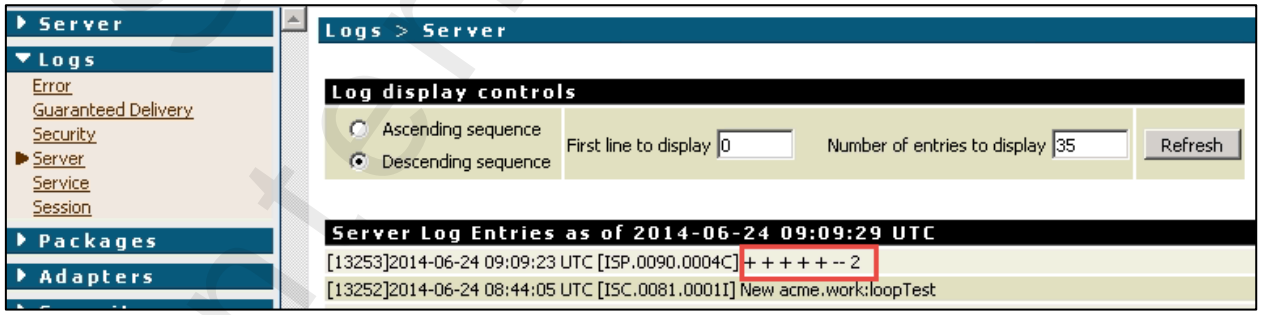




6. **Save** your work. In your debugLog's Pipeline Out, drop the following variables because they are not a part of the Input / Output for this service:
   a. **message**
   b. **function**
   c. **$iteration**
7. **Save** your work and run the service.

   When the service requests its input, select **employee**, 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 step was 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

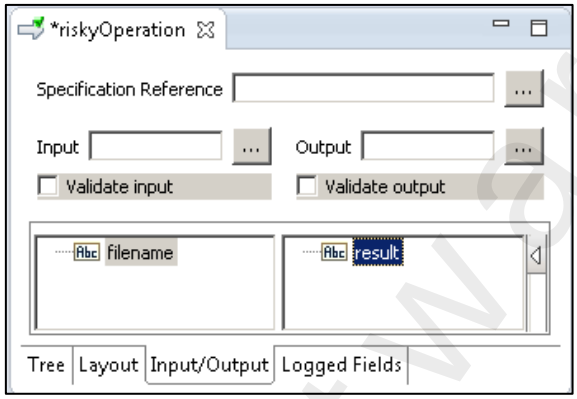
Overview

In this exercise, you will create a Flow service that implements error handling (Try/Catch) using three Sequences.

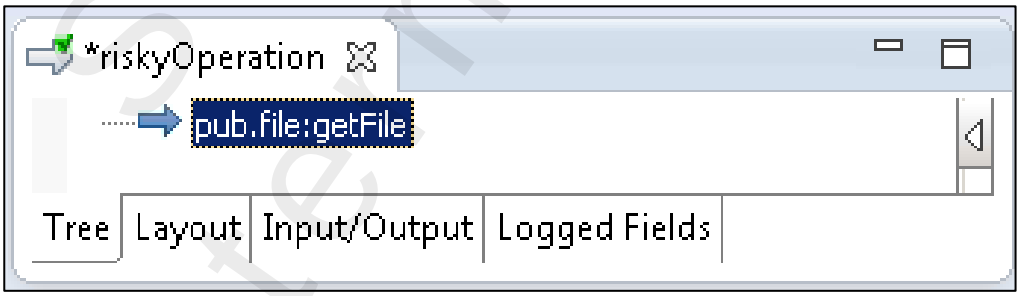
First you will create a service that returns an exception, and then you will embed this service in a Try/Catch sequence.

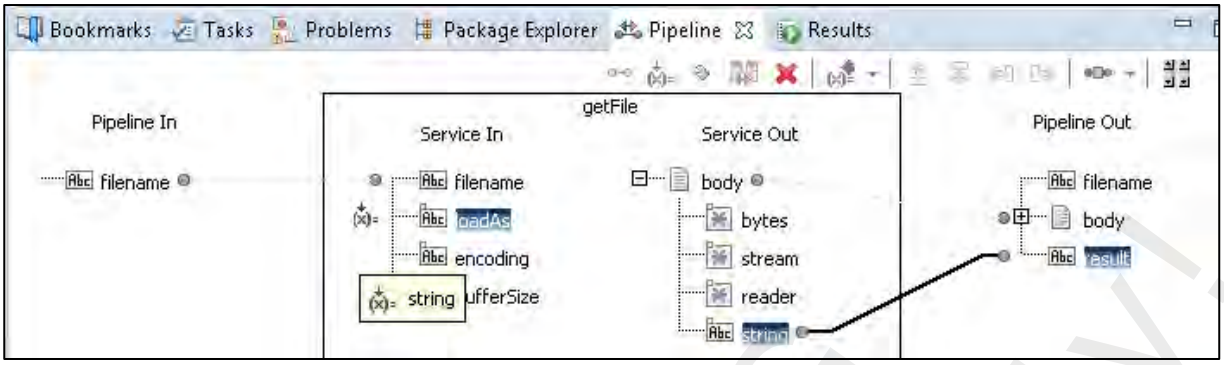
Steps

1. Create the flow service **acme.work:riskyOperation**. Set the input to a String called **filename** and the output to a String called **result**.



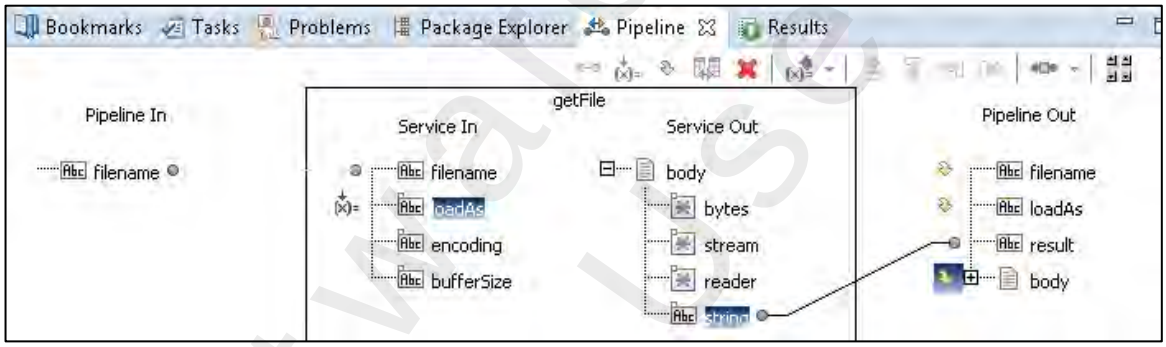
2. Add an invocation of the **WmPublic** package's **pub.file:getFile** to your service.
   a. Select **string** as input value for input field **loadAs** of getFile
   b. Link Service Out's **body/string** to **result**



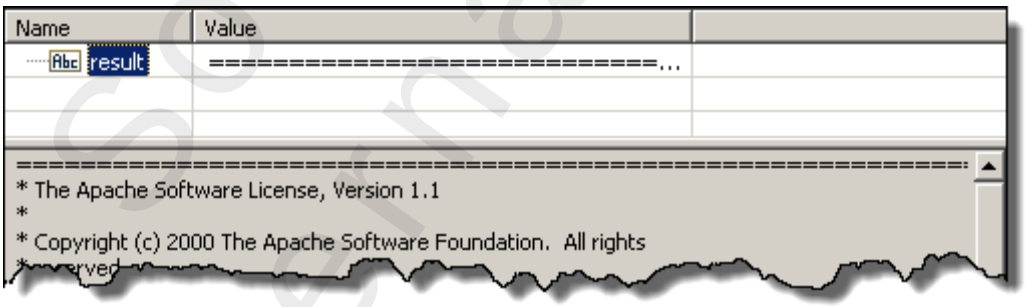


3. **Save** your work. In Pipeline Out, **Drop** the following variables:
   a. **filename**
   b. **loadAs**
   c. **body**

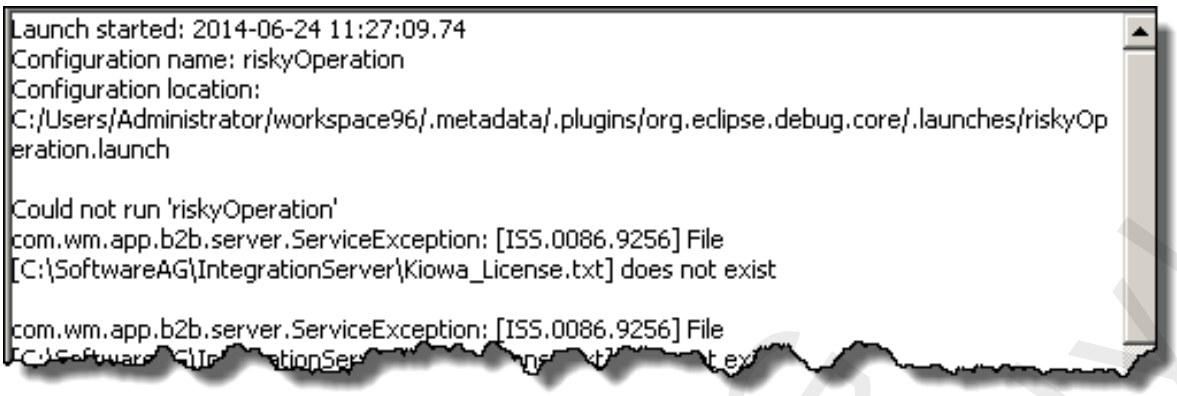
   Note: We are dropping the input **filename** because this service will be called by a client service later in this course.



4. Save and run the service twice, setting the following inputs:
   a. filename: **C:\SoftwareAG\IntegrationServer\Apache_License.txt** (this file exists). What result do you receive?

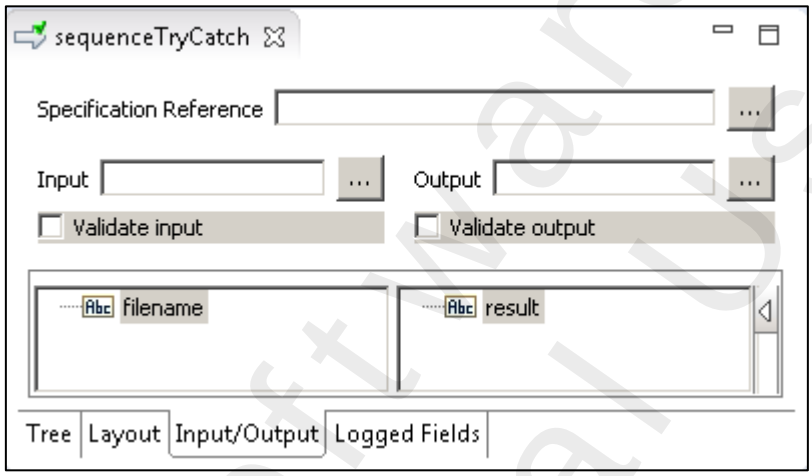


   b. filename: **C:\SoftwareAG\IntegrationServer\Kiowa_License.txt** (this file does NOT exist)**.** What result do you receive?



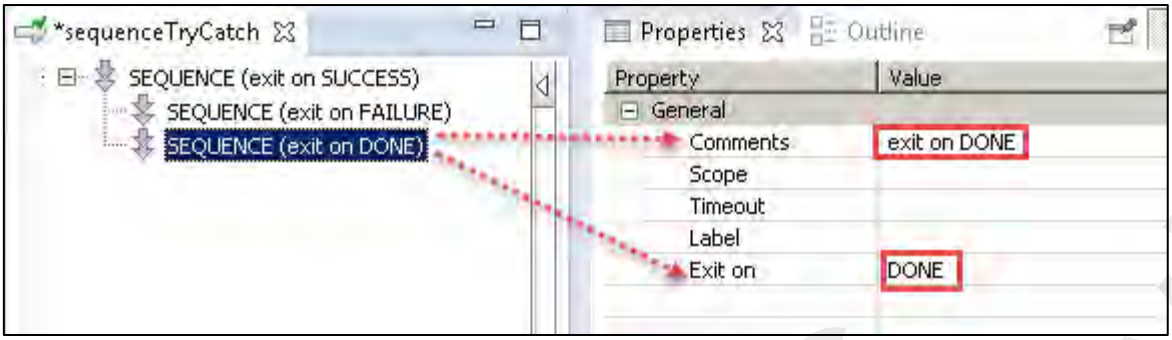
5. Now, create the flow service **acme.work:sequenceTryCatch.**

   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 **result**.

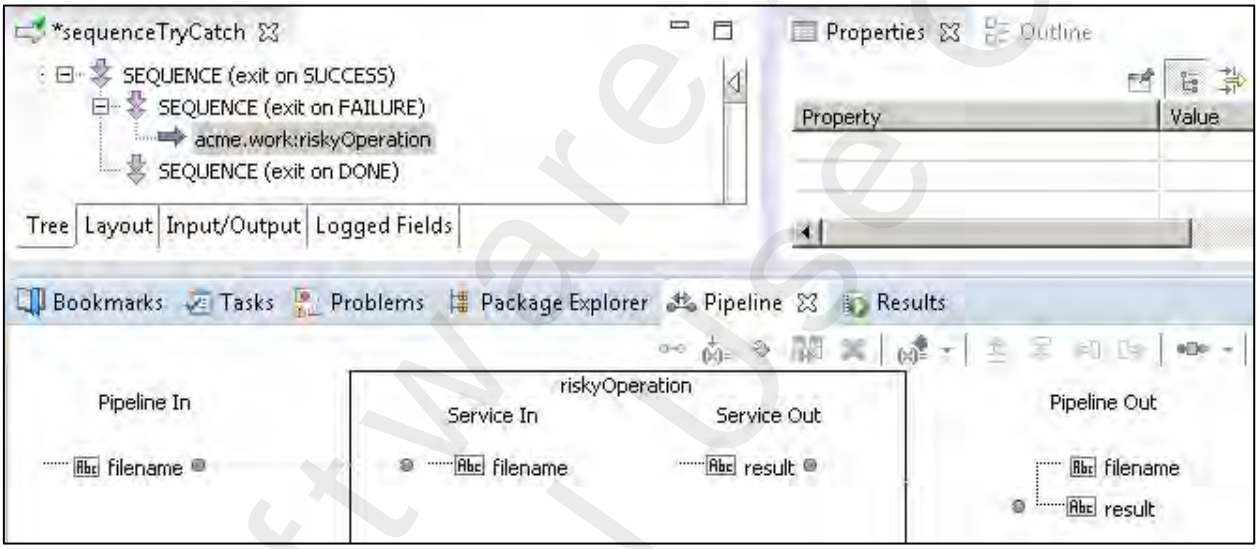


6. In the **sequenceTryCatch** service, add three **SEQUENCE** steps to create a Flow try/catch block, as follows:
   a. **SEQUENCE** set **Exit on** to **SUCCESS**
      i. **SEQUENCE** set **Exit on** to **FAILURE** (be sure it is indented under the first SEQUENCE)
      ii. **SEQUENCE** set **Exit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

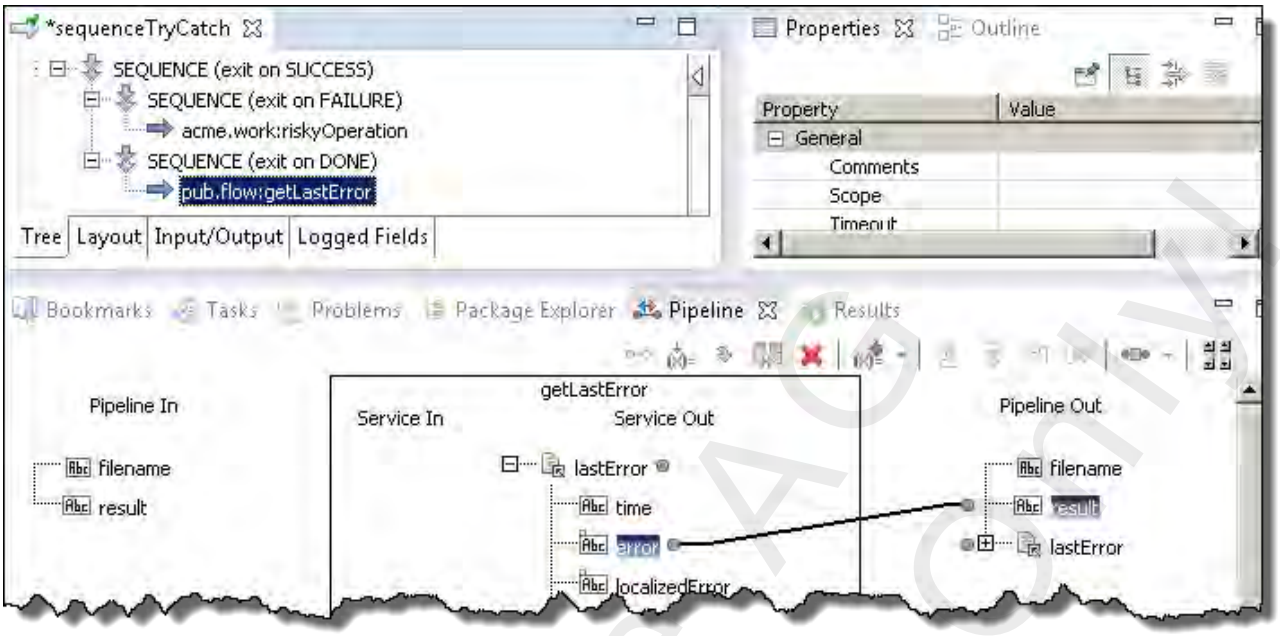
   While creating the individual SEQUENCE statements, set their comment property to reflect the **exit on** condition.



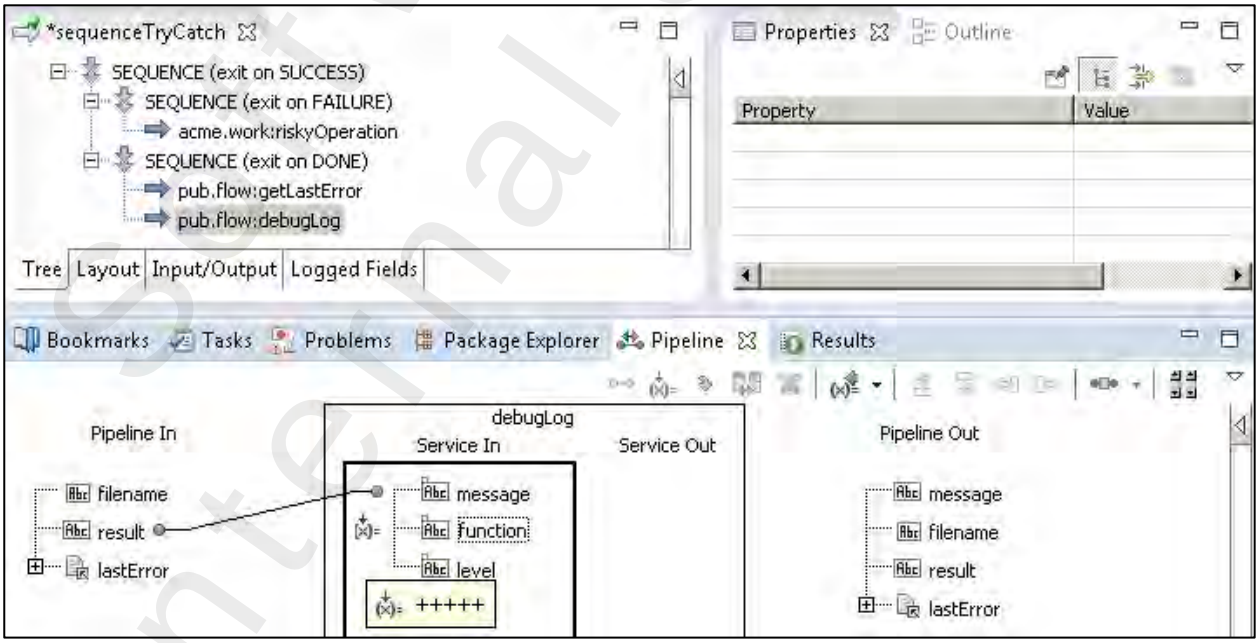
7. In the SEQUENCE Exit on FAILURE, add service **acme.work:riskyOperation** (be sure it is indented under the SEQUENCE exit on FAILURE). In the Pipeline view, confirm that the following are implicitly linked:
   a. **filename** to **filename**
   b. **result** to **result**



8. In the SEQUENCE Exit on DONE, add the **WmPublic** package's **pub.flow:getLastError** (be sure it is indented under the SEQUENCE exit on DONE). In the Pipeline view, link Service Out's **lastError/error** to Pipeline Out's **result.**

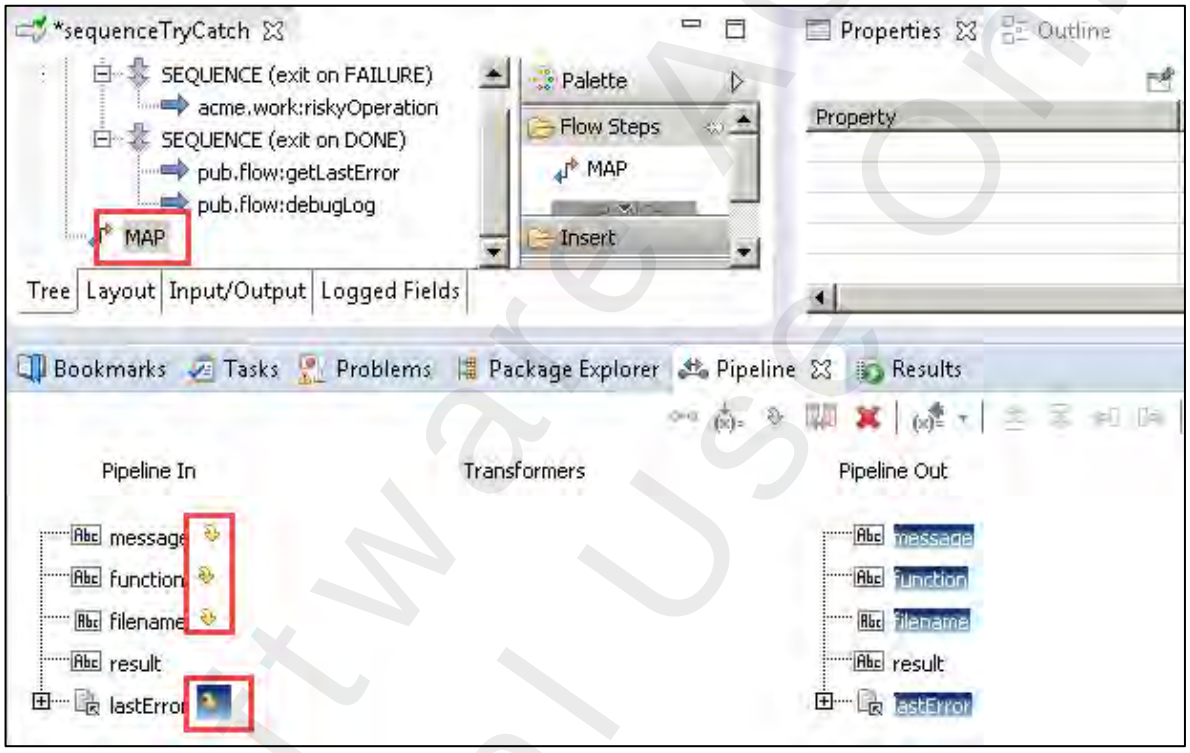


9. Add an invocation of the **pub.flow:debugLog** service below the invocation of **pub.flow:getLastError**.
   a. Link Pipeline In's **result** to Service In's **message.**
   b. Set **function** to **+++++ .**
   c. **Save** your work.



10. Add a **MAP** to the end of your service. Make sure it is NOT indented under any of your SEQUENCEs. In the MAP, **Drop** the following variables:
    a. **message**
    b. **function**
    c. **filename**
    d. **lastError**

    Save your work.
    Note: After saving your work, dropped variables will only appear in **Pipeline In** of the MAP statement.



11. Run your **sequenceTryCatch** service. Check the **service Results view** and the IS Server log.
    a. To succeed, provide **C:\SoftwareAG\IntegrationServer\Apache_License.txt.** Successful execution will show the file contents in the **service Results view** and no error message in the IS Server log.
    b. To fail, provide a file that does not exist (e.g. **C:\SoftwareAG\IntegrationServer\Kiowa_License.txt)**. Verify you see an error message in the IS Server log.

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)?

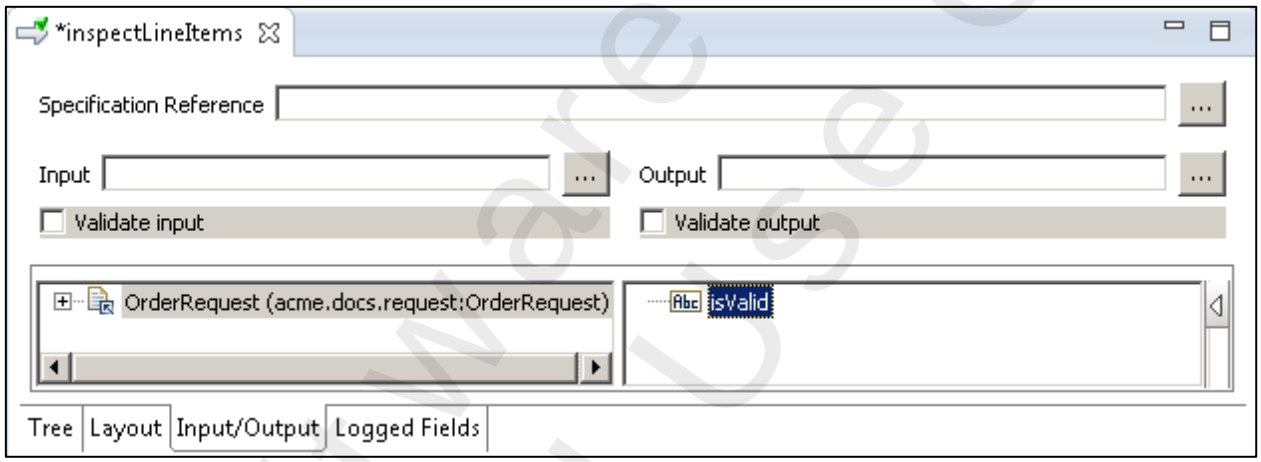
EXERCISE 8:
VALIDATION SERVICE

Overview

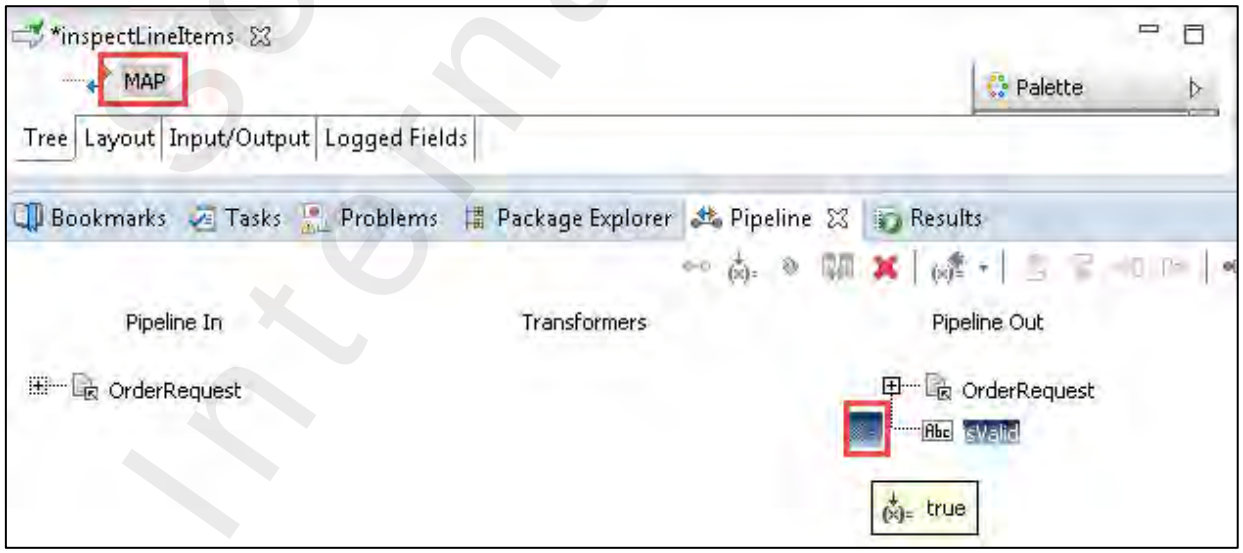
In this exercise, you will create business logic to validate an inbound Purchase Order. For now, this means to verify that the line items in the Purchase Order have a valid quantity. If this is not the case, you will flag the order as invalid for follow up by a customer service representative.

Steps

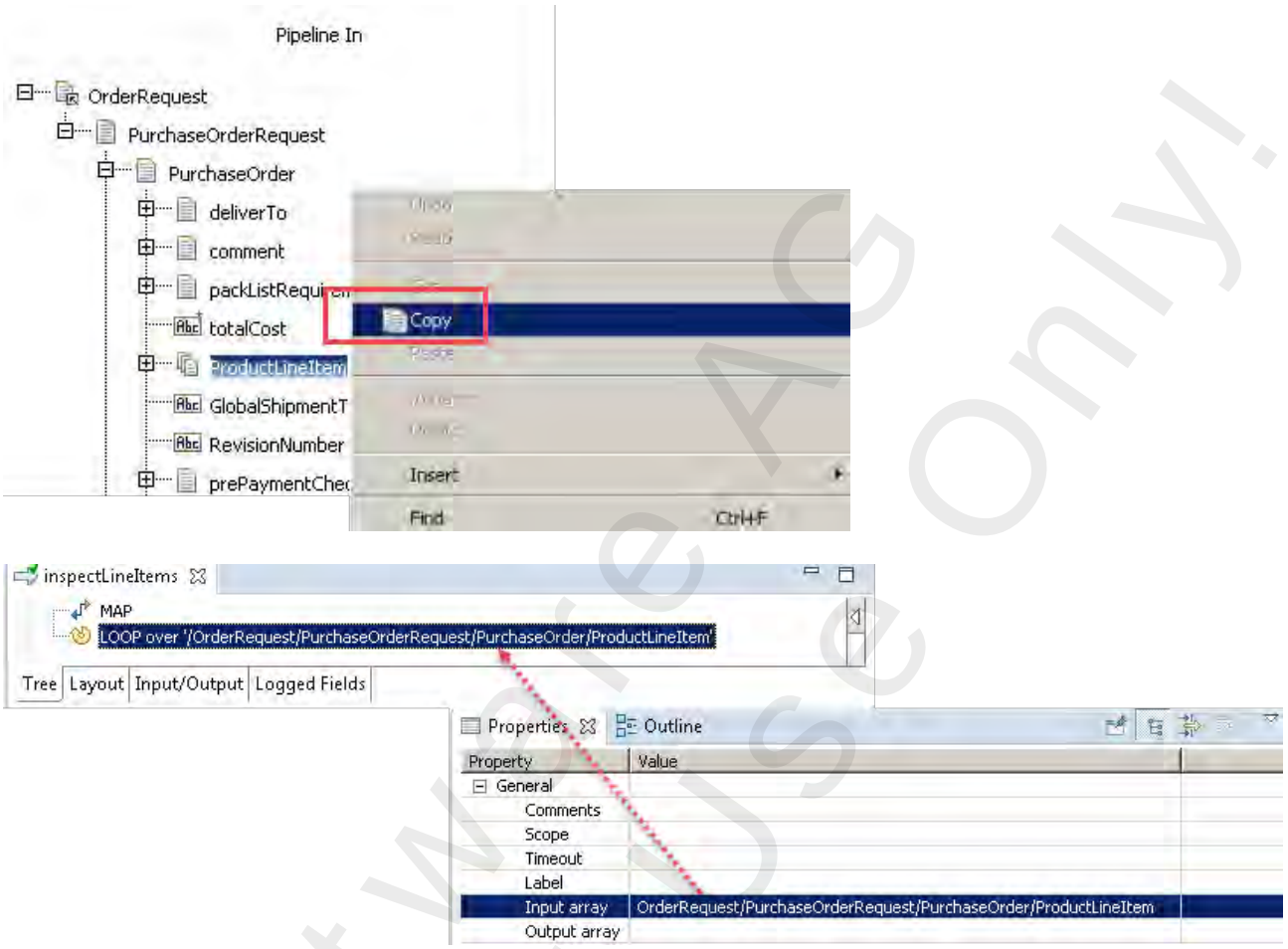
1. Create the flow service **acme.utils:inspectLineItems.** For input, specify a document reference to **acme.docs.request:OrderRequest.** Call this input Variable **OrderRequest**. As output variable, create a String called **isValid**.



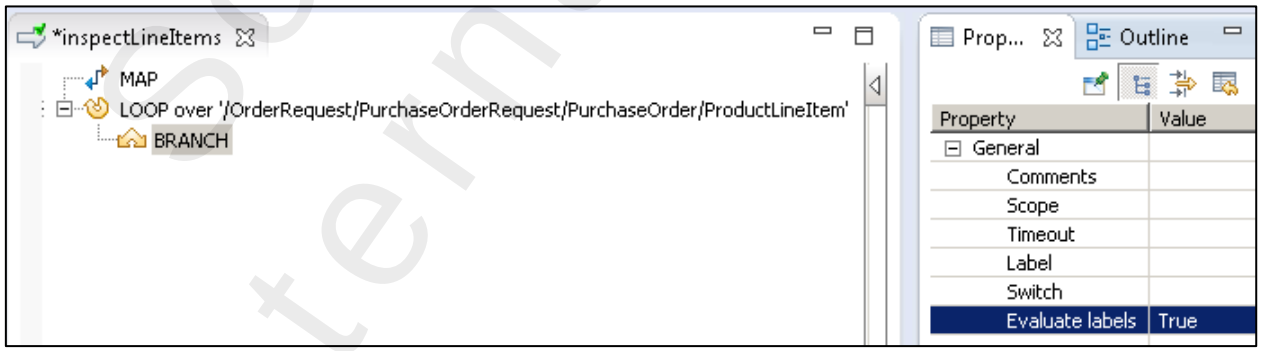
2. In the service, add a MAP step and then set the **isValid** variable to **true**.



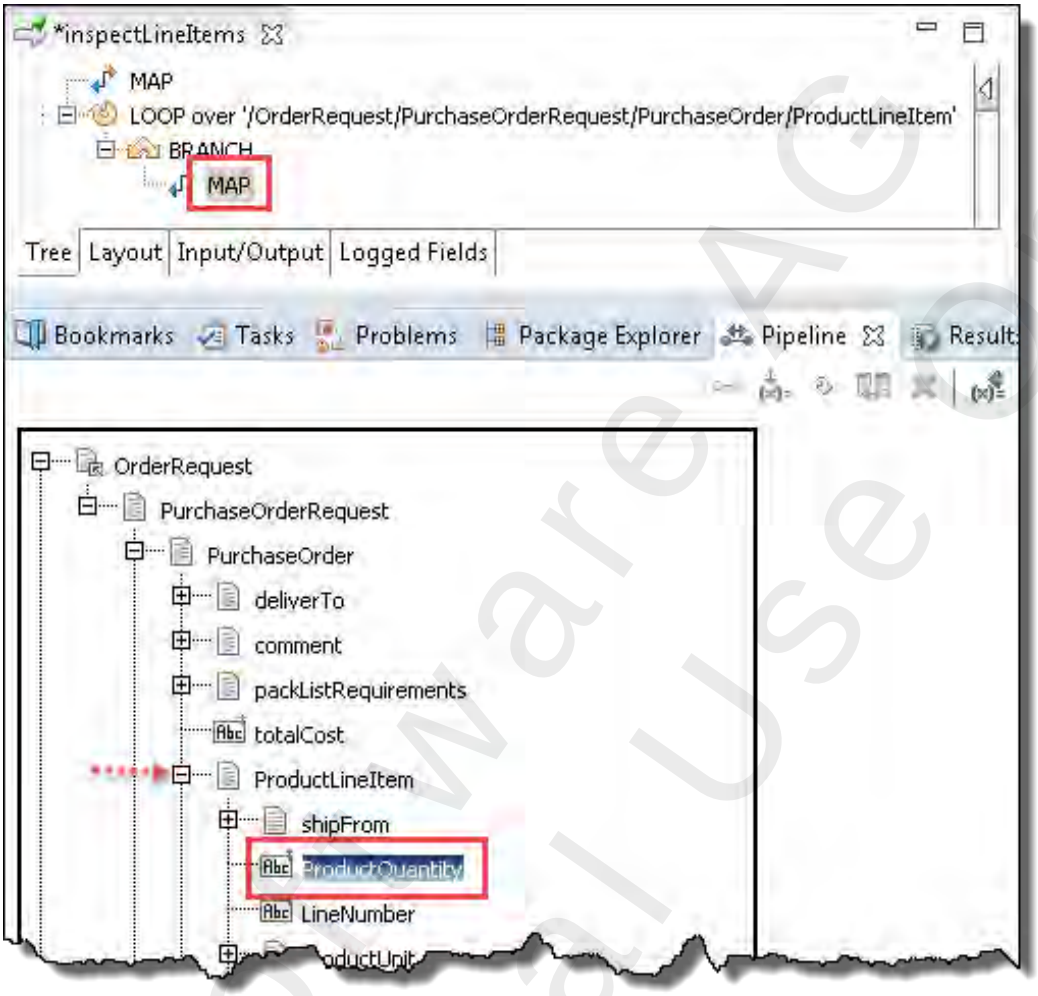
3. In the Pipeline view, locate and copy the **OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem** field. Then add a **LOOP** step to your service. Set its **Input array** property by pasting the copied value. **Save** your work.



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 the **Evaluate labels** property to **True**.



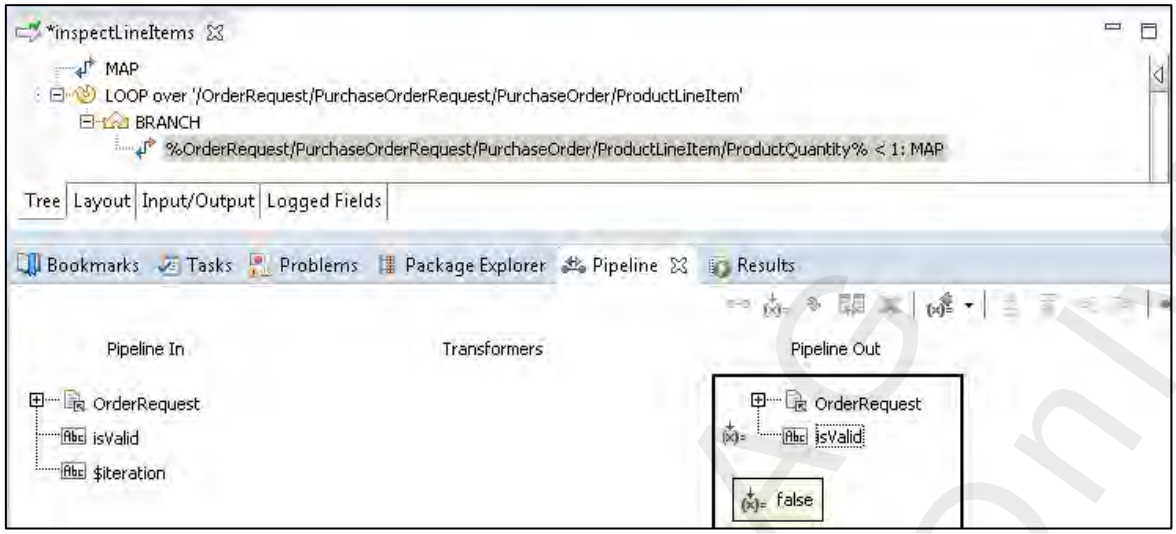
5. **Save** your work and add a **MAP** step below the BRANCH step (be sure it is indented under the BRANCH). In the Pipeline tab associated with the MAP you just added, locate and copy the **/OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/ProductQuantity** field.
   Hint: If the field is not visible in the Pipeline view, save your service. Also, confirm that **ProductLineItem** is a Document (NOT a Document List) in this MAP. If it is Document List, see your instructor to debug your service.



6. In the MAP's **Label** property, type **"%% < 1"** (six characters, no quotes, around the **less than** sign are spaces)

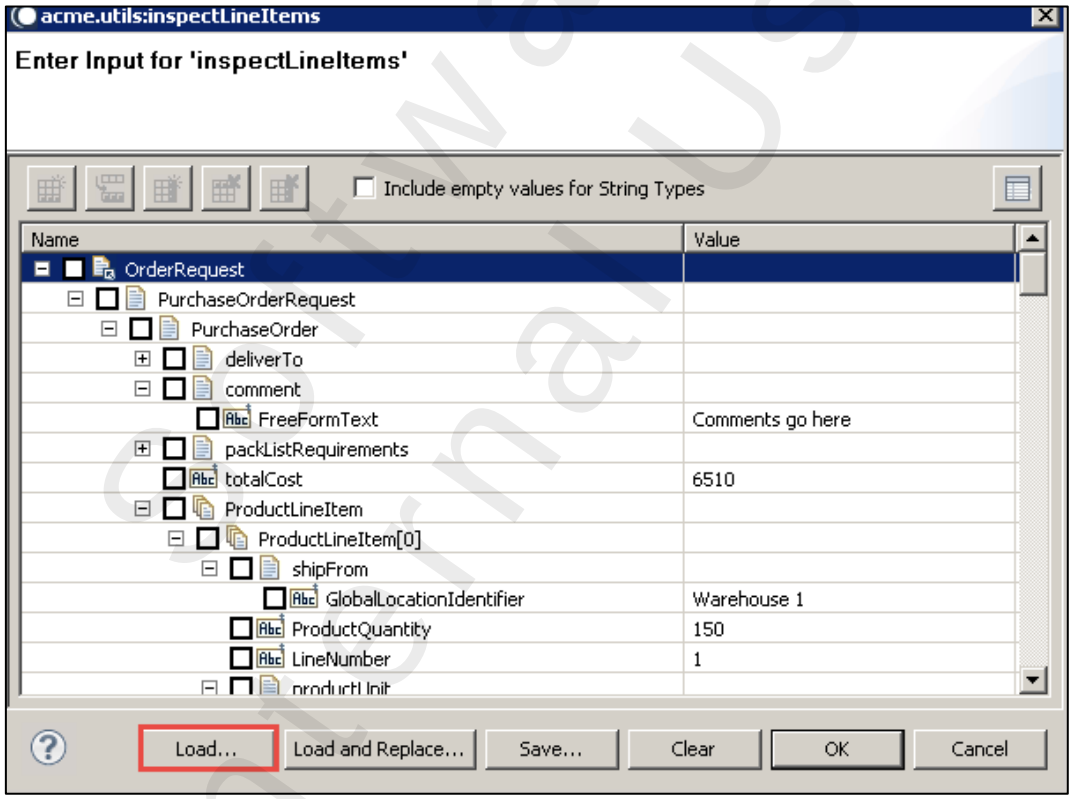
   Then paste the copied value between the two percent signs. You should end up with **%OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/ProductQuantity% < 1**.

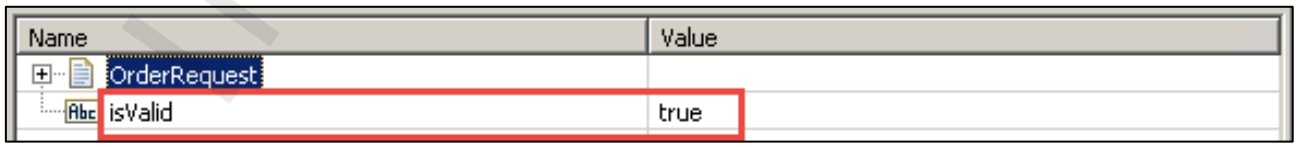
7. In the Pipeline of the MAP step, set **isValid** = **false**. Your service should look like this:



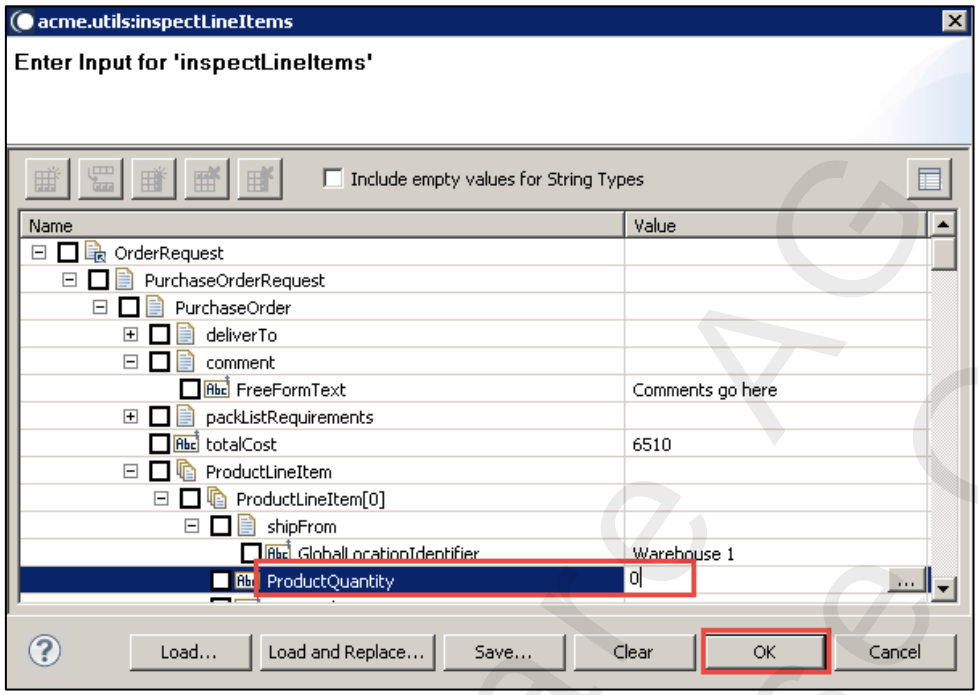
8. **Save** and run the service. To specify the input, use the **Load Inputs** button and select the file **order_request_input.txt** from folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub**.

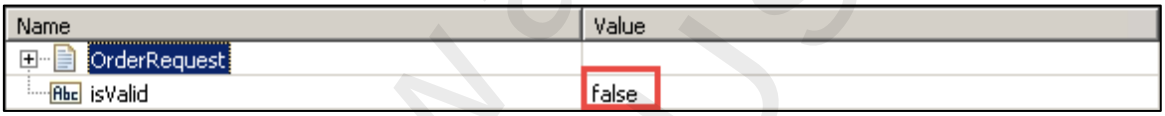
   After running the service, collapse the **OrderRequest** Document and then confirm that **isValid** is **true** in the Service Results tab.





Run the service again and change the first item's **ProductQuantity** value in the **ProductLineItem** array to **0**. **isValid** should be **false** in the service Results view.





9. *For extra credit*: Implement the service to 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 a way we could have validated all of the fields in the OrderRequest?

This Page intentionally left blank

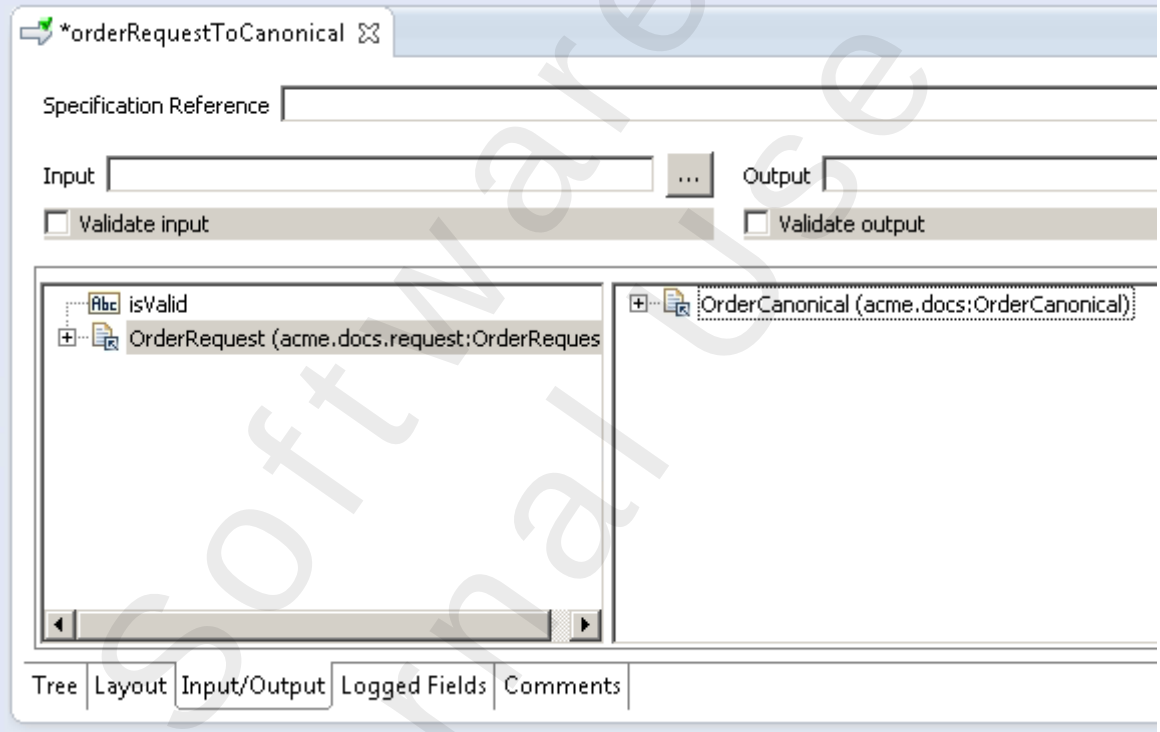
EXERCISE 9:
MAPPING SERVICE

Overview

In this exercise, you will create a Flow service that maps data from the OrderRequest document to the OrderCanonical document.

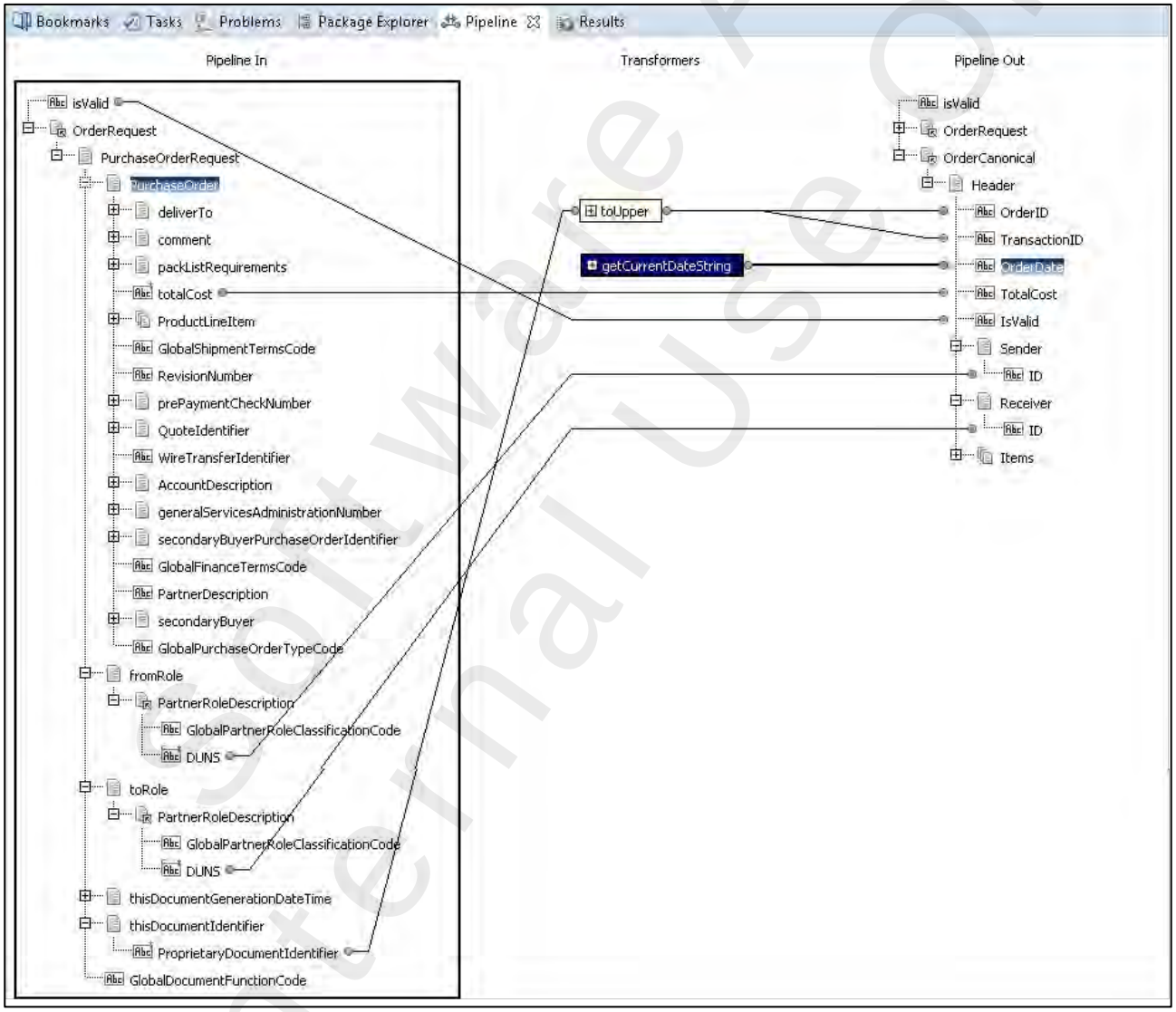
Steps

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

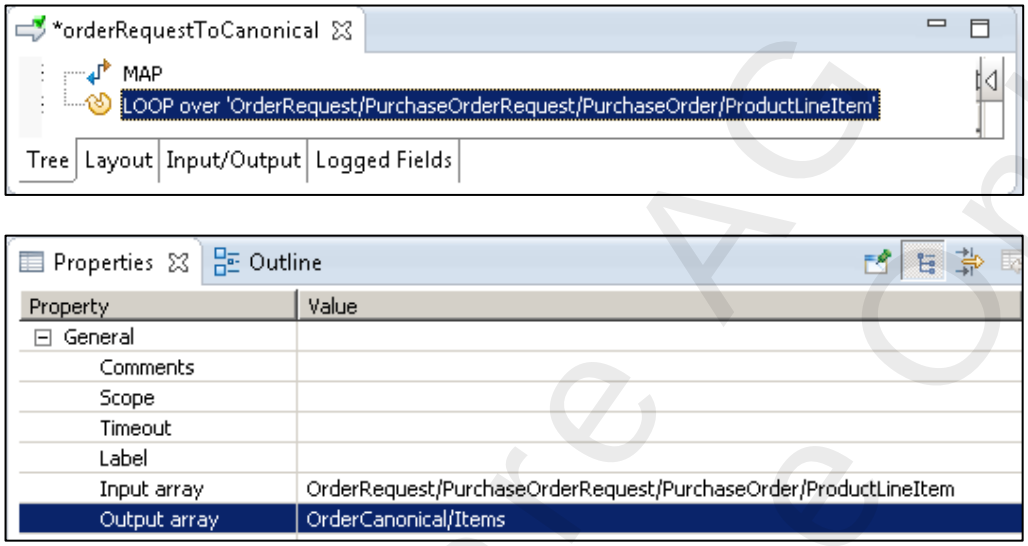


2. Add a **MAP** statement to your service. In the MAP statement, link the following Header variables from left to right on the Pipeline tab.
   a. **OrderRequest/PurchaseOrderRequest/PurchaseOrder/totalCost** to **OrderCanonical/Header/TotalCost**
   b. **isValid** to **OrderCanonical/Header/IsValid**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

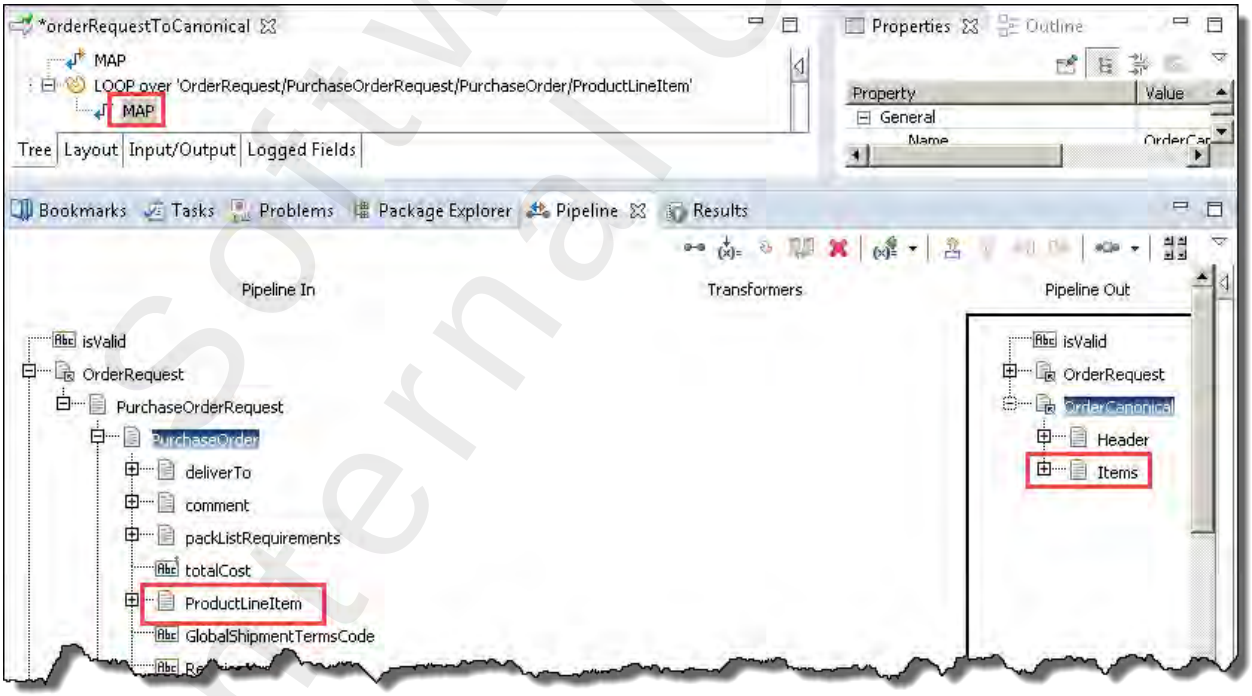
3. Add the service **pub.string:toUpper** as a **Transformer** in the MAP step. Link the following variables from **OrderRequest** to the transformer and from the Transformer to **OrderCanonical**:
   a. **OrderRequest/PurchaseOrderRequest/thisDocumentIdentifier/ ProprietaryDocumentIdentifer** to the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



5. Add a **LOOP** step to the service. In the following, please copy and paste the values from the Pipeline tab of the MAP you just worked with. This will minimize typographical errors.
   a. Set the **Input array** property to **OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem**
   b. Set the **Output array** property to **OrderCanonical/Items**

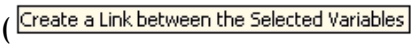


6. Add a **MAP** step in the LOOP (be sure it is indented under the LOOP statement). Confirm that the OrderRequest's **ProductLineItem** and the OrderCanonical's **Items** are now both of type Document. If they are not, please check the Input array and/or Output array.

