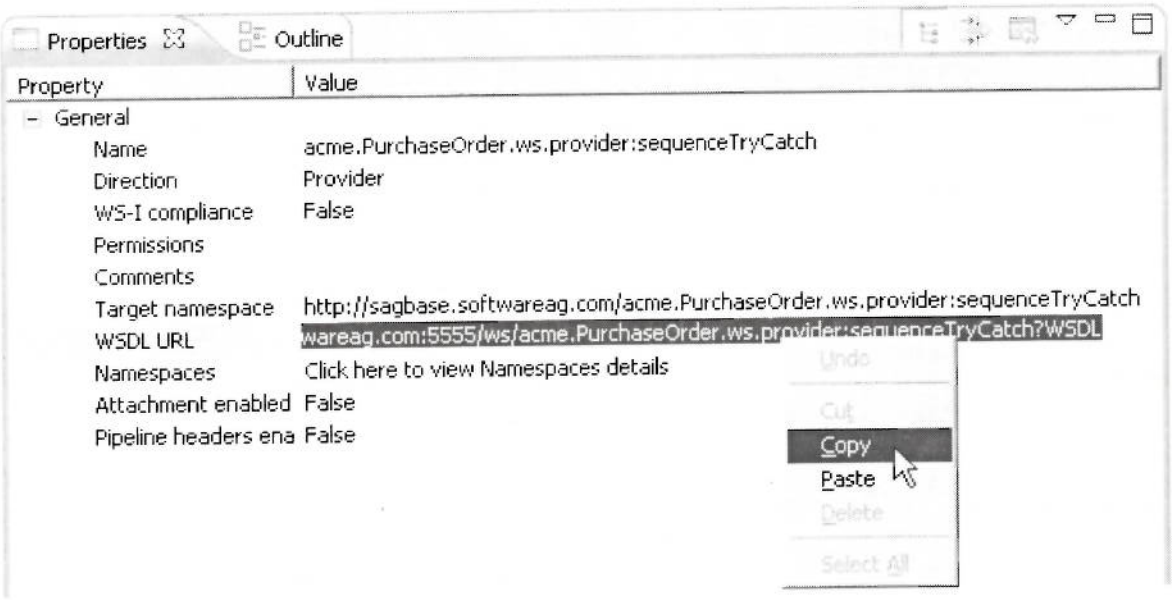