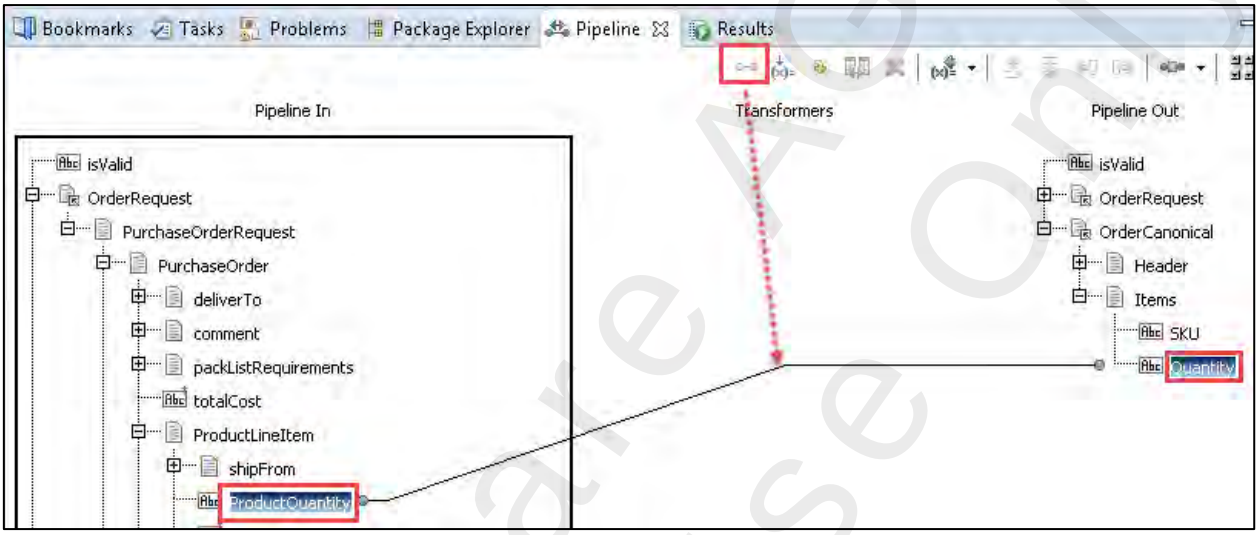


7. In the MAP, link the **OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/ProductQuantity** to the **OrderCanonical/Items/Quantity**.
   Hint: When mapping such deeply nested structures, it is often easier to create a link in two steps. First select the **from** and the **to** fields by clicking them with the mouse. In the second step connect the two selected fields by clicking on the

   Create a Link... (

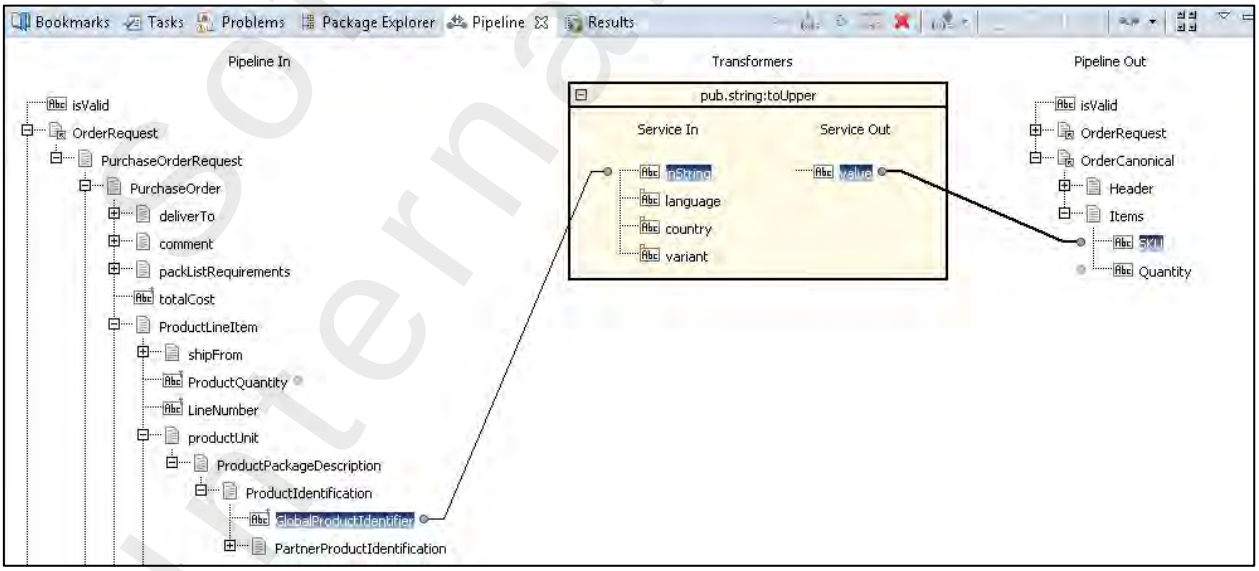
) button.
   This button is located in the title bar of the pipeline window.



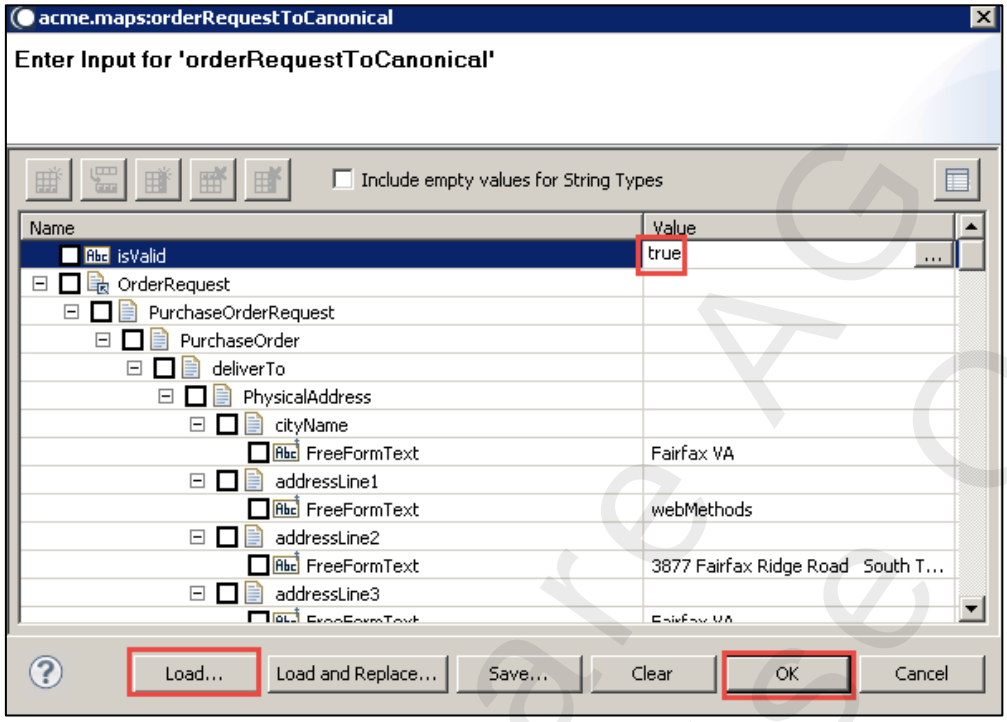
8. Add the service **pub.string:toUpper** as a **Transformer** in the MAP step. Link the following variables:

   a. **OrderRequest/PurchaseOrderRequest/PurchaseOrder/ProductLineItem/productUnit/ProductPackageDescription/ProductIdentification/GlobalProductIdentifier** to the Transformer's **inString**

   b. Transformer's **value** to **OrderCanonical/Items/SKU**



9. **Save** the service and run it.
   a. Use the **Load** button to load the input file **order_request_input.txt** from folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub.**
   b. Set the **isValid** to **true** and click **OK.**



Check the Results panel. Collapse **OrderRequest** and look at **OrderCanonical.** Confirm that the **OrderCanonical** data structure is completely populated. Check that **OrderDate** and **IsValid** have proper values. Also, check the **OrderID**, **TransactionID,** and **SKU** values to make sure that they are all uppercase.

Name	Value
isValid	true
OrderRequest	
OrderCanonical	
Header	
OrderID	021213153012A
TransactionID	021213153012A
OrderDate	June 24, 2014
TotalCost	6510
IsValid	true
Sender	
ID	88-888-8888
Receiver	
ID	11-111-1111
Items	
Items[0]	
SKU	CHAINSAW
Quantity	150
Items[1]	
SKU	TIME MACHINE
Quantity	163
Items[2]	
SKU	SLEDGEHAMMER
Quantity	120

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?

EXERCISE 10:
CREATE A JAVA SERVICE

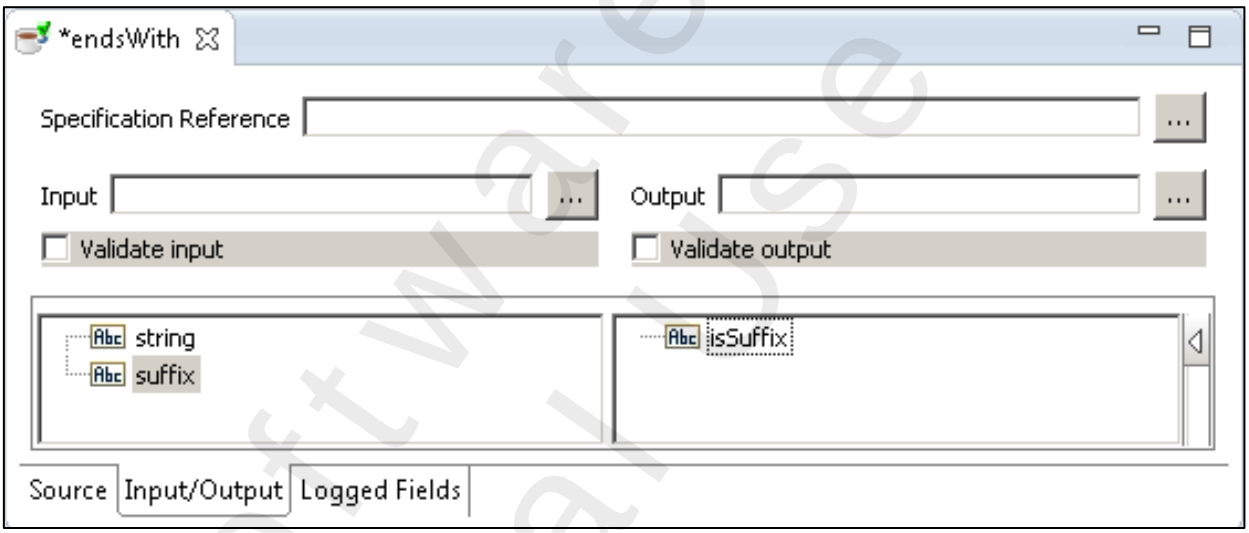
Overview

In this exercise, you will create, compile, and run an IS Java Service using Designer.

Imagine that you require a special utility service that tests if a string ends in a second string. There is no such service in the pub.string folder therefore you will create it. This new service will make use of the Java method **String.endsWith()**.

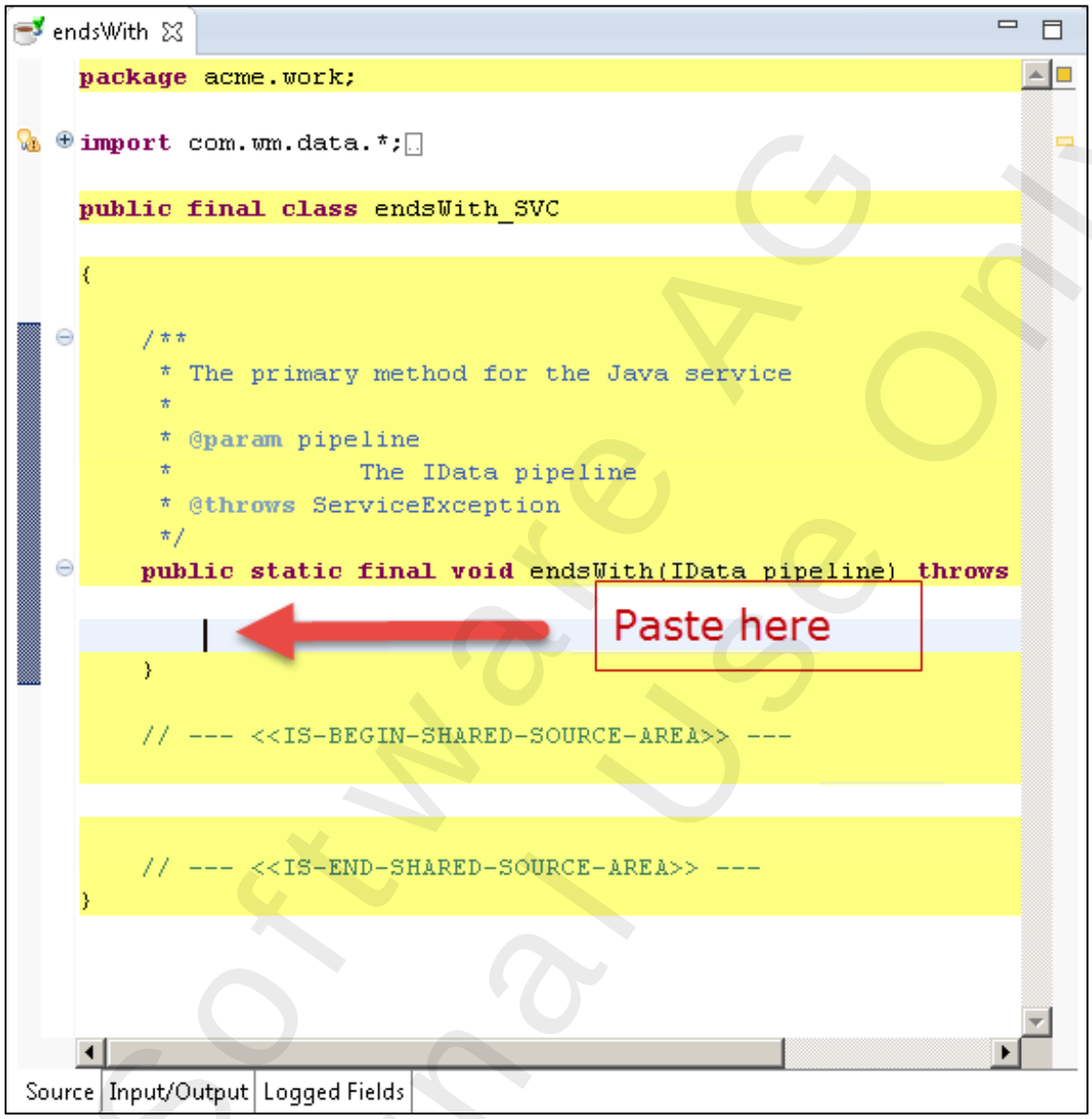
Steps

1. Create a new Java Service **acme.work:endsWith**. First visit the Input/Output tab. Add two String inputs called **string** and **suffix** and one String output called **isSuffix**.



2. **Save** your service.

3. Right click your Java Service in the Package Navigator view and select **Generate Code…** -> **For implementing this service** and click the **Finish** button.
   This places template code into the clipboard.
   Open the Source tab of your Java service and paste the generated code into the empty method body (just after the line “public static final void endsWith…”).



4. Edit it until it looks like the following. Note, the code shown within the boxes needs to be added/changed:

```
*/
public static final void endsWith(IData pipeline) throws ServiceException {

    // pipeline
    IDataCursor pipelineCursor = pipeline.getCursor();
        String  string = IDataUtil.getString( pipelineCursor, "string" );
        String  suffix = IDataUtil.getString( pipelineCursor, "suffix" );
    pipelineCursor.destroy();

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

    // pipeline
    IDataCursor pipelineCursor_1 = pipeline.getCursor();
    IDataUtil.put( pipelineCursor_1, "isSuffix", isSuffix );
    pipelineCursor_1.destroy();

}
```

Note: All Java development features available in Eclipse IDE, like code completion, are available to use.

5. Save and run (**Run As --> Run Service**) your Java Service with the following input values:
   a. Test 1:
      i. string: **wildest**
      ii. suffix: **est**

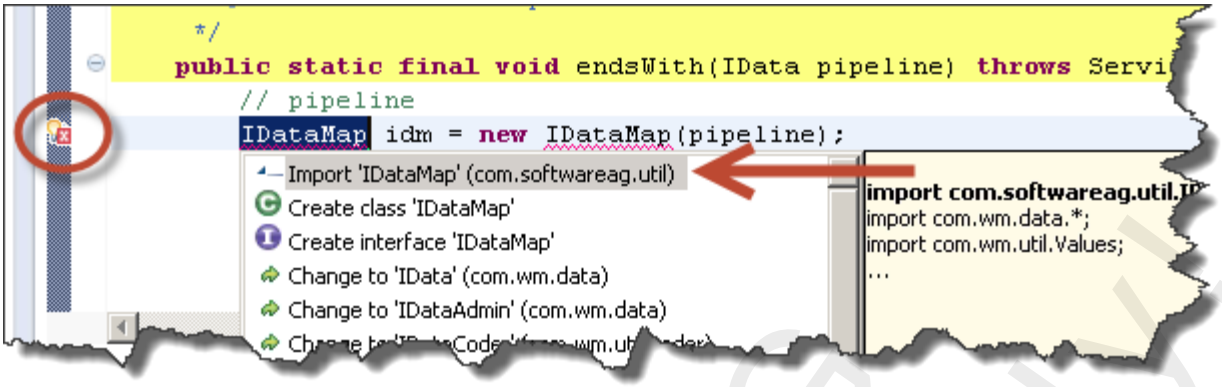
      isSuffix should be **true**
   b. Test 2:
      i. string: **wildest**
      ii. suffix: **ish**

      isSuffix should be **false**

6. Change the code to use the **IDataMap** class. To do this, use the following hints:
   a. As the first line of code for your service (before line "`IDataCursor pipelineCursor = pipeline.getCursor()`") add in the following:

      ```
      IDataMap idm = new IDataMap(pipeline);
      ```

      Note: You WILL get a syntax error!
   b. **Save** your work. In the Compiler Messages popup box, click the **OK** button.
   c. Any line(s) that have a lightbulb icon with a red X  indicate there is a problem. Double click on the icon and you will see a list of suggestions to fix the problem. Select **Import 'IDataMap' (com.softwareag.util)** by double clicking it.



Note: The icon will change to a grey X  . If you click this icon you will see other recommendations. This indicates that we have more modifications to make, so continue on with the next steps.

d. Change the code to initialize the Strings “string” and “suffix” using the “idm” variable:

```
String  string = idm.getAsString("string" );
String  suffix = idm.getAsString("suffix" );
```

e. Change the IDataUtil.put code to use the “idm” variable:

```
idm.put("isSuffix", isSuffix);
```

f. Since the IDataMap takes care of managing the cursors you can remove the following code:

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

pipelineCursor.destroy();

IDataCursor pipelineCursor_1 = pipeline.getCursor();

pipelineCursor_1.destroy();
```

When you are done the code should look like the following:

endsWith

```
package acme.work;

import com.wm.data.*;
import com.wm.util.Values;
import com.wm.app.b2b.server.Service;
import com.wm.app.b2b.server.ServiceException;
import com.softwareag.util.IDataMap;

public final class endsWith_SVC

{

    /**
     * The primary method for the Java service
     *
     * @param pipeline
     *            The IData pipeline
     * @throws ServiceException
     */
    public static final void endsWith(IData pipeline) throws ServiceException {
        // pipeline
        IDataMap idm = new IDataMap(pipeline);
        String  string = idm.getAsString( "string" );
        String  suffix = idm.getAsString( "suffix" );

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

        // pipeline
        idm.put( "isSuffix", isSuffix );

    }
```

g. **Save** and test your modified Java Service.

Check Your Understanding

1. In step 3, why was it important to set the Inputs and Outputs of the service before you generated the code?
2. Is the service thread safe? What would you have to do if not?
3. In step 6c, what was the purpose of “import com.softwareag.util.IDataMap”?

EXERCISE 11:
MONITORING SERVICES

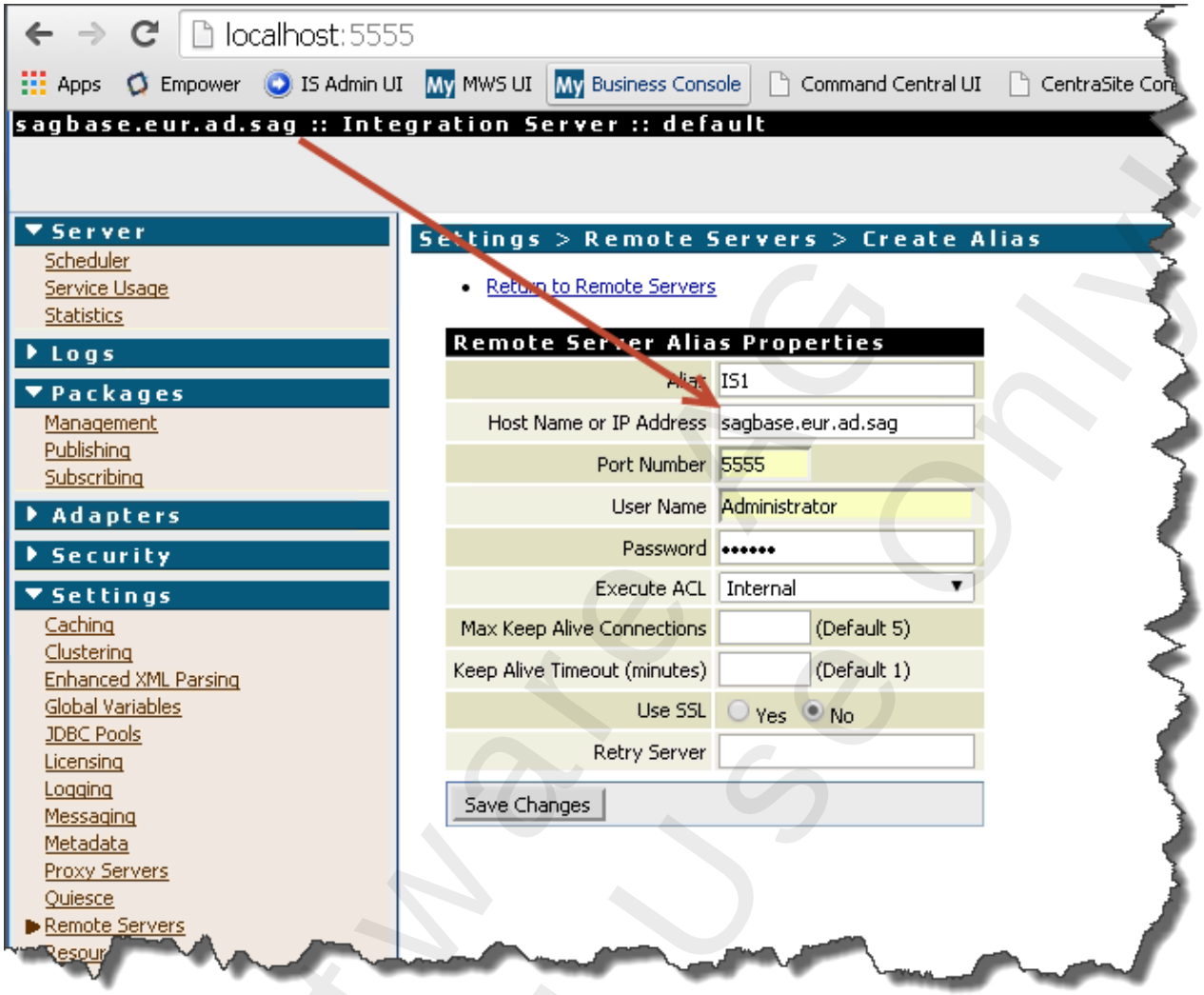
Overview

In this exercise, you will use My webMethods to track the execution of services and to perform service resubmission.

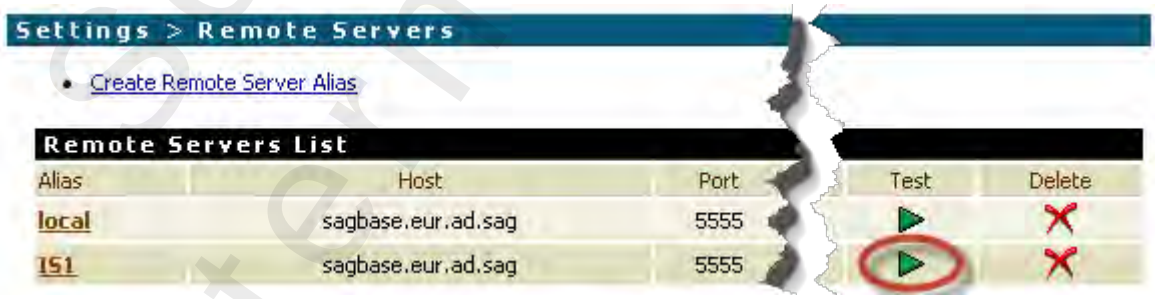
Steps

1. Open the IS Administrator UI (*http://localhost:5555*) and log in as **Administrator** using a password of **manage**. In the **Settings** menu select **Remote Servers** and then copy the Host name from the upper left corner of your browser. Click the **Create Remote Server Alias** link and enter the following:
   - Alias: **IS1**
   - Host Name or IP Address: (Paste the value you copied)
   - Port Number: **5555**
   - User Name: **Administrator**
   - Password: **manage**

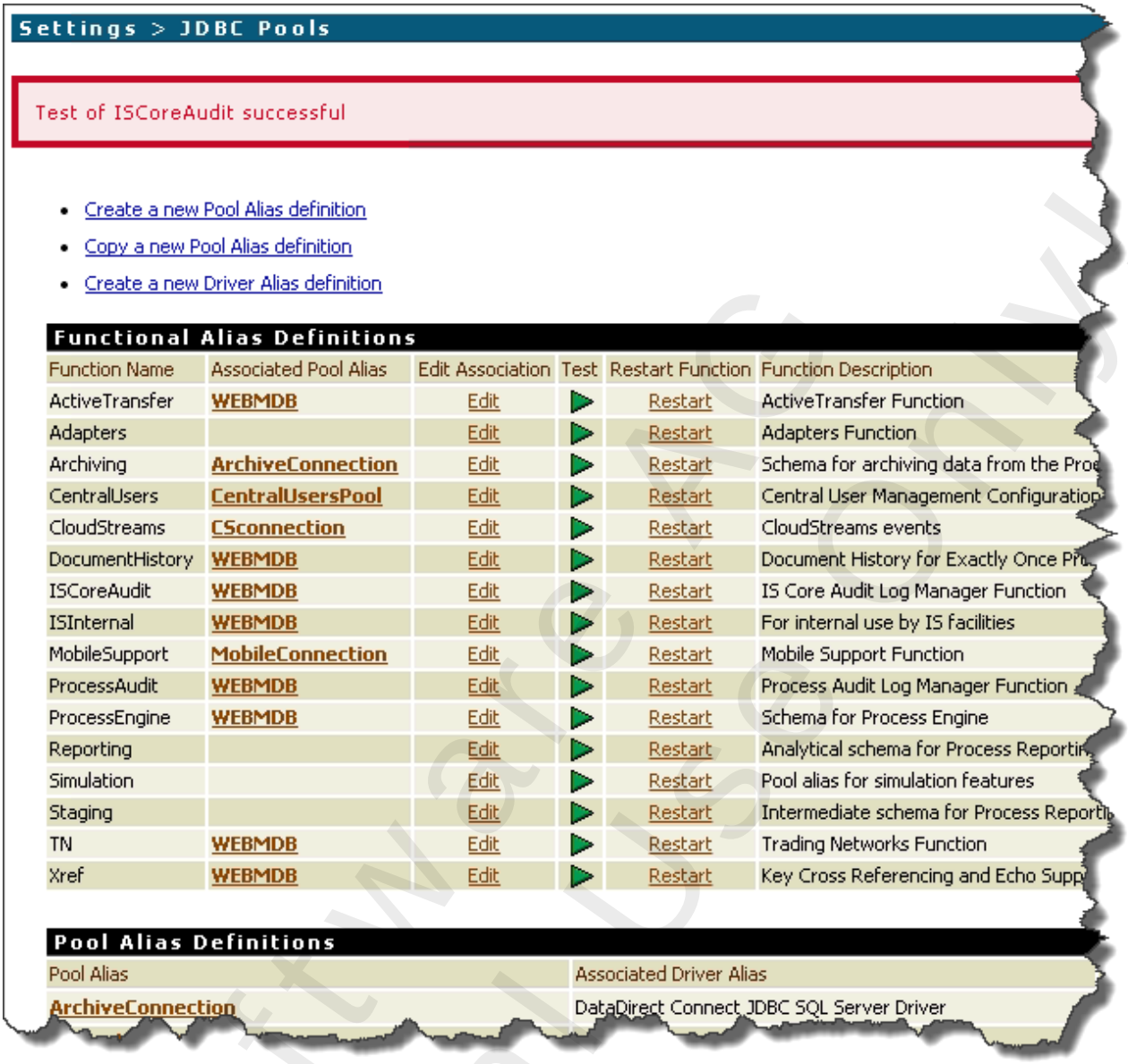
Click the **Save Changes** button to save your work.



Test the **IS1** alias by clicking on its Test icon (▶).



2. Go to the **Settings --> JDBC Pools** entry area in the IS Administrator UI and confirm that a **JDBC Pool Alias** named **WEBMDB** 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.



Settings > JDBC Pools

Test of ISCoreAudit successful

- Create a new Pool Alias definition
- Copy a new Pool Alias definition
- Create a new Driver Alias definition

Functional Alias Definitions

Function Name	Associated Pool Alias	Edit Association	Test	Restart Function	Function Description
ActiveTransfer	WEBMDB	Edit	▶	Restart	ActiveTransfer Function
Adapters		Edit	▶	Restart	Adapters Function
Archiving	ArchiveConnection	Edit	▶	Restart	Schema for archiving data from the Pro
CentralUsers	CentralUsersPool	Edit	▶	Restart	Central User Management Configuratio
CloudStreams	CSconnection	Edit	▶	Restart	CloudStreams events
DocumentHistory	WEBMDB	Edit	▶	Restart	Document History for Exactly Once Pr
ISCoreAudit	WEBMDB	Edit	▶	Restart	IS Core Audit Log Manager Function
ISInternal	WEBMDB	Edit	▶	Restart	For internal use by IS facilities
MobileSupport	MobileConnection	Edit	▶	Restart	Mobile Support Function
ProcessAudit	WEBMDB	Edit	▶	Restart	Process Audit Log Manager Function
ProcessEngine	WEBMDB	Edit	▶	Restart	Schema for Process Engine
Reporting		Edit	▶	Restart	Analytical schema for Process Reportin
Simulation		Edit	▶	Restart	Pool alias for simulation features
Staging		Edit	▶	Restart	Intermediate schema for Process Reporti
TN	WEBMDB	Edit	▶	Restart	Trading Networks Function
Xref	WEBMDB	Edit	▶	Restart	Key Cross Referencing and Echo Supp

Pool Alias Definitions

Pool Alias	Associated Driver Alias
ArchiveConnection	DataDirect Connect JDBC SQL Server Driver

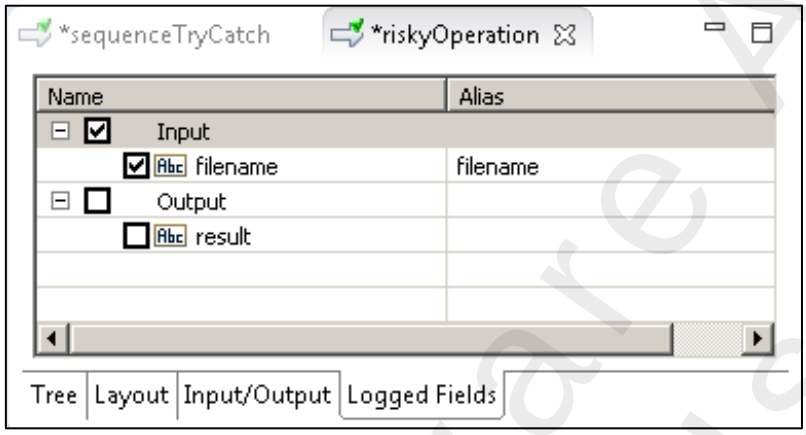
3. In Designer, open the service **acme.work:sequenceTryCatch**. If necessary, lock the service for edit. In its Properties view, enable Audit Logging. Set the **Audit** properties as follows:
   a. Enable auditing: **Always**
   b. Log on: **Error and success**
   c. Include pipeline: **Always**

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

4. Open the service **acme.work:riskyOperation** . If necessary, lock the service for edit. Enable auditing for the service, but make sure that **Include Pipeline** is set to **Never**. Your settings should look like the following:

Audit	
Enable auditing	Always
Log on	Error, success, and start
Include pipeline	Never

Click on the **Logged Fields** tab on the bottom of the services Editor panel.
Check the **filename** field:



5. Save the **sequenceTryCatch** and the **riskyOperation** services.
6. Run IS service **sequenceTryCatch** two times with the following values:
   a. Run 1, filename: **C:\SoftwareAG\IntegrationServer\Apache_License.txt**
   b. Run 2, filename: **C:\SoftwareAG\IntegrationServer\Kiowa_License.txt**

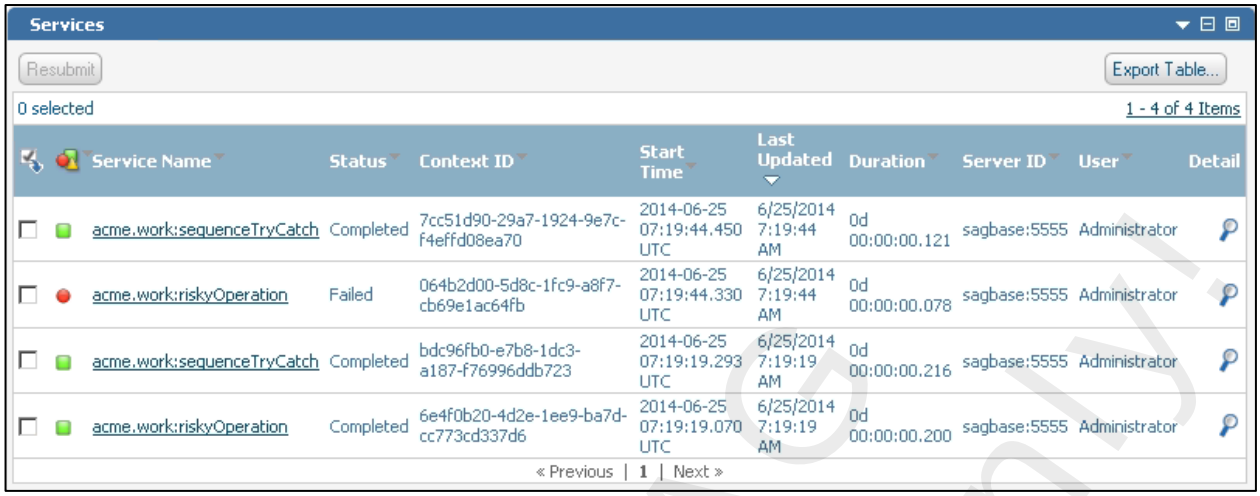
7. In a new browser tab, login to My webMethods (http://localhost:8585 or use the appropriate bookmark) and authenticate as **Administrator | manage**.

   Note: on a freshly started MWS, the response times can be a bit longer at first because MWS has to load Java classes when they are referenced for the first time. Be patient. Rapidly/randomly clicking on links and buttons will not speed things up. Keep in mind MWS is on a shared training VM with many other products. In your real webMethods environments the servers should have a lot more capacity and/or be installed on separate servers.

   Perform the following steps:

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

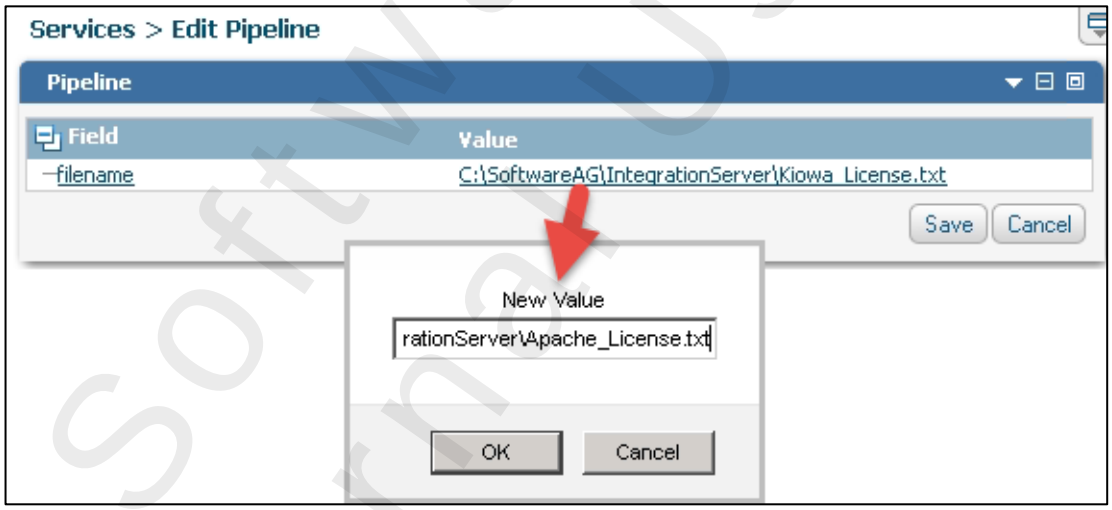
   b. On the Services page, find the two **acme.work:sequenceTryCatch** and the two **acme.work:riskyOperation** services in the **Services** list.
   Note: In case of an empty services list, click the Search button at the top of the page.



Note that, when reading the table from bottom to top, you first see the invocation of **riskyOperation** preceded by the matching call to **sequenceTryCatch**. You can also see this when matching the Context ID's, the Start Time and the Last Updated times.

c. For the top most **sequenceTryCatch** service click on the **Details** button ( ). The statistics about this individual service execution will be displayed.

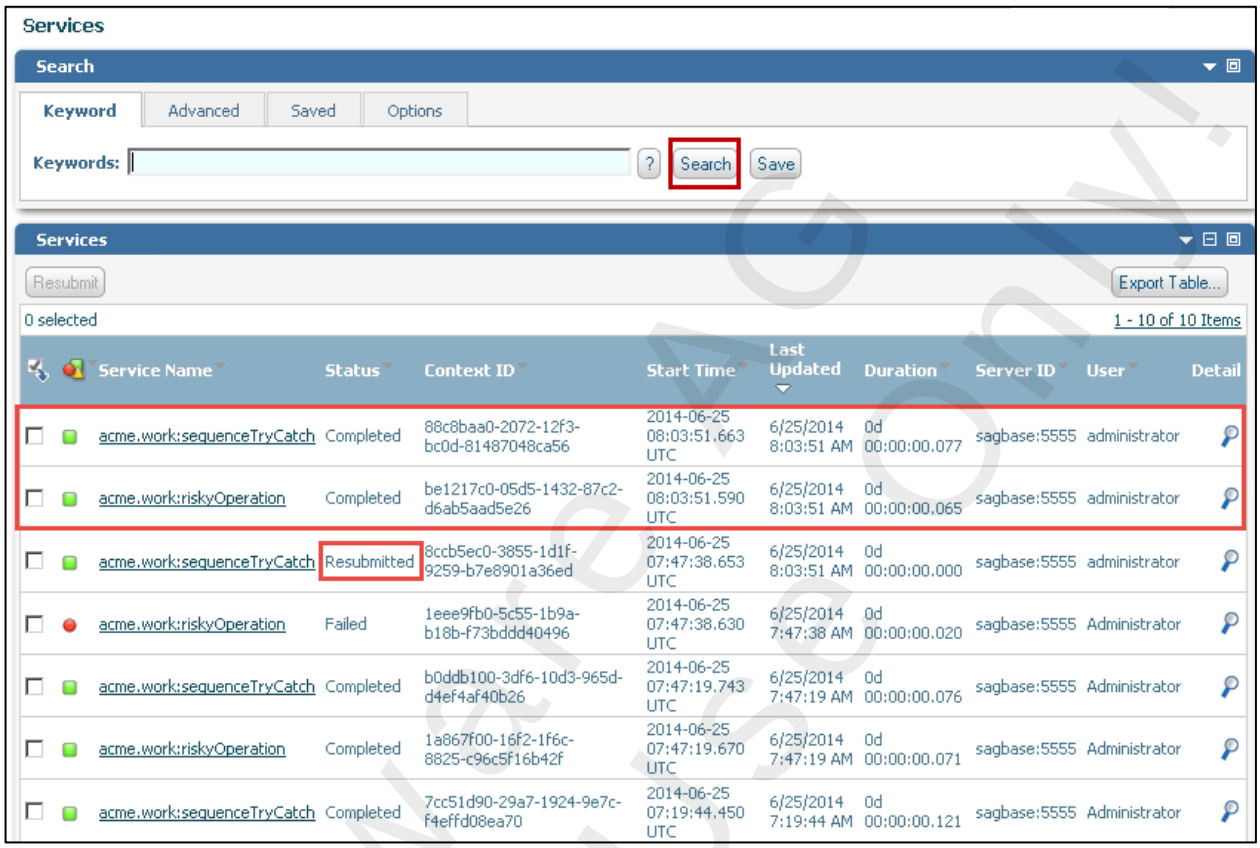
d. Use the **Edit Pipeline** button on this invocation of **sequenceTryCatch**, where the embedded risky operation failed. Change the value of field **filename** to point to the existing file (**C:\SoftwareAG\IntegrationServer\Apache_License.txt**).



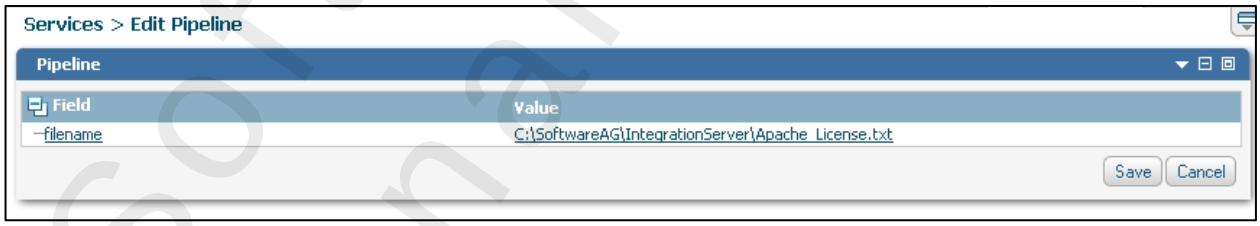
Click the **OK** and **Save** buttons.

e. Finally click on the **Resubmit** button and you will see a resubmission message like this one:

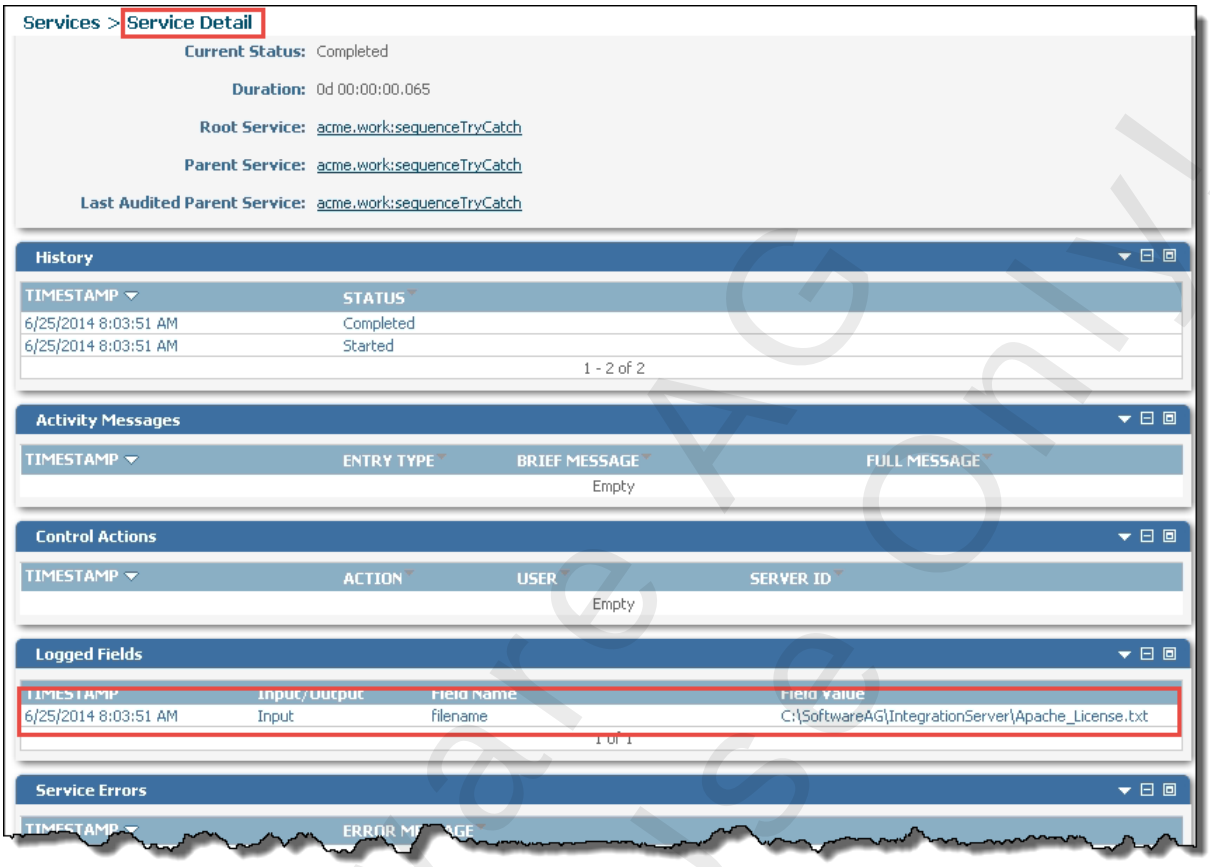
f. Select the **Cancel** button to go back to the Services page. Select the **Search** button to see that there is a new **Completed** entry in the **Services** list. The older **Completed** entry has its state changed to **Resubmitted**.



g. Finally, verify that the changed pipeline data is also shown in the Edit Pipeline view of the top most invocation of **sequenceTryCatch**.



h. From the Services list, choose the topmost **riskyOperation** entries, open its details and confirm that the logged field of **filename** is present.



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?

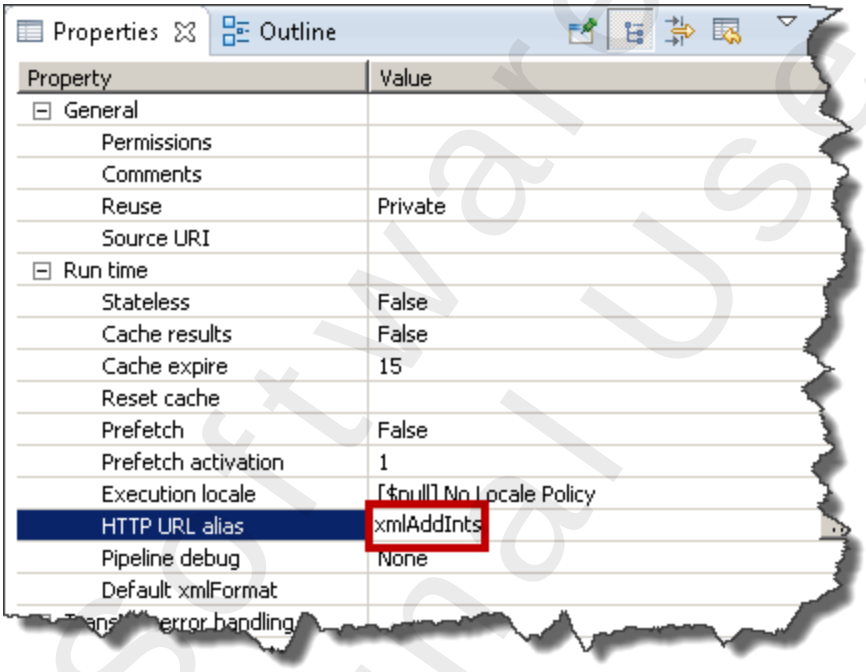
EXERCISE 12:
INVOKING SERVICES AND XML PROCESSING

Overview

In this exercise, you will invoke a service on the Integration Server via the following: HTTP, Designer Run Configuration, SMTP, FTP, and a Java client.
Input will be provided as text strings and XML data.

Steps

1. Invoke a service using HTTP:
   a. To invoke a service using HTTP, open Designer and in the **AcmeSupport** package, edit the **acmeSupport.xml:xmlAddInts** service. Make sure it is locked for editing.
   In its Properties view, set its **HTTP URL alias** property to **xmlAddInts.**



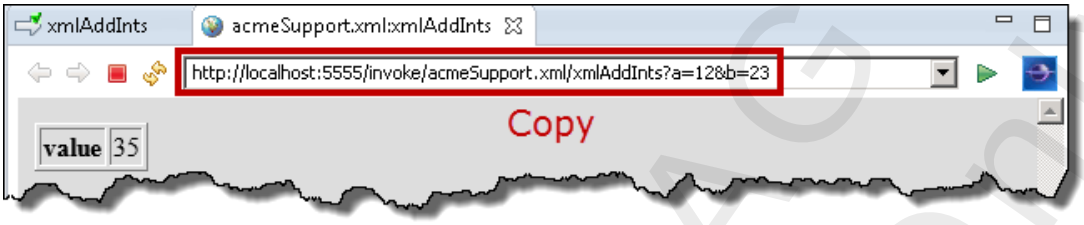
   Note: the HTTP URL alias can be any name. It does not have to match the service name.

   b. **Save** the service.

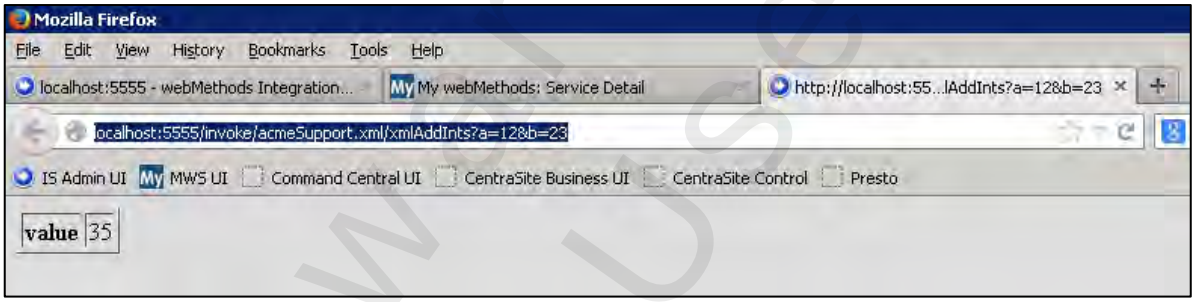
c. Right click the **acmeSupport.xml:xmlAddInts** service and choose **Run As --> Run in Browser.** Click the **OK** button in the popup window labeled **Tip.** Enter the following values:

- a: **12**
- b: **23**

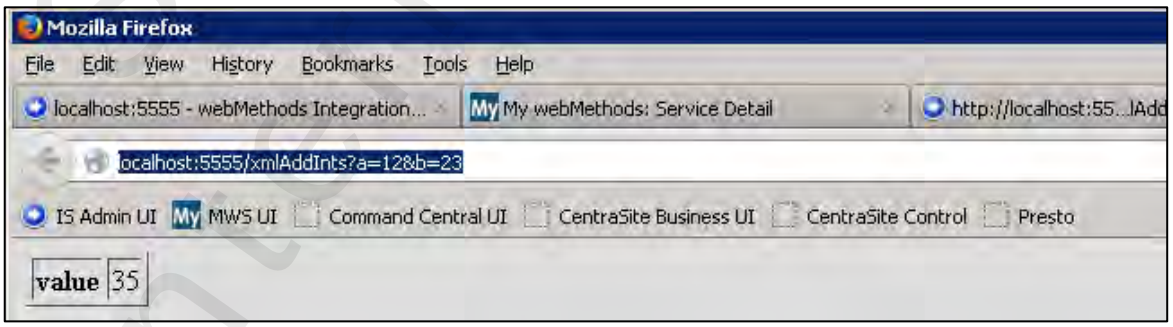
Click the **OK** button. If needed, enter credentials of **Administrator**/**manage.**
In the browser view that appears, **copy** the URL.



d. In a new browser tab. Paste the URL
http://localhost:5555/invoke/acmeSupport.xml/xmlAddInts?a=12&b=23 and press the <Enter> key. If asked for authentication, provide **Administrator** | **manage**.



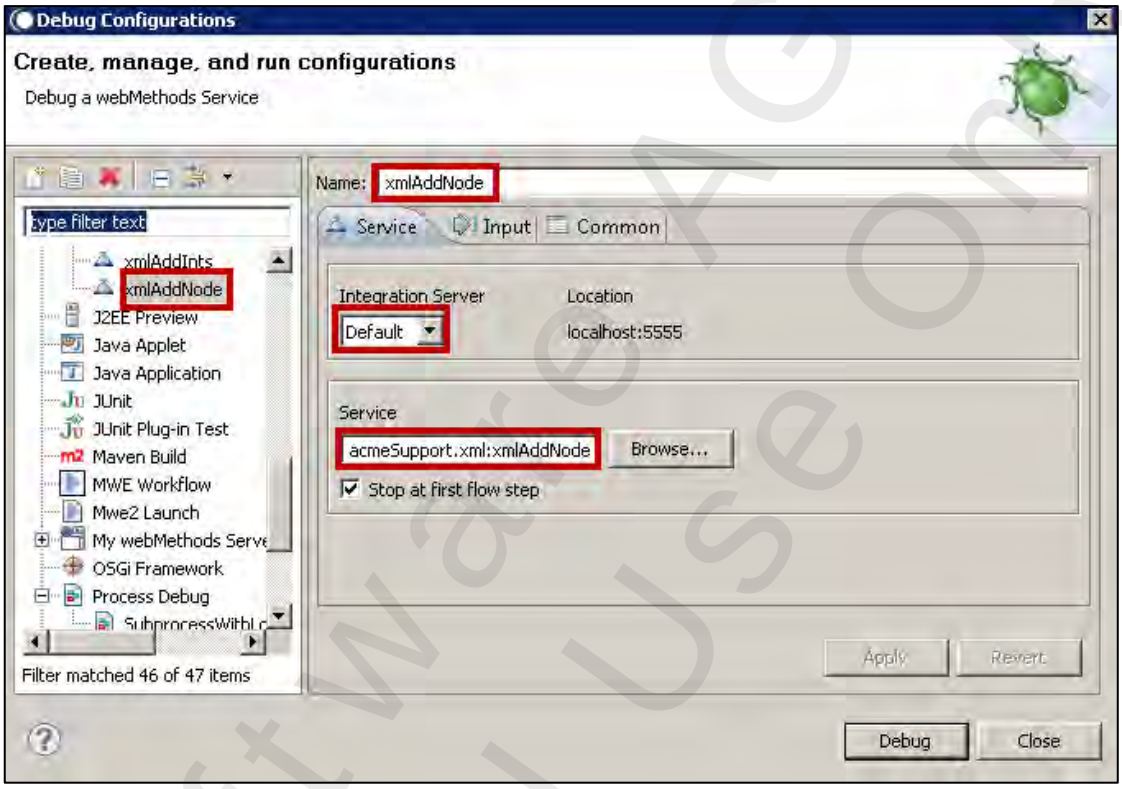
e. In another browser tab, enter the alternative URL
http://localhost:5555/xmlAddInts?a=12&b=23
Notice that the service was invoked by a shorter URL.
The HTTP URL alias is valuable if your service is nested within multiple folders in the namespace.



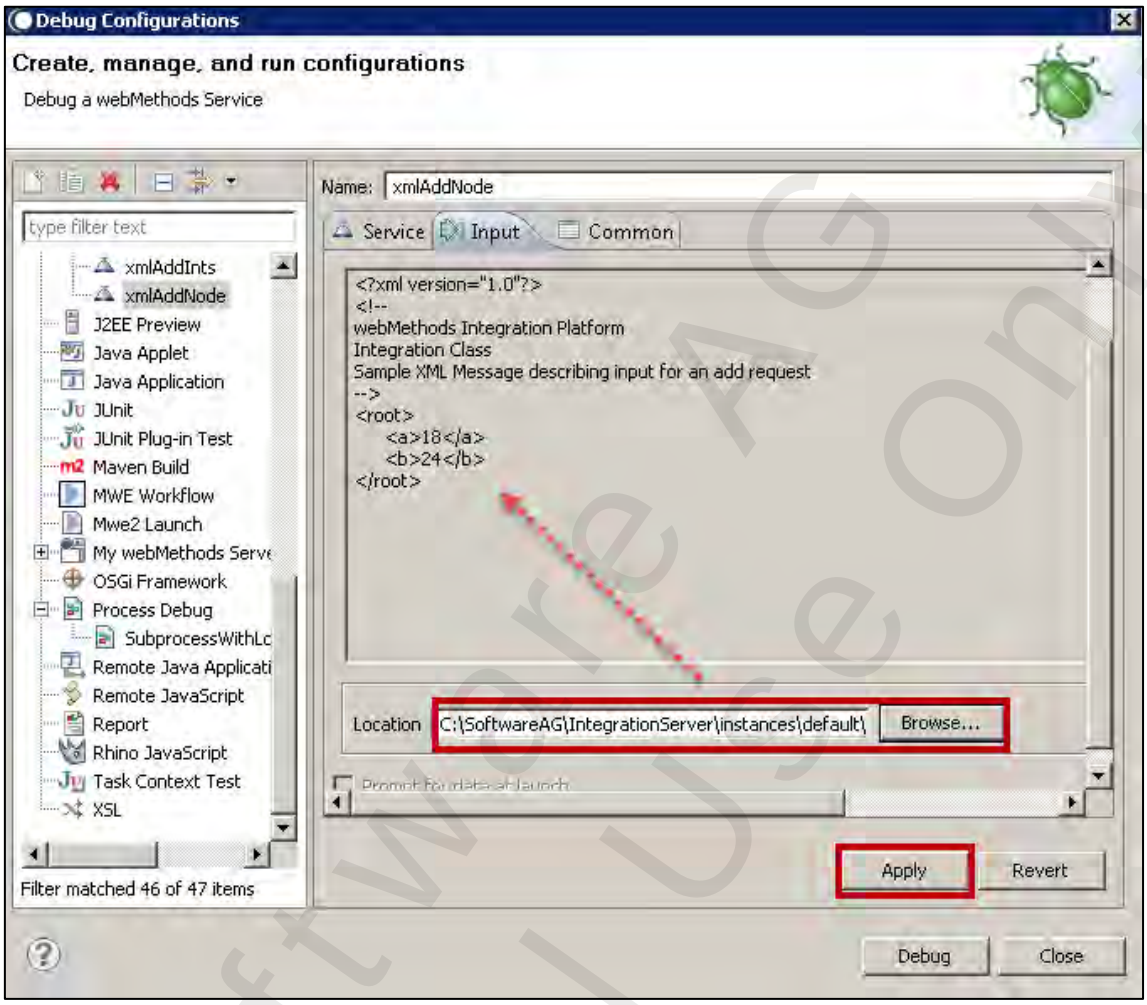
2. Invoke a service with XML input:

   a. Attempt to run the **AcmeSupport** package's **acmeSupport.xml:xmlAddNode** service in Designer.
   Notice that you cannot provide input for node since this is an object type.

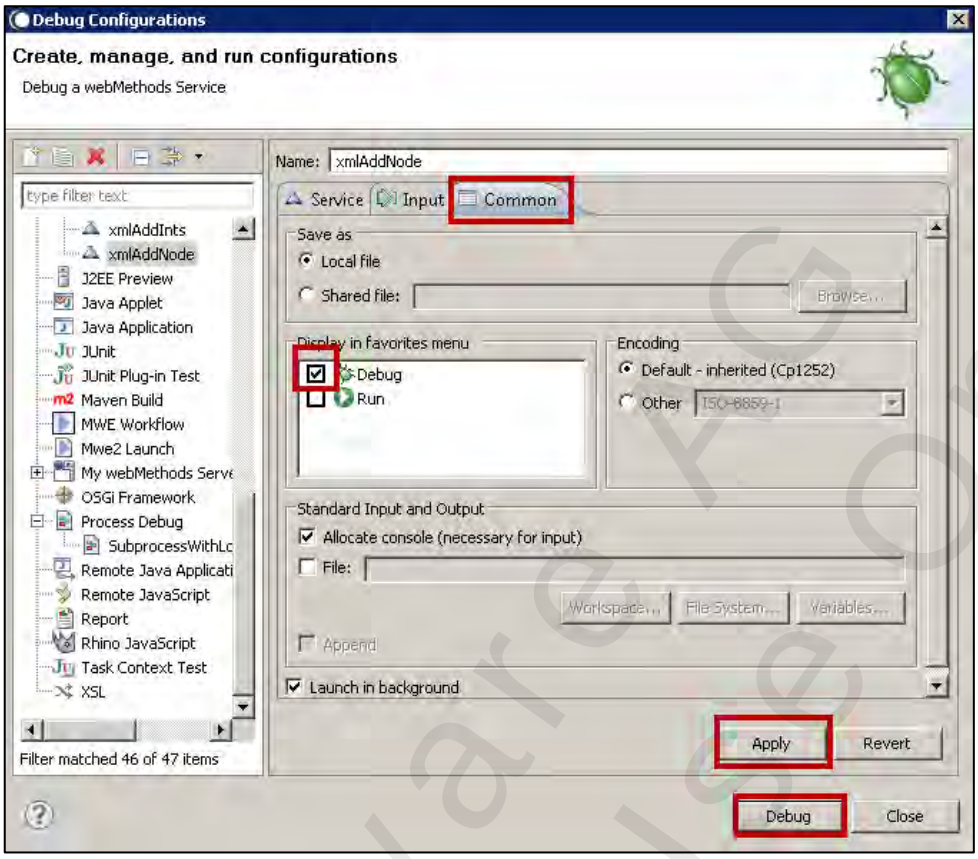
   b. Right-click the **acmeSupport.xml:xmlAddNode** service and select **Debug As ➡ Debug Configurations…** . In the upcoming dialog, note that the **Name, Integration Server,** and **Service** parameters are filled in with the values of the last service you ran:



c. Choose the **Input** tab and select the **Use XML** radio button. Click **Browse,** navigate to the folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub**, select the XML File **addInput.xml,** and click **Open.** Then click the **Apply** button.

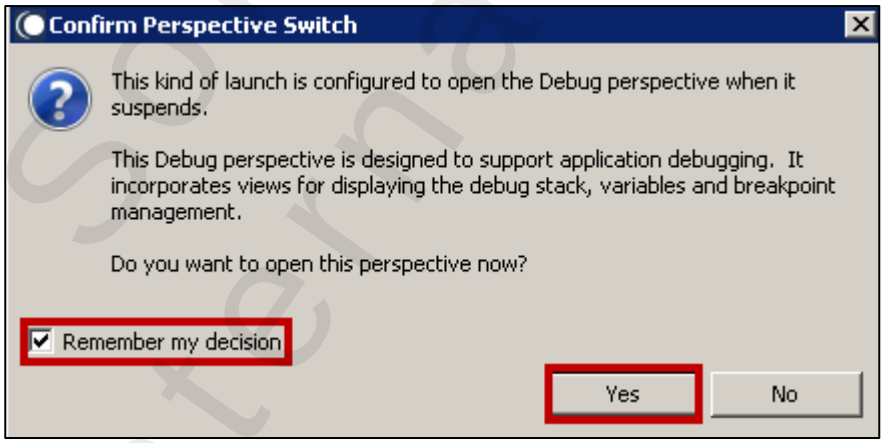


d. Now select the **Common** tab and check the checkbox **Debug** in the **Display in favorites menu** panel. Click **Apply** again and then click on **Debug**.



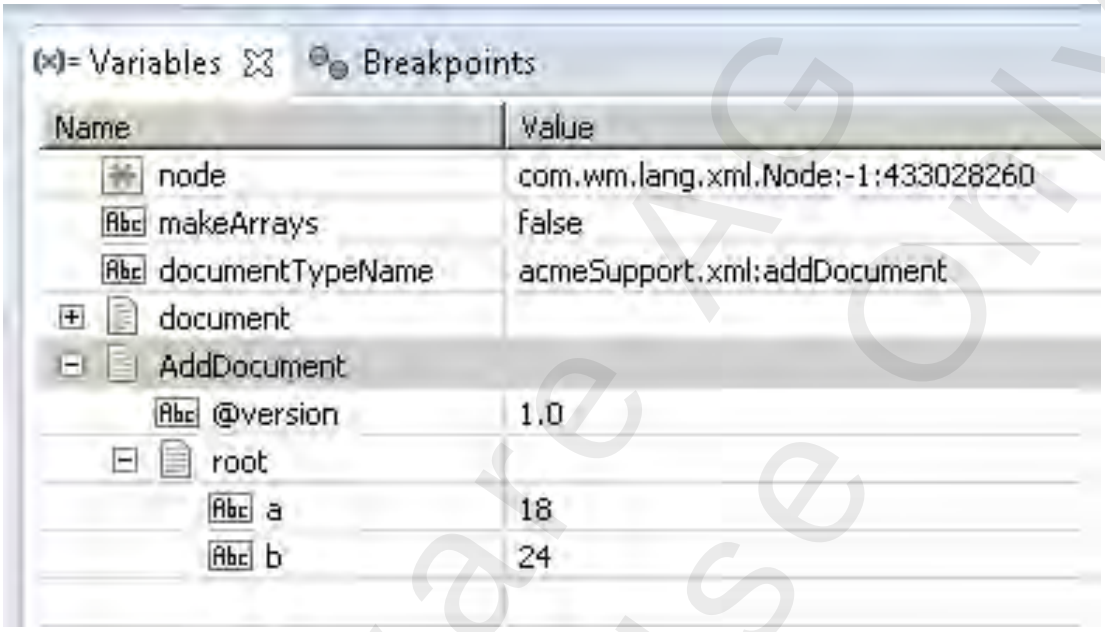
e. In the **Confirm Perspective Switch** popup box:

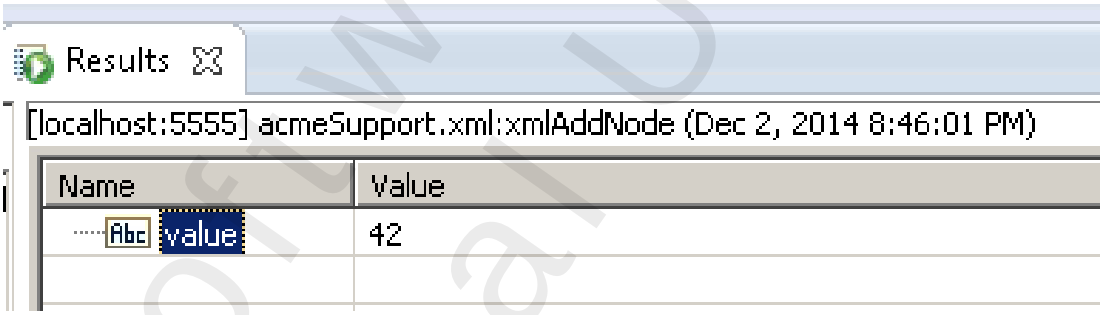
i. Select the **Remember my decision** checkbox
ii. Clicking **Yes** button to change to the Debug perspective.



f. Now the Debugger comes up and you are debugging the IS Service with a single input variable **node** of type Object already in the pipeline.

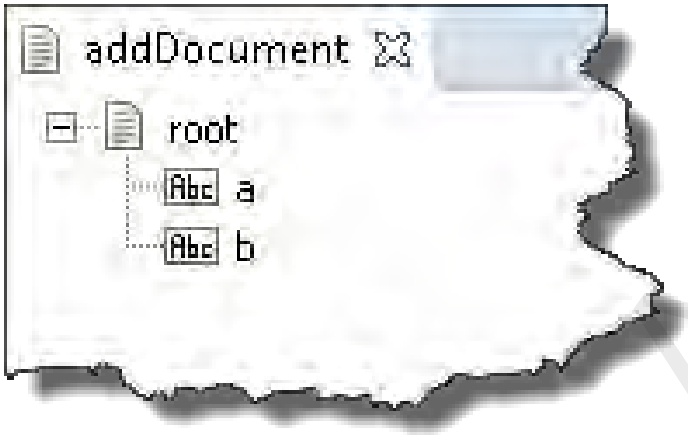
Use the **Step Over** icon () to see how the **node** object gets converted to the **AddDocument** document (by service pub.xml:xmlNodeToDocument), and then the **value** gets calculated (by service pub.math:addInts).





3. Invoke a service using SMTP (mail):

   a. In Designer, switch back to the Service Development perspective. From the Package Navigator view open the document **acmeSupport.xml:addDocument**. You should find that it contains a simple root node containing two variables called **a** and **b**.

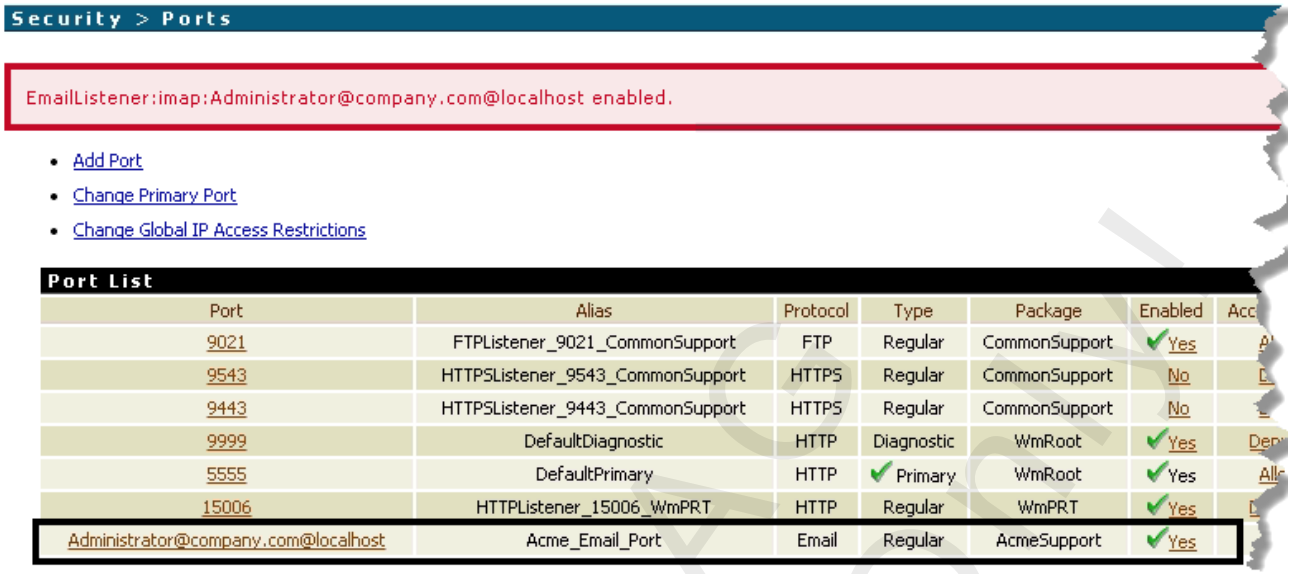


   b. Now open the **xmlAddNode** service in the same folder. Review the code so that you understand what it is doing.
   Also have a look at the content of the file **addInput.xml** as contained in the folder **C:\SoftwareAG\IntegrationServer\ instances\default \packages\AcmeSupport\pub**.

   c. Launch the Windows **Services** control panel. Start the **hMailServer** service to start the email server.

   d. Enable the Email port in Integration Server, which is turned off by default.

   To do so, open the IS Administration UI and go to the **Security ➡ Ports** menu.
   Enable the Email port by clicking on the word **No** in the **Enabled** column. You should see a success message at the top of the screen:



e. In the Designer, copy the **acmeSupport.xml:xmlAddNode** service to the clipboard.

f. From the main **Start** menu, choose **Start --> Mozilla Thunderbird**.

Click on the **Customer@company.com** account on the left side.

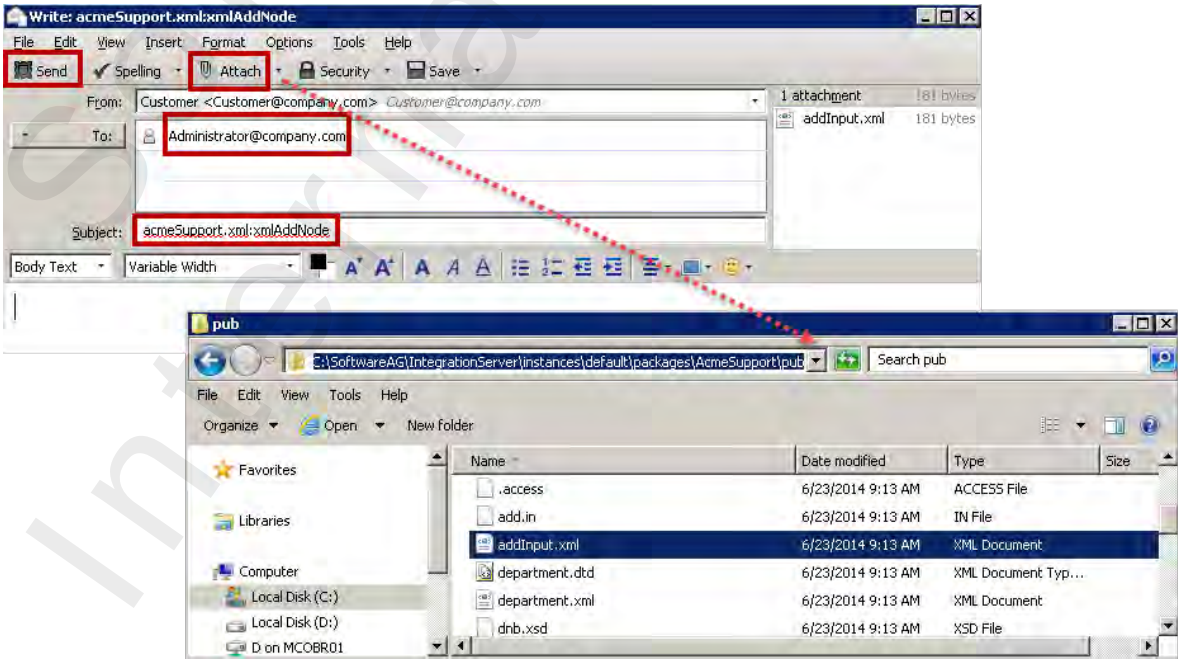
Write a new mail message from Customer@company.com specifying the following:

i. To: **Administrator@company.com**

ii. Subject: **acmeSupport.xml:xmlAddNode** (Use <Cntl -V> to paste the value)

iii. Attach the file **addInput.xml** from **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub**

iv. Once you completed your mail, click the **Send** button. If you are asked to enter a password, enter: manage

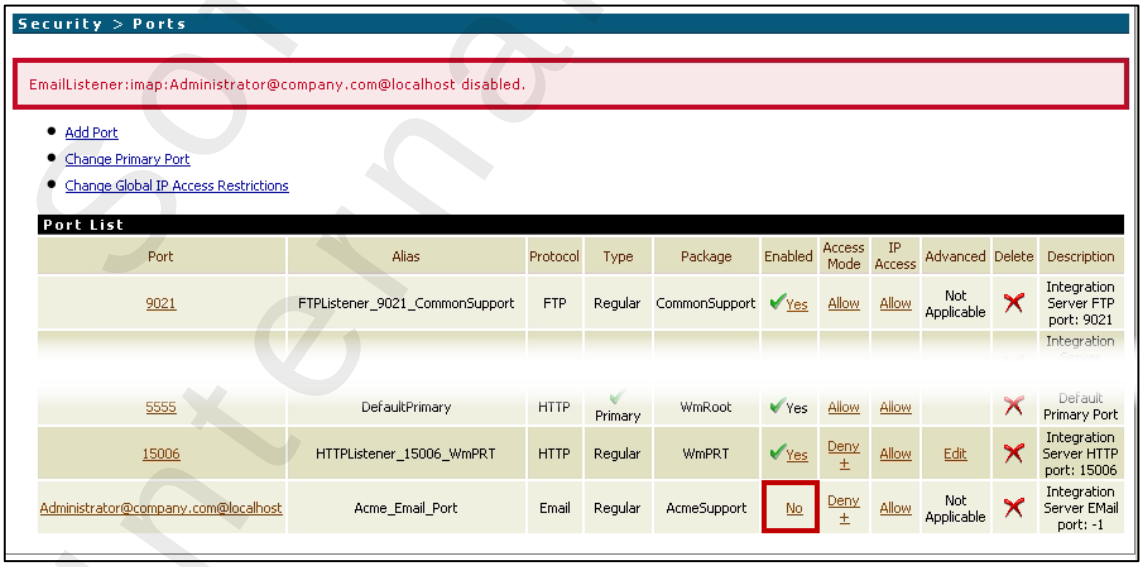


Note: The mail service and Thunderbird on your virtual machine are set up to handle all mail locally. There is no connectivity to any outside mail system. Thunderbird and the hMail Server are set up to serve the company.com domain. Please do not change any of the configuration settings unless otherwise noted.

g. After you sent your mail, wait about 15 Seconds, then press the **Get Mail** (

) button in Thunderbird and you will receive a reply from Integration Server with the result of the message processing.

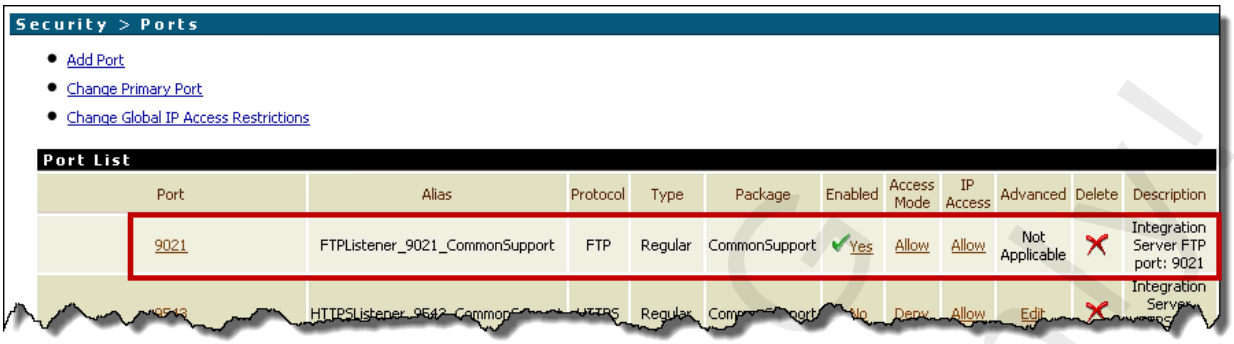
h. When you are done with this part of the exercise, you should disable the Email port in Integration Server, because constant polling is a waste of resources.



4. Invoke a service using FTP:

a. Before using ftp, make sure there is an enabled FTP port in Integration Server available. Open the IS Administrator UI and go into the **Security --> Ports** submenu.

Make sure an FTP port exists for port 9021 and that it has an **Access Mode** setting that allows execution by default:



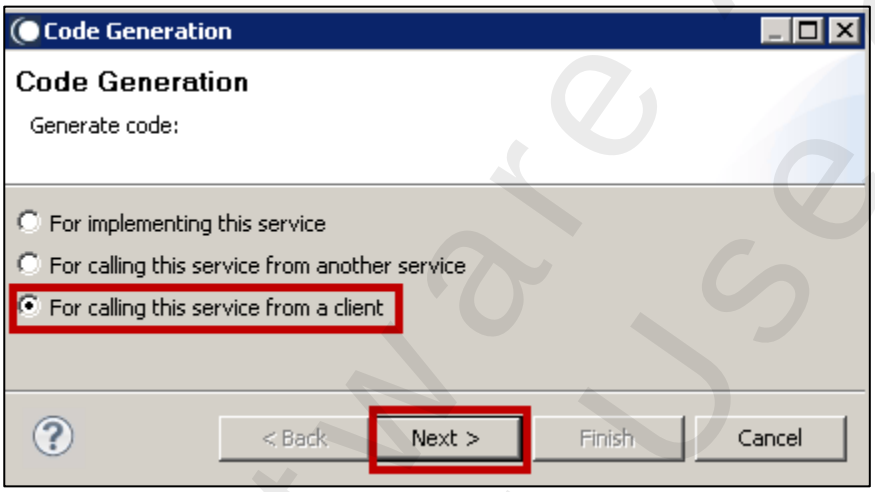
b. Now open a windows command prompt window (**Start --> Run... --> cmd**), and execute the commands as shown below **in bold font**.
Make sure you understand what each command is doing before typing it in.

```
C:\Users\Administrator>cd /d
C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub
C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub>dir addInput.xml
 Volume in drive C has no label.
 Volume Serial Number is D0D6-FA14
 Directory of C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub
06/23/2014  02:56 PM               181 addInput.xml
               1 File(s)            181 bytes
               0 Dir(s)  12,190,478,336 bytes free
C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub>ftp
ftp> open localhost 9021
Connected to DAEAP83210.eur.ad.sag.
220 sagbase:9021 FTP server (webMethods Integration Server version 9.7.x.x) ready.
User (sagbase:(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 xmlAddNode
250 CWD command successful.
ftp> send addInput.xml
200 PORT command successful.
150 ASCII mode  data connection for addInput.xml (127.0.0.1,52951).
226 ASCII transfer complete.
ftp: 181 bytes sent in 0.06Seconds 3.23Kbytes/sec.
ftp> dir
200 PORT command successful.
150 ASCII mode  data connection for /bin/ls (127.0.0.1,52975).
total 1
dr-xr-xr-x  3 root      root                      1 Jul 22 16:41 .
dr-xr-xr-x  3 root      root                      1 Jul 22 16:41 ..
-r--r--r--  1 tx        tx                      106 Jul 22 16:41 addInput.xml.out
226 ASCII transfer complete.
ftp: 232 bytes received in 0.00Seconds 232000.00Kbytes/sec.
ftp> get addInput.xml.out -
```

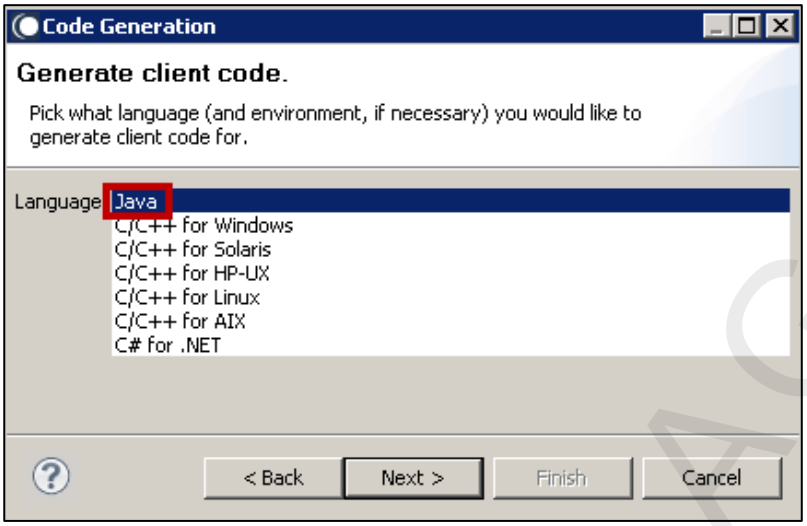
```
200 PORT command successful.
150 ASCII mode  data connection for addInput.xml.out (127.0.0.1,53000) (106
bytes).
<?xml version="1.0" encoding="UTF-8"?>

<Values version="2.0">
  <value name="value">42</value>
</Values>
226 ASCII transfer complete.
ftp: 106 bytes received in 0.00Seconds 106000.00Kbytes/sec.
ftp> quit
221 Goodbye.
```

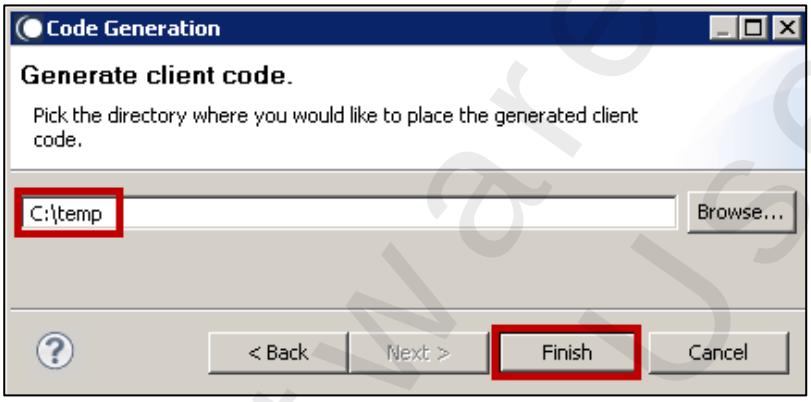
5. Invoke a service using Java:
   a. In the Designer, select the **acmeSupport.xml:xmlAddInts** service.
   b. Right click the service and choose the **Generate Code…** entry. In the dialog box that opens select **For calling this service from a client**. Click **Next**.



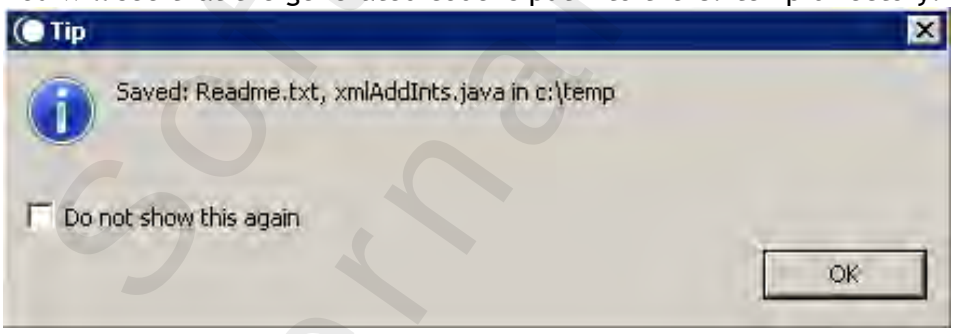
c. On the next panel choose **Java** as language and click **Next**.



d. On the last panel type **C:\temp** as directory for code generation. Click the **Finish** button.



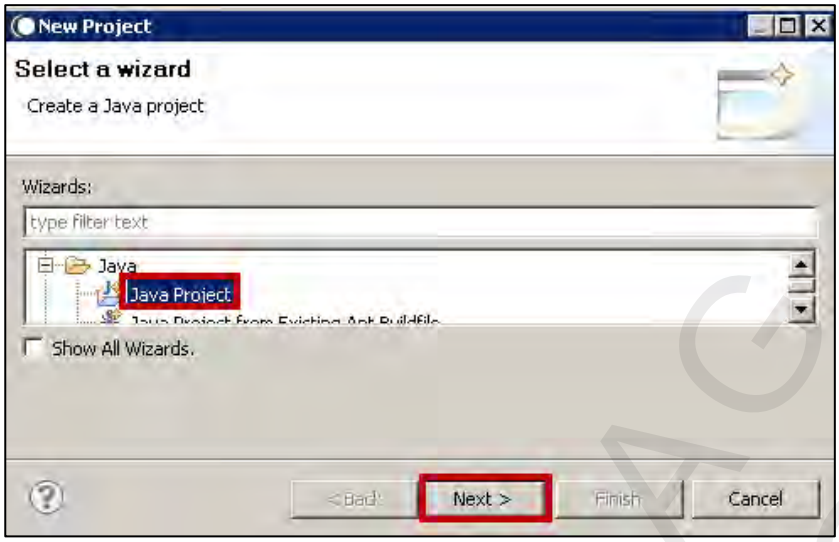
You will see that the generated code is put into the C:\temp directory.



e. Now create a Java project, by doing the following:

i. From the Main menu, choose **File --> New --> Project… .**

ii. In the **New Project** popup window, choose **Java** --> **Java Project** and click **Next.**

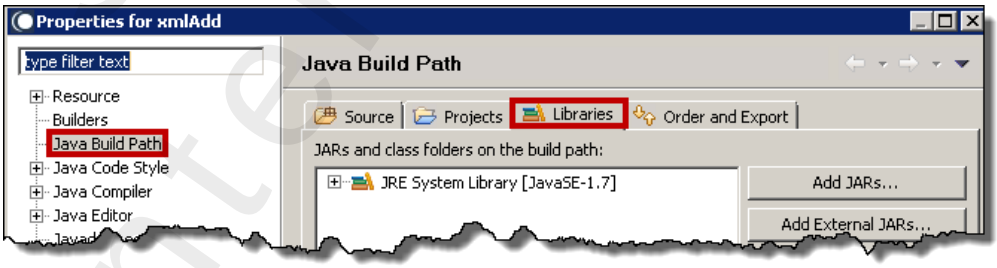


iii. In the **New Java Project** Popup window, for the **Project name** enter **xmlAdd.**

Do not change any of the defaults and click the **Finish** button. Click Yes when asked if you want to switch to the Java perspective.

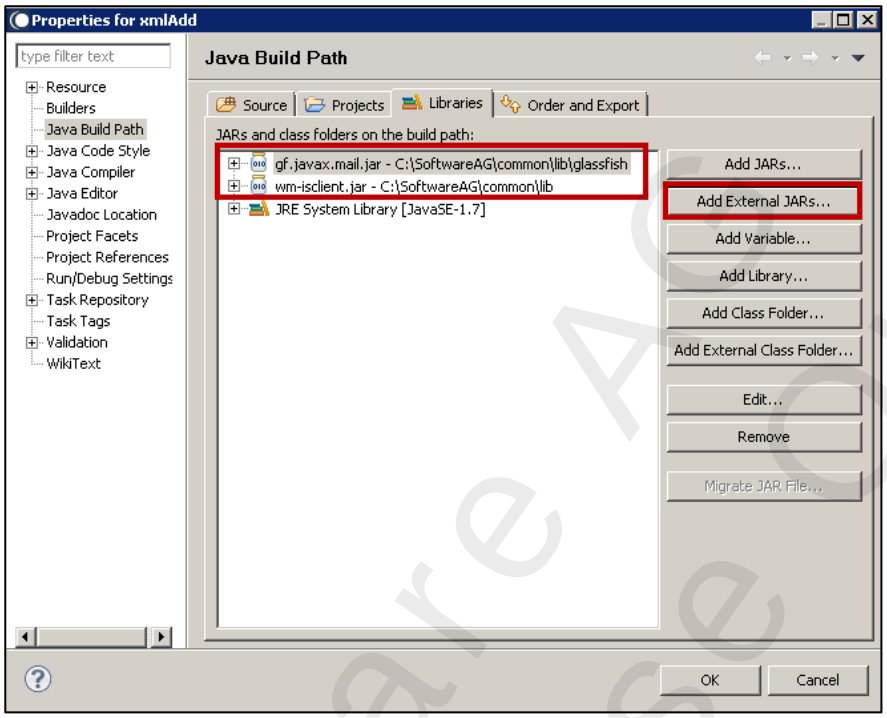
f. This Project requires two additional external Jar files. To add them:

i. While in the Java perspective, right-click on the project node (xmlAdd) and select the **Build Path --> Configure Build Path… .**
ii. In the **Properties for xmlAdd** popup box, select the **Libraries** tab.



iii. Choose the **Add External JARs…** button and add (in two steps) the libraries:

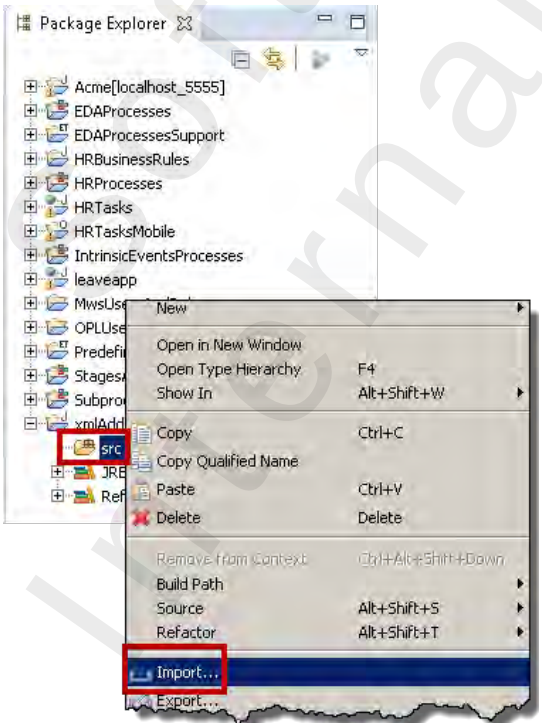
- **C:\SoftwareAG\common\lib\wm-isclient.jar**
- **C:\SoftwareAG\common\lib\glassfish\gf.javax.mail.jar**



Close the dialog by clicking the **OK** button.

g. Now import the Java source generated in the first step by doing the following:

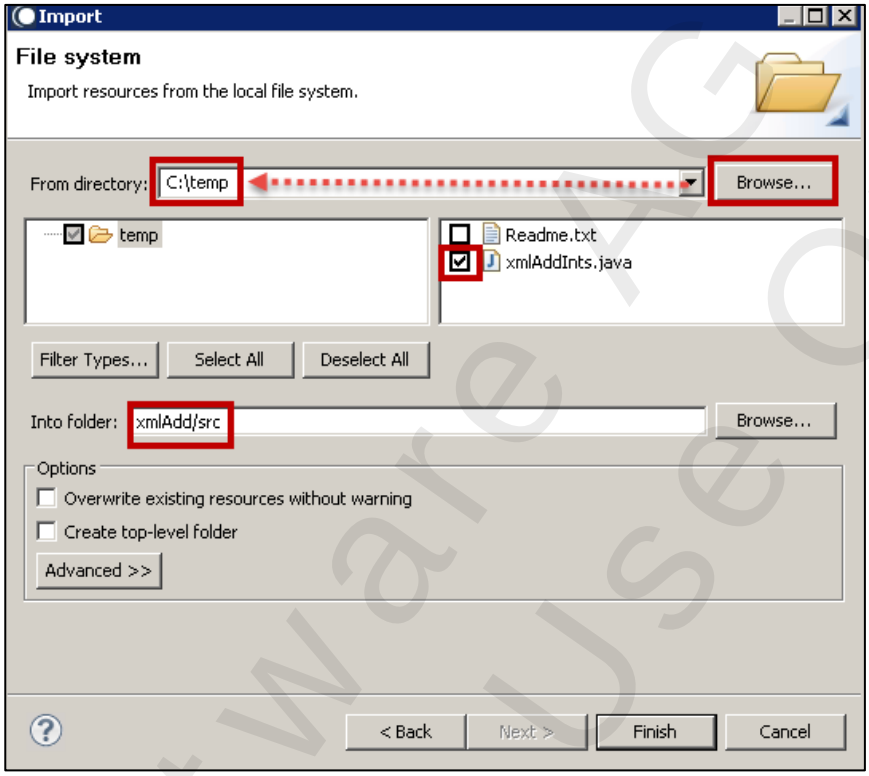
i. Right-click the **xmlAdd\src** node and select the **Import** option from the menu.



ii. Choose **General ➡ File System** and click **Next**.

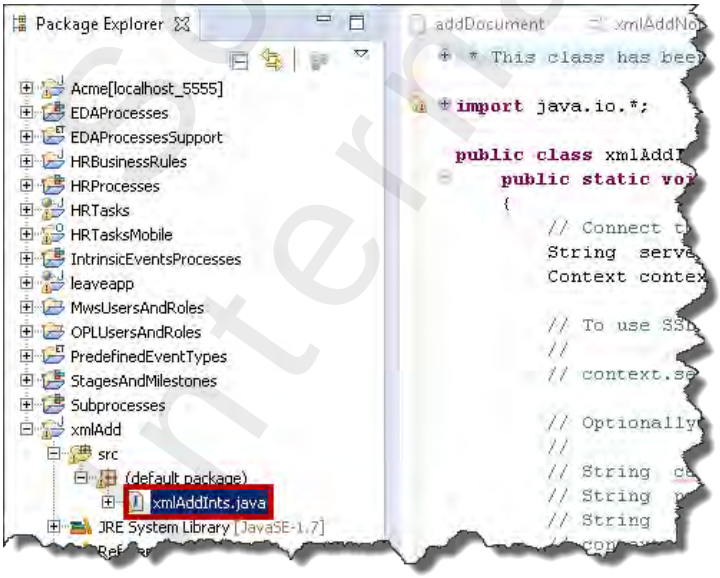
iii. In the upcoming window, select / confirm the following:

- From directory: **C:\temp** (use **Browse** button)
- Select the **xmlAddInts.java** checkbox
- Into folder: **xmlAdd\src** (should already be set to this value)



Click the **Finish** button.

h. Double-click **xmlAdd\src\(default package)\xmlAddInts.java** to open it in the editor.



Change the lines:

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

to the provide the correct credentials:

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

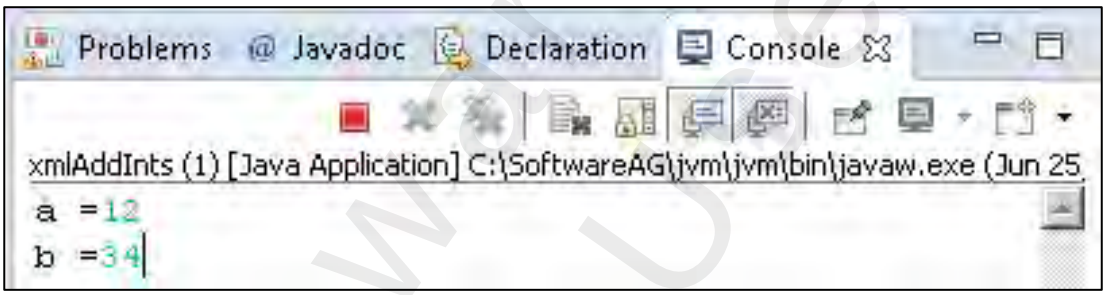
Save your changes.
Note: The Java code will be automatically compiled in the background when you save.

i. Run your Java client by right-clicking the **xmladdInts.java** file and selecting **Run As --> Java Application**.

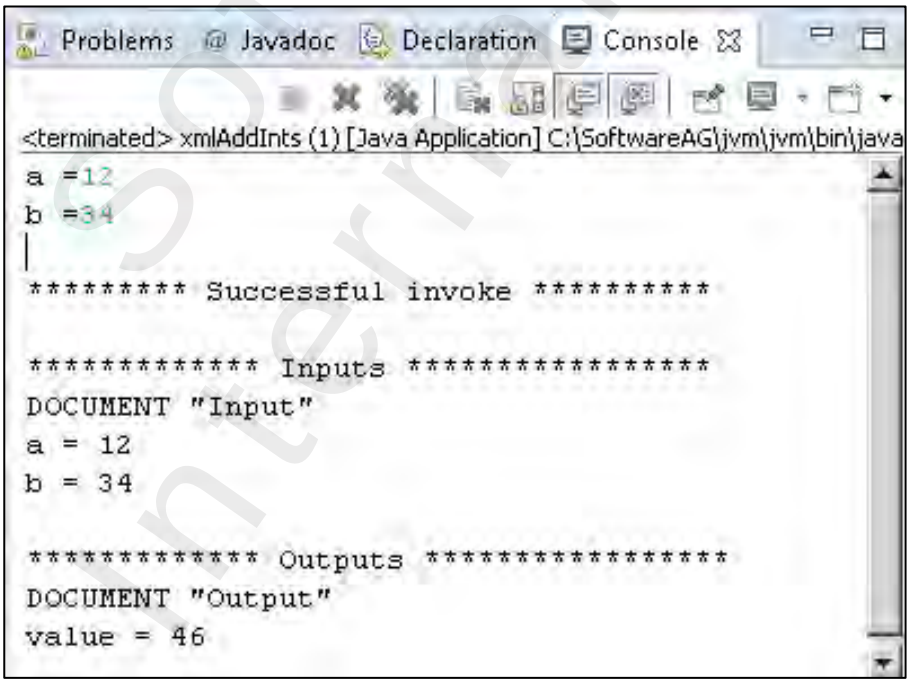
j. In the Console view, enter the following value shown in bold:

   i. a=**12** (press <Enter> to continue)

   ii. b=**34** (press <Enter> to continue)



Press the <Enter> key to run the Java client.



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:
WORKING WITH FLAT FILES

Overview

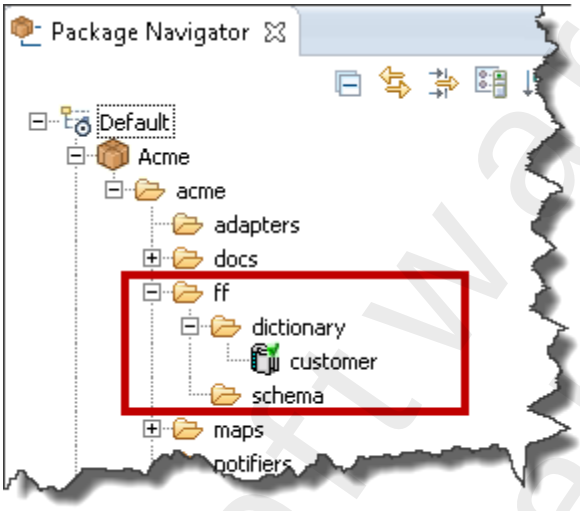
In this exercise, you will create a new Flat File Dictionary, create a reusable record definition in the Dictionary, and reference this record definition in a new Flat File Schema.

You will create a service to test your work.

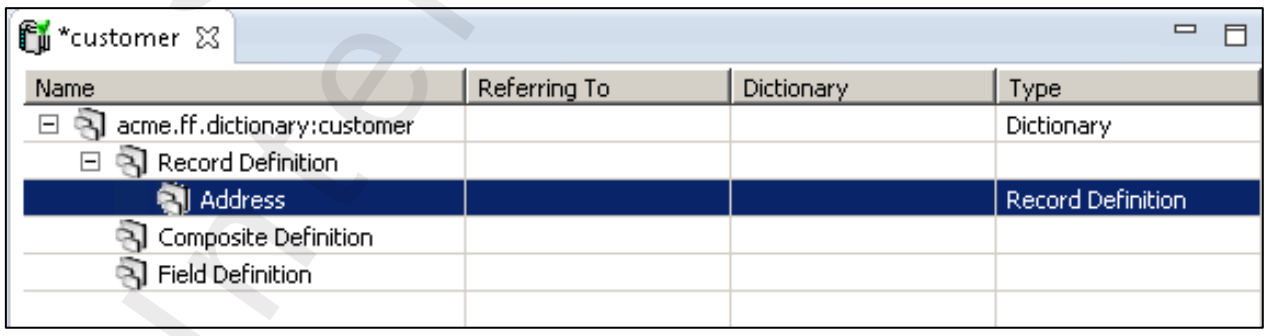
Steps

1. In the Acme package create an **acme.ff** folder. Create two new sub-folders called **schema** and **dictionary**.

   Create a **Flat File Dictionary** called **acme.ff.dictionary:customer.**



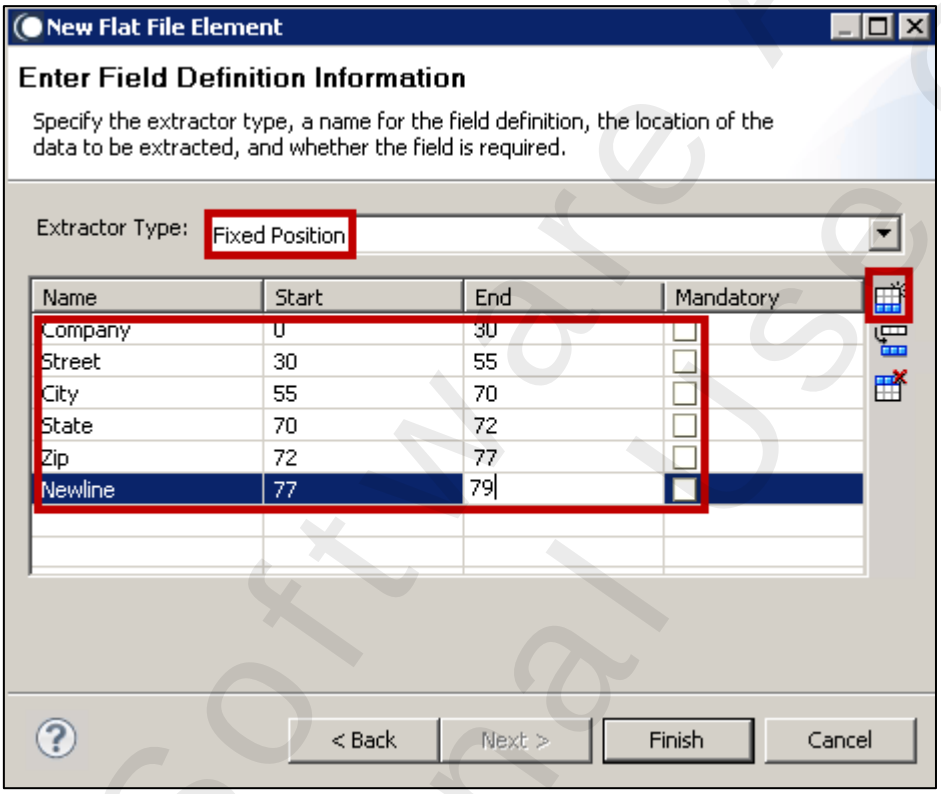
2. In the **customer** Flat File Dictionary, right-click **Record Definition --> New** to create a record definition called **Address**.



3. Right-click **Address.** Select **New --> Field definition --> Next.**

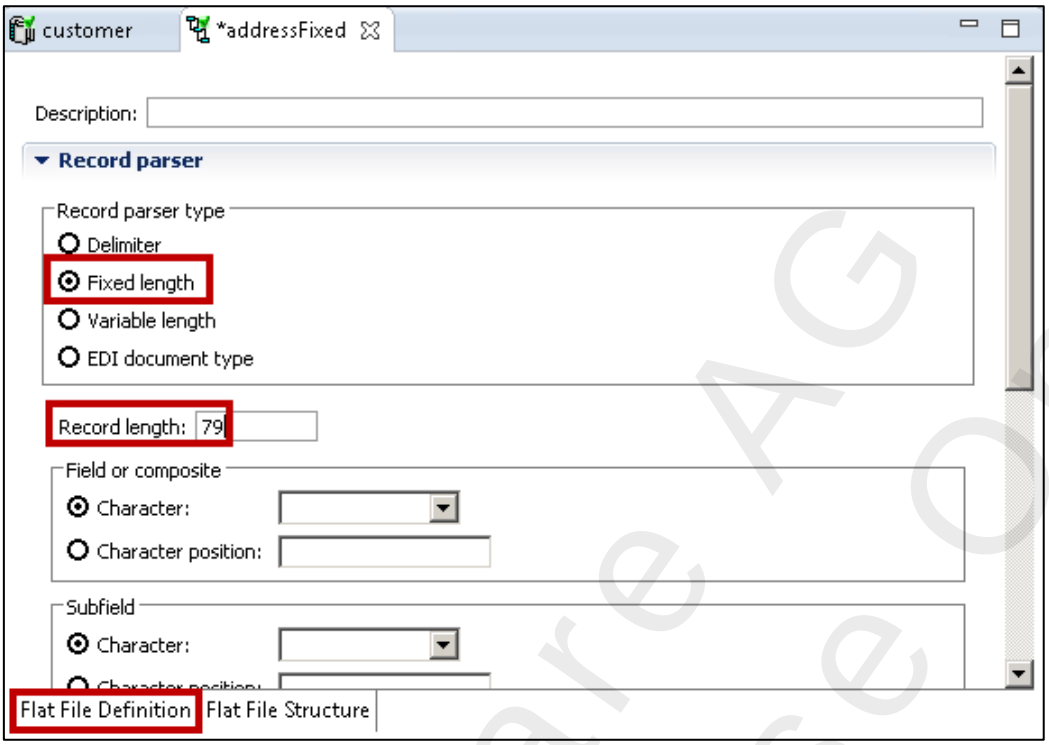
   Confirm that the **Extractor Type** is of type **Fixed Position**, use the **Add Row** button to add 6 new Field definitions to the **Address** record as shown in the table below. All fields are non-mandatory:

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

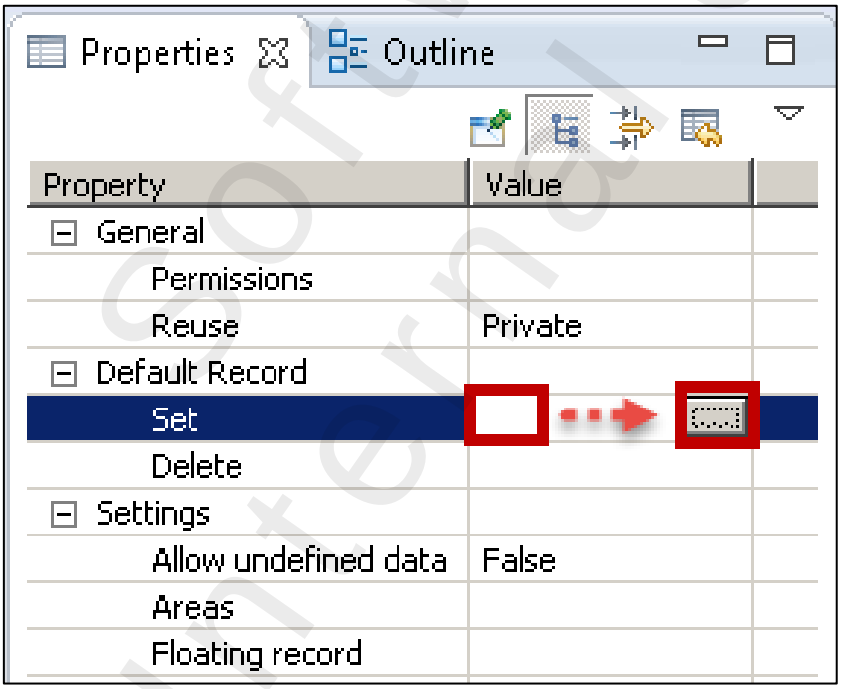


   Click **Finish,** then **save** your work.

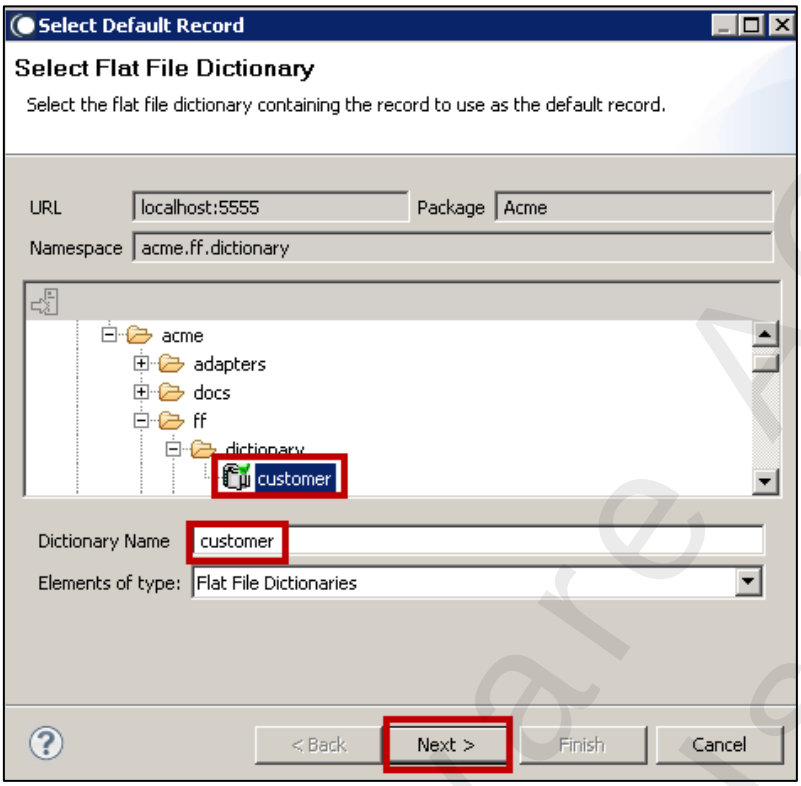
4. Create a **Flat File Schema** called **acme.ff.schema:addressFixed**. Specify that it is **Fixed Length** with a **Record length** of **79** characters.



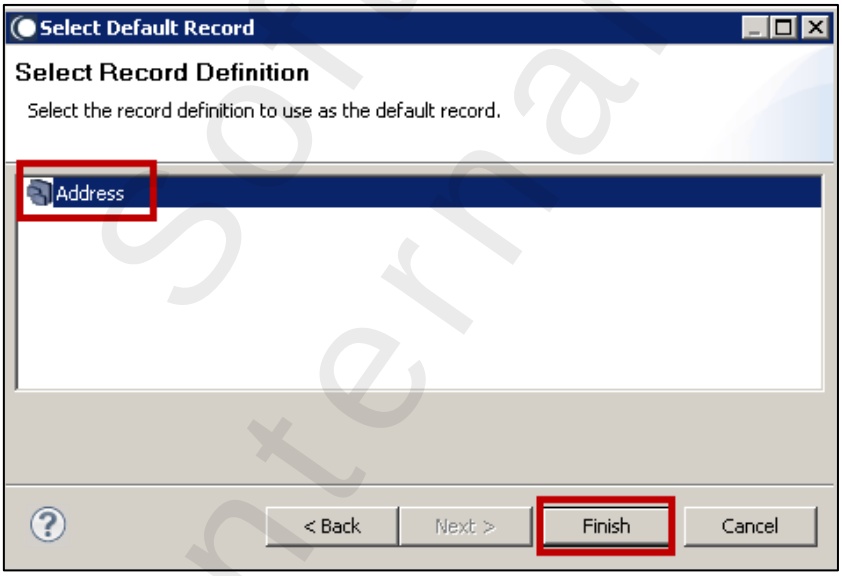
5. In the **Default Record** property of the **addressFixed** schema, click the white area in the **Value** column of the **Set** row. Then, click the **(…)** box to bring up the **Select Flat File Dictionary** dialog box:



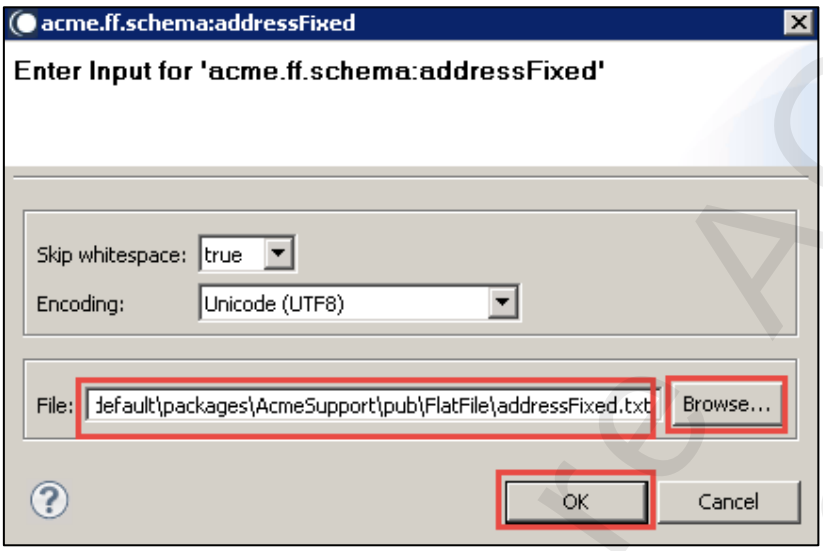
In the **Select Flat File Dictionary** dialog box, navigate to the **customer** dictionary and select it. Click **Next**.

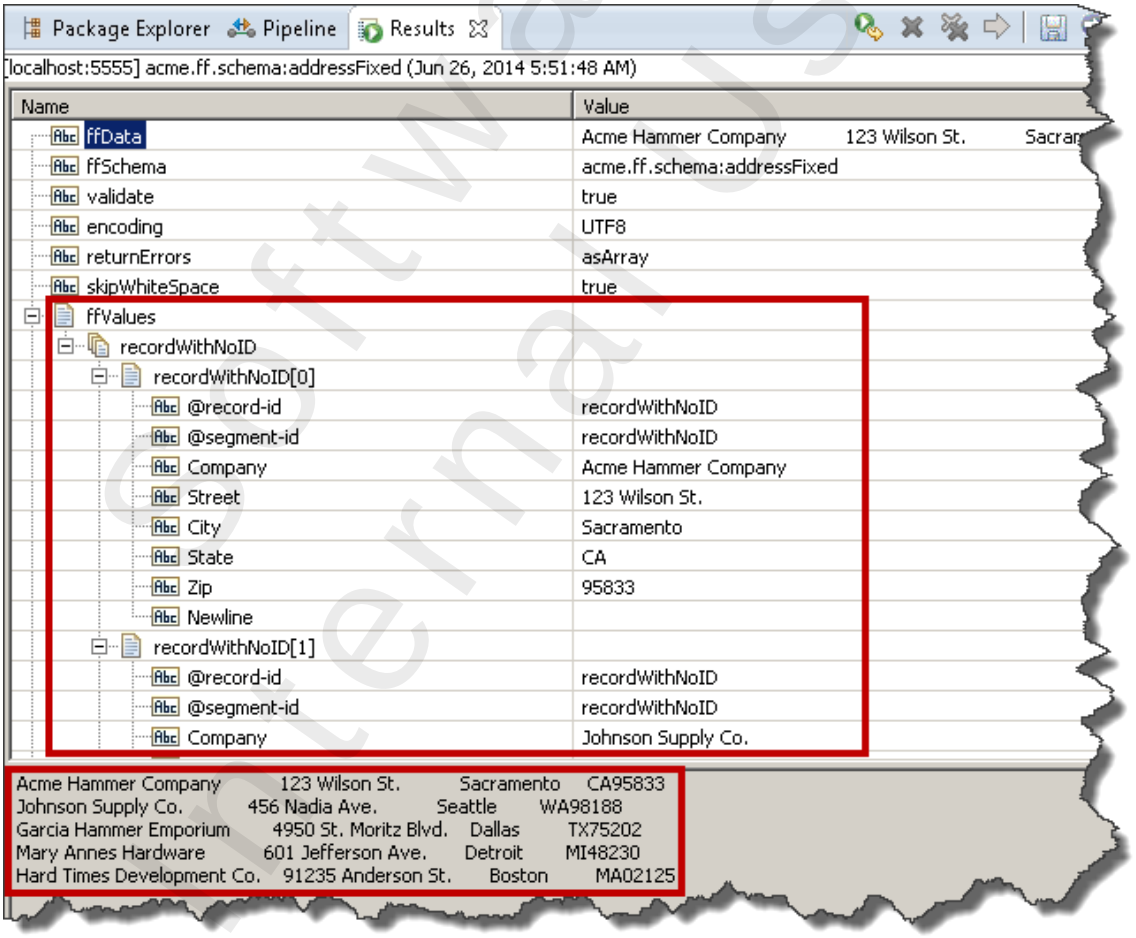


In the **Select Record Definition** dialog box, choose the **Address** record and click **Finish**.

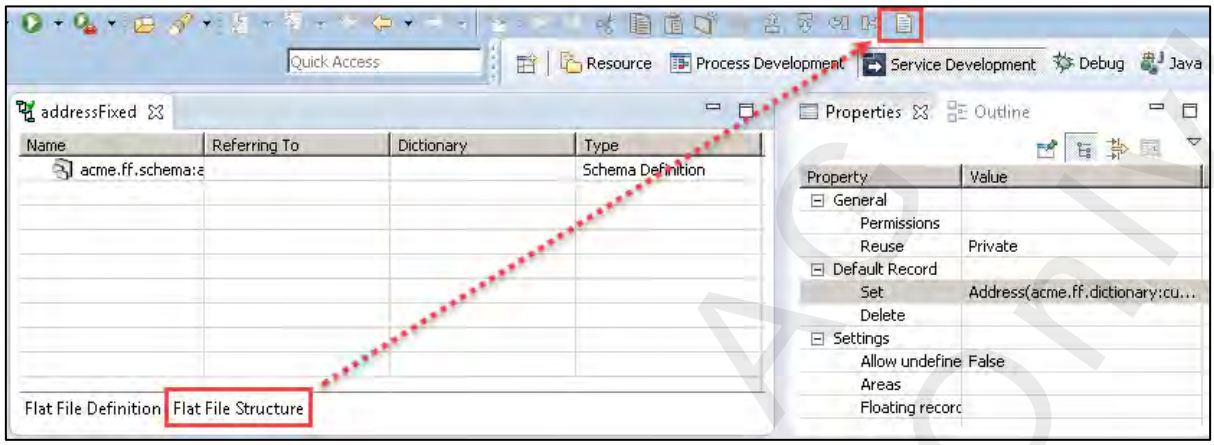


6. **Save** your work and test the **addressFixed** Flat File Schema by right clicking on it in the Package Navigator.
   Then, select **Run As --> Flat File Schema.** As input, browse for the file **addressFixed.txt** contained in the package folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub\FlatFile.**
   Inspect the test results in the Flat File Results view.

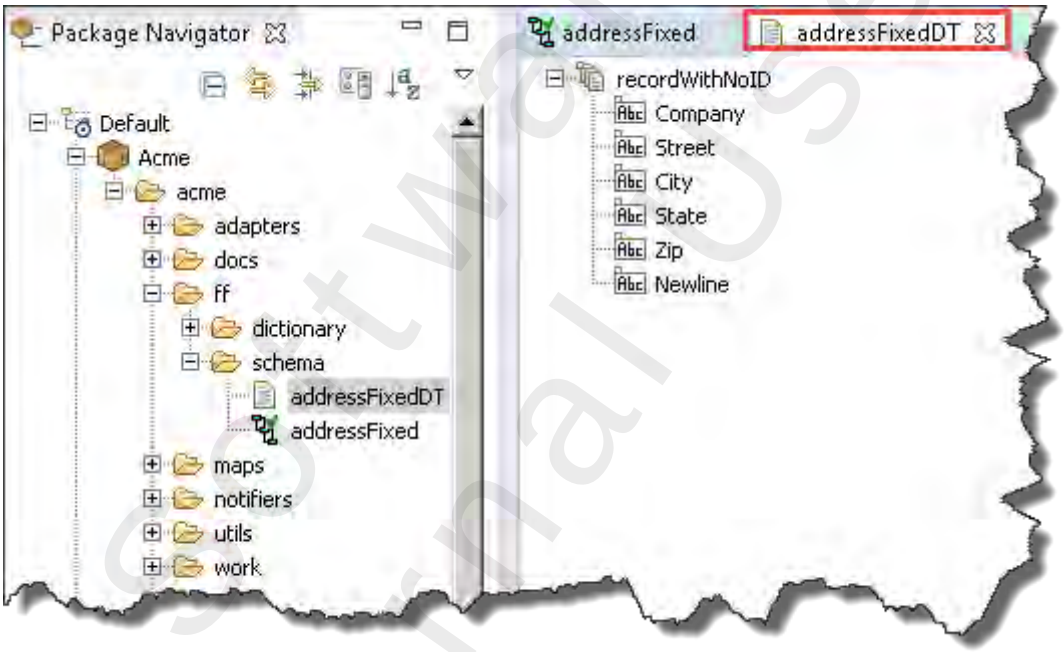




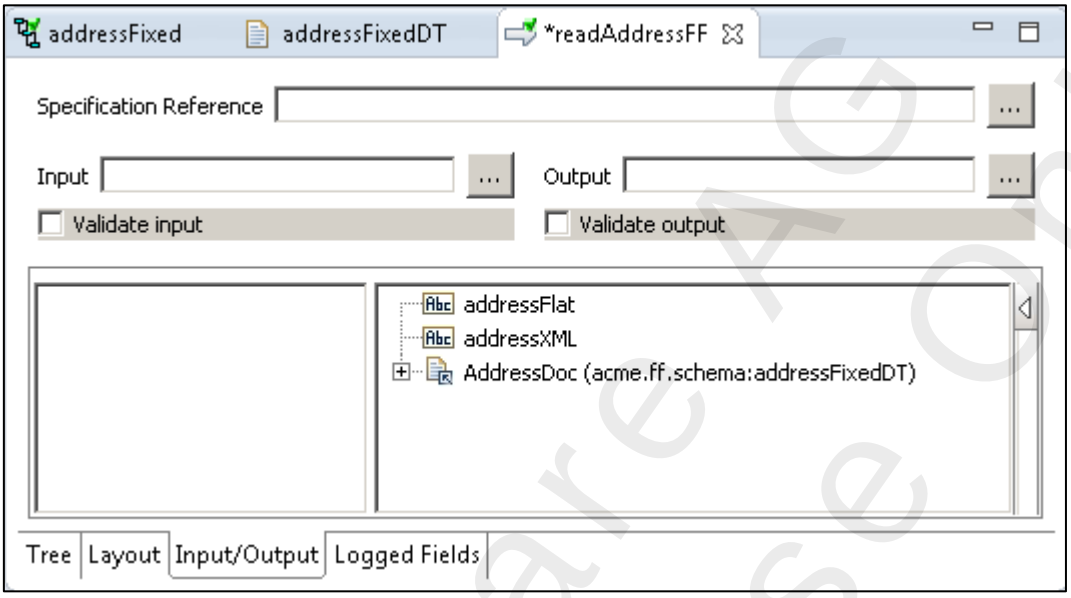
7. Once the **addressFixed** schema functions correctly, select the **Flat File Structure** tab, and then select the **Create Document Type** icon to create the **addressFixedDT** IS document type.
   Note that you must select the **Flat File Structure** tab in order for the **Create Document Type** to be active.



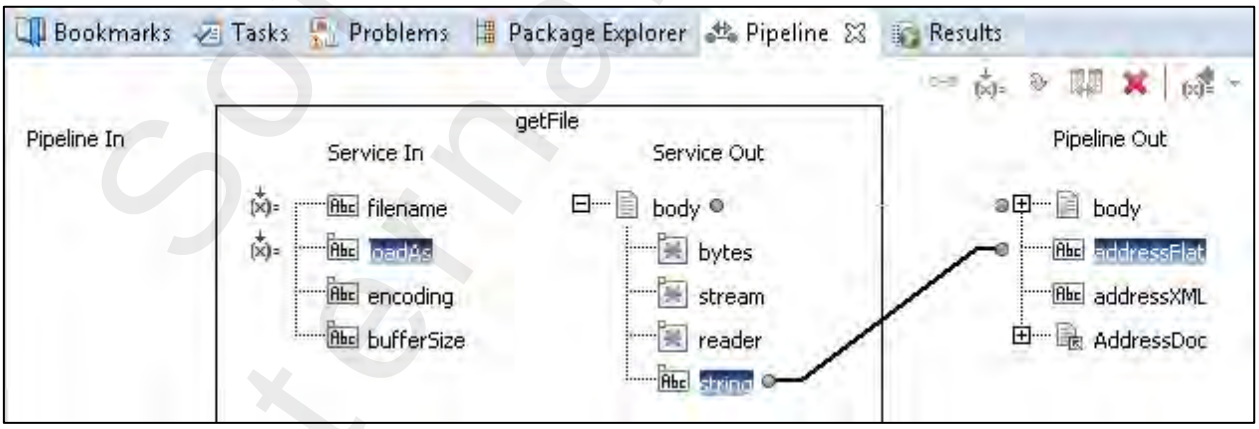
8. View the **acme.ff.schema:addressFixedDT** document type (it was automatically opened after it was created). Note, that it contains **recordWithNoID** (Document List) which contains the 6 fields you defined earlier in the exercise.



9. Create an empty flow service named **acme.ff:readAddressFF**. Add the following three output variables:
   - **String** named **addressFlat**
   - **String** named **addressXML**
   - Document Reference to **acme.ff.schema:addressFixedDT** named **AddressDoc**

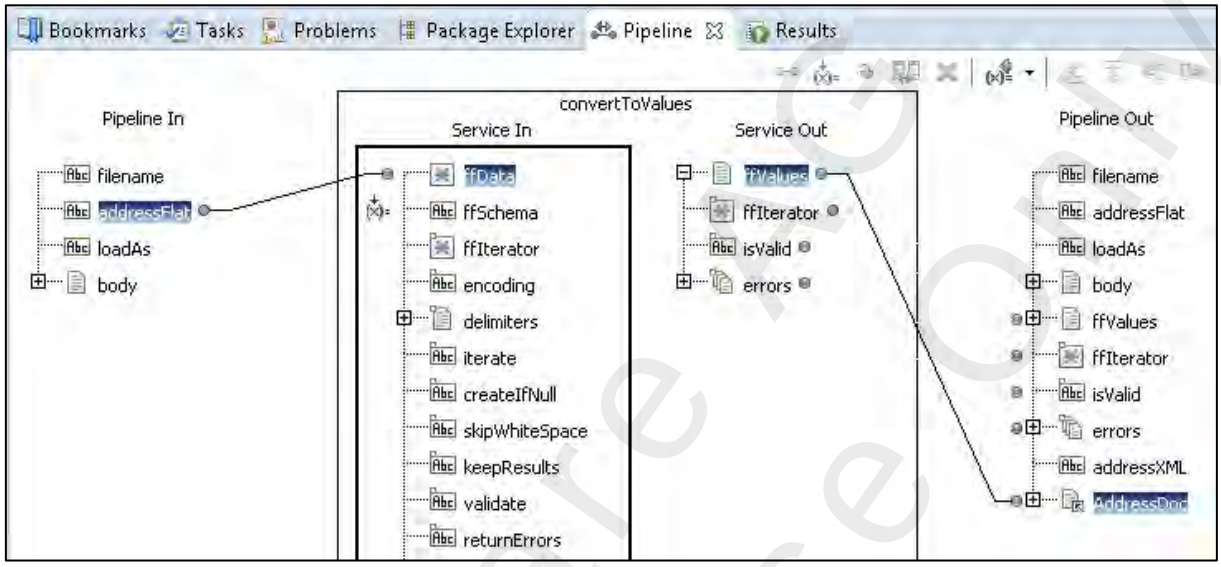


10. In the Tree tab, add an Invoke of the **WmPublic** package's **pub.file:getFile** service. Set and link the following in the Pipeline view:
    - Set **filename** (Service In) to **packages/AcmeSupport/pub/FlatFile/addressFixed.txt**
    - Set **loadAs** (Service In) to **string** (select from drop down)
    - Link **body/string** (Service Out) to **addressFlat**



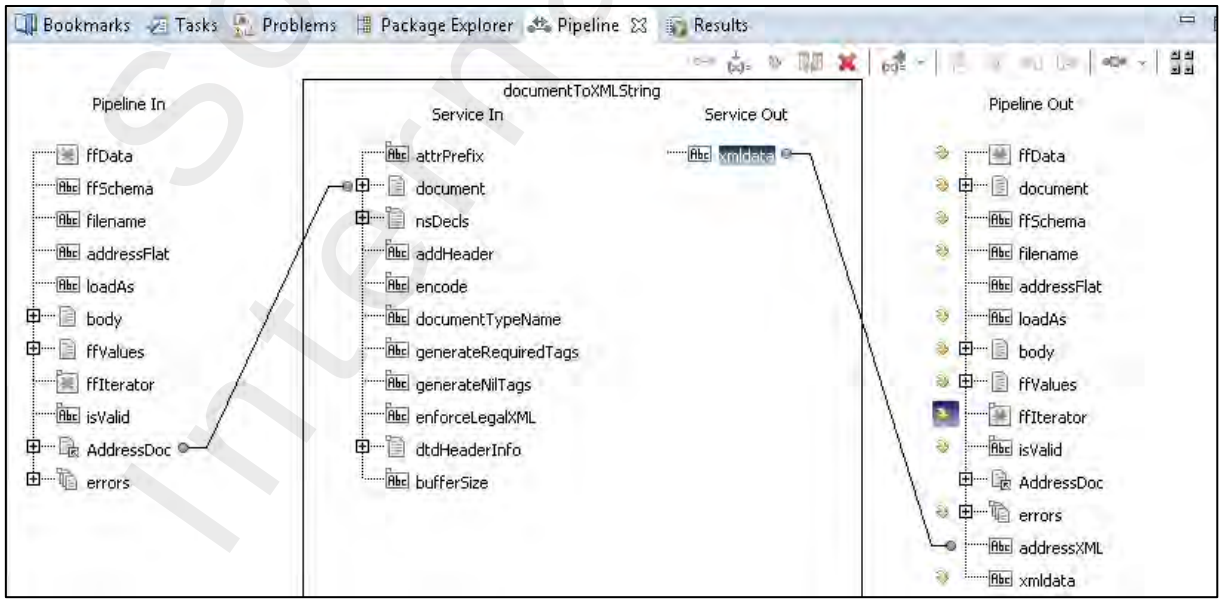
11. In the Package Navigator view, copy the **acme.ff.schema:addressFixed** schema to the clipboard.

12. Add an Invoke of the **WmFlatFile** package's **pub.flatFile:convertToValues** service. In the Pipeline view, set and link the following:
    - Link **addressFlat** (Pipeline In) to **ffData**
    - Set **ffSchema** (Service In) to **acme.ff.schema:addressFixed** (paste value from clipboard)
    - Link **ffValues** (Service Out) to **AddressDoc**

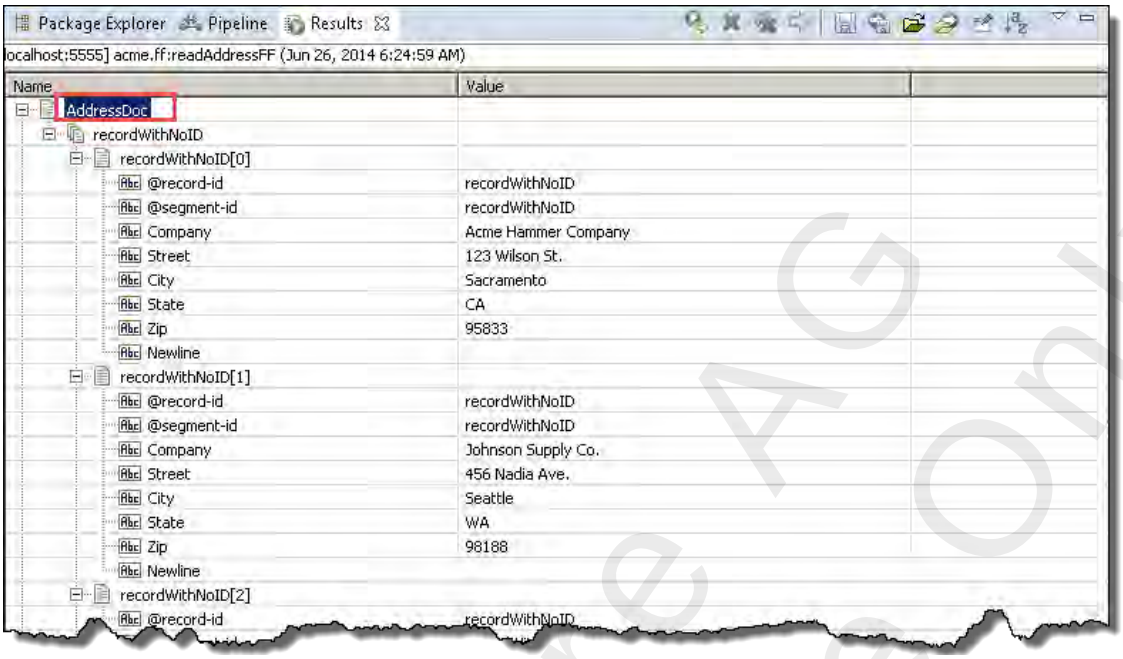


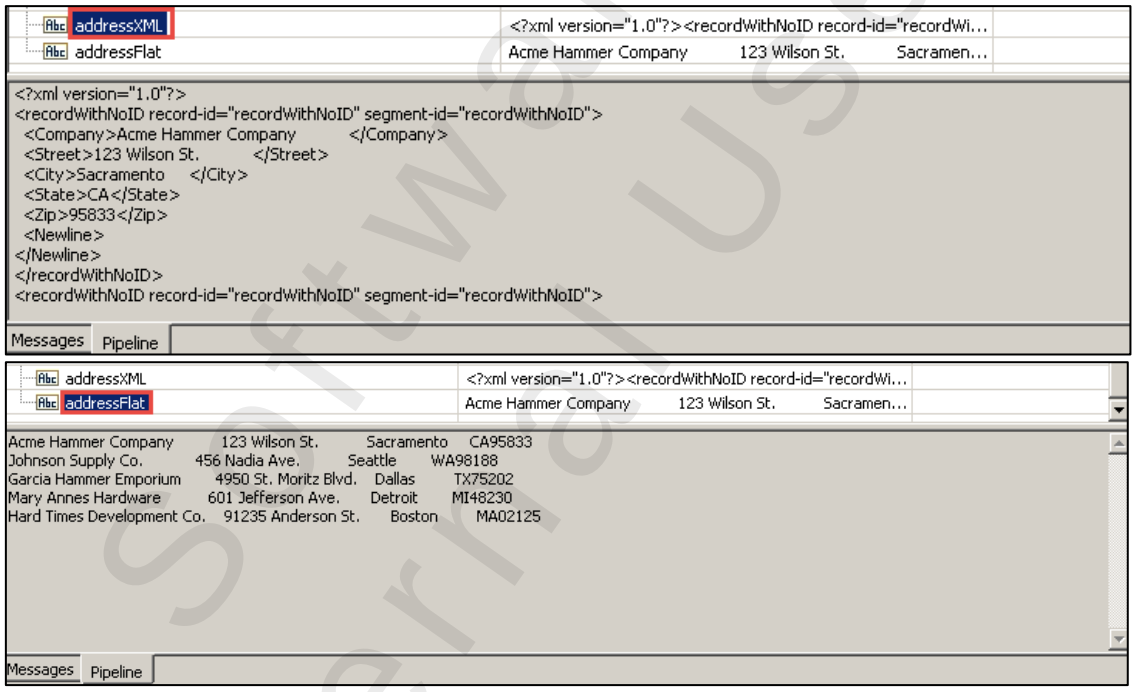
13. Add an Invoke of the **WmPublic** package's **pub.xml:documentToXMLString** service. In the Pipeline view, do the following:
    - Link **AddressDoc** (Pipeline In) to **document**
    - Link **xmldata** (Service Out) to **addressXML**

    Save your work.
    Finally drop all variables in Pipeline Out EXCEPT your three output variables **addressFlat**, **addressXML**, and **AddressDoc.**



14. **Save** and run the **readAddressFF** service. View the three output parameters which should each show the file's addresses in various formats:





Check Your Understanding

1. Why can't flat files be imported like XML documents?
2. What is the meaning of Nth field?
3. What is the difference between a dictionary and a schema?
4. Why should you create the IS document type when the schema is complete?

This page is intentionally left blank.

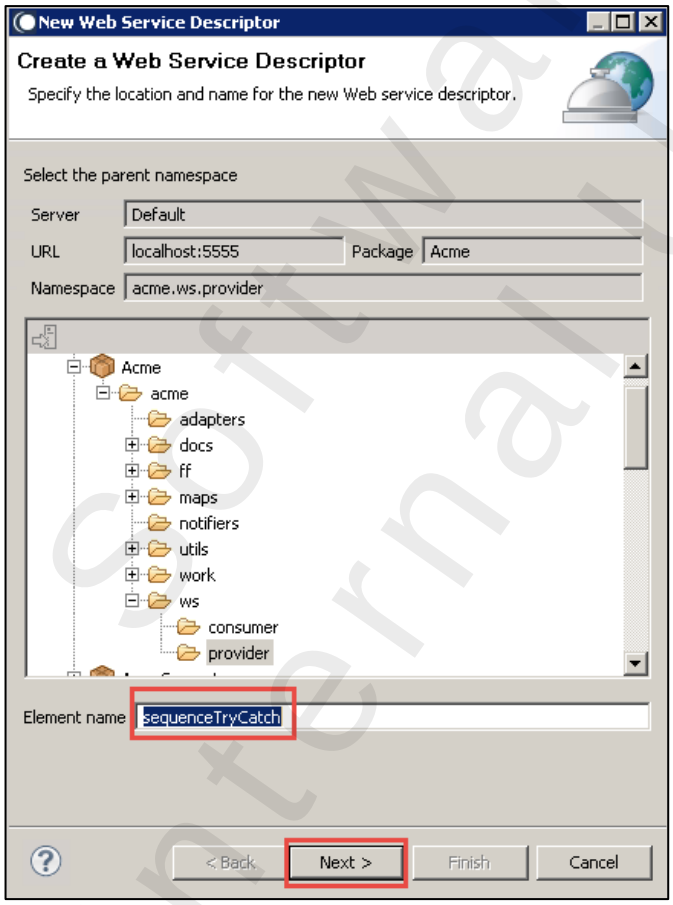
EXERCISE 14:
WEB SERVICE DESCRIPTORS AND CUSTOM FAULTS

Overview

In this exercise, you will take the Flow service you 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 and use this as a custom SOAP Fault response message. Then you test to see if the custom SOAP Fault gets returned as expected.

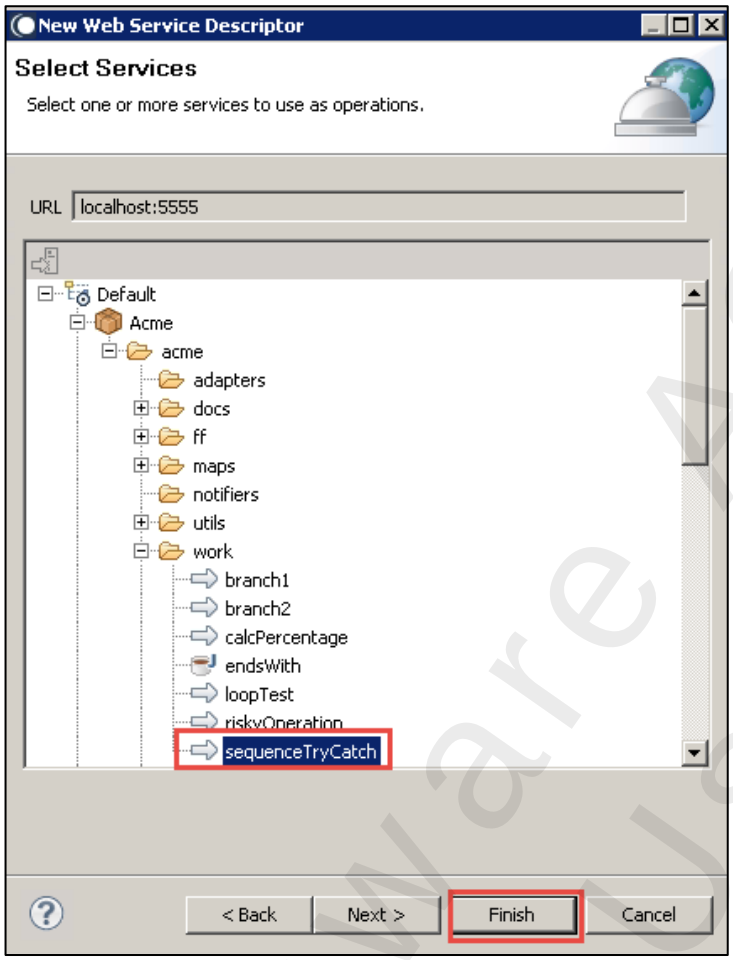
Steps

1. Add a Web Service Descriptor of type Provider to your Acme package. To do so:
   a. Use the Package Navigator view to add a **Web Service Descriptor** named **acme.ws.provider**:**sequenceTryCatch**. Click **Next**.

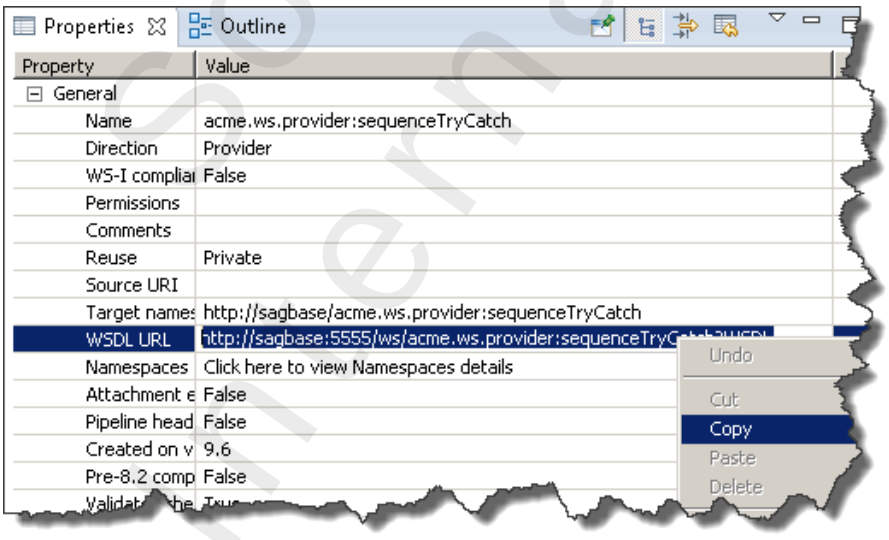


   b. Accept all of the defaults (**Provider**, **Existing IS service(s)**) and click the **Next>** button.

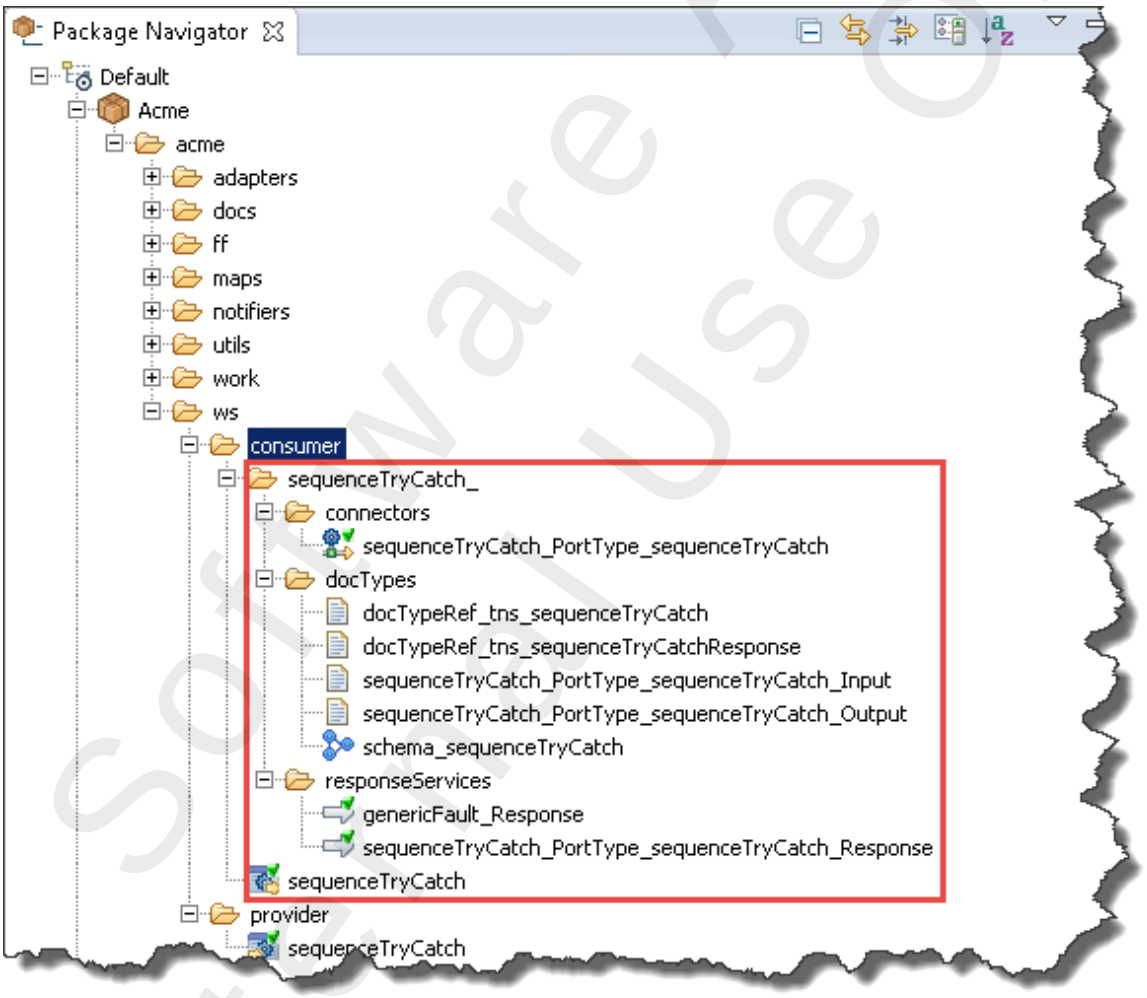
c. Select the IS service **acme.work:sequenceTryCatch** as service to use as Web service operation and click the **Finish** button.



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

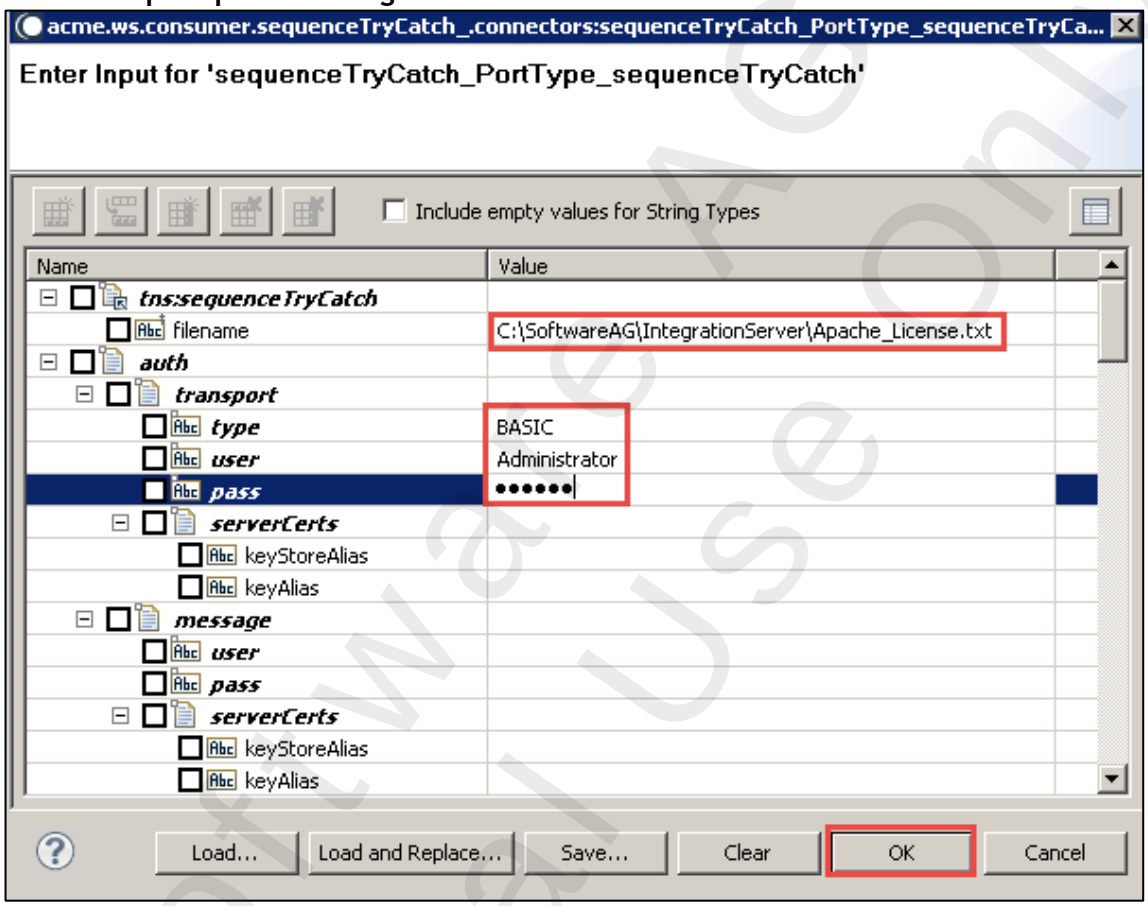


3. Open a browser tab, paste the WSDL URL in the address field and press return. The WSDL for the **sequenceTryCatch** web service should appear. This is the URL that consumers of the Web service must use to access its WSDL.
4. In Designer, add another **Web Service Descriptor** of type Consumer to your Acme package. To do so:
   a. Use the Package Navigator view to add a **Web Service Descriptor** named **acme.ws.consumer:sequenceTryCatch**. Click **Next**.
   b. This time select the **Consumer** radio button and accept the default for all other fields. Click **Next**.
   c. In the **Select the WSDL Location** dialogue select **File/URL** and paste the just copied WSDL URL into the text field. Then click the **Next** button.
   d. Accept all the defaults and click the **Finish** button.
   Notice all the newly generated objects.

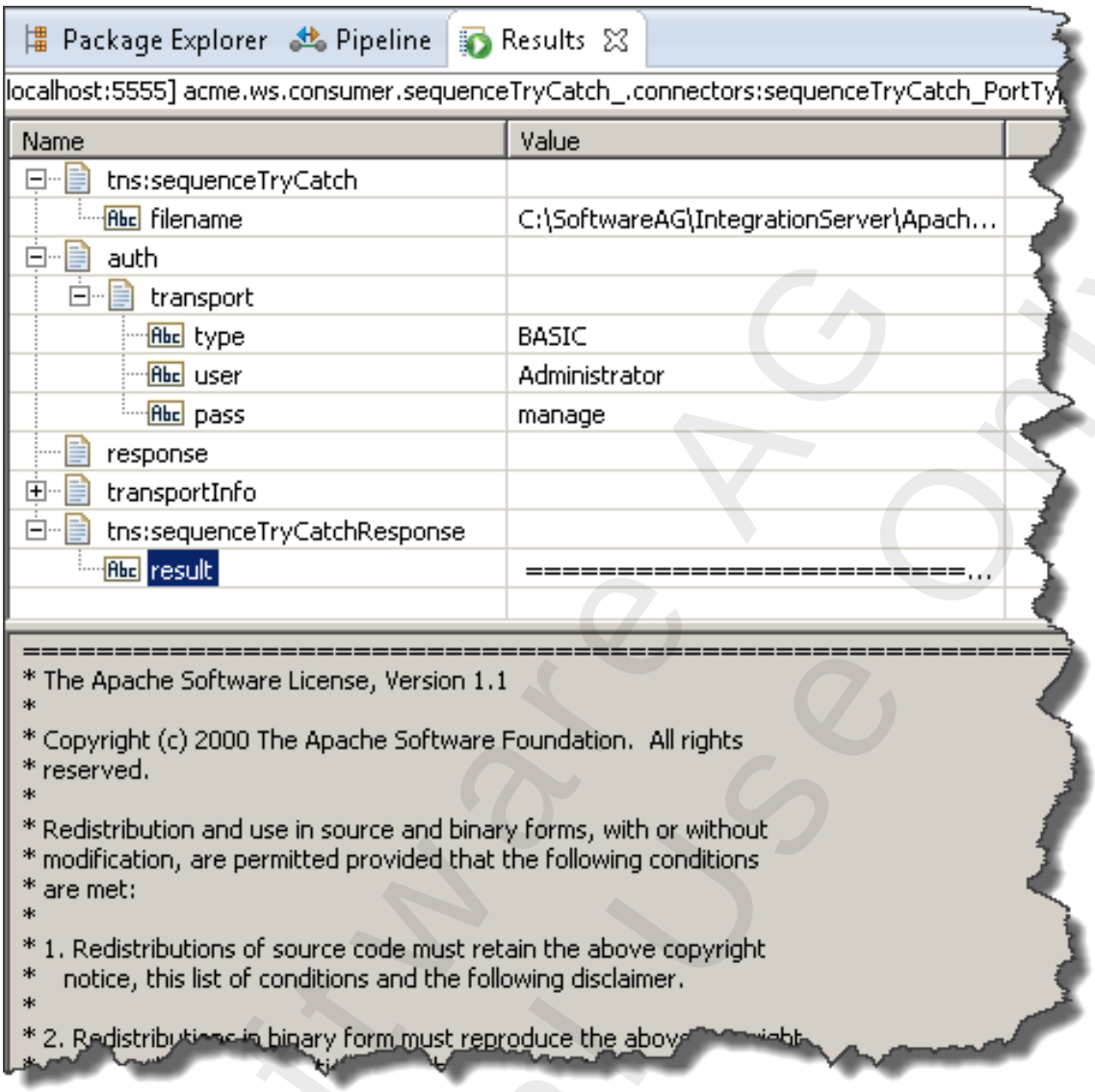


5. Open the Web Service Connector **acme.ws.consumer.sequenceTryCatch_.connectors: sequenceTryCatch_PortType_sequenceTryCatch.**
   Inspect the generated Flow code in the Tree tab.

6. In the Package Navigator view, right-click the **sequenceTryCatch_PortType_SequenceTryCatch** Web Service Connector and select **Run As --> Run Flow Service**. Provide the following values as input and click **OK**:
   a. **tns:sequenceTryCatch/filename** = **C:\SoftwareAG\IntegrationServer\Apache_License.txt**
   b. **auth/transport/type** = **BASIC**
   c. **auth/transport/user** = **Administrator**
   d. **auth/transport/pass** = **manage**



7. Review the results in the service Results view.

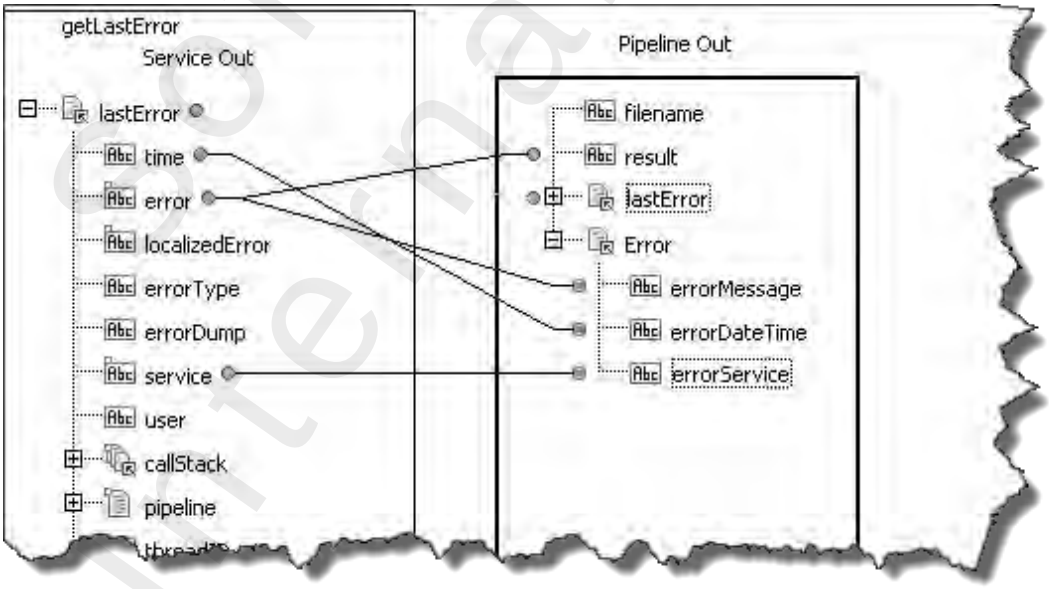


8. Now add a Document Type to your Acme package called **acme.docs:Error**. Add three String fields: errorMessage, errorDateTime and errorService. **Save** your work.

Note: this is just an example of a generic document used for SOAP Faults. You can create any error documents to meet your project needs.

9. Open the Flow service **acme.work:sequenceTryCatch** and modify it to use the new **Error** doc. If needed, lock the service for edit first:
   a. In the service, open the **Tree** tab. Expand the service and select the **pub.flow:getLastError** step.
   b. Click on the **Pipeline** tab. In the **Pipeline Out** section, select the **result** variable.
   c. Right click in the whitespace of the **Pipeline Out** section. Select **Insert ➡ Document Reference**, then navigate to the **acme.docs:Error** document type. Name the reference **Error**.
   d. Link the following from the getLastError Service Out to the Error document in the Pipeline Out:

lastError/error to **Error/errorMessage**
lastError/time to **Error/errorDateTime**
lastError/service to **Error/errorService**



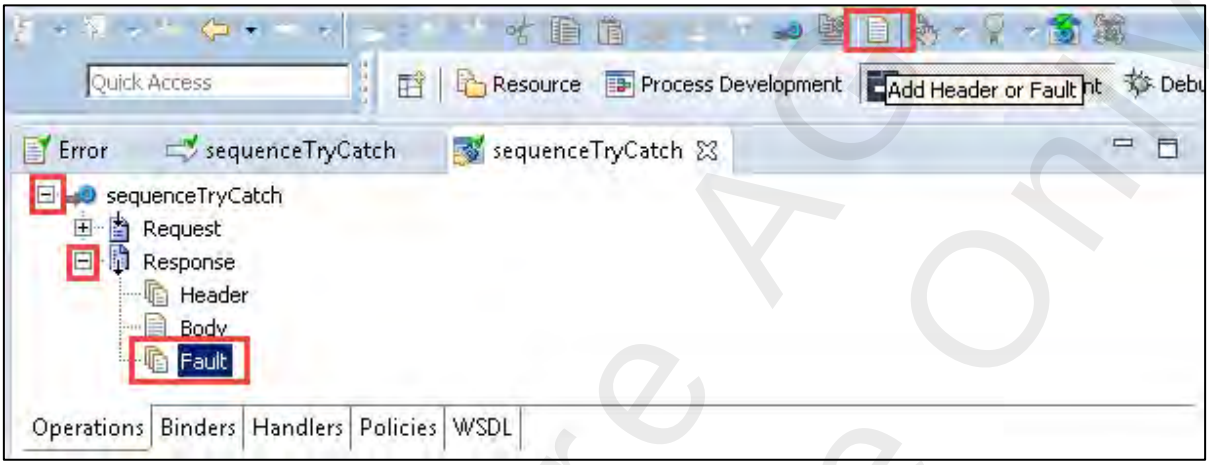
Note: Do not save the service before you set/link at least one field in the Error document, otherwise the Error document will disappear from the Pipeline Out.

e. **Save** your work.

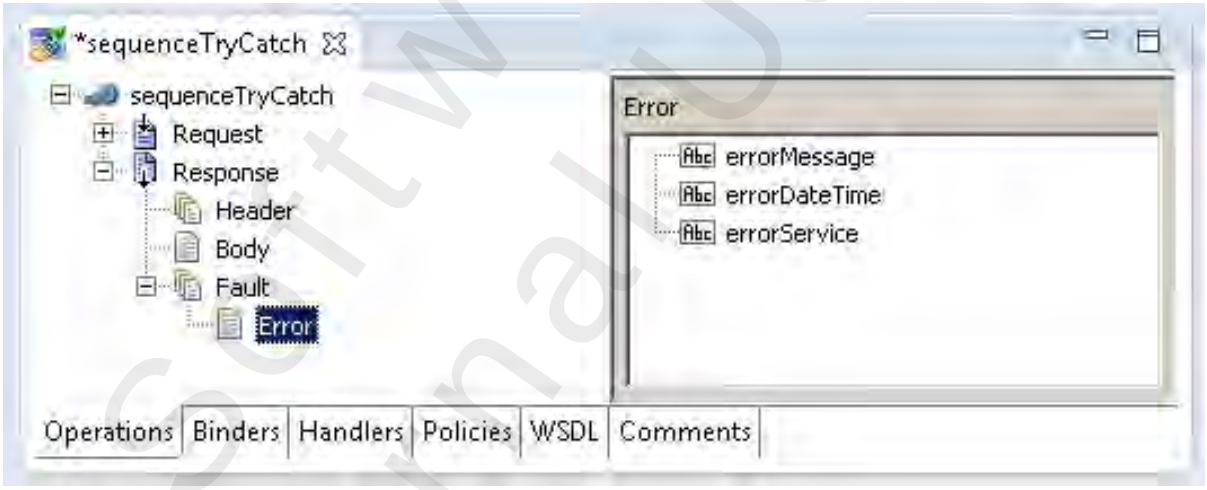
10. Open the Provider Web Service Descriptor **acme.ws.provider:sequenceTryCatch.**

    Expand the **sequenceTryCatch** operation, and 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.



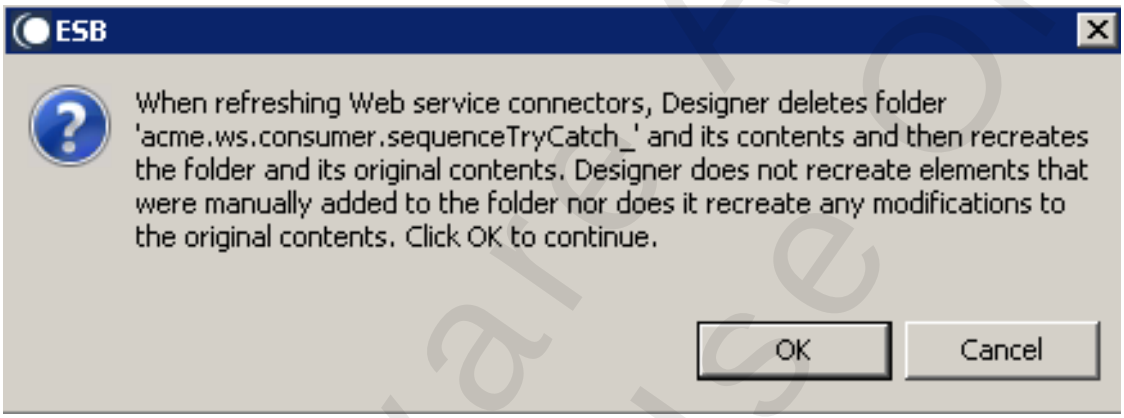
    In the popup dialog, navigate to the **acme.docs:Error** document type, select it, and click the **OK** button. Your **sequenceTryCatch** Web Service Descriptor (Provider) should look like the following:



    Save your changes.

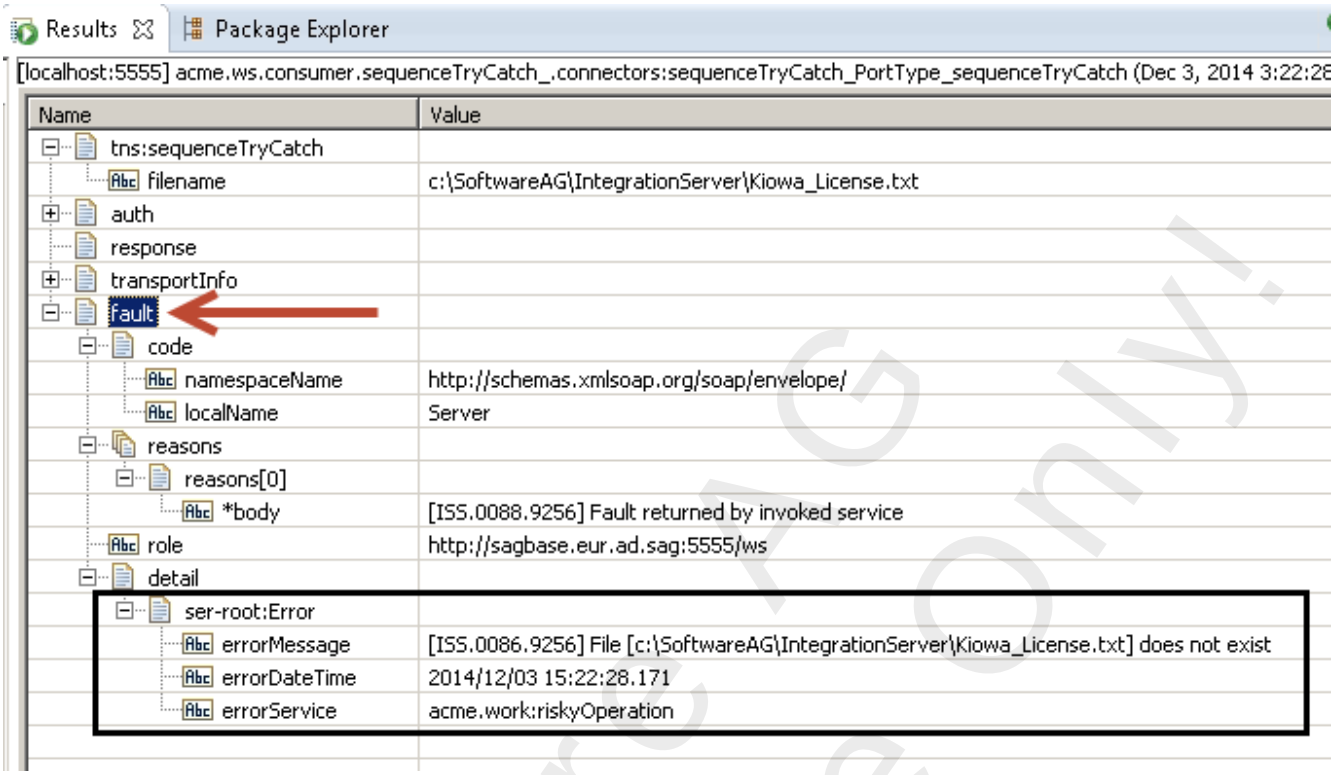
11. Now, you must regenerate the consumer **WSD** Connector Flow services so that it captures the change to the Provider Web Service Descriptor. To do this:
    a. Edit the **acme.ws.consumer:sequenceTryCatch** Consumer Web Service Descriptor
    b. Select the **sequenceTryCatch** operation
    c. Click the **Refresh Web Service Connectors** icon in the main menu's tool bar.

You will get a popup saying:



After reading the message, click the **OK** button.

12. Repeat steps 6 and 7 from above to run the consumer Web Service Connector **sequenceTryCatch_PortType_SequenceTryCatch** Flow Service, but this time enter a **filename** that does not exist (e.g. **C:\SoftwareAG\IntegrationServer\Kiowa_License.txt)**. You should see your custom fault document in the Service Result view.



Check Your Understanding

1. When would you create a Provider Web Service Descriptor (WSD) and when a Consumer WSD?
2. How and when are Web Service Connectors (WSC) created?
3. Can you have more than one custom SOAP Fault Document?

This Page intentionally left blank

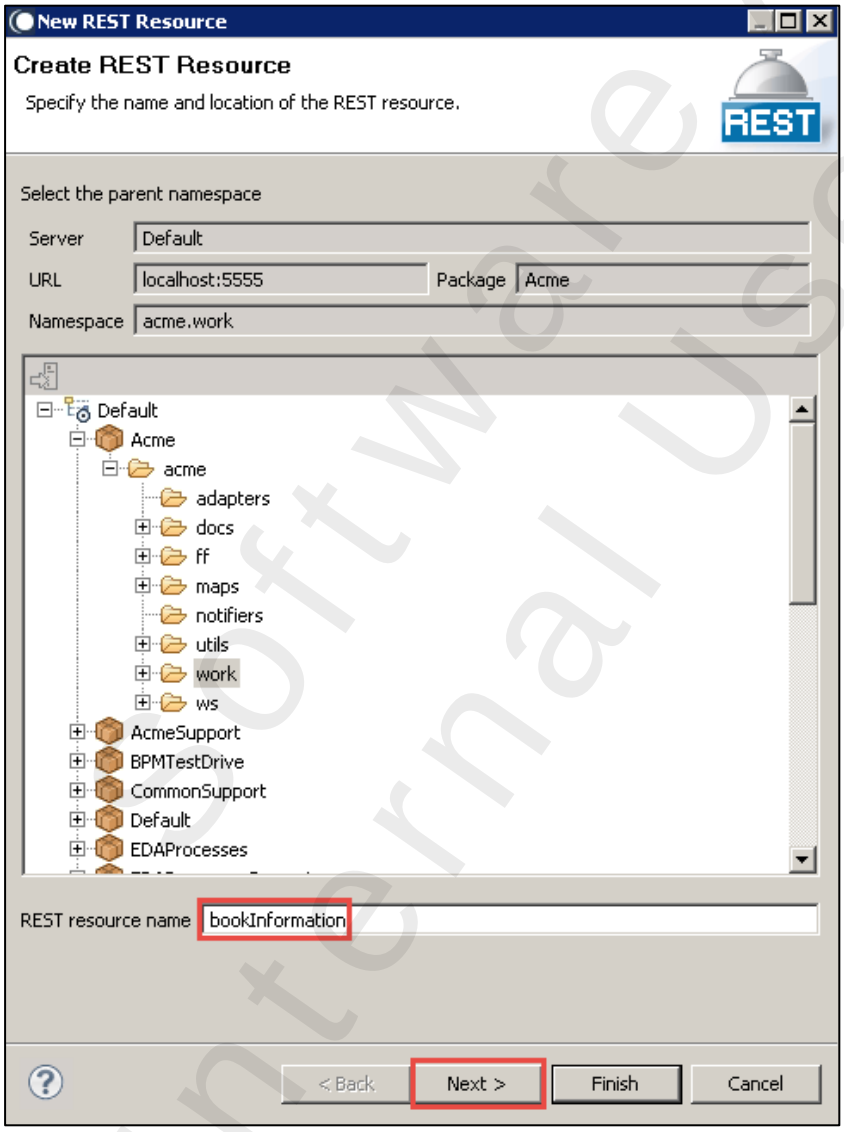
EXERCISE 15:
CREATE REST SERVICE IN INTEGRATION SERVER

Overview

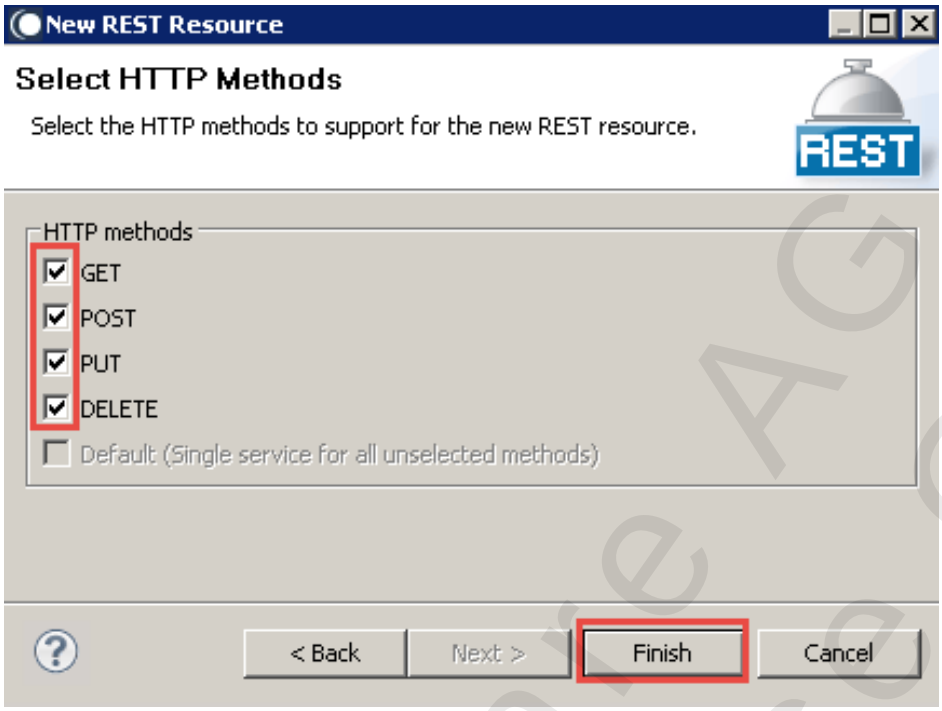
In this exercise, you will create a REST resource in the Integration Server and test it by sending REST requests using the Firefox REST Client.

Steps

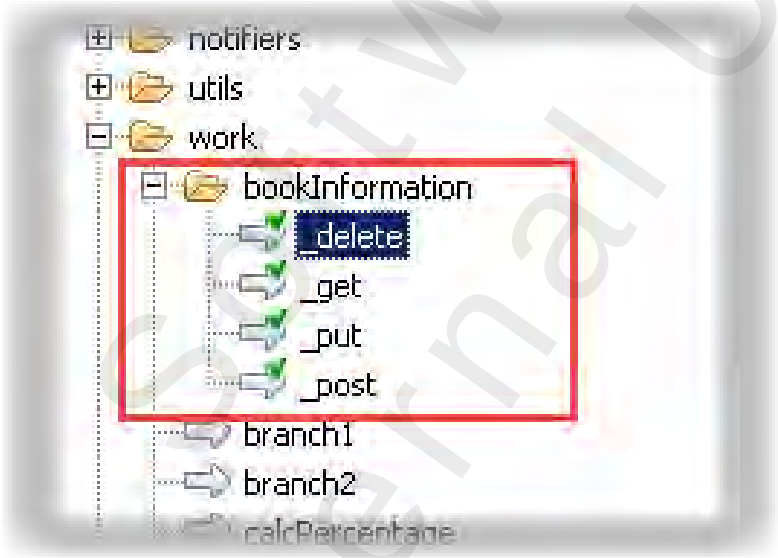
1. In Designer, create an object of type **REST Resource** under the **acme.work** folder. Call it **bookInformation**. Click **Next**.



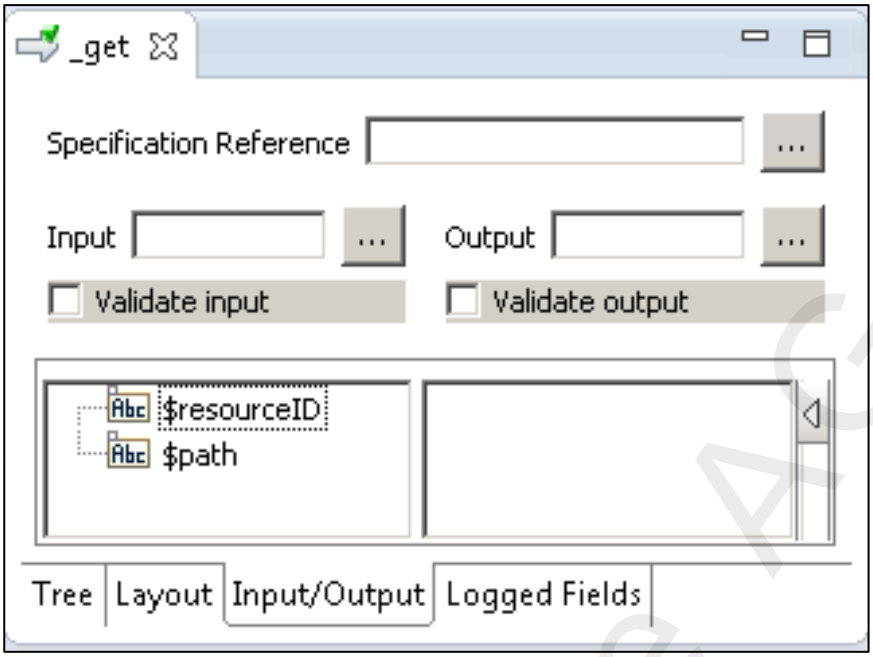
On the next panel select the HTTP methods **GET**, **POST**, **PUT** and **DELETE**. Click **Finish**.



You should end up with the following structure:

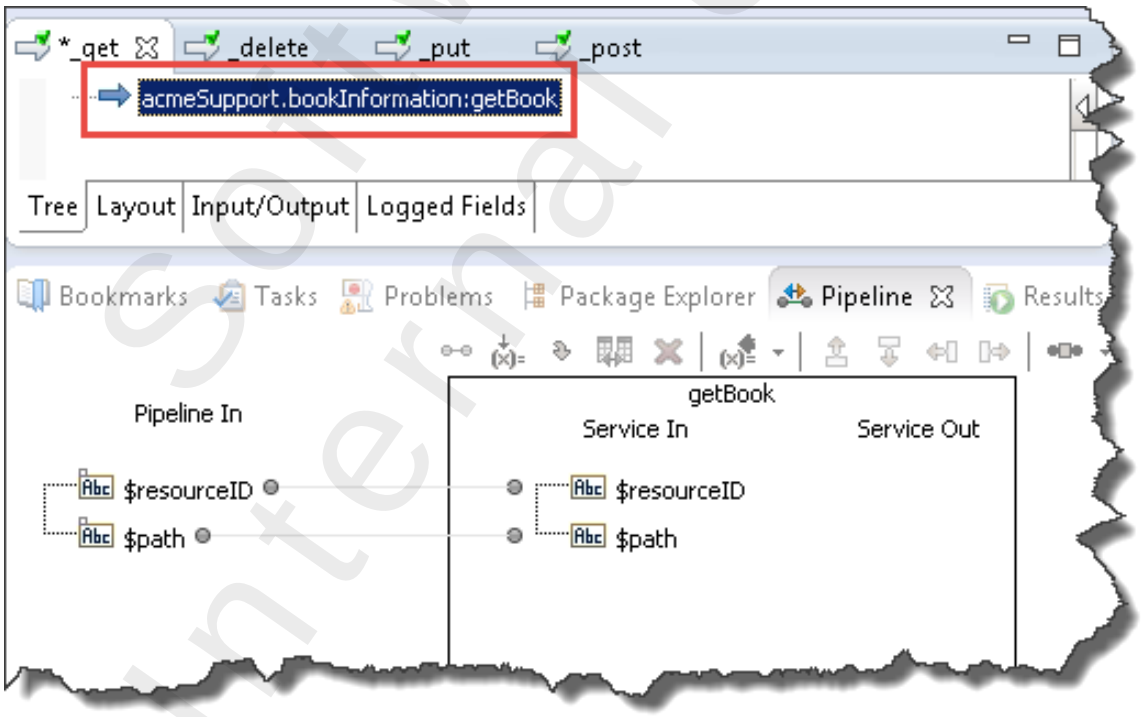


Each of the four services will have the following Input/Output:



2. To keep the implementation of the REST services simple in this exercise we have pre-created four Java services in the **AcmeSupport** package, in folder **acmeSupport.bookInformation**: deleteBook, getBook, postBook and putBook. Consider these services the "business logic" behind the generated REST (wrapper) services.

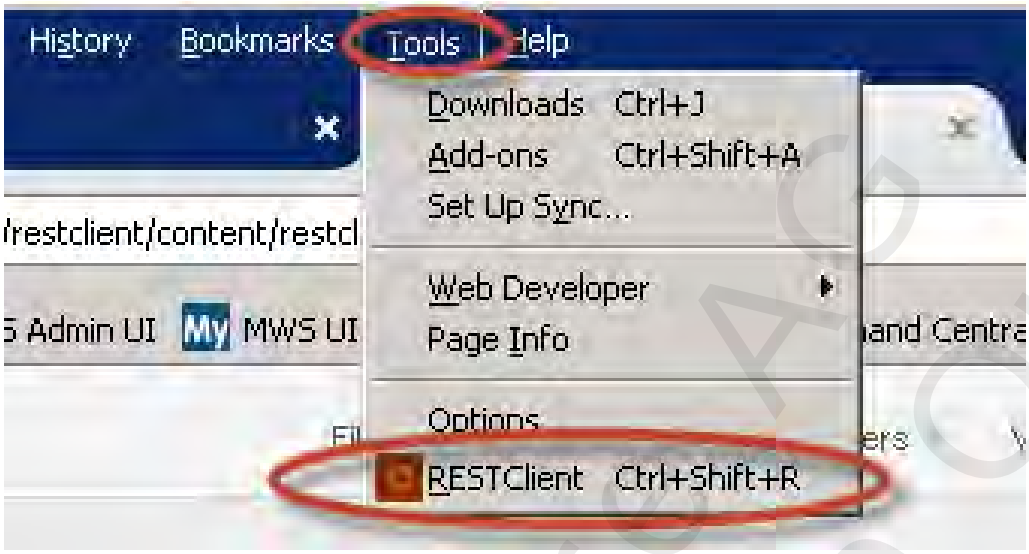
   Drag and drop each of these services into the appropriate generated Flow REST service under **acme.work.bookInformation**.
   For example, to implement the **_get** service, drag **acmeSupport.bookInformation:getBook** into it. The completed **_get** service will look like this:



   Do the same thing for the **_delete**, **_post** and **_put** services.

Save all your work.

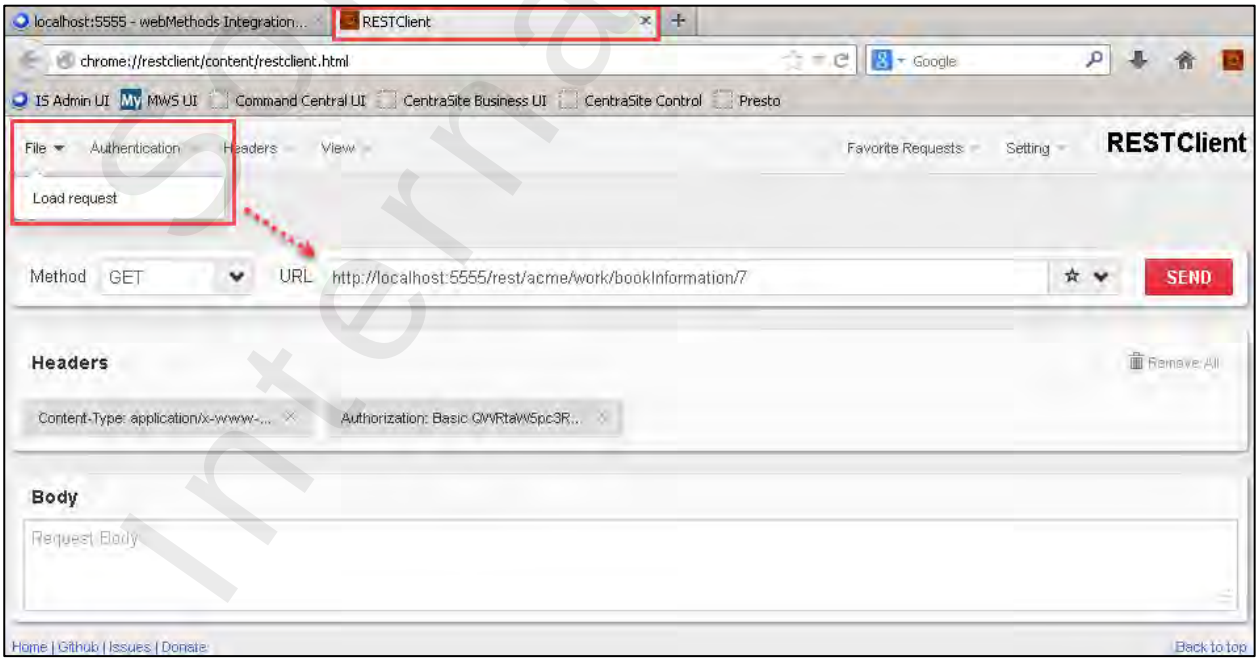
3. Test your **_get** service with the Firefox REST client. To do so, start Firefox, go to **Tools** and select **RESTClient** or click the appropriate icon in the menu bar.



4. In the **RESTClient** browser tab, load the arguments for a REST request by choosing **File --> Load request** (from the **RESTClient**'s menu**).**

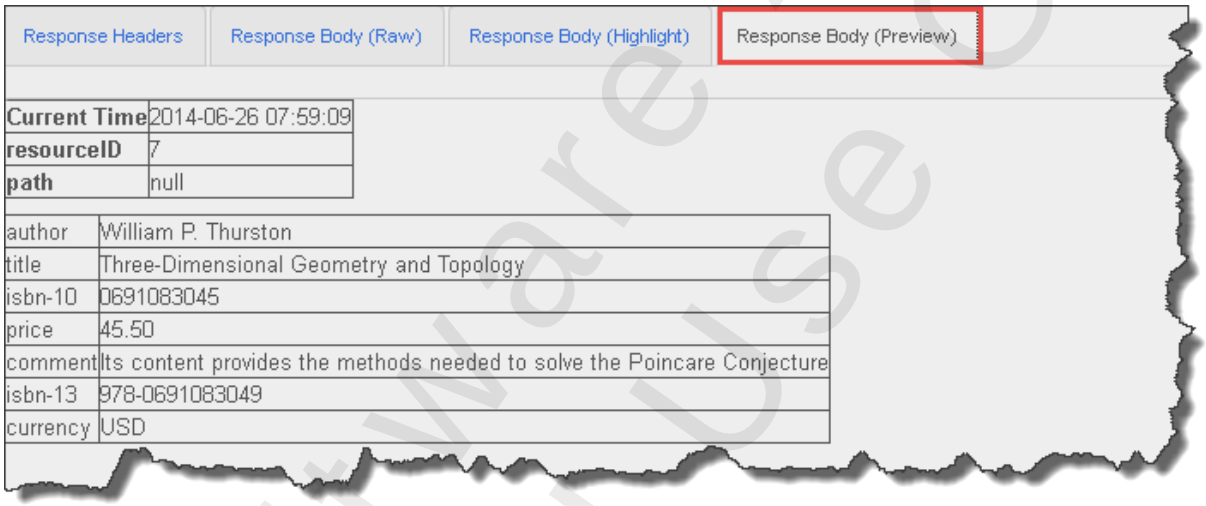
   Navigate to the file **GetBookBySKU.txt** contained in folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub\REST** and open it.

   After loading the file, you will see the following information in the input fields:



Notice that the Content-Type in the header is **application/x-www-form-urlencoded** (for a description of the format see http://en.wikipedia.org/wiki/Percent-encoding).

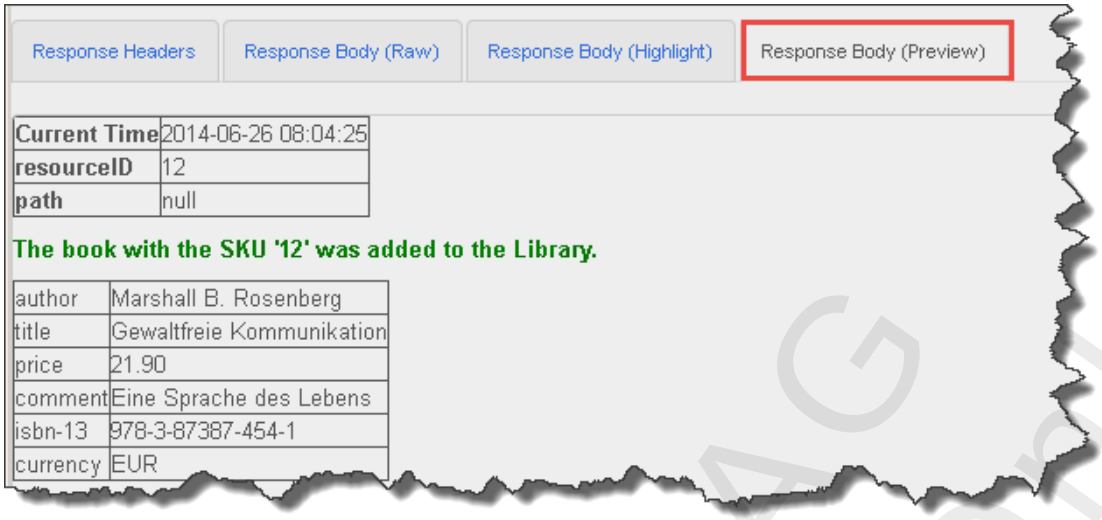
Now click the **Send** button. If prompted for authentication, provide **Administrator** | **manage**. You should get data about book number **7**:



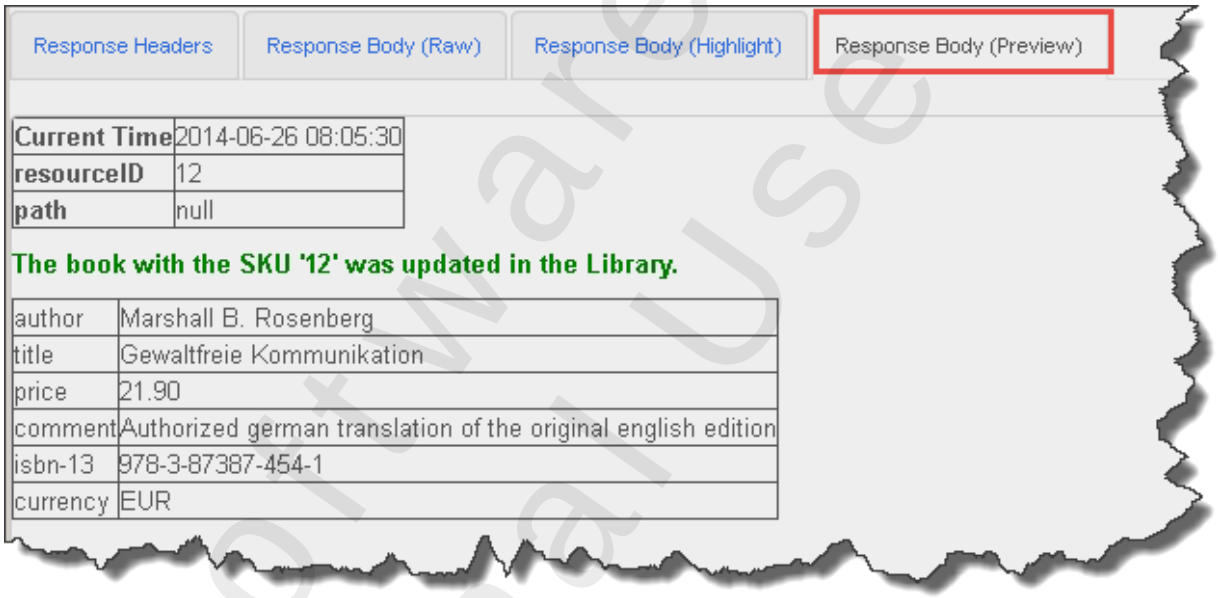
5. From the REST Client's menu, choose **File --> Load request** and browse for the file **DeleteBookBySKU.txt** from the folder mentioned above.

   Use the settings to delete book **3** from the library.

6. In the same way, load the **PostBookBySKU.txt** file and use it to store a new book number **12** in the library.



7. Finally load the **PutBookBySKU.txt** file to update the **comment** field of the book number **12** loaded by the previous step.



8. *For extra credit*: Create a Flow service called **acme.work:getBookInformation** in your **Acme** package. This service should have one string input argument named **SKU**, and one string output argument named **result**.

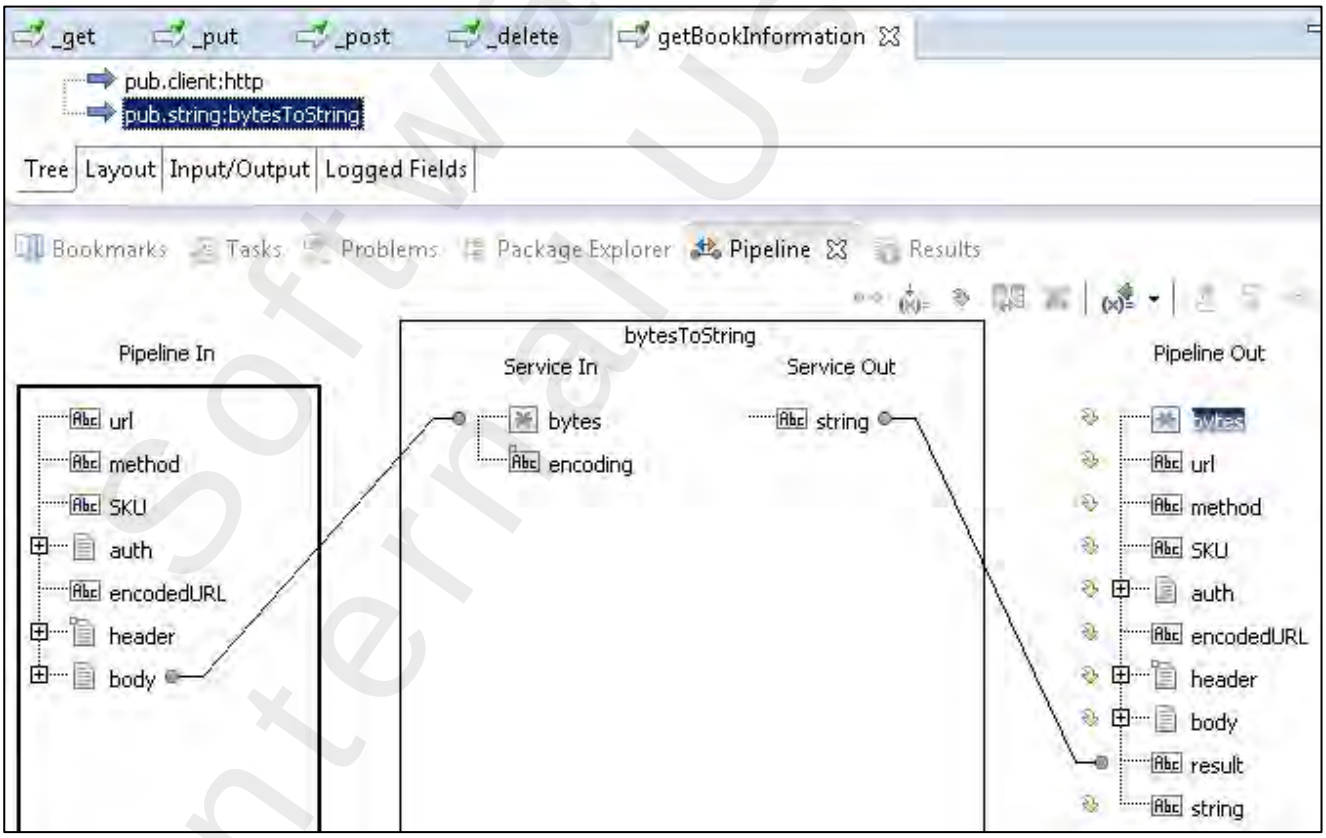
   In the new Flow service, you should invoke the WmPublic package's service **pub.client:http** with the following inputs to get a book description via the REST service you created above:

url	**http://sagbase:5555/rest/acme/work/bookInformation/%SKU%** **Hint:** Make sure you check **"Perform pipeline variable substitution"**
method	**get**
auth/type	**BASIC**
auth/user	**Administrator**
auth/pass	**manage**

   After **pub.client:http,** add an invoke to **pub.string:bytesToString** to convert the results in body/bytes (from pub.client:http) to a string which is linked to the output **result**.

   Save your work.

   Drop everything from Pipeline Out except the **result** variable.



   Save and run your service in Designer. Look for the **result** output string.

Check Your Understanding

1. What is the difference between a POST and a PUT operation?
2. Why wasn't it necessary to perform any mapping when dragging the pre-created services into the generated REST services?
3. Where did the initial list of books come from?
4. Why was it necessary to specify credentials when invoking **pub.client:http** in the extra credits section?

EXERCISE 16:
MESSAGING USING JMS

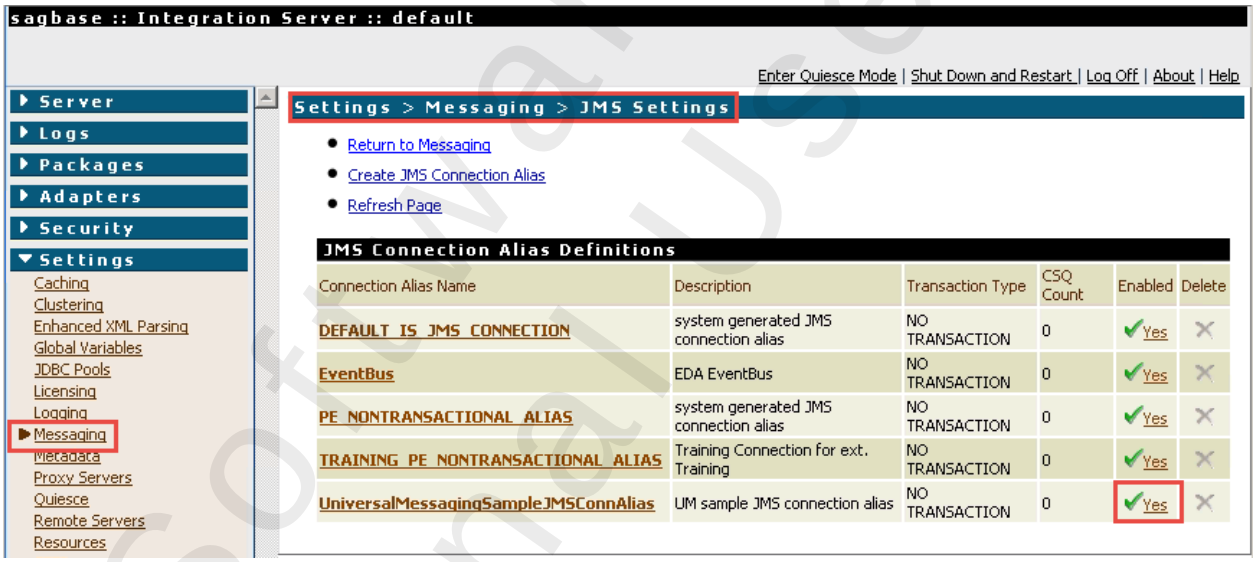
Overview

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.

You will create a document type (**acme.docs:Validation**) and publish it using a JMS send service.

Steps

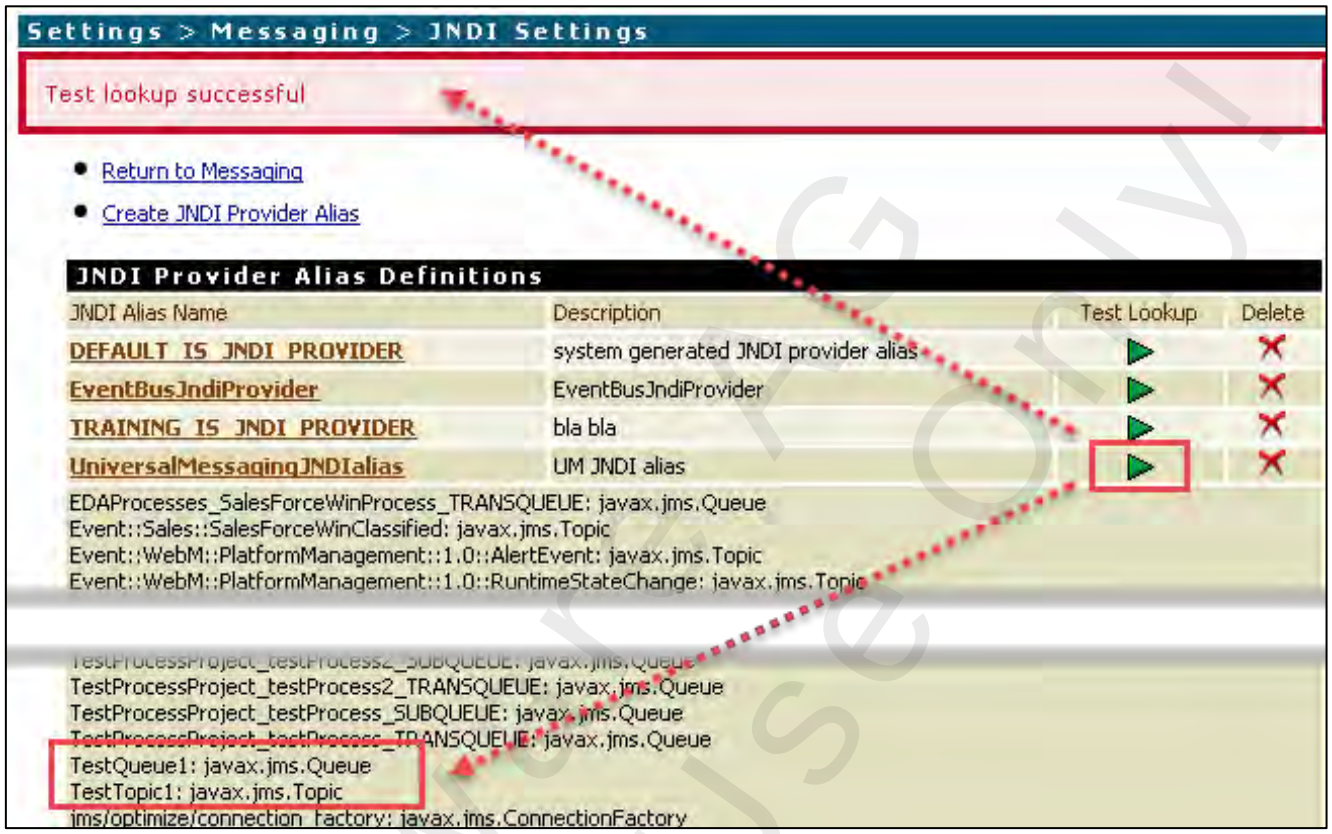
1. Login to the IS Administrator UI (http://localhost:5555) with credentials of **Administrator** | **manage**.
2. In **Settings --> Messaging --> JMS Settings**, enable the **UniversalMessagingSampleJMSConnAlias** connection alias to a Universal Messaging realm. This is our JMS provider for this exercise.



Note: JMS Connection Alias **UniversalMessagingSampleJMSConnAlias** uses JNDI Connection Alias **UniversalMessagingJNDIalias** to access Universal Messaging.

3. In **Settings --> Messaging --> JNDI Settings**, on the **UniversalMessagingJNDIalias** row, click the green triangle to view the items available in this JMS Provider.

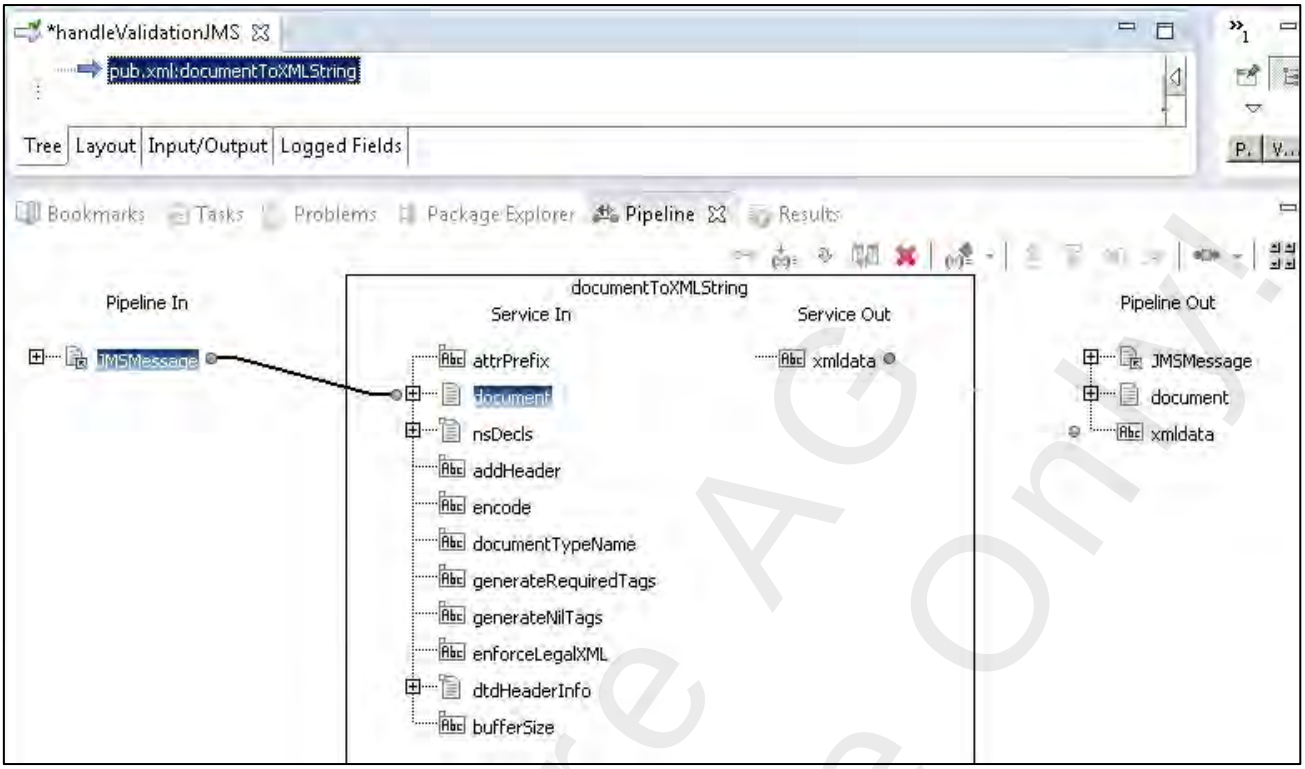
   Note: We will use the **TestTopic1** topic in this exercise.



4. In Designer, add a Flow service named **acme.work:handleValidationJMS** to your **Acme** package.
   This service will be the handling service for the JMS trigger.

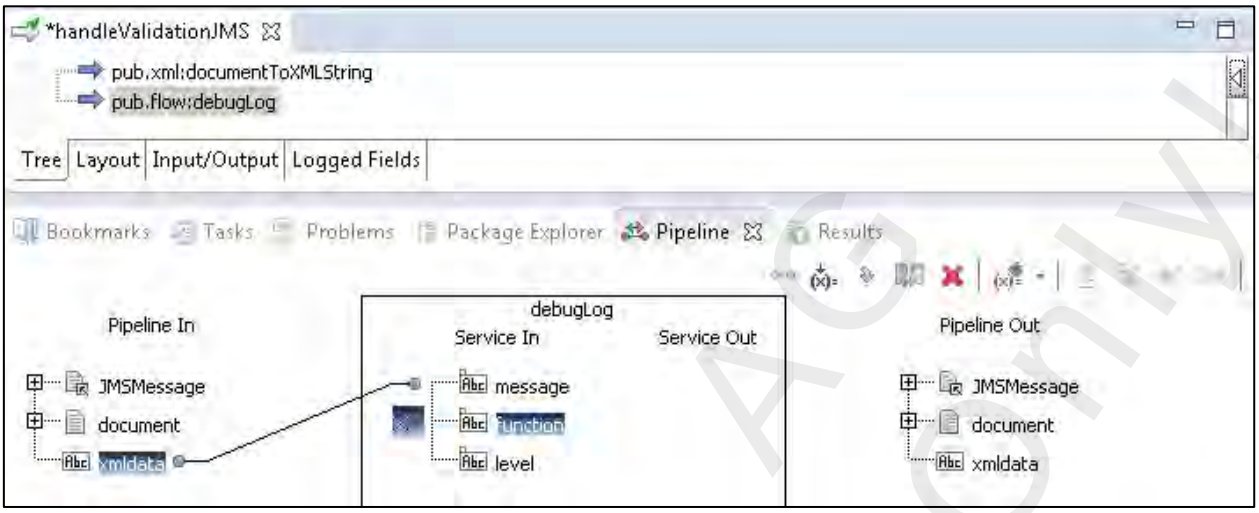
   On the Input/Output tab, to the right of the **Specification Reference** field, click the ... button. Browse for and select the **WmPublic** package's **pub.jms:triggerSpec**.

5. In the Tree tab of the service editor, add a **pub.xml:documentToXMLString** step and link **JMSMessage** to **document**.



Note: pub.xml:documentToXMLString is being done to convert the JMS message to XML so that you can display it in the IS log (for testing purposes). In a real solution you will map (link) the payload from with **JMSMessage\body** into an appropriate target (e.g. pipeline variable represented by an IS Document Type reference that matches the payload format).

6. Next add an invocation of **pub.flow:debugLog** to the service and map **xmldata** to **message**. Set the value of **function** to + + + + + (or some other characters that will catch your attention) to make the message more visible in the IS log.

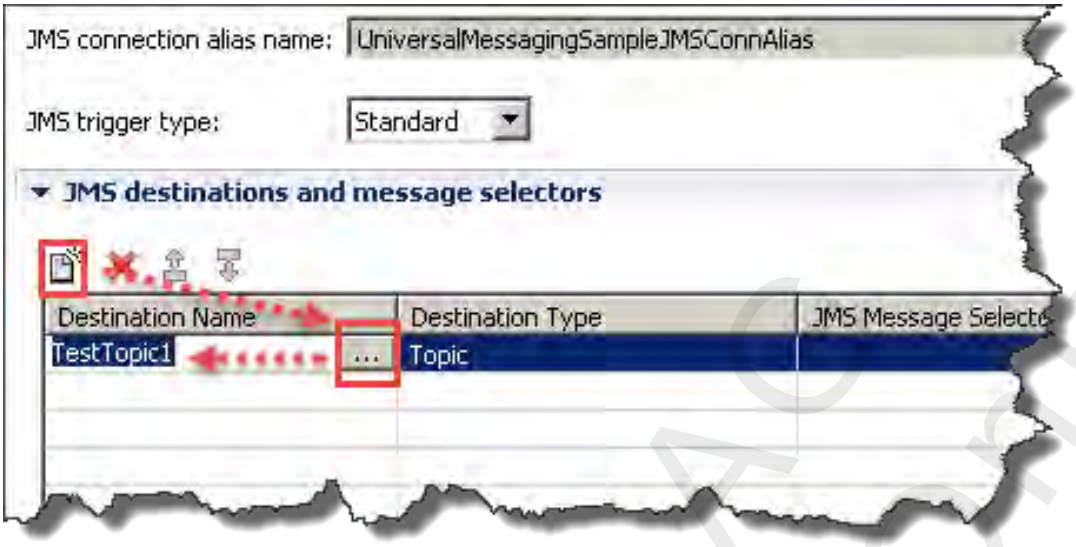


Save your Flow service.

7. Create a JMS Trigger called **acme.work:listenForValidationJMS:**
   a. Select the **JMS connection alias name** to be the **UniversalMessagingSampleJMSConnAlias** you enabled earlier in this exercise.

   b. Configure the triggers **JMS destinations and message selectors** by doing the following:
      i. Click the **Insert destinations** button .
      ii. In the first row, select the box under **Destination Name** and then choose the (…) button.

c. Select the **TestTopic1 (TestTopic1)** topic. Leave all other fields empty.



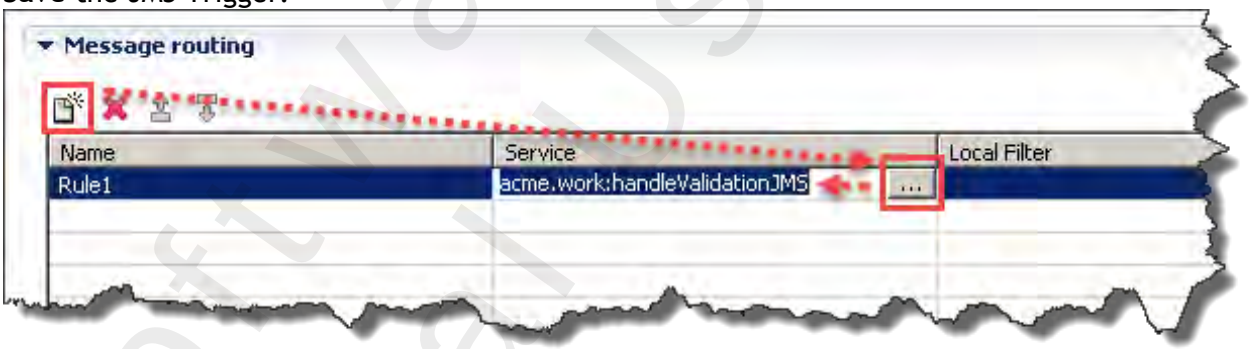
Configure the JMS Trigger **Message routing** by doing the following:

i. Selecting the **Insert routings** button .

ii. In the first row, select the box under **Service** and then choose the **(…)** button.

iii. Select the service named **acme.work:handleValidationJMS**

iv. **Save** the JMS Trigger.



8. In the **Acme** package, create an **acme.docs:Validation** Document type containing one String field named **Valid**. **Save** your work.

9. To test your JMS Trigger immediately, right-click Document type **acme.docs:Validation** in the Package Navigator view and select **Run As** -> **Publish as JMS Message**.
In the panel that appears, specify the following input:

a. **JMS connection alias name** = **UniversalMessagingSampleJMSConnAlias** (browse for it)

b. **Destination name** = **TestTopic1** (browse for it)

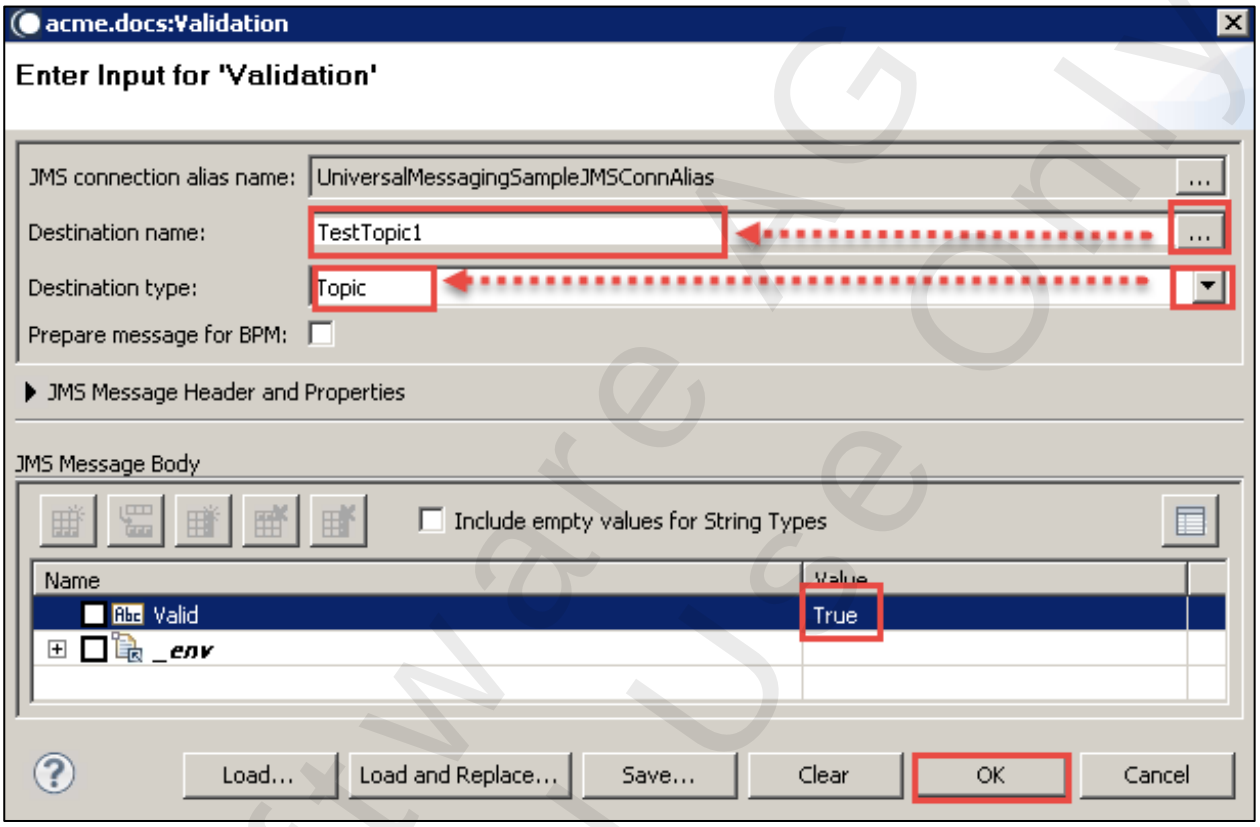
c. **Destination type** = **Topic** (set automatically)

d. **Prepare message for BPM** = *leave unchecked*

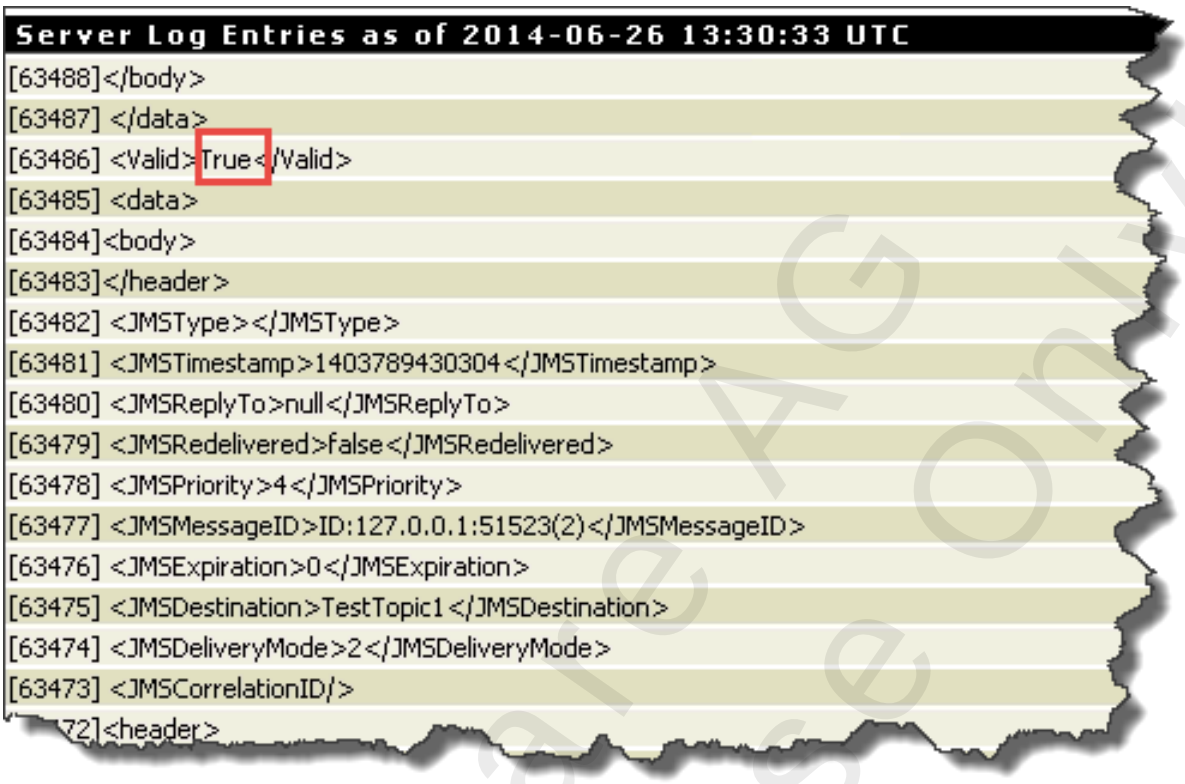
e. Do not provide any value in the **JMS Message Header and Properties** section.

f. In the section titled **JMS Message Body**, set Valid to: **True** (Note: "Valid" is just a field in the Document. You can type anything in here for our simple testing)

Click **OK** to send the data as a JMS message to be received by the JMS Trigger.

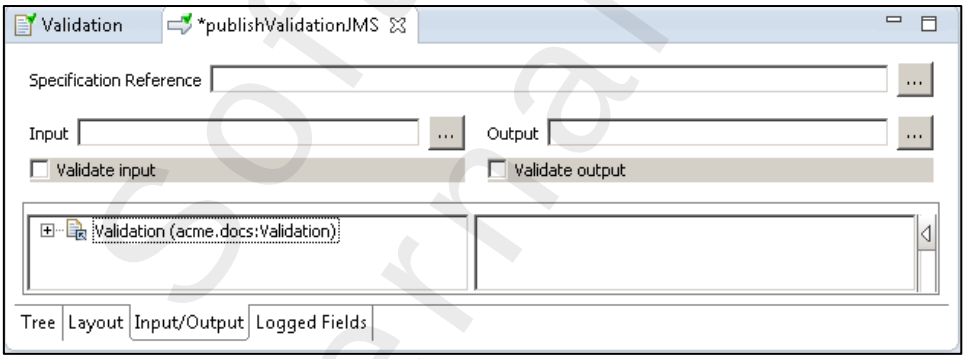


10. Use the IS Administrator UI to ensure the value **True** is shown, embedded in the JMS Message, in the IS Server Log:

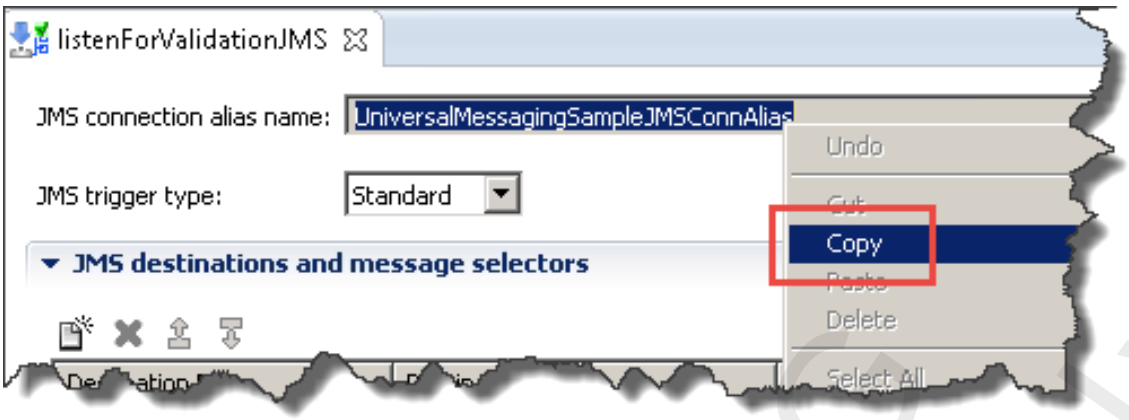


Add another Flow service called **acme.work:publishValidationJMS** in your **Acme** package.

In the Input of this service, add a document reference to your **Acme** package's **acme.docs:Validation** doctype. Name it **Validation.**

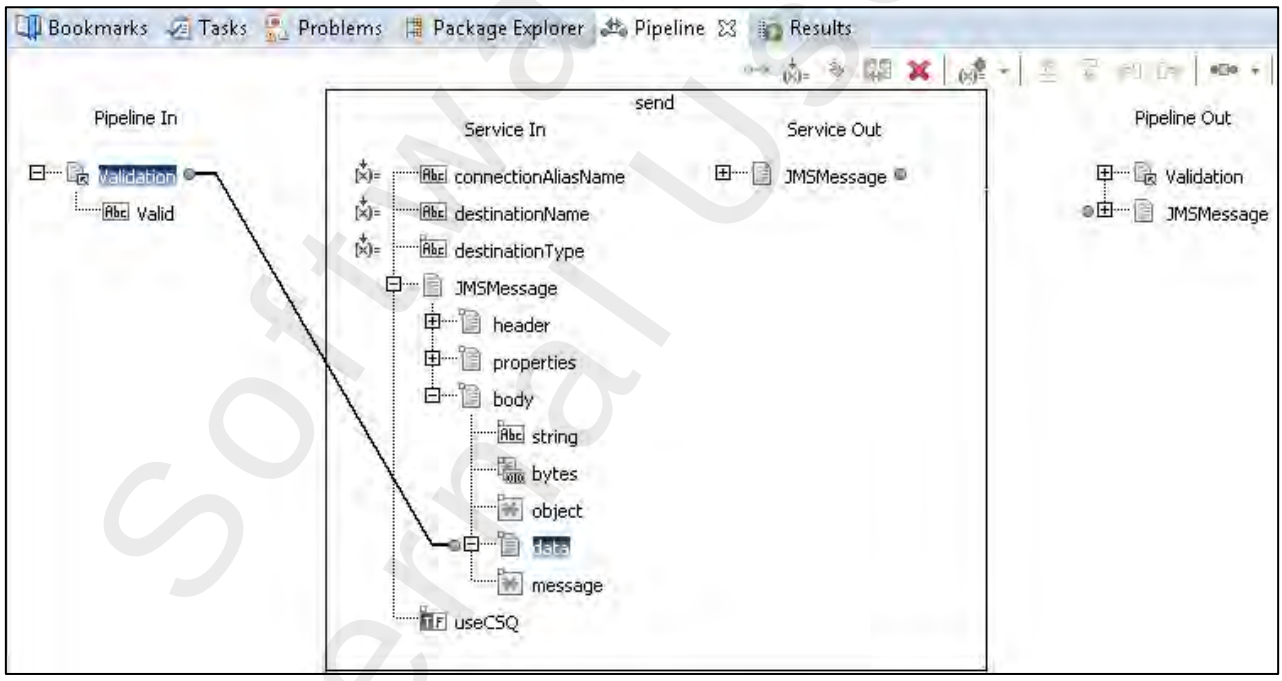


11. Copy the JMS Connection Alias name, **UniversalMessagingSampleJMSConnAlias,** from the top of your JMS Trigger definition into the clipboard.

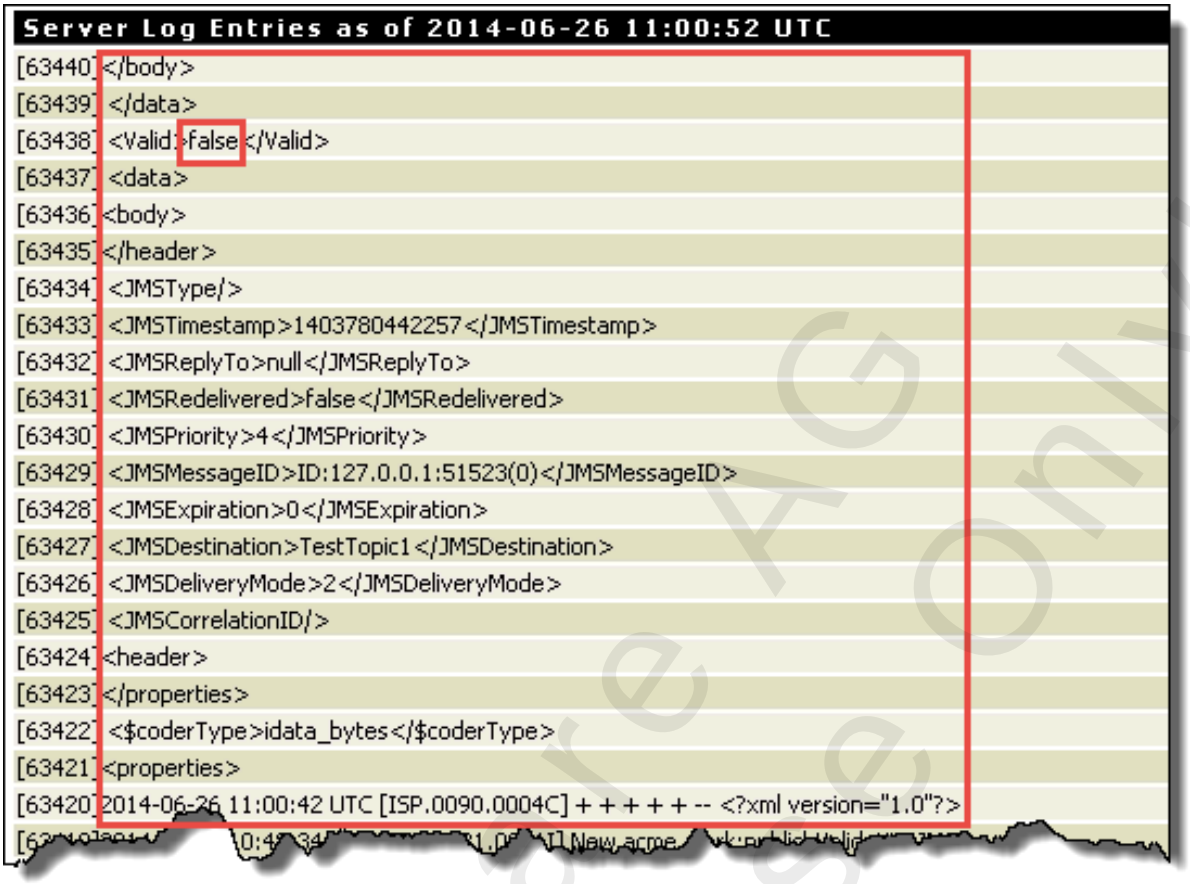


12. In the new service **acme.work:publishValidationJMS**, add a step to invoke the **WmPublic** package's **pub.jms:send** service. Then set / link the following:
    a. Set **connectionAliasName** to **UniversalMessagingSampleJMSConnAlias** (paste this value)
    b. Set **destinationName** to **TestTopic1** (Type it or copy it from your JMS Trigger and paste it here. Just make sure you get the spelling and case correct)
    c. Set **destinationType** to **TOPIC** (select value from dropdown list)
    d. Link **Validation** (Pipeline In) to **JMSMessage/body/data**

    Save your work.



13. Run the **publishValidationJMS** service. Enter some text into the Valid field.
14. Use the IS Administrator UI to ensure that the JMS Message just published appears in the IS server log (in XML format). This proves that the JMS trigger fired and passed the JMS Message to the handling service which converted it into XML and wrote it to the IS log.



Check Your Understanding

1. What is a Topic versus a Queue?
2. Name the five JMS message types?

This page is intentionally left blank

EXERCISE 17:
WEBMETHODS MESSAGING

Overview

In this exercise, you will create a publishable document type, a handling service and a subscribing webMethods Messaging Trigger. Then you create a service to publish the document.

We will use Universal Messaging (UM) as the messaging provider in this exercise however the components involved are the same if Broker is used.

Note: The publishable IS Document Type defines the messaging provider Connection Alias.

Steps

First, create the subscribing components: Document Type, handling service, and subscriber:

1. In the **Acme** package, edit the **acme.docs:Validation** Document type.

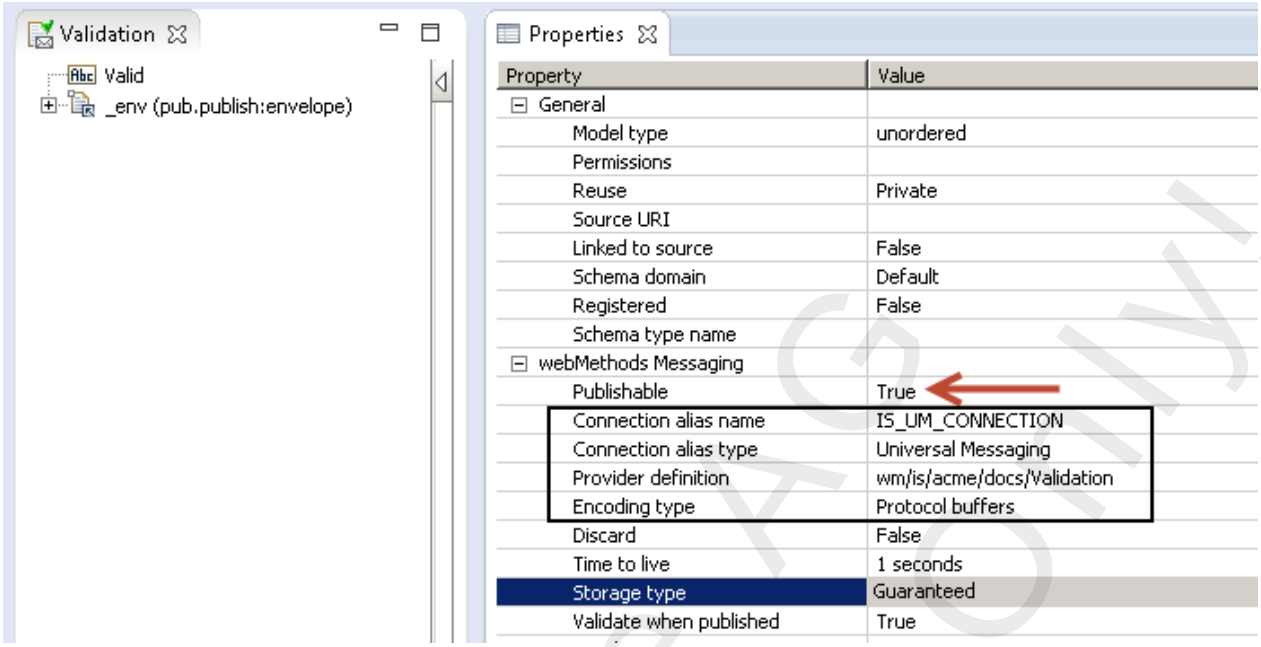
   Make this document publishable to a webMethods Messaging provider by setting the **Publishable** property for the document to **True** in the Properties view.

   Also select **IS_UM_CONNECTION** as webMethods Messaging **Connection alias name**.
   Save the Document type.

   In the Properties view, note that saving sets the **Connection alias type** to **Universal Messaging** and the **Provider definition** is filled with the channel name that will be created in the Universal Messaging realm (**wm/is/acme/docs/Validation).**

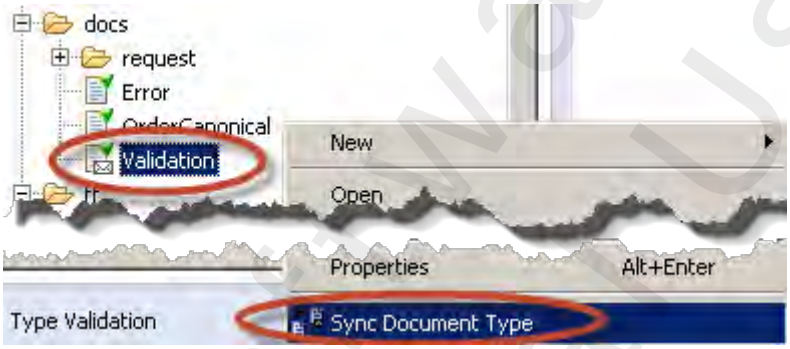
   Set the **Encoding type** to **Protocol buffers.**

   An envelope (_env) is also added to the document structure when you save the document type.



2. The publishable document type has to be synchronized with the Provider (Universal Messaging). Synchronizing will create/update the channel definition in the UM realm.

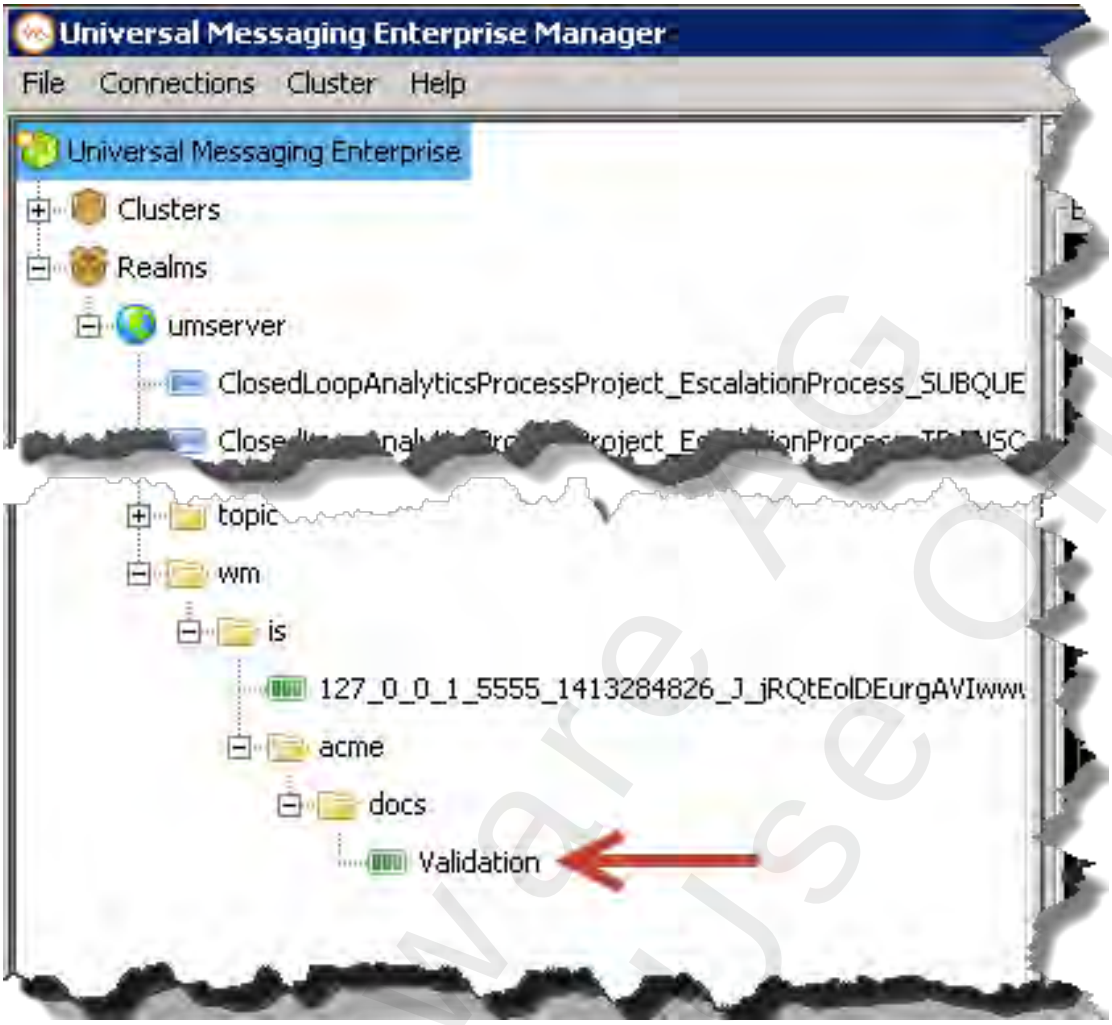
   Right click on the **Validation** document and select **Sync Document Type**.



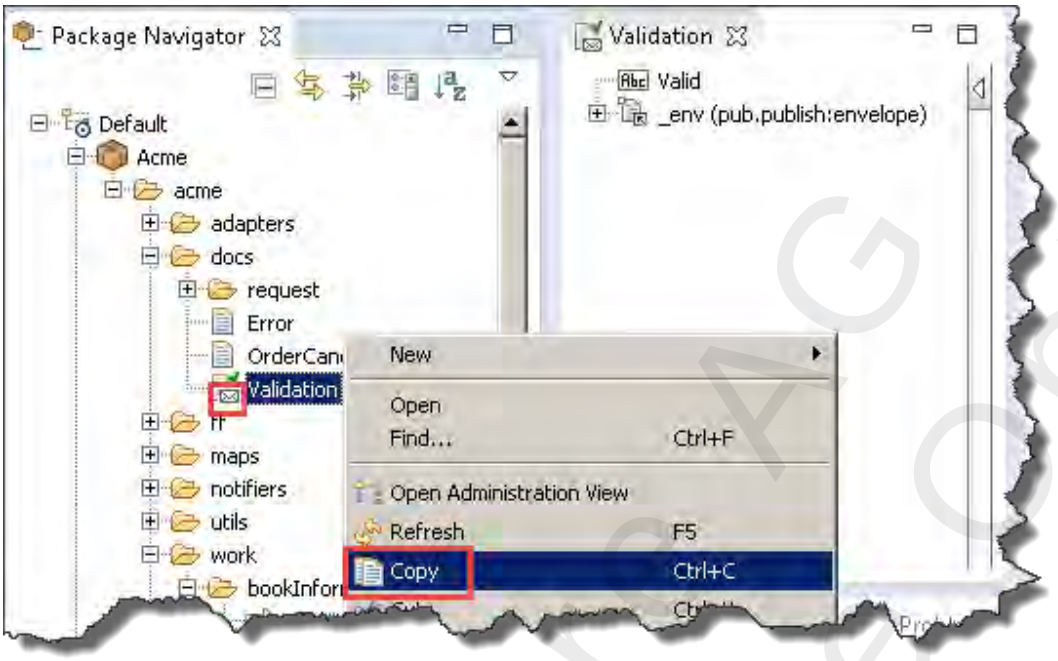
3. Open the Universal Messaging Enterprise Manager.
   Click the Windows **Start --> All Programs --> Software AG --> Administration --> Universal Messaging Administration 9.7 --> umserver --> Enterprise Manager**

   Expand Realms and umserver. You will see the channel called Validation that was created in

wm\is\acme\docs.



4. Go back into Designer. In the Package Navigator view, copy the **acme.docs:Validation** document type to the clipboard.

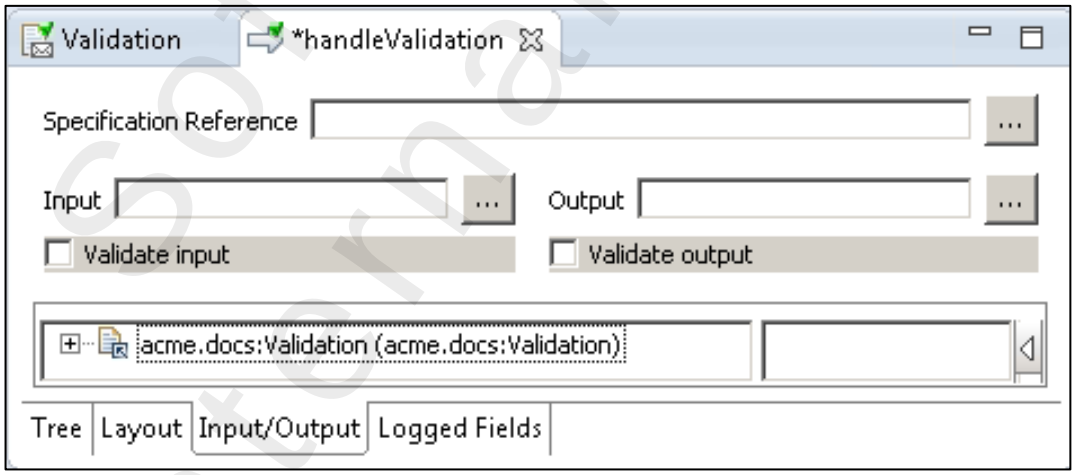


5. Create a Flow service named **acme.work:handleValidation** in your **Acme** package. In the Input/Output tab:

   a. Use Paste to create a document reference to your document **acme.docs:Validation** as service input.

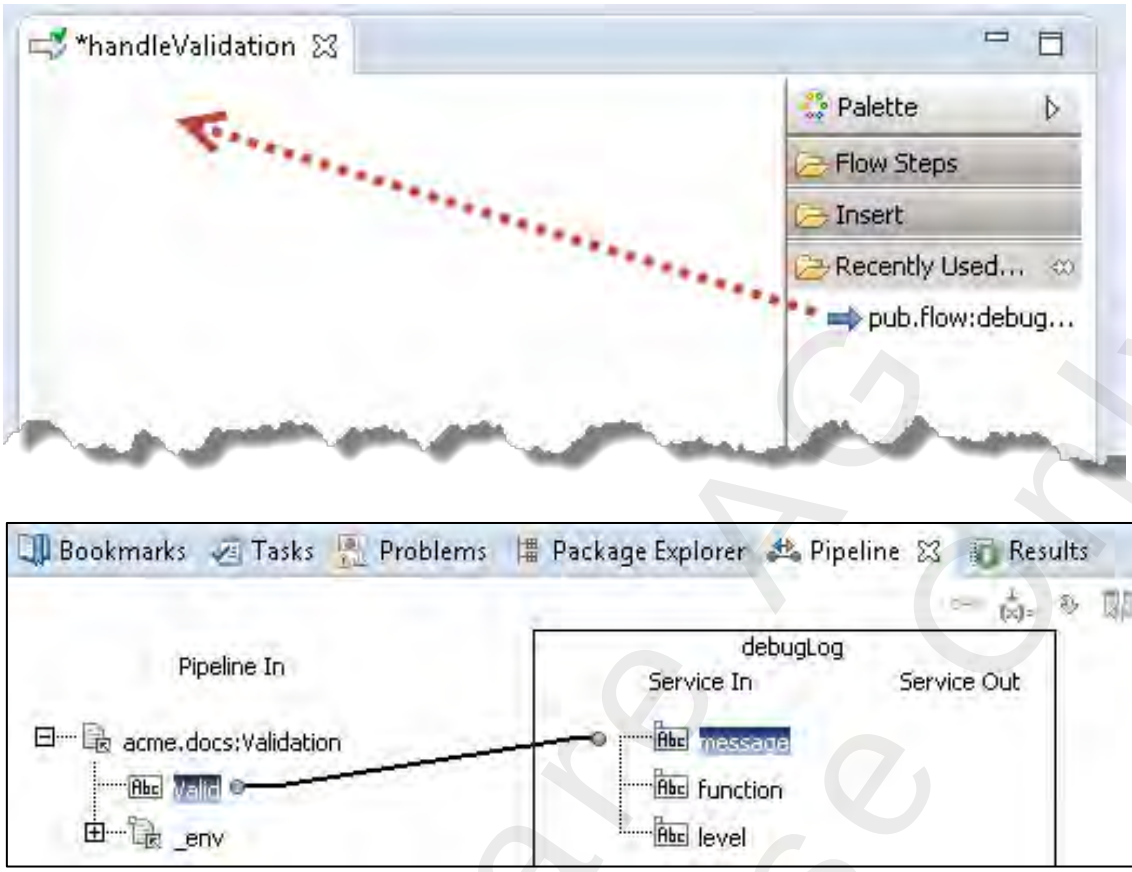
   b. Again, use Paste to set the name of the document reference to be the fully-qualified name of the document type: **acme.docs:Validation.**

   Note: Use the Copy and Paste technique to avoid typographical errors!



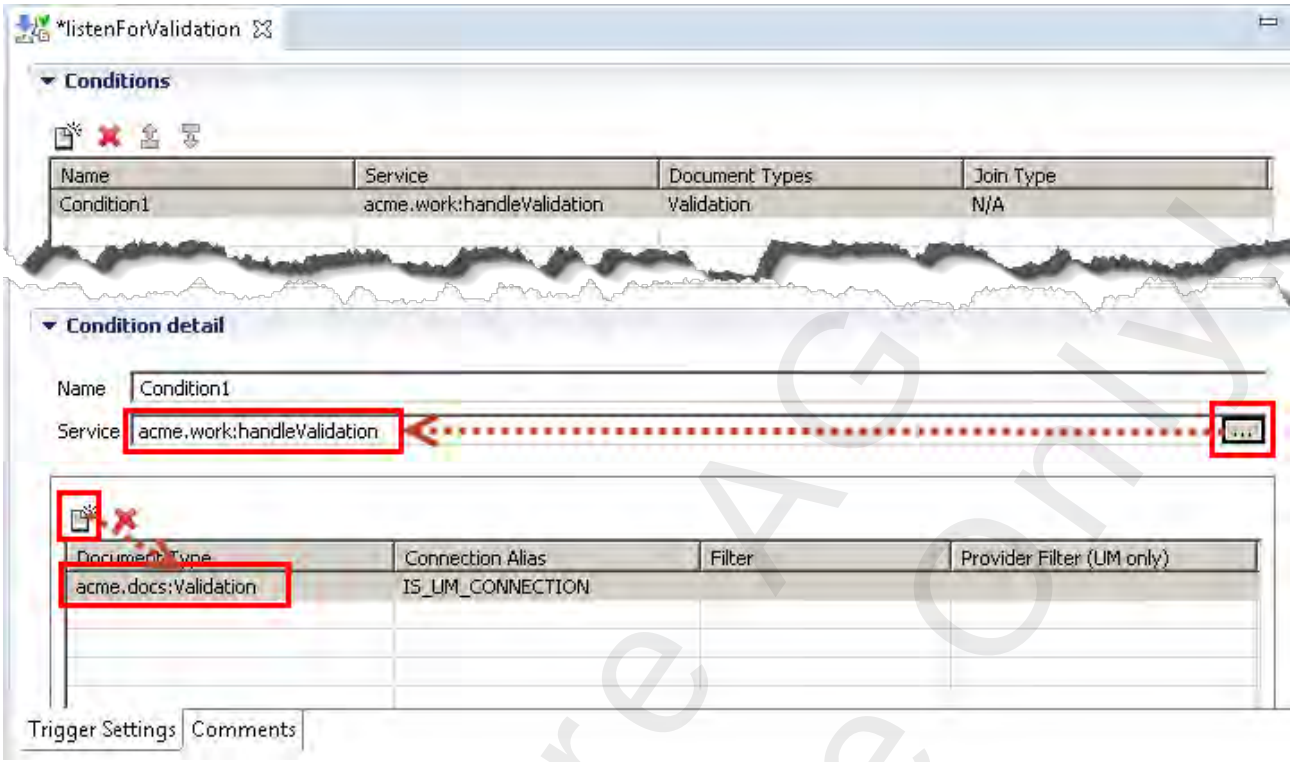
6. In the Tree tab of the service editor, add a **pub.flow:debugLog** step and link **acme.docs:Validation/Valid** (Pipeline In) to **message**. **Save** your Flow service.

   Tip: You can drag pub.flow:debugLog from the list of Recently Used services in the Palette.



Note: in a real handling service you would create logic to parse your incoming document and do something with it (e.g. insert the data into a database, etc…)

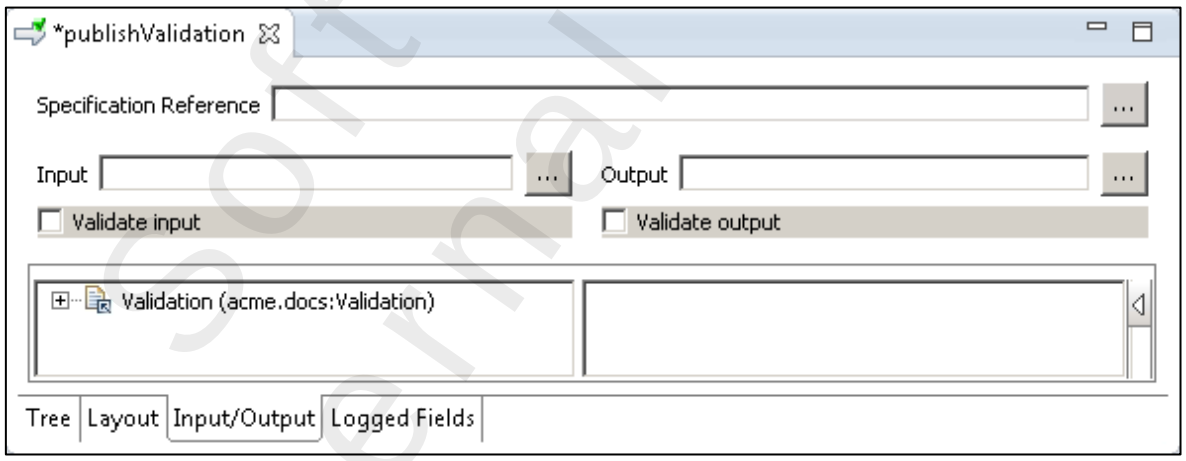
7. Create a **webMethods Messaging Trigger** named **acme.work:listenForValidation**. Configure the Trigger **Condition details** as follows:
   a. **Name** = **Condition1** (don't change)
   b. **Service** = **acme.work:handleValidation**
   c. **Document type** = **acme.docs:Validation**
   d. **Connection Alias = IS_UM_CONNECTION** (gets filled automatically)
   e. **Filter** = ***leave this empty***
   f. **Provider Filter (UM only)** = ***leave this empty***



8. Next, create a publishing service to publish the publishable document.

   To do so, add a Flow service named **acme.work:publishValidation** to your **Acme** package.

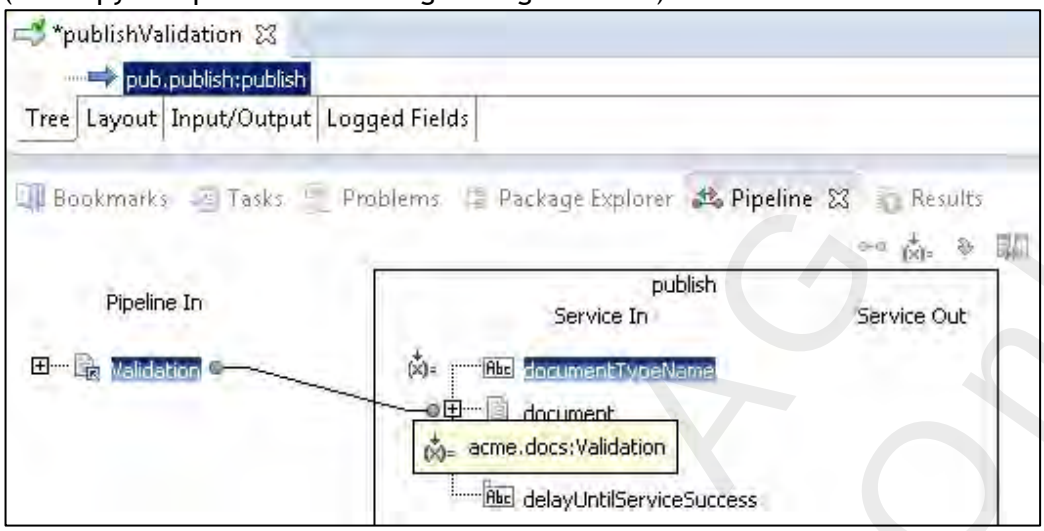
   Add a document reference to **acme.docs:Validation** doctype as input of this service. Name it **Validation.**



9. In the service, add a step to call the **WmPublic** package's **pub.publish:publish** service. Perform mapping in the Pipeline view as follows:

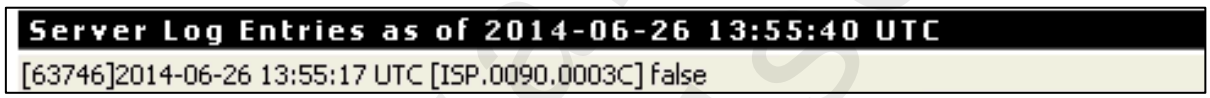
a. Link **Validation** (Pipeline In) to **document**

b. Set **documentTypeName** to **acme.docs:Validation**
(use copy and paste from Package Navigator view)



10. Save and run the **publishValidation** Flow service.

Enter something for the value of the Valid field (e.g. true, false, helloworld…….whatever…..). The value you entered should be shown in the IS Server Log:



Check Your Understanding

1. What happens when a document is made publishable?
2. What two objects are required for publishing?
3. What two objects are required for subscribing?
4. What name should you give to the Document(s) on the Input of the trigger handling service (e.g. handleValidation)?

This Page intentionally left blank

EXERCISE 18:
CREATE JDBC ADAPTER SERVICES

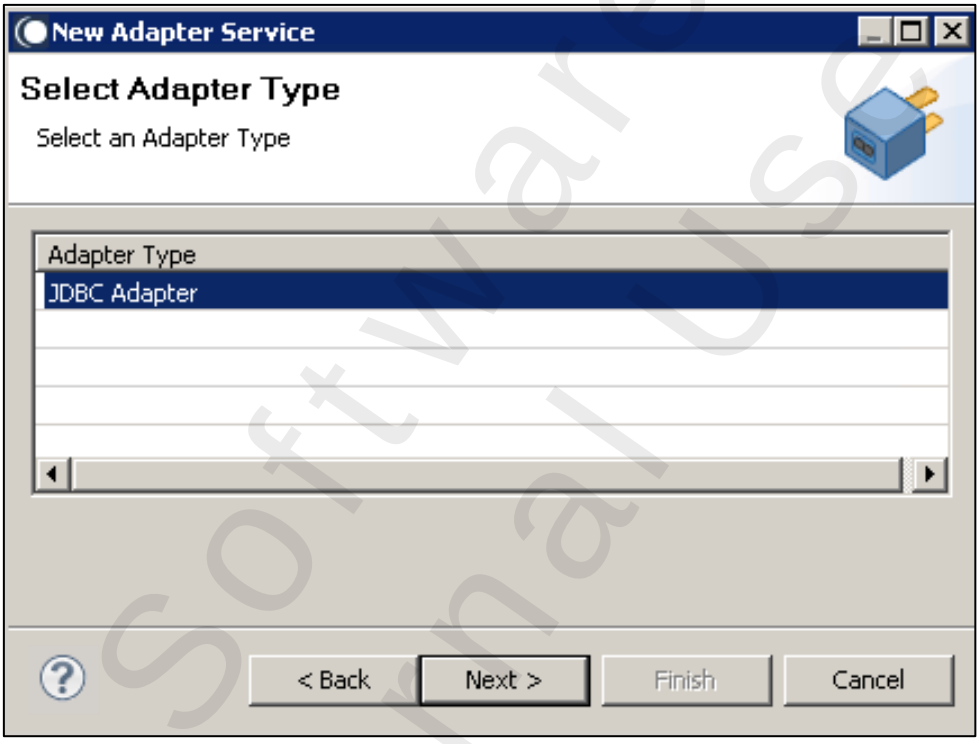
Overview

In this exercise, you will create insert and select JDBC Adapter services.

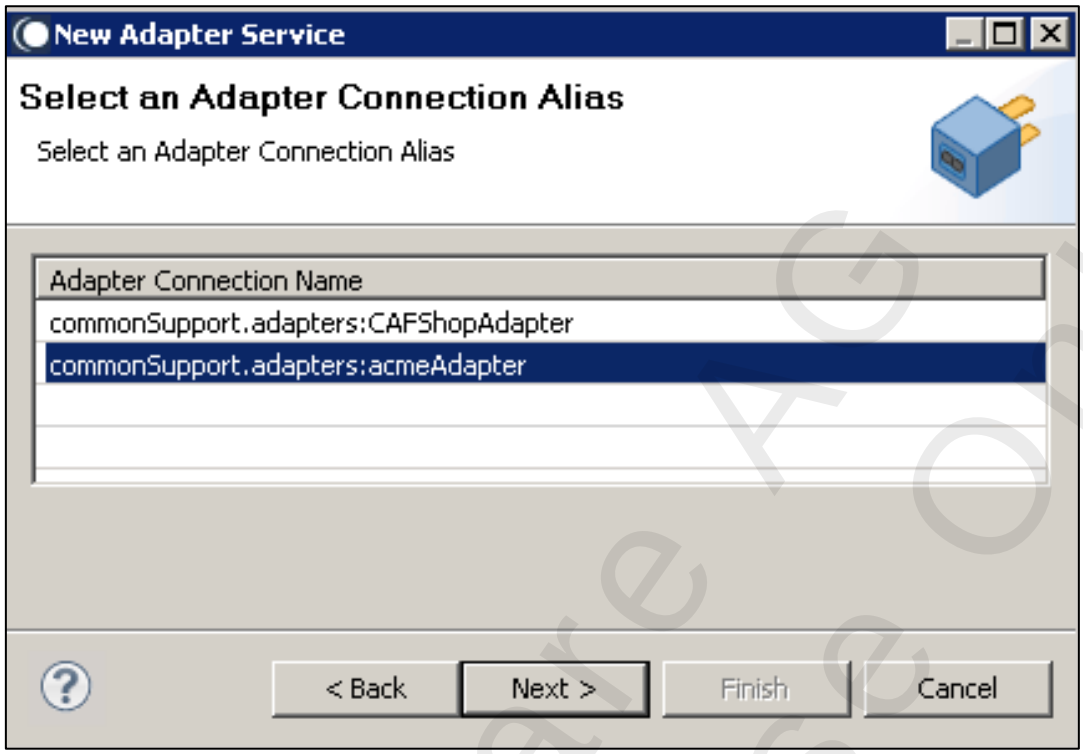
You will combine these services with a parent Flow service to insert the OrderCanonical data into the Order Management database tables.

Steps

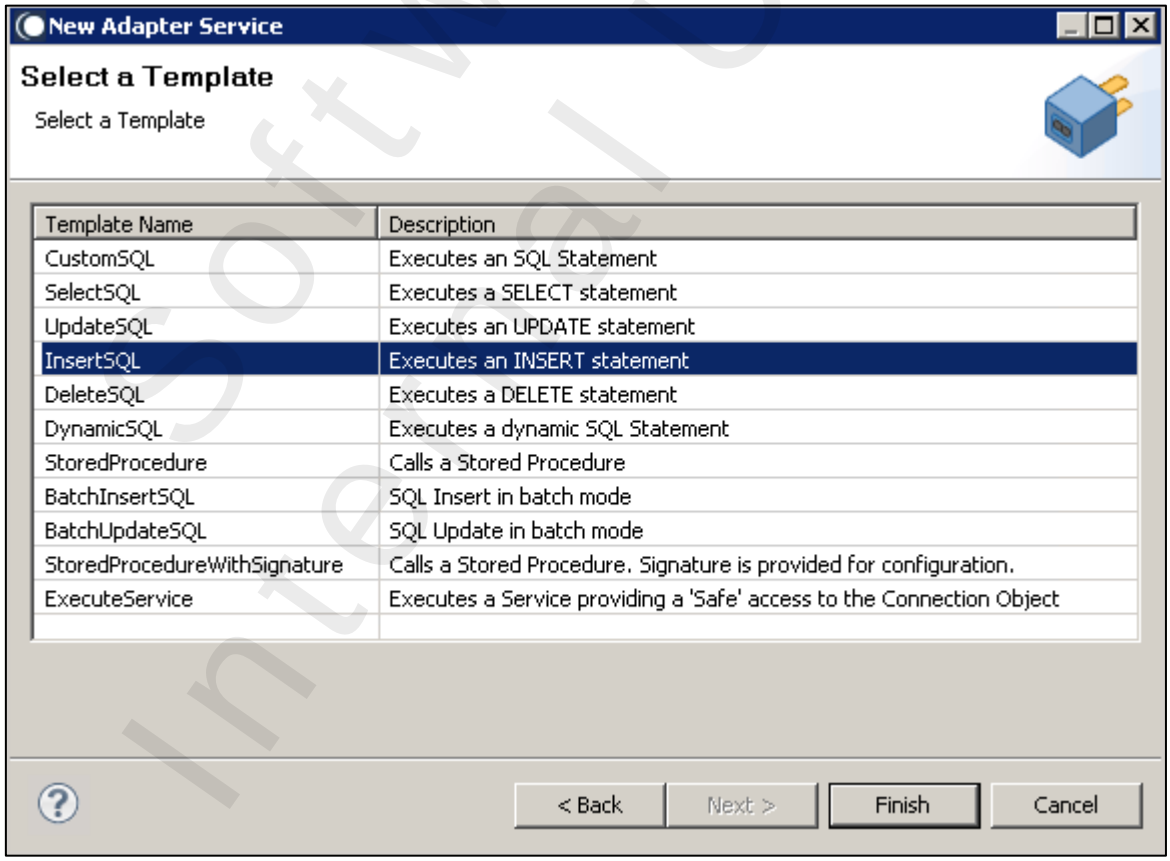
1. In Designer add a new Adapter Service called **acme.adapters:insertOrderHeader** to your **Acme** package.
2. For the Adapter Type select **JDBC Adapter** and click **Next**.



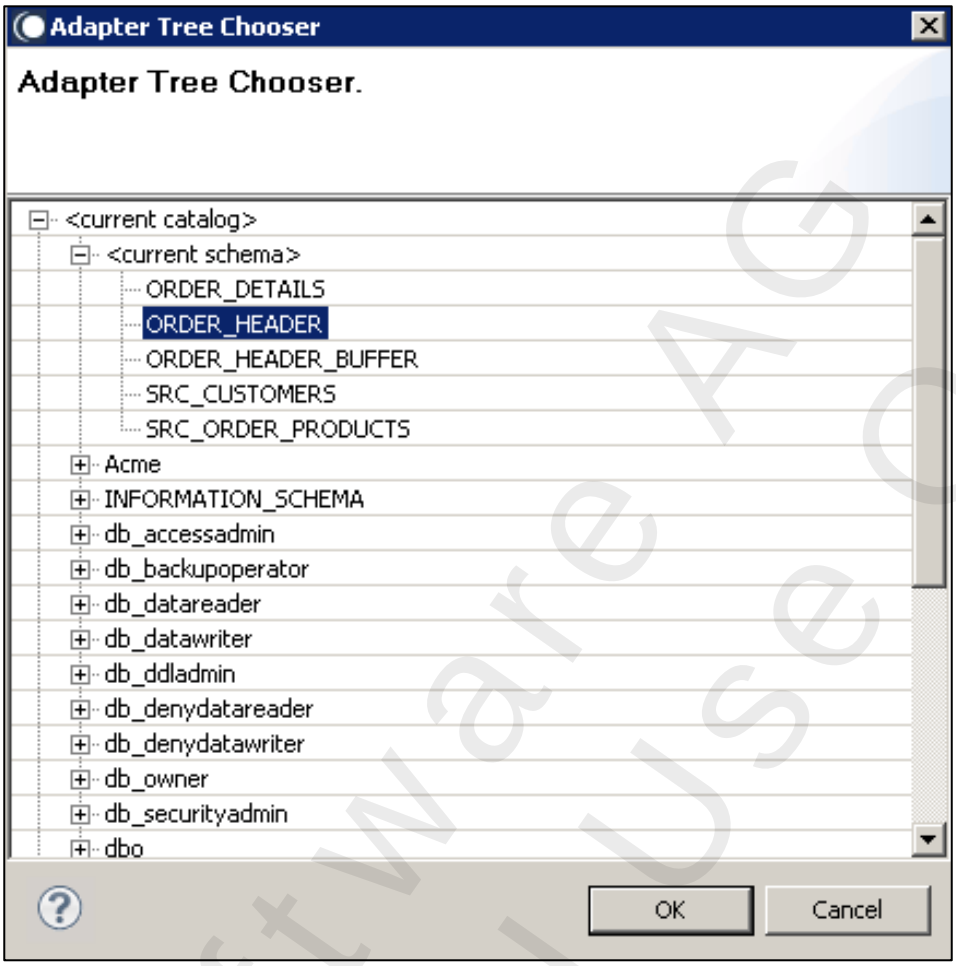
3. Then choose the existing **commonSupport.adapters:acmeAdapter** as Adapter Connection Alias and click **Next.**



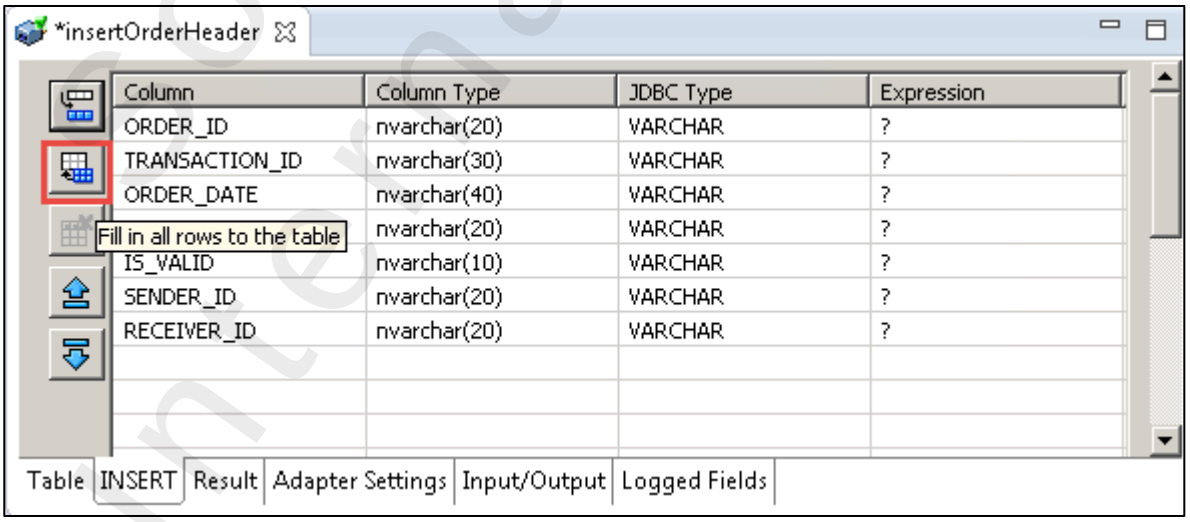
4. Finally select the **InsertSQL** as the template and click **Finish**.



5. Configure the generated JDBC Adapter Service **insertOrderHeader** as follows:
   a. On the **Table** tab, click into the first row of column Table Name and use the (…) button to select **<current catalog>.<current schema>.ORDER_HEADER**. Click **OK.**



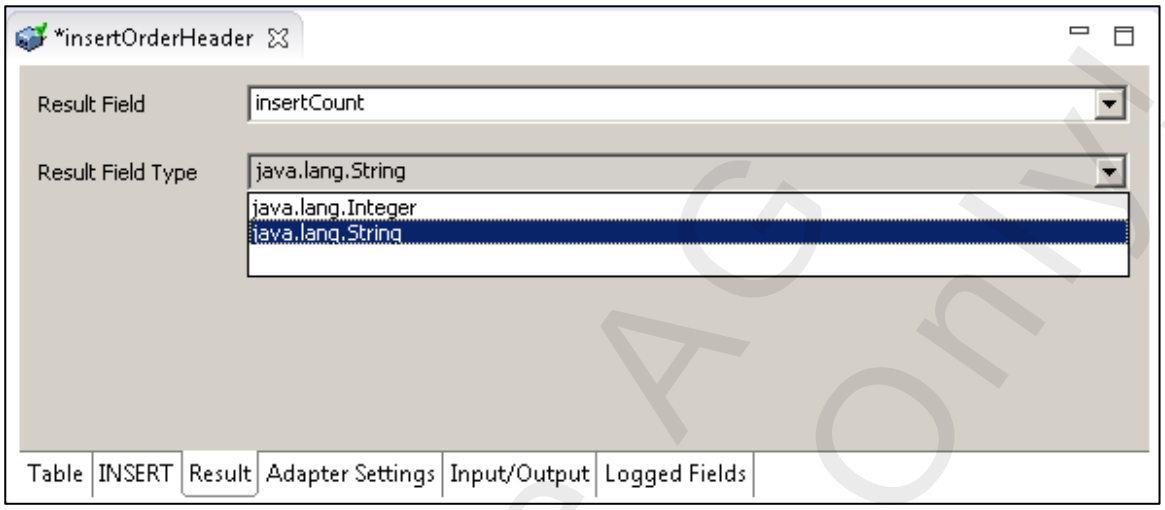
   b. On the **INSERT** tab, click the **Fill in all rows to the table** button.



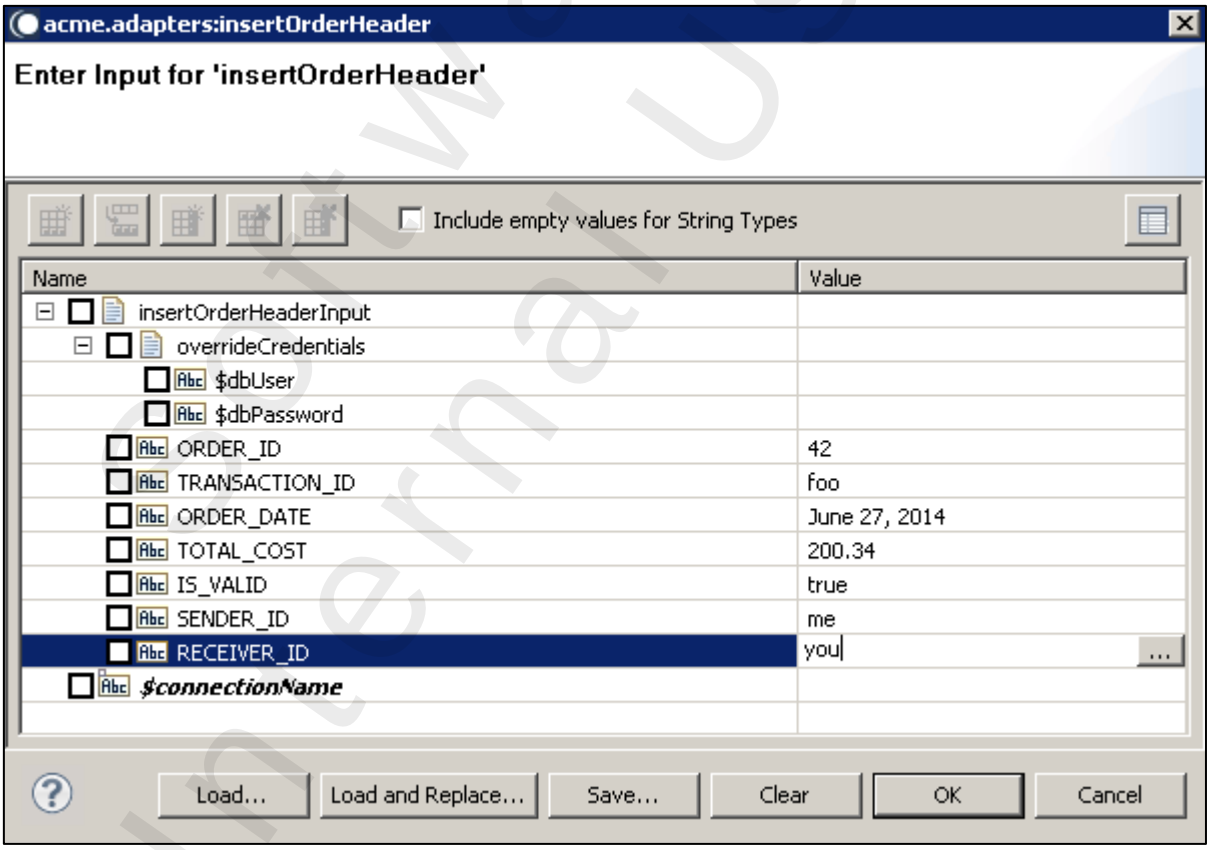
c. On the **Result** tab, set:

i. Result Field: **insertCount**

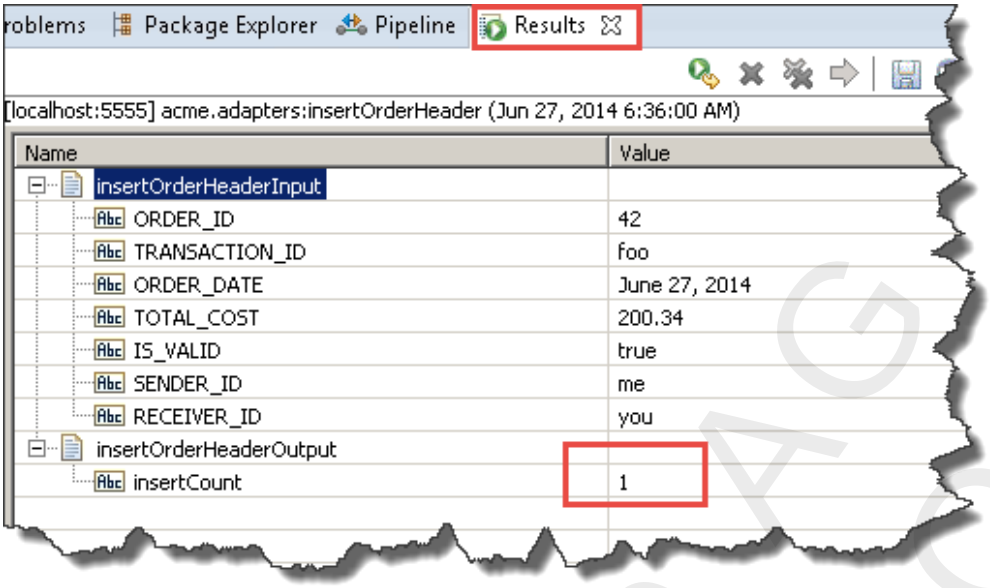
ii. Result Field Type: **java.lang.String**.



6. Save and run the **insertOrderHeader** Adapter Service from the Package Navigator view. Insert data for every input field except for **overrideCredentials** and **$connectionName.** Click **OK**.



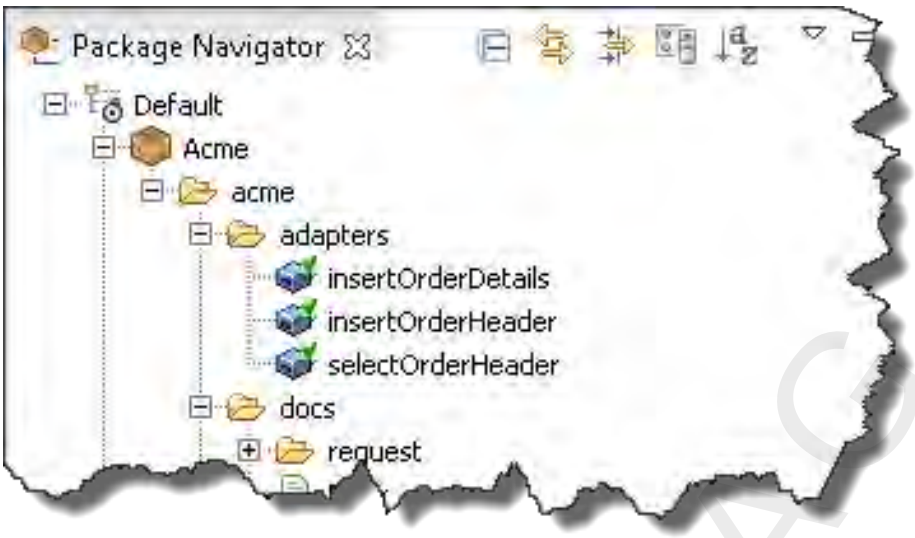
In the Results view, confirm that the **insertCount** returns with a value of **1**.



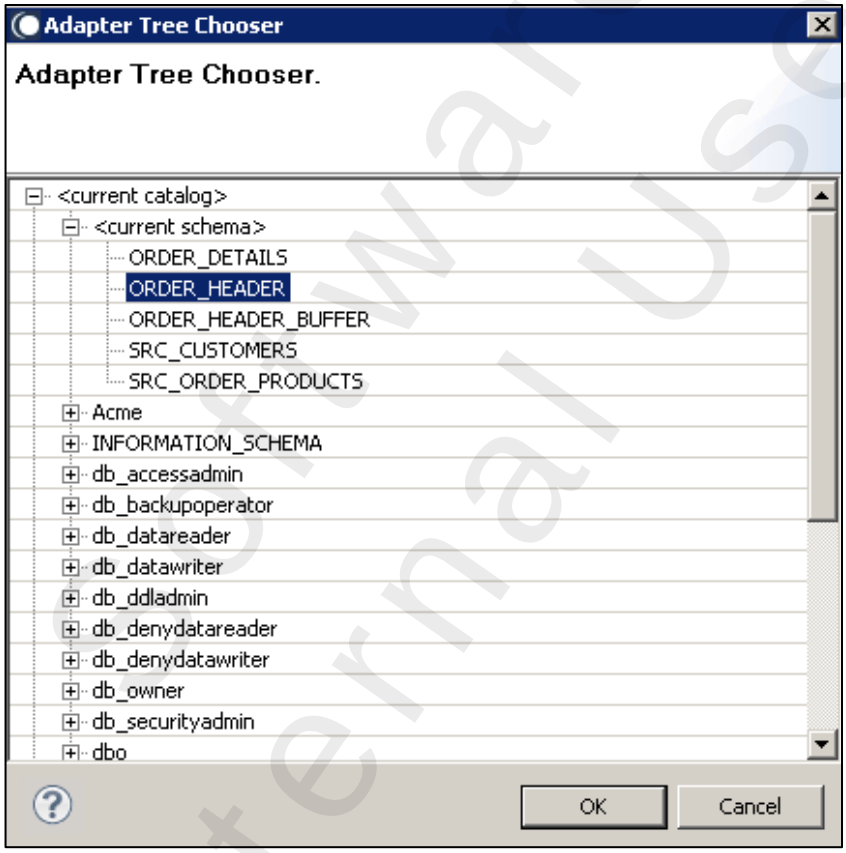
7. Like you did in step 1, add another Adapter Service to package **Acme** named **acme.adapters:insertOrderDetails**.

   Specify this will be an **Adapter Service** of type **JDBC Adapter** using Adapter Connection **commonSupport.adapters:acmeAdapter** and select the **InsertSQL** template.

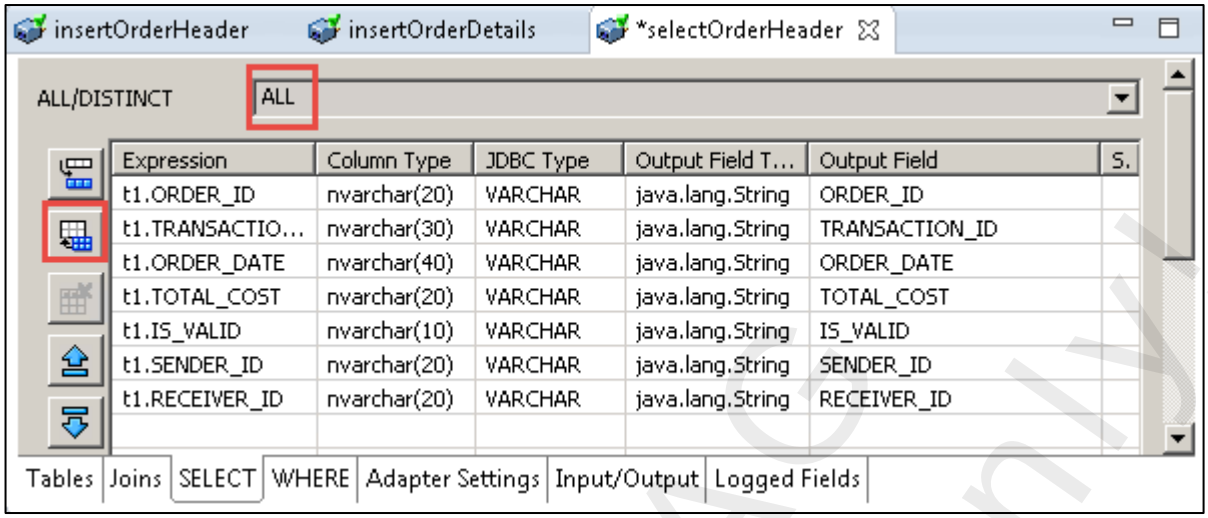
8. Configure the generated JDBC Adapter Service **insertOrderDetails** as follows:
   a. On the **Table** tab, select **<current catalog>.<current schema>.ORDER_DETAILS**.
   b. On the **INSERT** tab, click the **Fill in all rows in the table** button.
   c. On the **Result** tab, set
      i. Result Field: **insertCount**
      ii. Result Field Type: **java.lang.String**.
9. Save and run the **insertOrderDetails** Adapter Service.
   Insert data for every input field except for **overrideCredentials** and **$connectionName.**
   Click **OK**.
   Confirm that the **insertCount** returns with a value of **1**.
10. Add a third Adapter Service in the **acme.adapters** folder named **selectOrderHeader**. Specify this will be an **Adapter Service** of type **JDBC Adapter** using Adapter Connection **commonSupport.adapters:acmeAdapter,** and select the **SelectSQL** template.



11. Configure the generated JDBC Adapter Service **selectOrderHeader** as follows:
    a. On the **Tables** tab, in the first row, browse for **<current catalog>.<current schema>.ORDER_HEADER** as the **Table Name**.

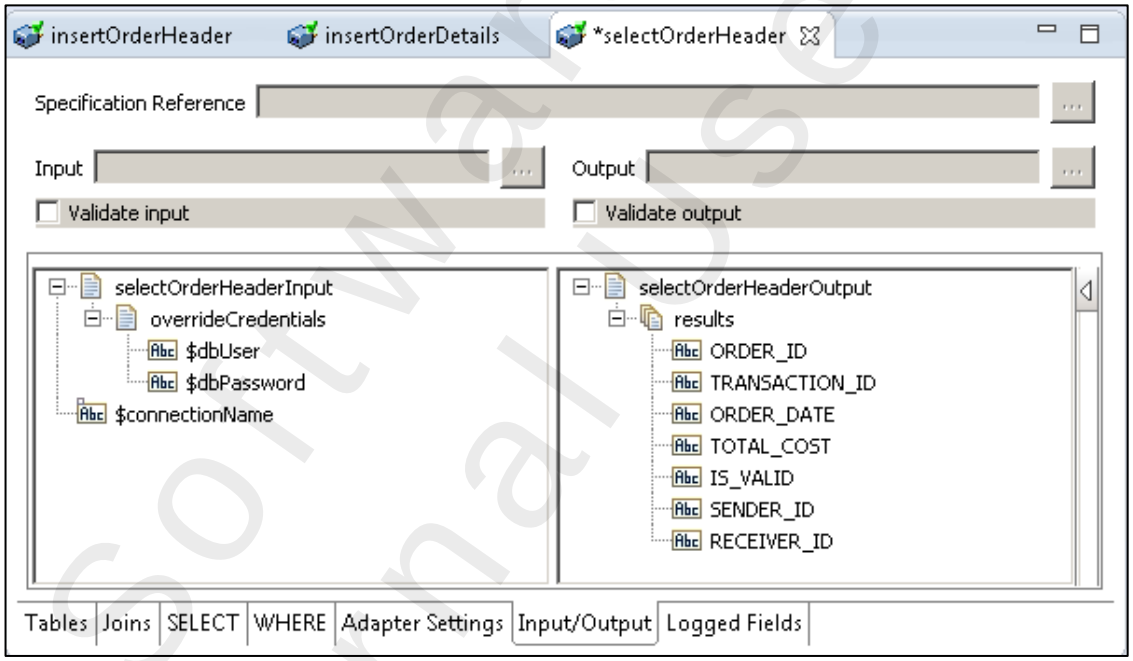


    b. Skip the **Joins** tab.
    c. On the **SELECT** tab, select **ALL** from the **ALL/DISTINCT** dropdown. Click on the **Fill in all rows to the table** button.

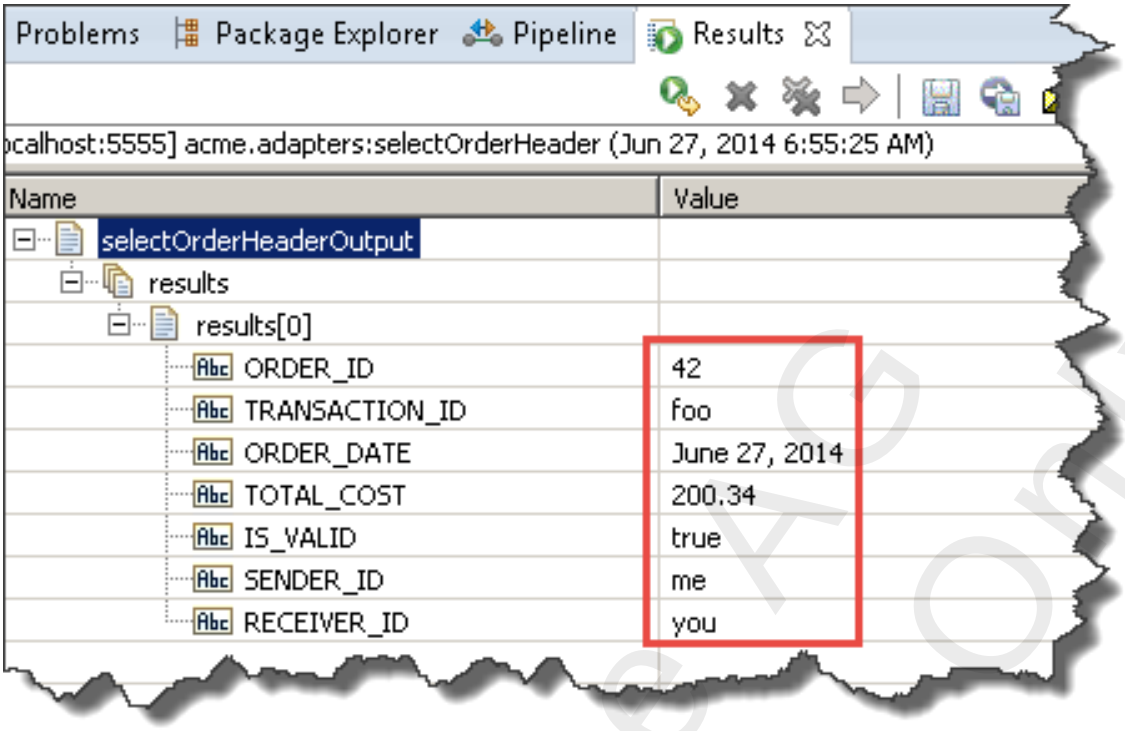


d. **Save** your work.

e. In the **Input/Output** tab, review the generated Document type. You should see the database columns that you selected under the results Document List.



12. Save and run the **selectOrderHeader** Adapter Service. Do not provide any inputs. Confirm that the database table was populated with your data from step 6.



13. Create a fourth Adapter Service named **acme.adapters:selectOrderDetails**. Specify this will be an **Adapter Service** which is of type **JDBC Adapter**, using Adapter Connection **commonSupport.adapters:acmeAdapter,** and select the **SelectSQL** template.

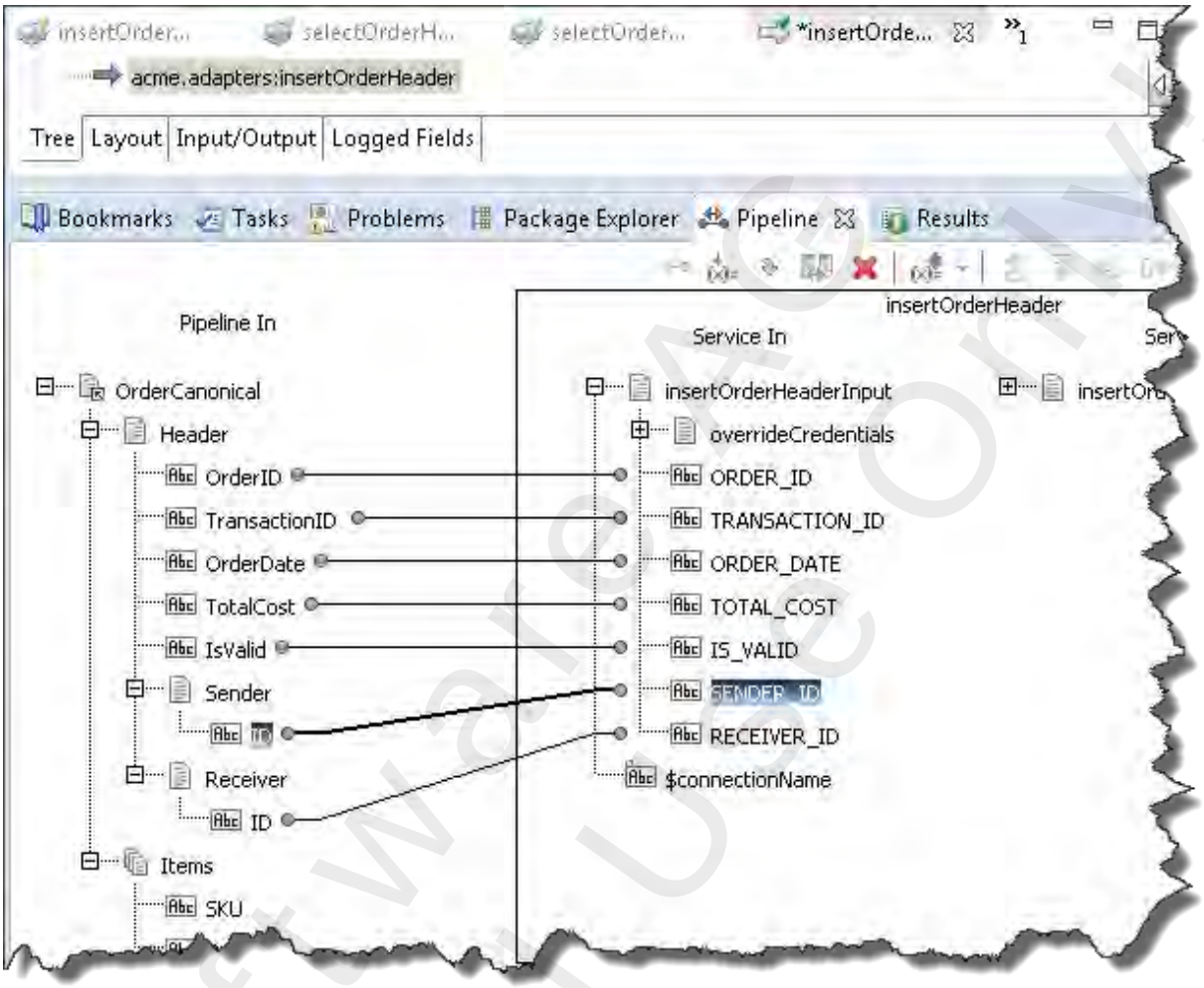
14. Configure the generated JDBC Adapter Service **selectOrderDetails** as follows:
    a. On the **Tables** tab, select Table Name **<current catalog>.<current schema>.ORDER_DETAILS.**
    b. Skip the **Joins** tab.
    c. On the **SELECT** tab, select **ALL**. Click the **Fill in all rows to the table** button**.**
    d. **Save** your work.
    e. In the **Input/Output** tab, review the generated Document type. You should see the database columns that you selected under the results Document List.

15. Save and run the **selectOrderDetails** service and confirm that the database table was populated with your data from step 9.

In the next couple of steps, you will create the service logic that is required to store a canonical order in the database. This means storing order header and order details belonging to this order.

16. In the **acme.utils** folder, create a Flow service called **insertOrderCanonical.**
    a. Set the input to be a Document Reference to **acme.docs:OrderCanonical** document, named **OrderCanonical**.
    b. Switch to the **Tree** tab and drag your **acme.adapters:insertOrderHeader** into your service.

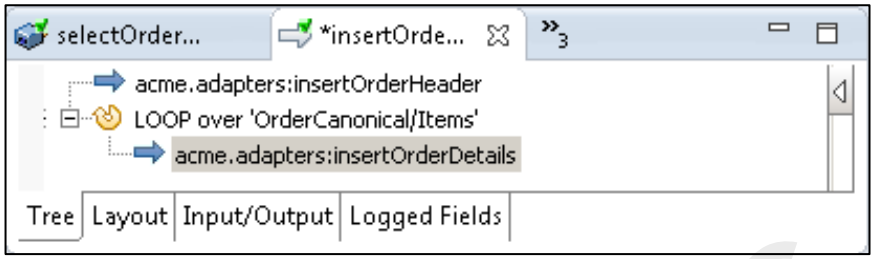
c. Use the Pipeline view to link all the fields from **OrderCanonical/Header** into the similarly named fields in **insertOrderHeaderInput**. Be sure to link from the Sender/ID and Receiver/ID fields! Do not specify links to **overrideCredentials** or **$connectionName**.



In the Pipeline view, copy the **OrderCanonical/Items** document list from Pipeline In to the clipboard.

d. Add a **LOOP** step to your service. For the **Input array** property, paste **/OrderCanonical/Items** from the clipboard.

e. Drag **acme.adapters:insertOrderDetails** into your service and make sure it is indented under the LOOP step.



Use the Pipeline view to make sure that in the invocation of the insertOrderDetails service Pipeline input **OrderCanonical/Items** is no longer a document list.

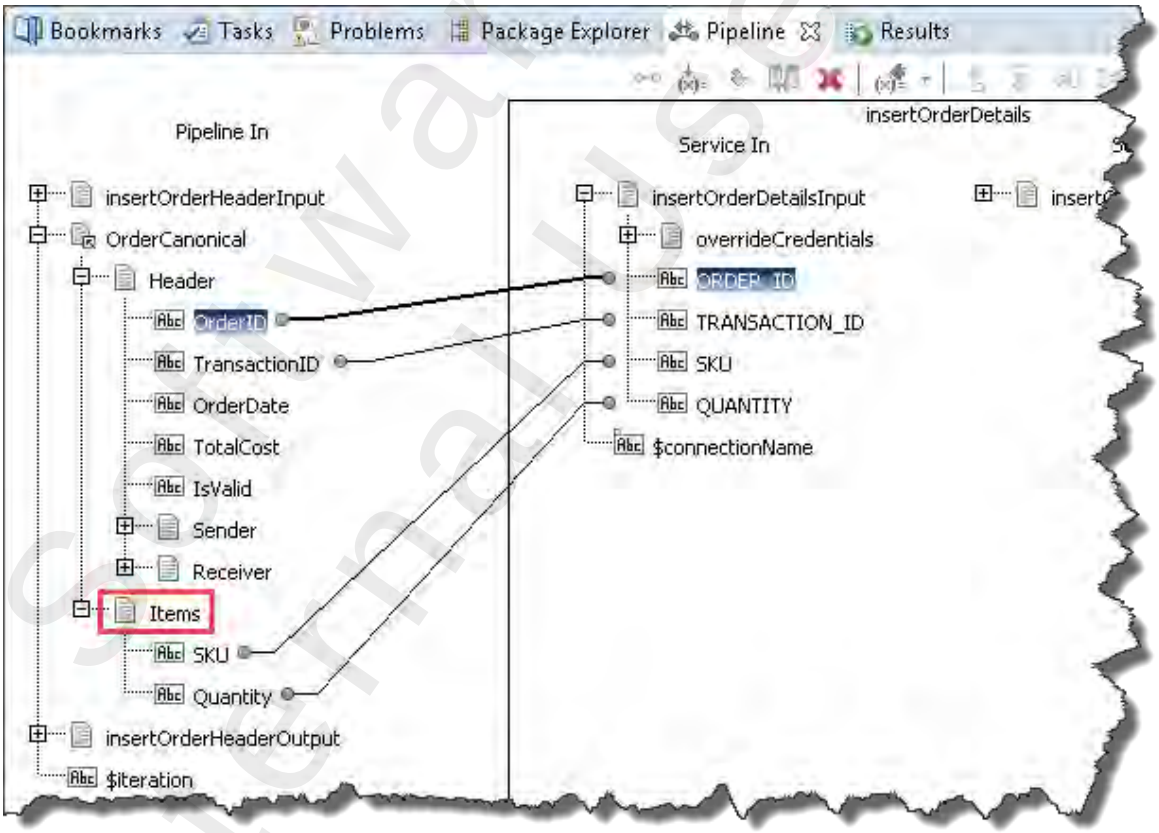
Then, link the following:

i. **OrderCanonical/Header/OrderID** to **ORDER_ID**

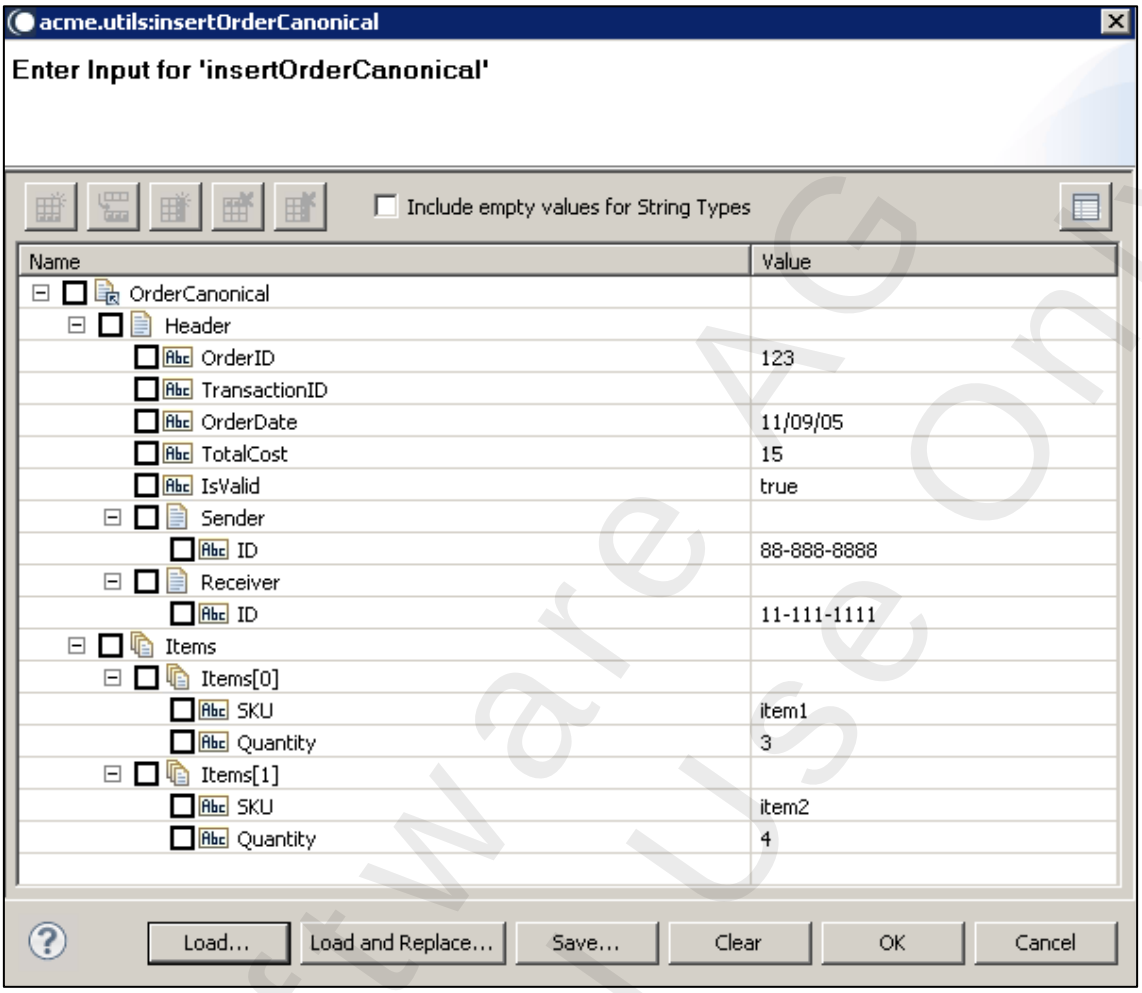
ii. **OrderCanonical/Header/TransactionID** to **TRANSACTION_ID**

iii. **OrderCanonical/Items/SKU** to **SKU**

iv. **OrderCanonical/Items/Quantity** to **QUANTITY**



17. Save and run the **acme.utils:insertOrderCanonical** service by loading the data from the file **order_canonical_input.txt** contained in the folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub**.



18. Use the two Select JDBC 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?

This Page intentionally left blank

EXERCISE 19:
JDBC 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.

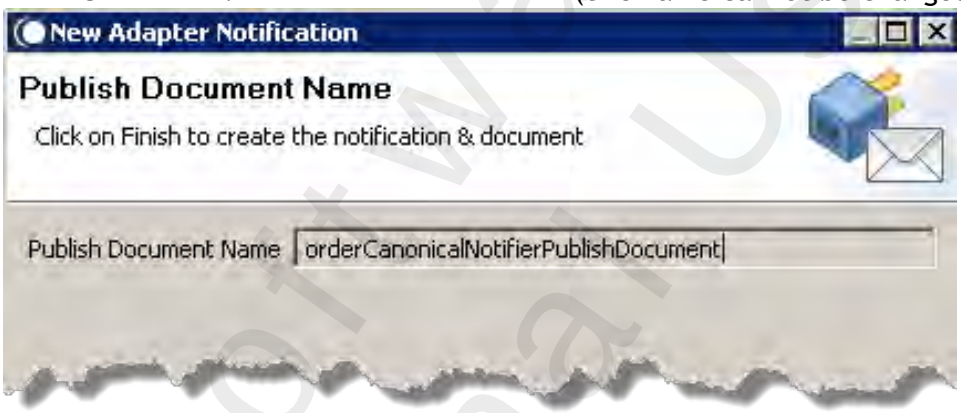
Note: This exercise assumes that Exercise 16 and 18 (Messaging using JMS and Create JDBC Adapter Services) are completed.

Steps

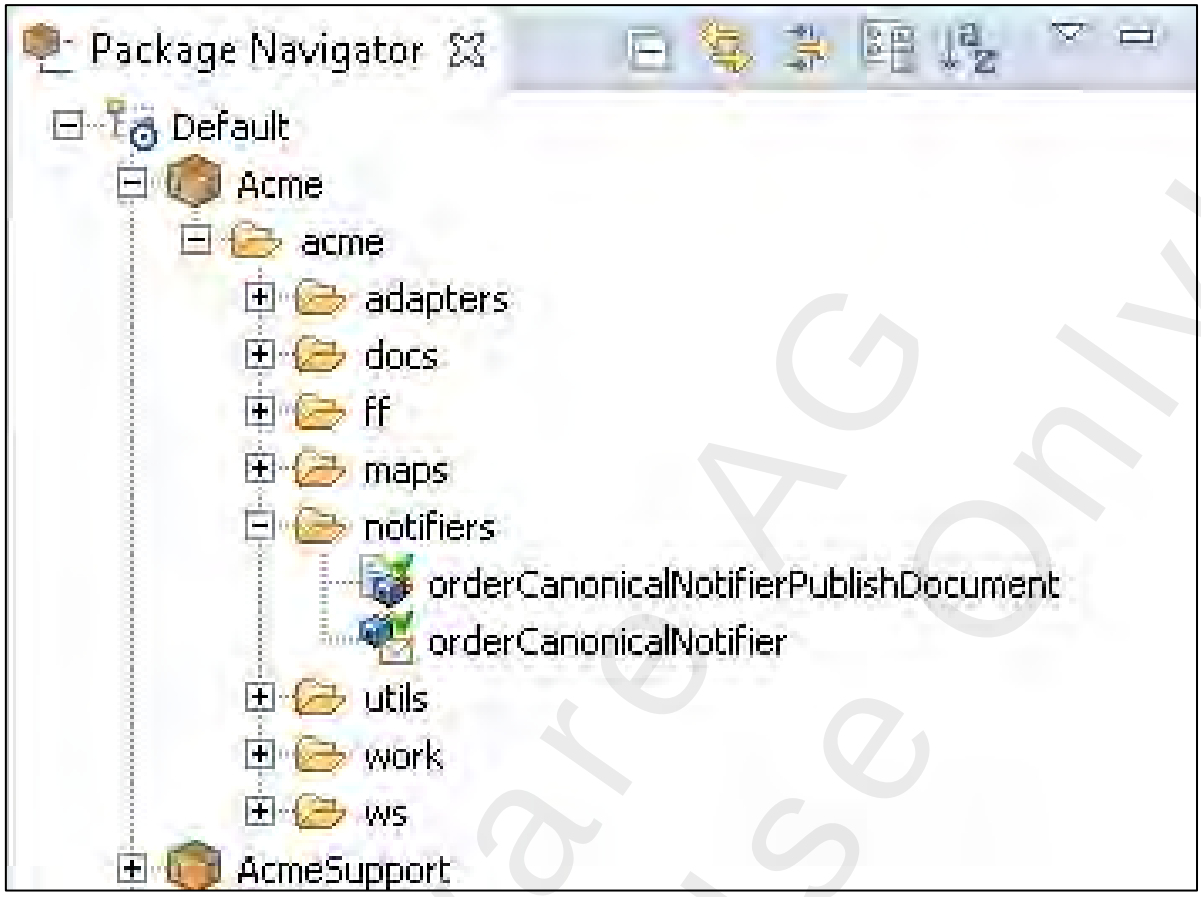
1. Create a new Adapter Notification in the **acme.notifiers** folder named **orderCanonicalNotifier**.

   Specify this will be of type **JDBC Adapter**, select template **InsertNotification**, and use the Adapter Connection **commonSupport.adapters:acmeAdapter**.

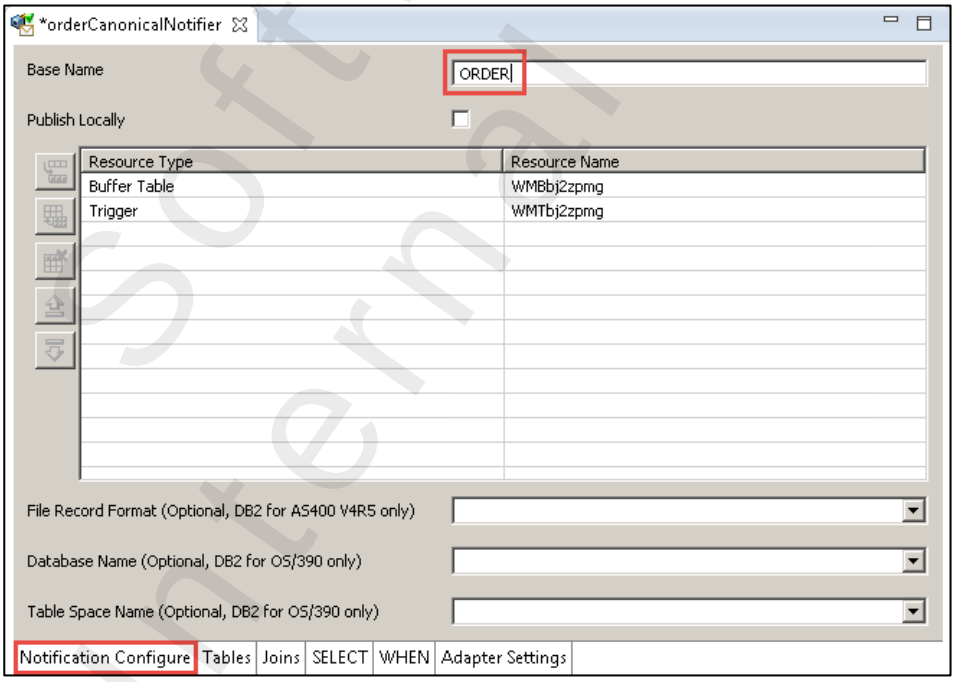
   You will be told that the publishable document used with this notification is **orderCanonicalNotifierPublishDocument** (this name cannot be changed):

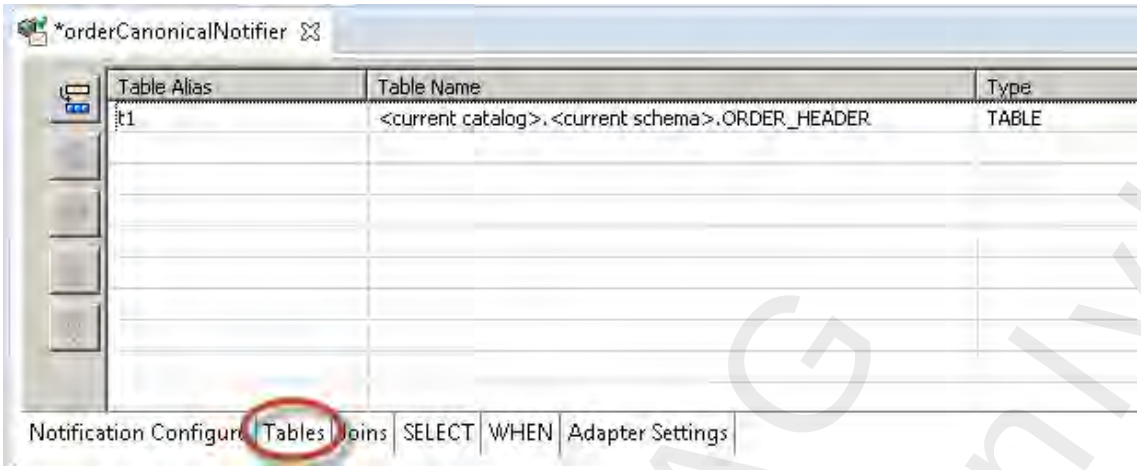
   Click **Finish**.
   You will see that the Adapter Notification and corresponding publishable document type were created:



2. Configure the Adapter Notification like this:
   a. On the **Notification Configure** tab, specify **ORDER** as the base name.

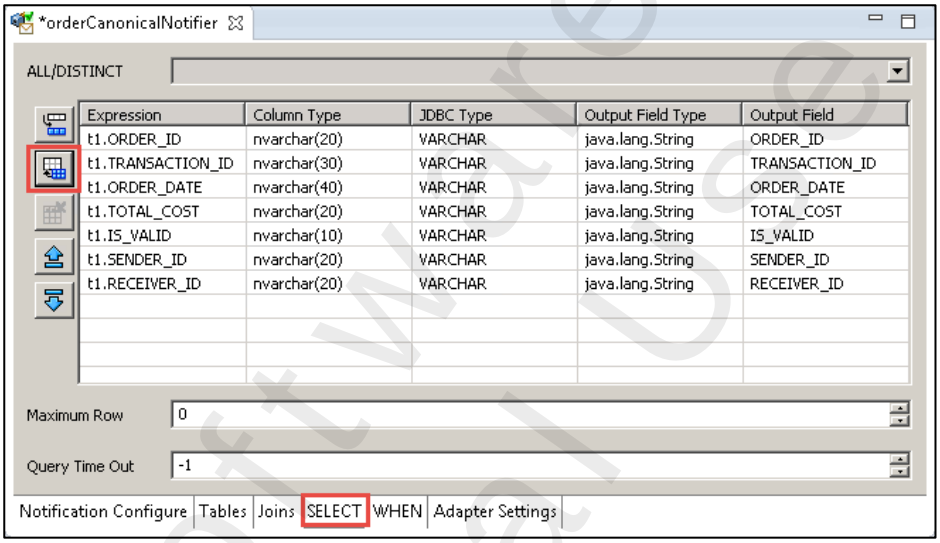


   b. On the **Tables** tab, select the Table Name **<current catalog>.<current schema>.ORDER_HEADER**.



c. Skip the **Joins** tab.

d. On the **SELECT** tab, click the **Fill in all rows to the table** button.



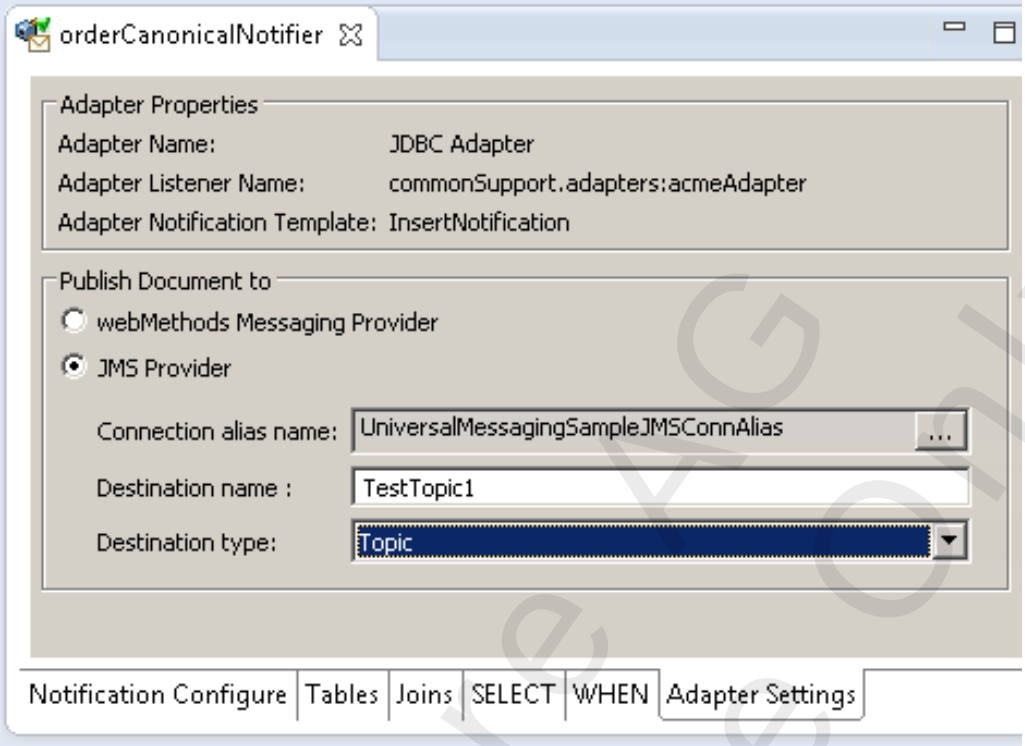
e. Skip the **WHEN** tab.

f. On the **Adapter Settings** tab, select the **JMS Provider** radio button. Then, specify the following:

   i. Connection alias name: **UniversalMessagingSampleJMSConnAlias** (browse for it)

   ii. Destination name: **TestTopic1**

iii. Destination type: **Topic**

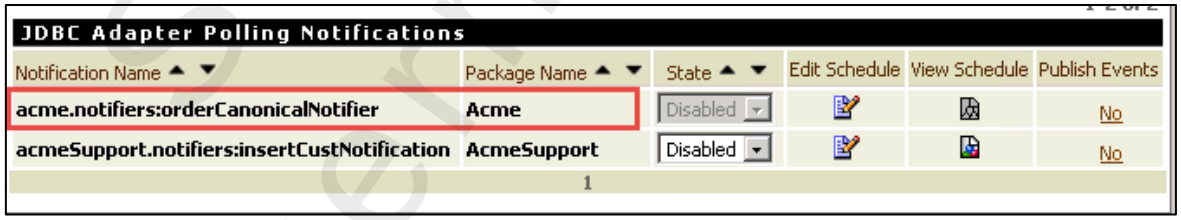


g. **Save** your work.

3. As you recall from exercise 16, you already have a JMS trigger (listenForValidationJMS) listening on **TestTopic1**. The trigger will pass the notification document to the handling service acme.work:handleValidationJMS.
This handler service will convert the document to an XML string and display it in the IS log.

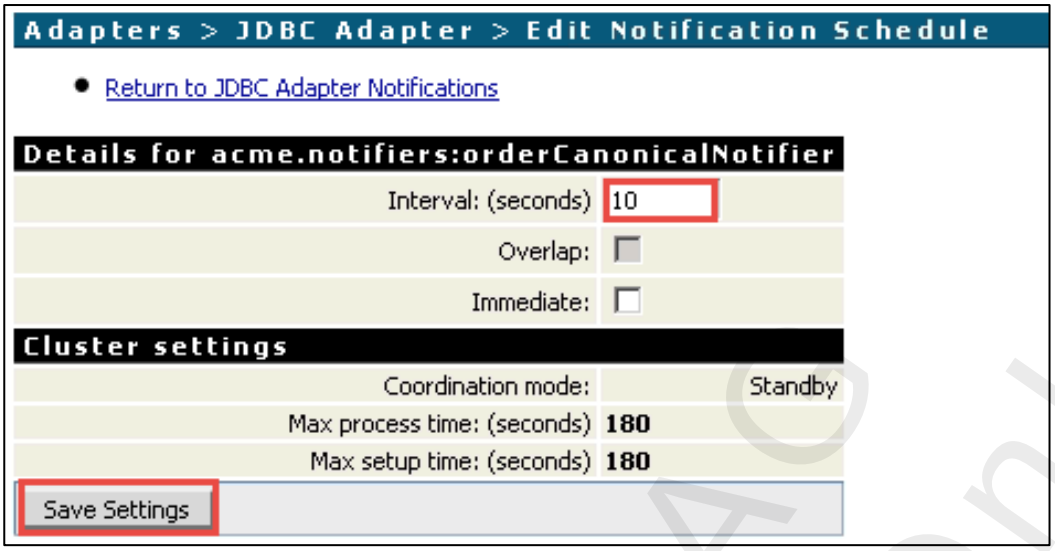
   Therefore, in this exercise you do not have to create a new trigger and handle service to test.

4. Next you will have to configure the JDBC adapter polling notification and enable it.
To do this, go into the IS Administrator UI, select the **Adapters --> JDBC Adapter --> Polling Notifications** link. You should see your new notification service listed.

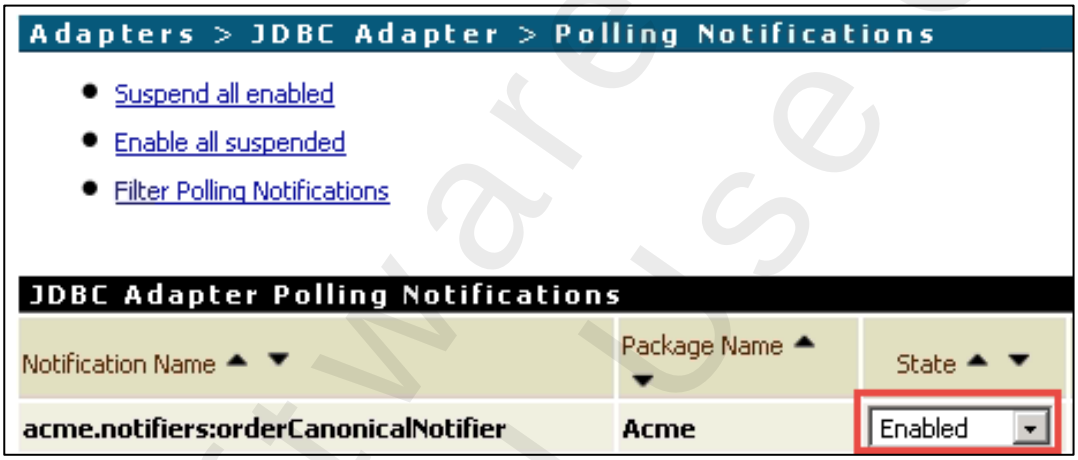


a. Edit the Notification Schedule by clicking the **Edit Schedule** icon for your notification service. Specify the following parameters and then select **Save Settings**:

   - **Interval** = **10**
   - **Overlap** = ***unchecked***
   - **Immediate** = ***unchecked***



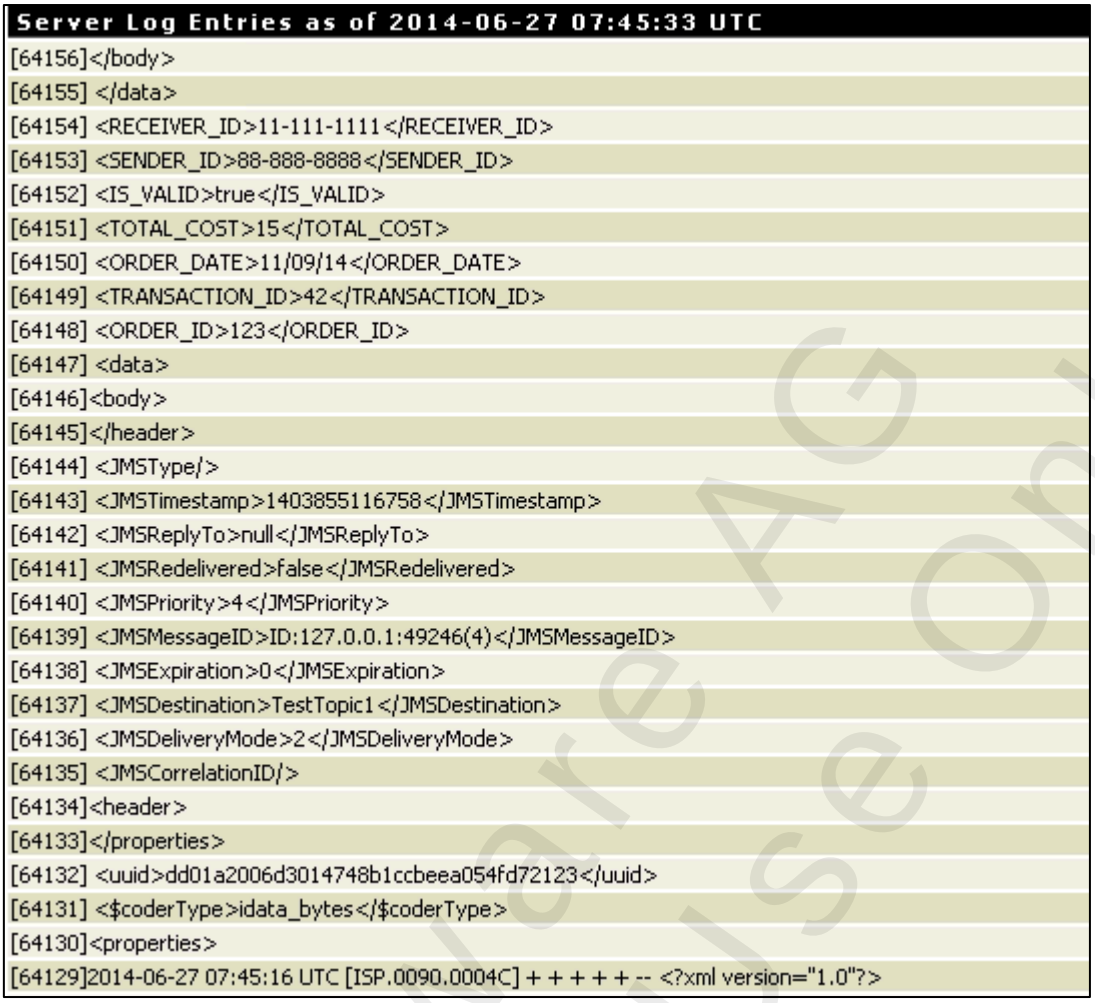
b. Enable the Polling Notification for your Adapter Notification.



5. Switch back to Designer. Test your work by running the **acme.utils.insertOrderCanonical** service from the previous exercise.

   You can load the same file again, which was **order_canonical_input.txt** contained in folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub**.

   You should see an XML representation of your OrderHeader document, within the polling interval (10 seconds), in the IS Server Log.



Server Log Entries as of 2014-06-27 07:45:33 UTC

```
[64156]</body>
[64155] </data>
[64154] <RECEIVER_ID>11-111-1111</RECEIVER_ID>
[64153] <SENDER_ID>88-888-8888</SENDER_ID>
[64152] <IS_VALID>true</IS_VALID>
[64151] <TOTAL_COST>15</TOTAL_COST>
[64150] <ORDER_DATE>11/09/14</ORDER_DATE>
[64149] <TRANSACTION_ID>42</TRANSACTION_ID>
[64148] <ORDER_ID>123</ORDER_ID>
[64147] <data>
[64146]<body>
[64145]</header>
[64144] <JMSType/>
[64143] <JMSTimestamp>1403855116758</JMSTimestamp>
[64142] <JMSReplyTo>null</JMSReplyTo>
[64141] <JMSRedelivered>false</JMSRedelivered>
[64140] <JMSPriority>4</JMSPriority>
[64139] <JMSMessageID>ID:127.0.0.1:49246(4)</JMSMessageID>
[64138] <JMSExpiration>0</JMSExpiration>
[64137] <JMSDestination>TestTopic1</JMSDestination>
[64136] <JMSDeliveryMode>2</JMSDeliveryMode>
[64135] <JMSCorrelationID/>
[64134]<header>
[64133]</properties>
[64132] <uuid>dd01a2006d3014748b1ccbeea054fd72123</uuid>
[64131] <$coderType>idata_bytes</$coderType>
[64130]<properties>
[64129]2014-06-27 07:45:16 UTC [ISP.0090.0004C] + + + + + -- <?xml version="1.0"?>
```

6. Use the IS Administrator UI and select **Adapters --> JDBC Adapter --> Polling Notifications.** Disable your Polling Notification **acme.notifiers:orderCanonicalNotifier**.

Check Your Understanding

1. What is automatically created and updated when you work with your JDBC Adapter Notification?
2. What occurs when the JDBC Adapter Notification Schedule is enabled?
3. Why do you see the XML representation of your OrderHeader document in the IS Server Log?
4. Why is the JDBC Adapter Notification Schedule an administrative task?

EXERCISE 20:
USE SERVICES IN A BUSINESS PROCESS

Overview

Use Designer to modify an existing business process to:

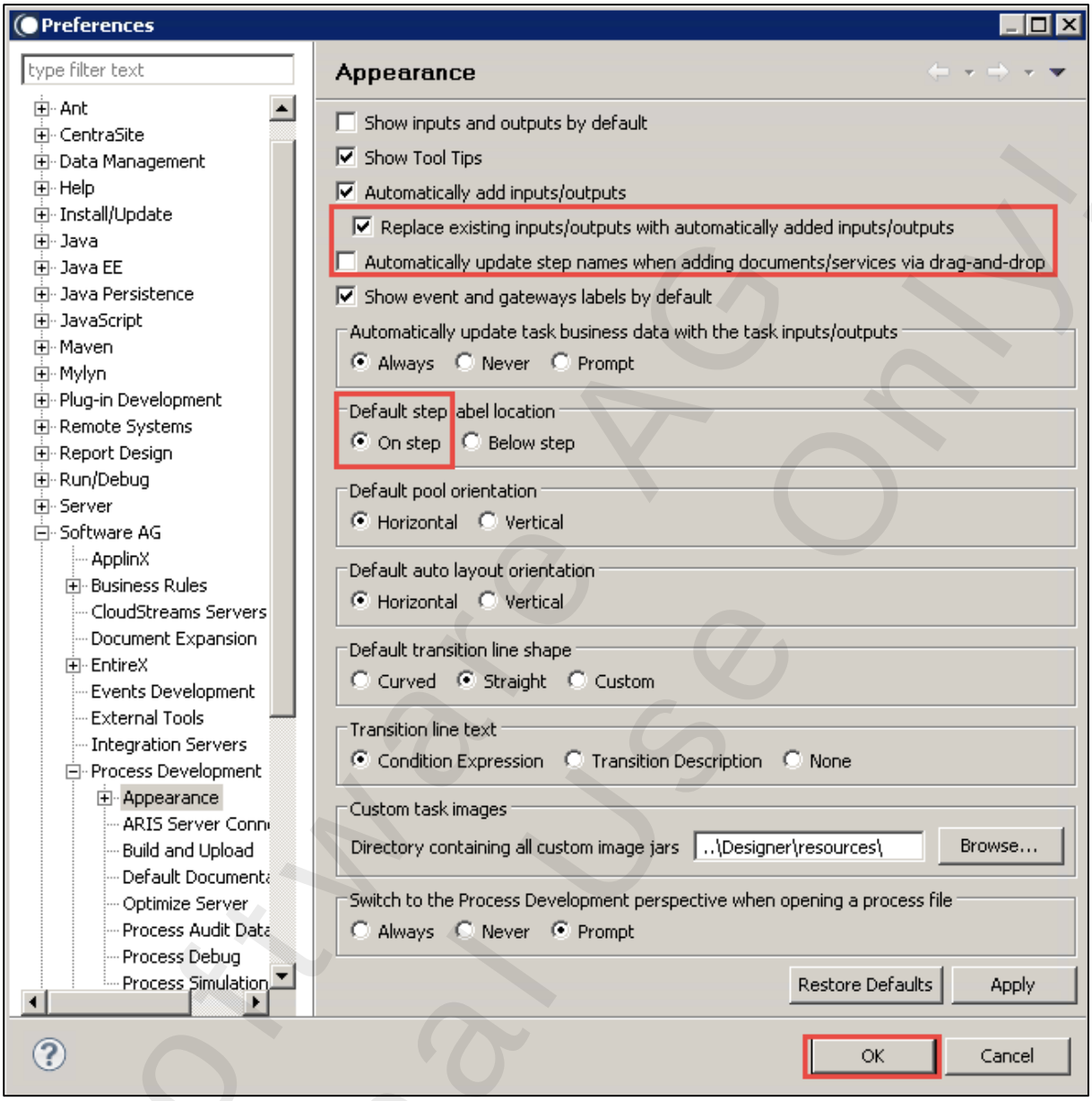
- subscribe to an IS document to start the process
- include services created in previous exercises
- incorporate an error handling service

After you modify the process, you will do a Build and upload, and then test your process in Designer.

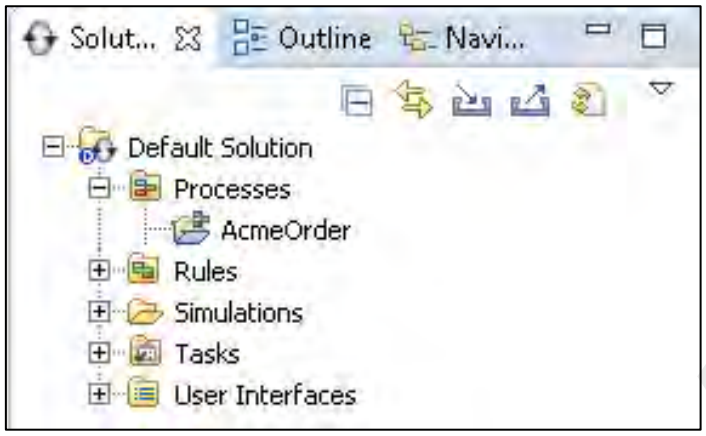
Steps

1. In Designer change to the **Process Development** perspective (**Window --> Open Perspective**).
   Open the Preferences view under **Window --> Preferences** and select **Software AG --> Process Development --> Appearance**. Change the following selections:
   a. Select the **Replace existing inputs/outputs with automatically added inputs/outputs** checkbox.
   b. Unselect the **Automatically update step names when adding documents/services via drag-and-drop** checkbox.
   c. Set Default step label location to **On Step**

d. Select the **OK** button to save your changes.

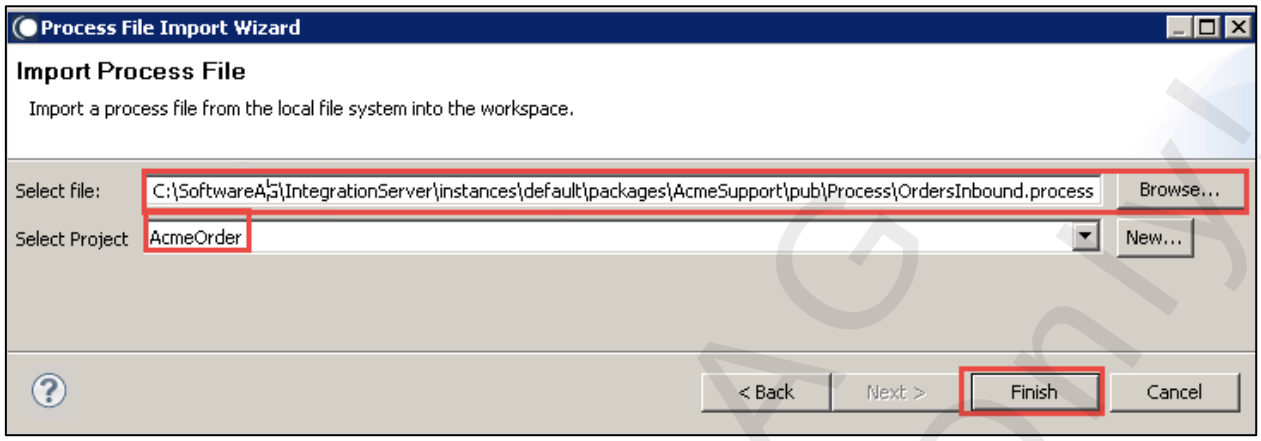


2. Using the Solutions view, right-click **Processes** to create a **New Process Project** called **AcmeOrder** in the default location. Click **Finish**.



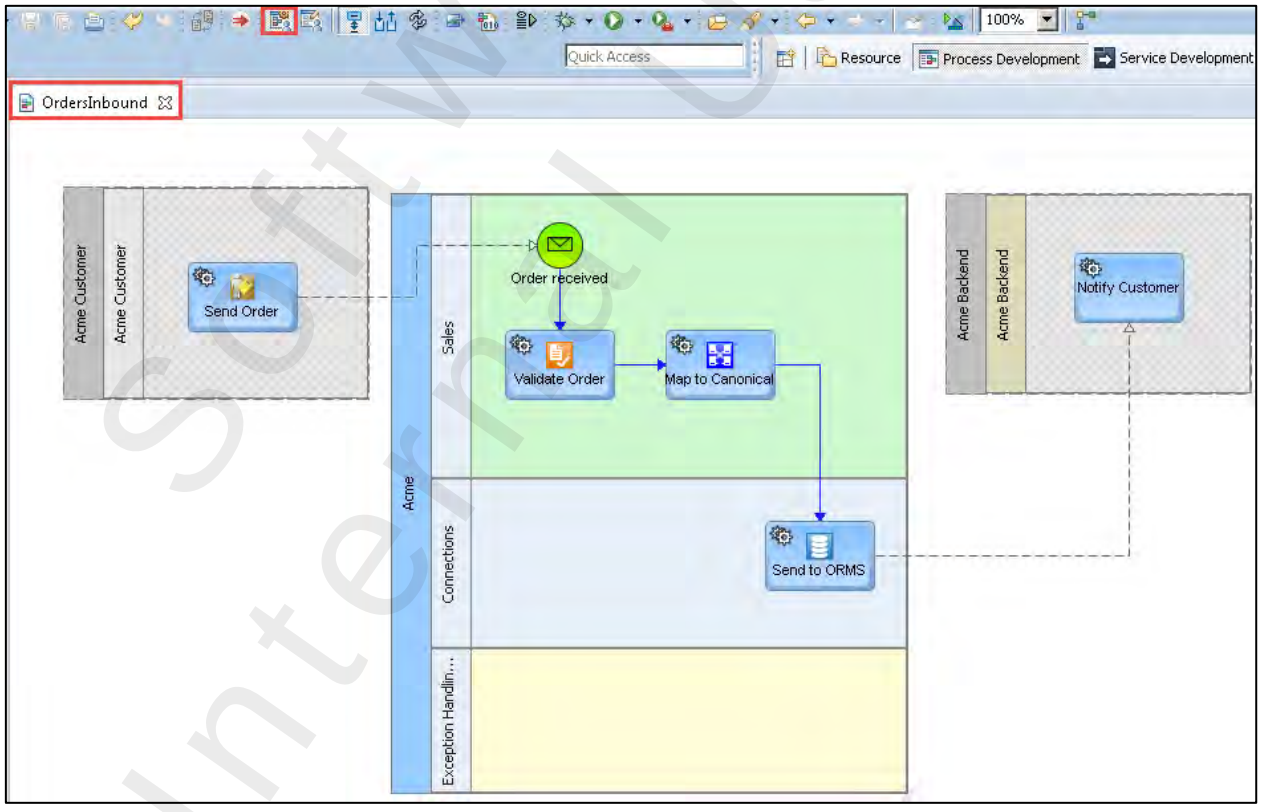
3. From the File menu, select **Import** from an import source of **SoftwareAG --> Process File.** Click **Next.**

4. Choose the process file **OrdersInbound.process** contained in folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub\Process** and the target process project **AcmeOrder**. Click **Finish**.

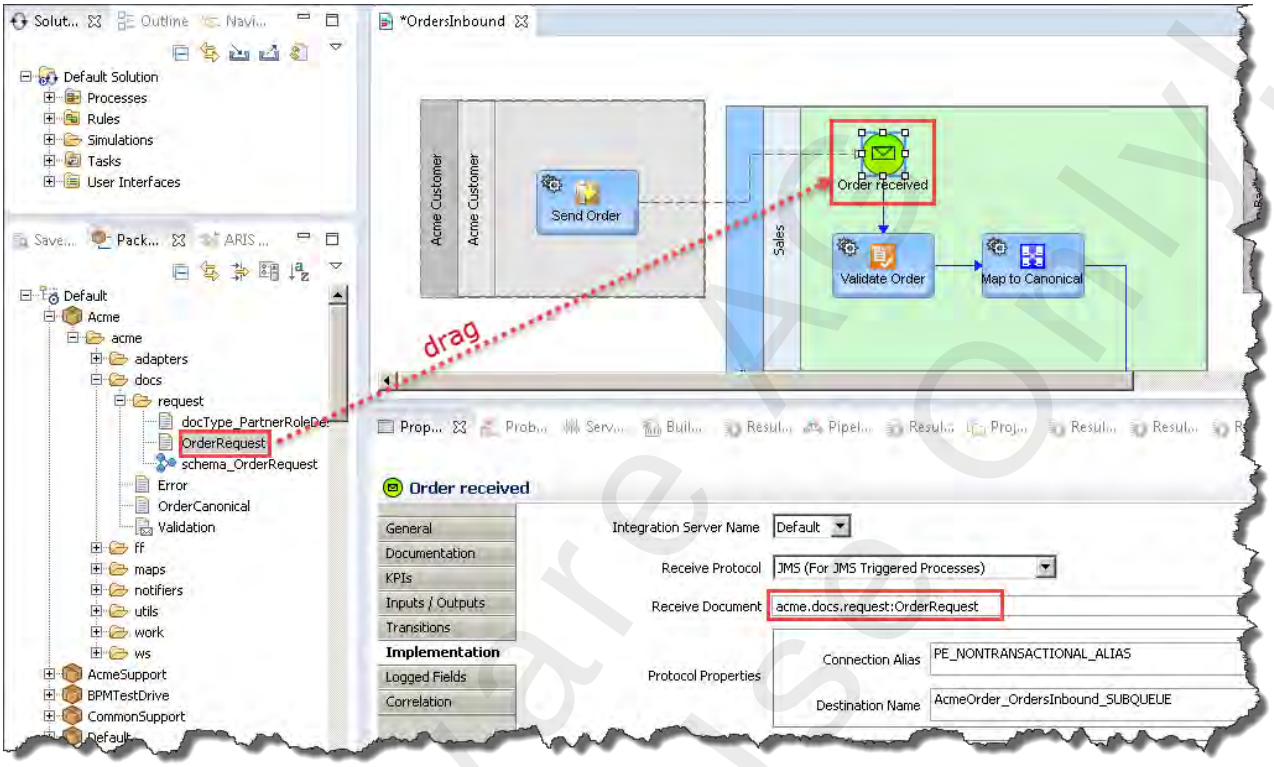


5. Make sure you are in **Process Developer** mode

Examine the flow of the process so that you understand what it is intended to do.
Note: Double-click the tab labeled with **OrdersInbound** to display the process model in full-screen mode.

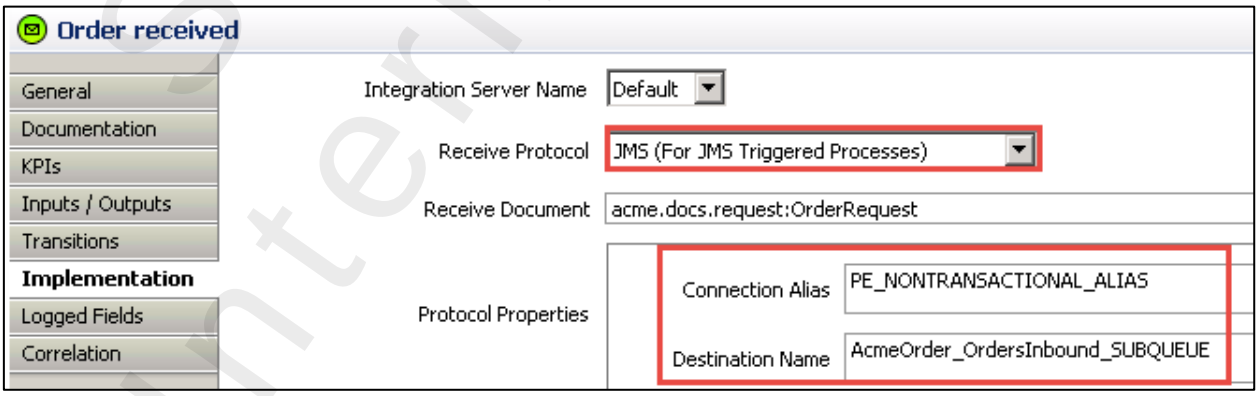


6. Drag the **acme.docs.request:OrderRequest** doctype from the Package Navigator view onto the **Order received** Start Message Event.
   Choose the **Order received's Implementation** tab and confirm that its **Receive Document** text box is set to **acme.docs.request:OrderRequest.**

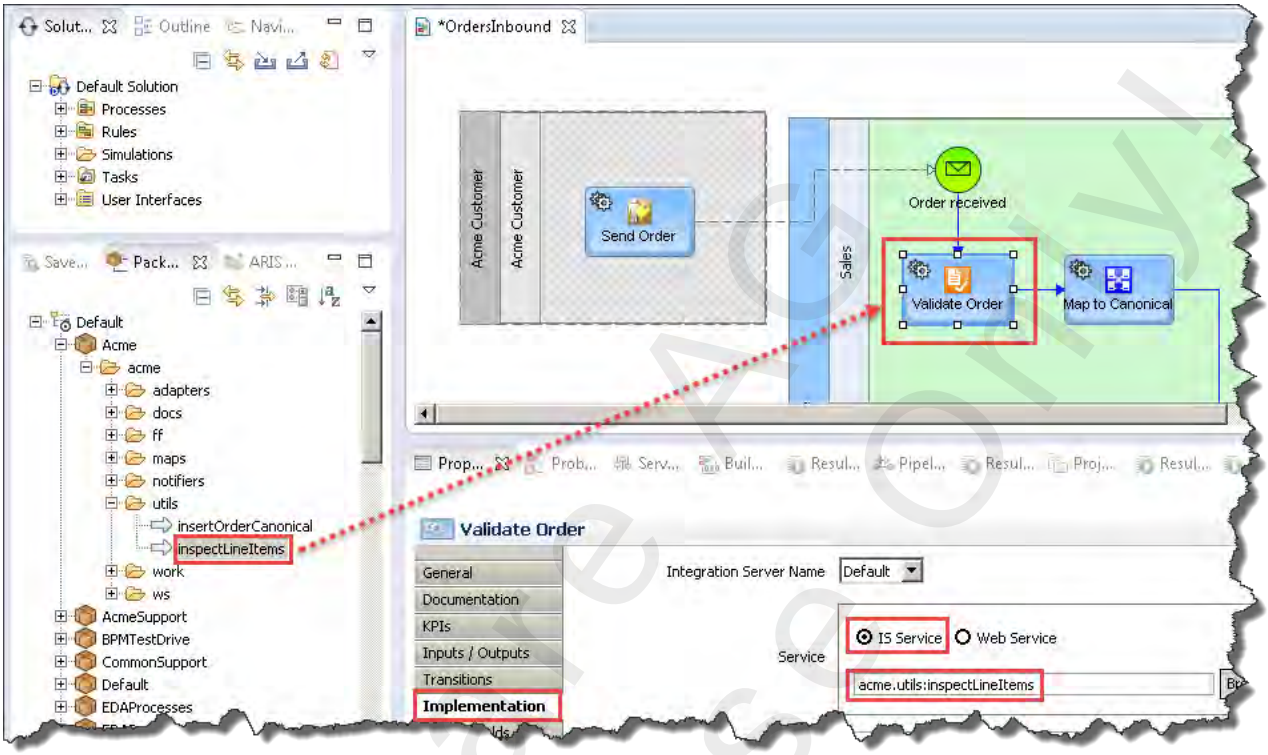


Dragging the IS document type onto the **Order received** step has also configured the JMS protocol properties of the process model. At runtime, the process will use by default the following JMS properties to wait for and receive an OrderRequest document (for this exercise don't change these settings):

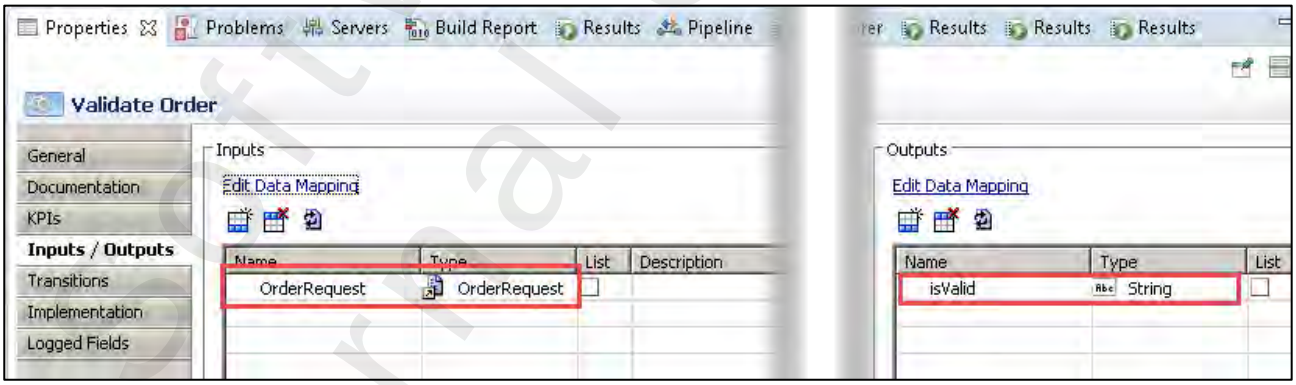
- Connection Alias: **PE_NONTRANSACTIONAL_ALIAS**
- Destination Name: **AcmeOrder_OrdersInbound_SUBQUEUE**



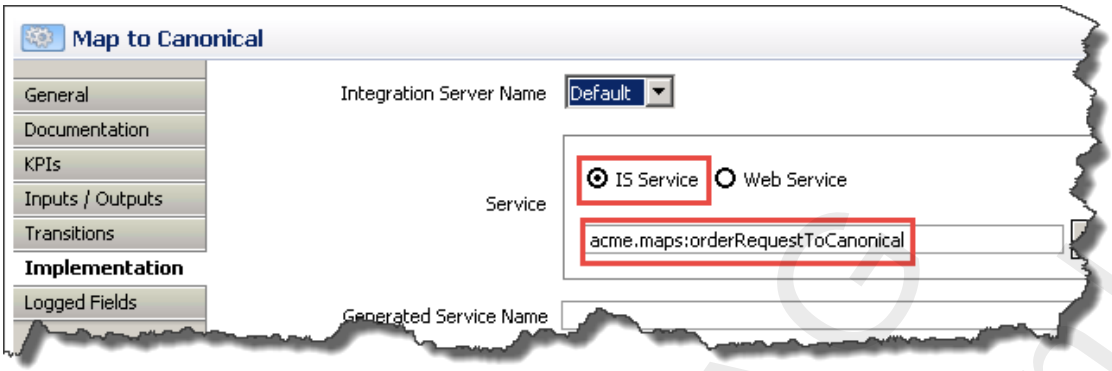
7. Drag the **acme.utils:inspectLineItems** service onto the **Validate** step. Confirm that the **Validate** step's:
   a. **Implementation** tab's **Service** field is filled in:

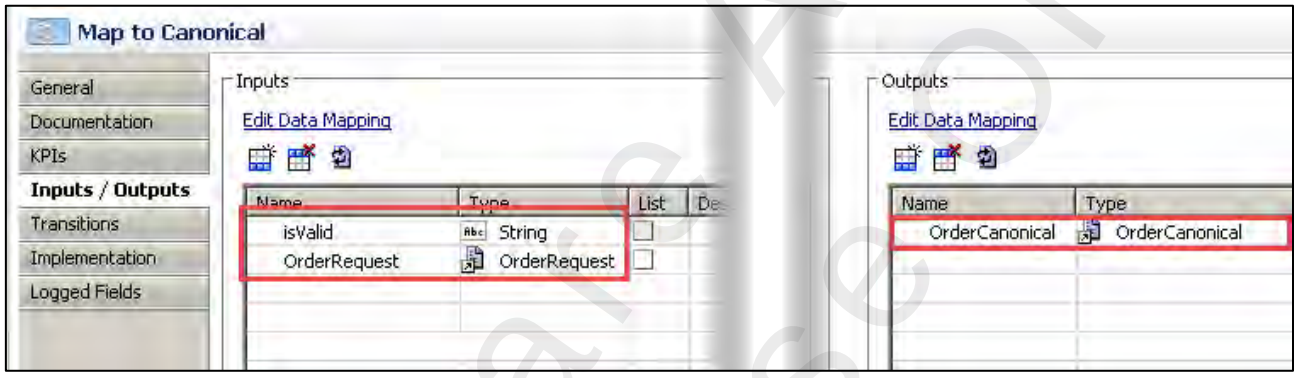


   b. **Input / Outputs** tab's, Inputs and Outputs are filled based on the IS service inputs/outputs:

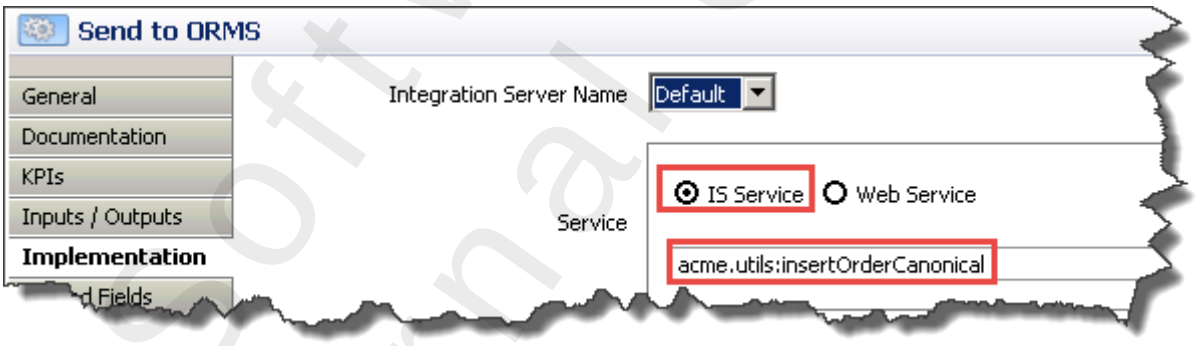


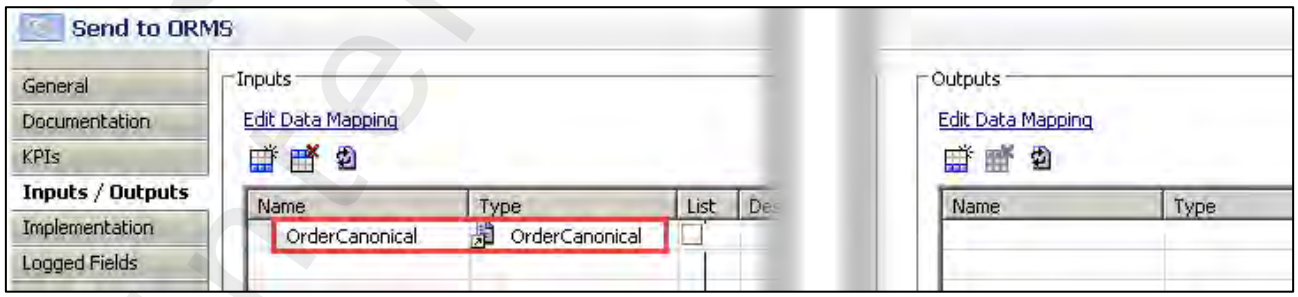
8. Drag the **acme.maps:orderRequestToCanonical** service onto the **Map to Canonical** step. As before, confirm the **Map to Canonical** step's **Implementation** and **Input / Output** tabs are now correctly filled in:





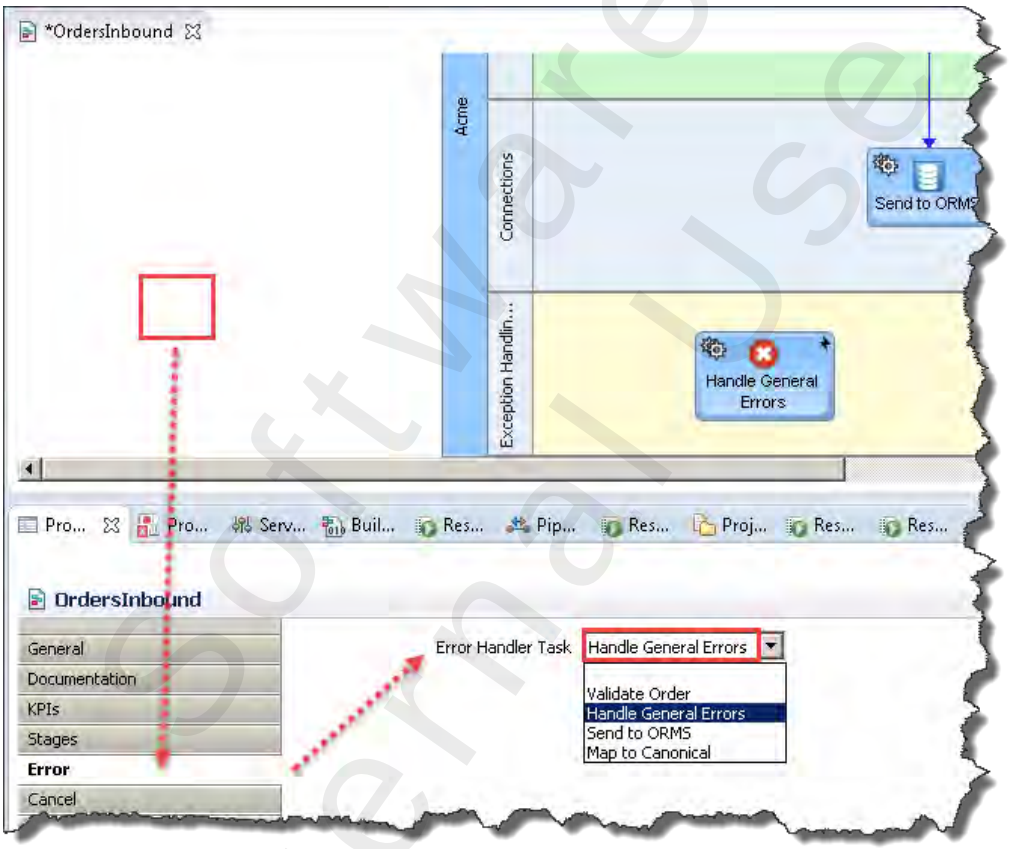
9. Drag the **acme.utils:insertOrderCanonical** service onto the **Send to ORMS** step. Confirm this step's **Implementation** and **Input / Output** tabs. Note that this service has no output.





10. Drag the **AcmeSupport** package's **acmeSupport.process:processErrorHandler** service into the empty **Exception Handling** swimlane (yellow row).
    a. Select the step's **General** tab and change its **Label** to **Handle General Errors**
    b. Right click the step and use the **Choose Image…** option to select the image that shows a stop sign with the X in the middle.

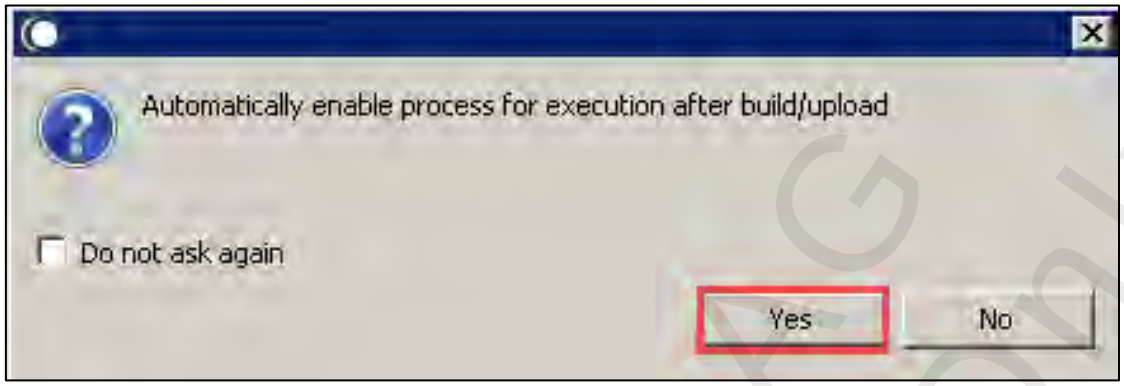
11. You will now set the process level **Error** properties. To get access to the process properties, click somewhere in your process diagram where there is only white background. In the **Error** tab, select the **Error Handler Task** of **Handle General Errors**.



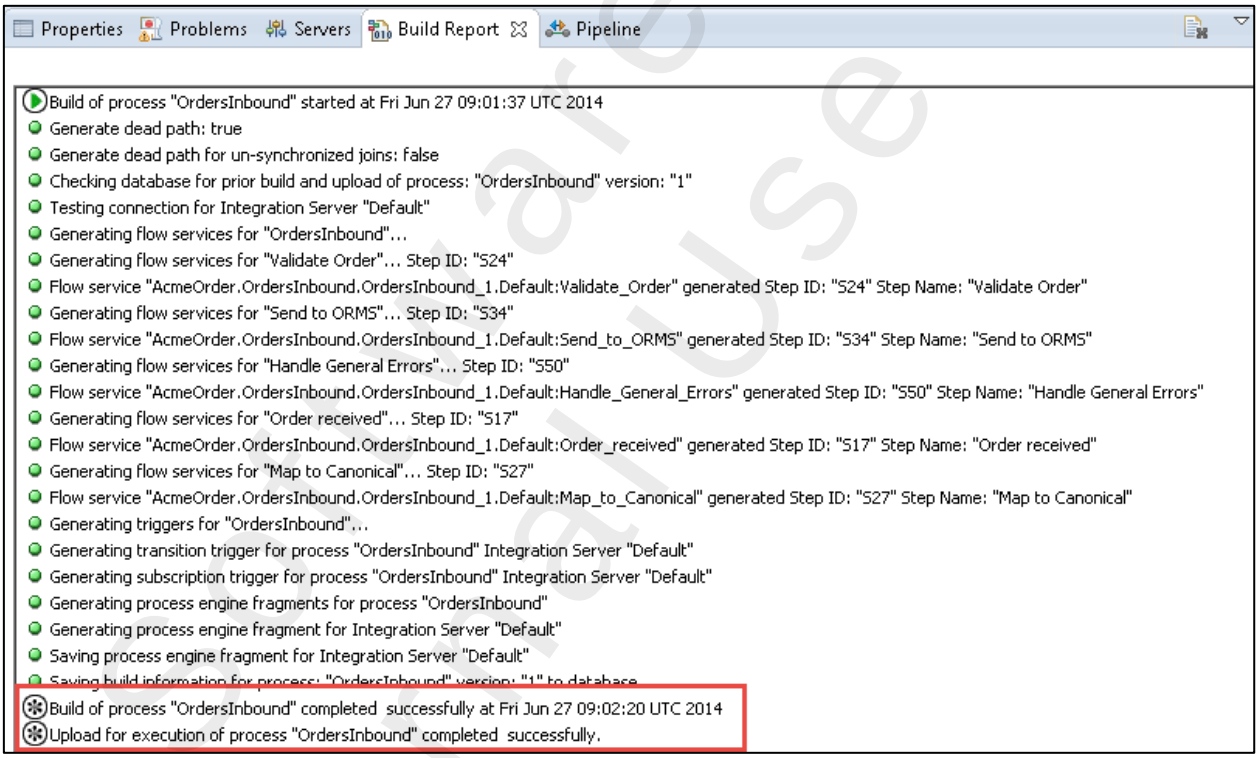
12. **Save** the process.

13. Then, select the **Build and upload for execution**  Icon. Check for errors in the Build Report, and make corrections as necessary.

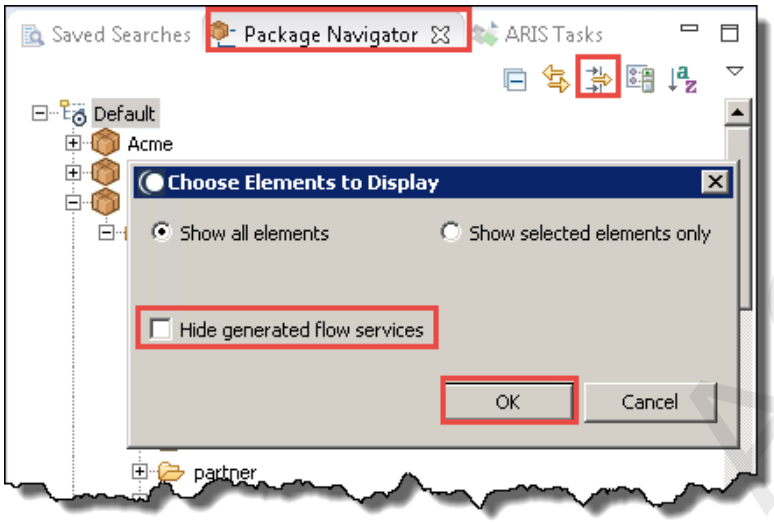
    If asked whether you want to enable you process for execution, choose **Yes**.



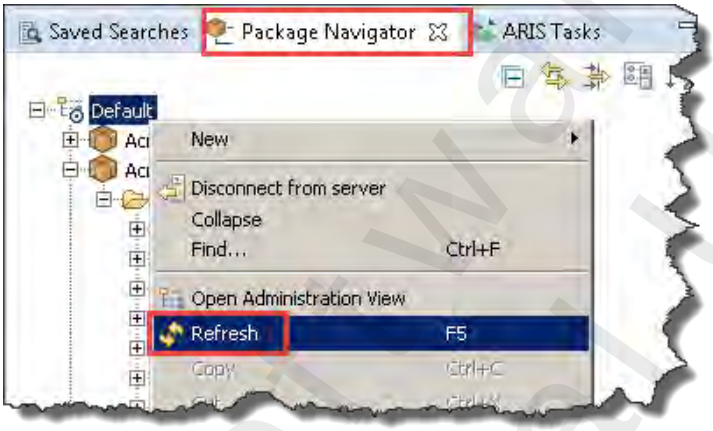
Your build report should look like the following:



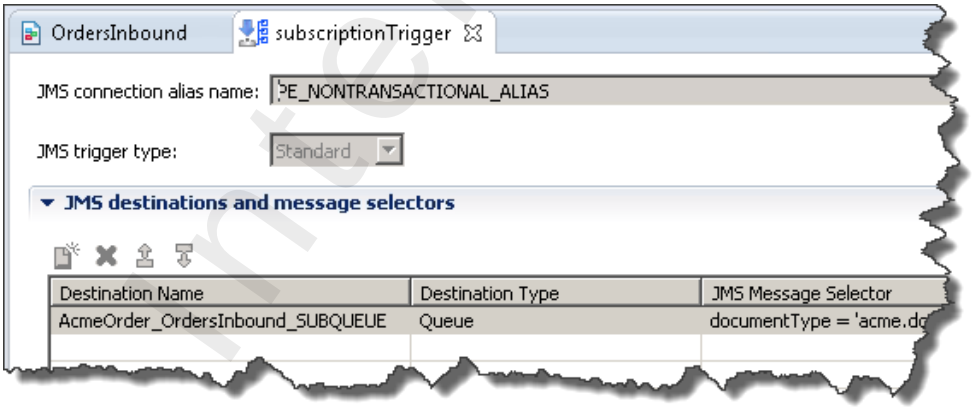
14. In the Package navigator view, click the **Filter contents of Navigator** icon , and in the popup box, unselect the **Hide generated flow services** checkbox.



15. Refresh the Package Navigator view to see the generated assets. To do this right-click on **Default** in the Package Navigator view and select **Refresh** (or press **F5**). This will reload everything under the Integration Server packages folder.



16. Expand the generated package **AcmeOrder**. Open the generated process JMS Subscription Trigger **AcmeOrder.OrdersInbound.Default:subscriptionTrigger**. It should use the (default) JMS connection alias name **PE_NONTRANSACTIONAL_ALIAS** and the Destination Name **AcmeOrder_OrdersInbound_SUBQUEUE**.



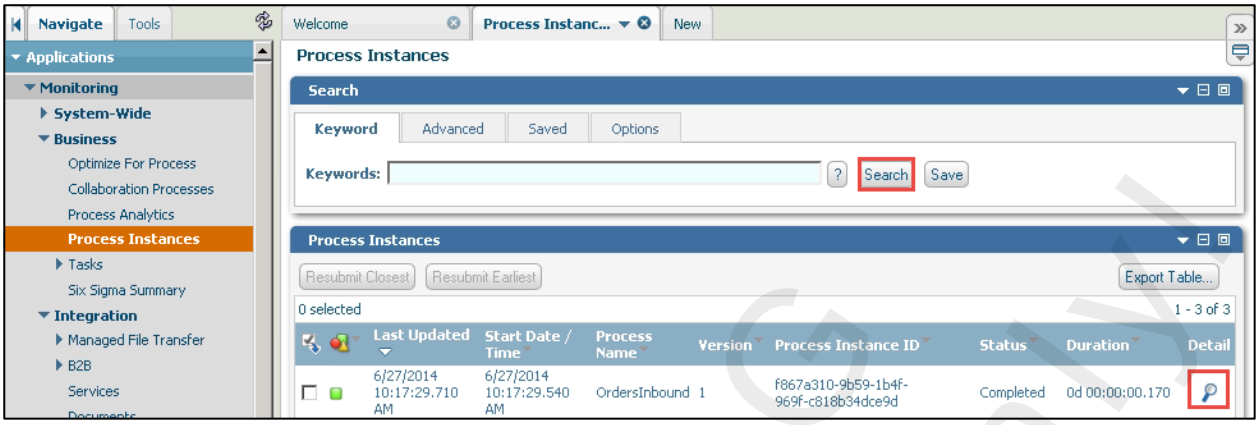
17. To launch an instance of your process model, perform the following steps:
    a. From the Package Navigator view, right-click Document type **acme.docs.request:OrderRequest** contained in your Acme package. Select **Run As --> Publish As JMS Message**.
    b. In the panel that appears, load sample data from file **ProcessInput.txt** contained in folder **C:\SoftwareAG\IntegrationServer\instances\default\packages\AcmeSupport\pub.**

       Note that loading the input data will also set the following JMS properties:

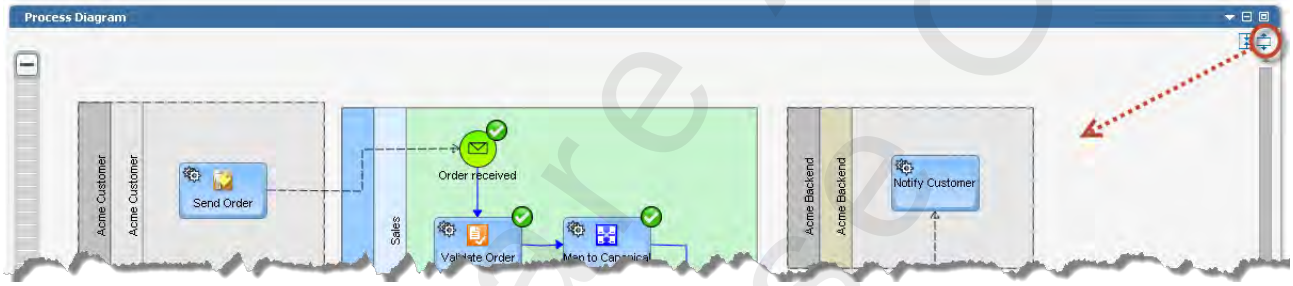
       i. **JMS connection alias name** = **PE_NONTRANSACTIONAL_ALIAS**
       ii. **Destination name** = **AcmeOrder_OrdersInbound_SUBQUEUE**
       iii. **Destination type** = **Queue** (set automatically)
       iv. **Prepare message for BPM** = *checked*
    c. Click **OK** to send the data as a JMS message to be caught by a process instance.

18. Open up a browser tab and login to **My webMethods (**http://localhost:8585 or use the appropriate shortcut) as **Administrator | manage.**

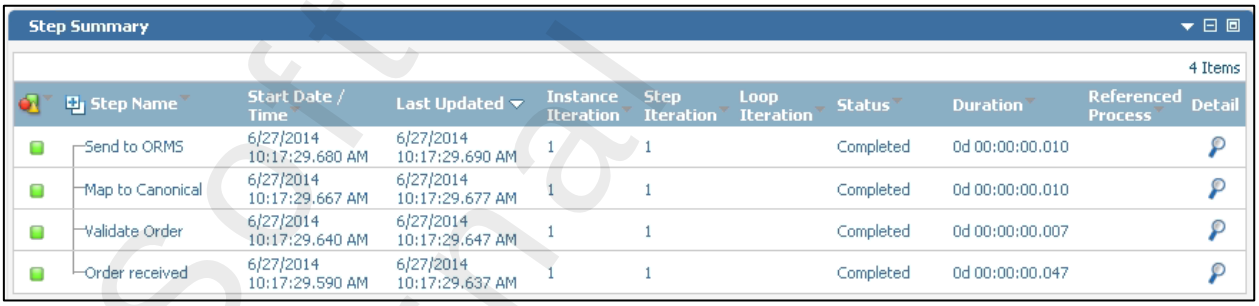
    a. In My webMethods, navigate to **Applications --> Monitoring --> Business --> Process Instances**. Look for a new process instance of type **OrdersInbound**. Click **Search** to refresh the process list, if necessary. On the **OrdersInbound** row, click the magnifying glass icon under the **Detail** column.



b. Scroll down to the **Process Diagram** section. Click the **Increase diagram height** icon to view the entire model. You should see green checkmarks by each of your steps:



19. Scroll down to the **Step Summary** section. You should see four completed steps. If you see an error, ask instructor to help you debug your process.



Check Your Understanding

1. Why do you create swim lanes 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. Name different start mechanisms for a process.

CHECK YOUR UNDERSTANDING: ANSWERS TO THE QUESTIONS

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

1. What is the URL to access the Integration Server? http://<IS**_hostname>:5555/**
2. What is the URL for MWS? **http://<MWS_hostname>:8585/**
3. Why should you set the Broker Monitor service 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.
4. Do you have to start Broker Monitor/Server and Universal Messaging in your webMethods environment?
   Not necessarily. New webMethods customers will typically use Universal Messaging (UM) as the messaging provider. Broker is typically only required for existing customers who have already been using Broker and are not ready to migrate yet to UM.

Exercise 2: Packages and Folders

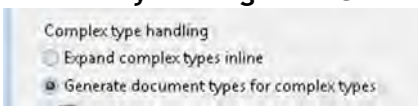
1. If you place the folders in the wrong parent folder, how could you correct it? **You can drag and drop folders with the mouse. You can use "Cut and Paste".**
2. Why is a consistent folder structure in all packages important?
   This makes it easier for new people on a project to understand the inner workings of a package.
   It helps standardize on a folder structure and prevent integration projects from using different folder structures. Non-standard folder structures can be very confusing when developing, maintaining and supporting the integration solutions.

Exercise 3: Create a Flow Service

1. Why is the order of the services important? **The order of service invocation is the order of execution; therefore this determines the runtime behavior of your service.**
2. What would appear in the Server Log if in the message parameter, you did the following:
   a. Selected "Perform pipeline variable substitution" but NOT "Perform global variable substitution". **60 points is 75.0%percentSign% of 80 possible points**
   b. Selected "Perform global variable substitution" but NOT "Perform pipeline variable substitution".
      i. In Server Log: **%earnedPoints% points is %percentage%% of %possiblePoints% possible points**
      ii. In Designer: **com.wm.lang.flow.FlowException: Global Variable possiblePoints does not exist**
   c. Did NOT select either "Perform pipeline variable substitution" or "Perform global variable substitution". **%earnedPoints% points is %percentage%%percentSign% of %possiblePoints% possible points**

Exercise 4: Document Types

1. Why are additional documents created in the acme.docs.request folder when the OrderRequest schema is imported? **Because you explicitly said so when importing the schema by leaving the "Generate document types for complex types" radio button**



on its default value. 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 (XSD) for import over using a DTD? **XML Schemas allow for constraints (positive Integer, Float, or matches a pattern). A DTD does not have this feature. 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 three ways to indent a Document Type element under another? **You can use drag and drop with the mouse, or the arrow buttons in the top toolbar, or the arrow buttons in the menu bar.**

Exercise 5: Building Flow Services - BRANCH

1. When are regular expressions useful in branch? **When you want to match a pattern or use more complex evaluation criteria.**
2. Can you combine a switch variable with Evaluate labels=True? **No, this wouldn't make sense.**
3. What are the special test values that can be used as labels in a branch statement?
   "$default" - which is used when none of the other input values matches
   "$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 and debugLog steps were not indented under the Loop? **The 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? **Document references allow you to share IS Document Type definitions. Create the Document Type once in the namespace and then re-use in multiple services that require that same Document Type. Think of the Document Reference as a "pointer" to the Document Type definition (metadata).**
   If you create a Document Type definition directly in a service it is then only available for that specific service (the definition is not re-usable).

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**

2. What happens if the riskyOperation service works (doesn't fail)? **Only the first two 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 be initialized.**
2. Is there a way we could have validated all of the fields in the OrderRequest? **We could have used the WmPublic package's pub.schema:validate service.**

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. In step 3, why was it important to set the Inputs and Outputs of the service before you generated the code? **Because the generated code is based on the inputs and outputs you define in the Input/Output tab.**
   The generated code will include logic to retrieve the Input variables into local Java objects, and "put" statements to put the Output variables back into the pipeline.
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. In step 6c, what was the purpose of "import com.softwareag.util.IDataMap"? **To allow access to all the resources in the IDataMap class.**

Exercise 11: Monitoring Services

1. Why is it necessary to create Remote Server Aliases?
    a. **They are used for resubmits from Monitor in MWS - 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 executing IS.**
    b. **They are also used with the webMethods Deployer.**
2. Under what circumstances would it be acceptable to resubmit a service? Why?
Using the (Out of the box) Monitor resubmit capability is an decision that has to be made by the Integration Architect(s) and Developer(s) based on the integration requirements. Also keep in mind that a service has to be designed to be "resubmittable". For example, if a failed service only executed half of the code, then you may have a case where some updates were made but not all updates. (Imagine a service giving a 3% raise to all employees, who 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, easier to remember 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 HTML encoding the content of the output pipeline.**

Exercise 13: Working with Flat Files

1. Why can't flat files be imported like XML documents? **Because they do not contain markup to describe their structure and require additional information.**
2. What is the meaning of Nth field? **The Nth field identifies the position of the record identifier in a flat file record. Numbering starts at 0.**
3. What is the difference between a dictionary and a schema? **A schema describes the various record types 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.**
4. Why should you create the IS document type when the schema is complete? **This document type is used in your services to read from or write to a flat file.**

Exercise 14: Web Service Descriptors and Custom Faults

1. When would you create a Provider Web Service Descriptor (WSD) and when a Consumer WSD? **You create a provider when your Integration Server wants to host a web service. You create a consumer WSD when you want to consume (invoke) a web service.**
2. How and when are Web Service Connectors (WSC) created? **They are implicitly created when you create a Web Service Descriptor of type Consumer.**

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 15: Create REST service in Integration Server

1. What is the difference between a POST and a PUT operation? **The POST service creates a new object (Book), while the PUT service is used to update an existing object.**
2. Why wasn't it necessary to perform any mapping when dragging the pre-created services into the generated REST services? **Because the provided services had the same input parameters, implicit mapping took place.**
3. Where did the initial list of books come from? **They are stored in the static section of the code implementing the provided services.**
4. Why was it necessary to specify credentials when invoking pub.client:http in the extra credits section? **Because the pub.client:http service entered the Integration Server at its HTTP port, without providing a session cookie. Hence authentication was requested by the server.**

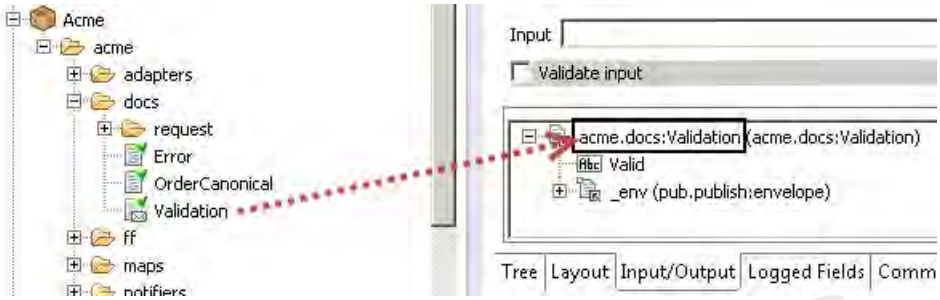
Exercise 16: Messaging with JMS

1. What is a Topic versus a Queue? **A Topic supports many to many communication, while a Queue supports many to one communication channel.**
2. Name the five JMSMessage types? **TextMessage, ByteMessage, ObjectMessage, MapMessage (used for IData), javax.jms.Message**

Exercise 17: webMethods Messaging

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 contains data that is used internally by the messaging provider to process the document.**
   The publishable document type will also activate the following properties: Connection alias name, Connection alias type and Provider definition.
2. What two objects are required for publishing? **A publishable Document Type (the structure) and a service to publish the document instance (e.g. pub.publish:publish)**
3. What two objects are required for subscribing? **A service to handle (receive) the published document and a webMethods Messaging Trigger.**
   The webMethods Messaging Trigger must be configured with the Document Type(s) it is subscribing to AND the (handling) service that will receive the document.
4. What name should you give to the Document(s) on the Input of the trigger handling service (e.g. handleValidation)? **The name must be the fully qualified name of the publishable document type. The fully qualified name of a publishable document type conforms to the following format: folder.subfolder:PublishableDocumentTypeName**
   For example, in the exercise you configured the webMethods messaging trigger (listenForValidation) to listen for the acme.docs:Validation publishable document type and you specified the handling service of acme.work:handleValidation.
   On the Input/Output tab of the handleValidation service, the input signature must

contain a document reference named acme.docs:Validation.



Exercise 18: Create JDBC 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: JDBC Adapter Notifications

1. What is automatically created and updated when you work with your JDBC Adapter Notification? **The Notification's Publishable Document.**
2. What occurs when the JDBC Adapter Notification Schedule is enabled? **The Database is polled and the data corresponding to the Notification is collected.**
3. Why do you see the XML representation of your OrderHeader document in the IS Server Log? **Running the IS service insertOrderCanonical inserts a new record in the ORDER_HEADER database table and causes your Insert Adapter Notification orderCanonicalNotifier to publish the entire record as a JMS message. This message is received by the JMS Trigger you created in exercise 16. Its handling service converts the message into XML and logs it into the IS Server Log.**
4. Why is the JDBC Adapter Notification Schedule an administrative task? **Database Administrators like to know what happens to their databases and especially want to control repeating operations to their database.**

Exercise 20: Use Services in a Business Process

1. Why do you create swim lanes 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. Name 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 segments as a service (Simple Service Protocol).**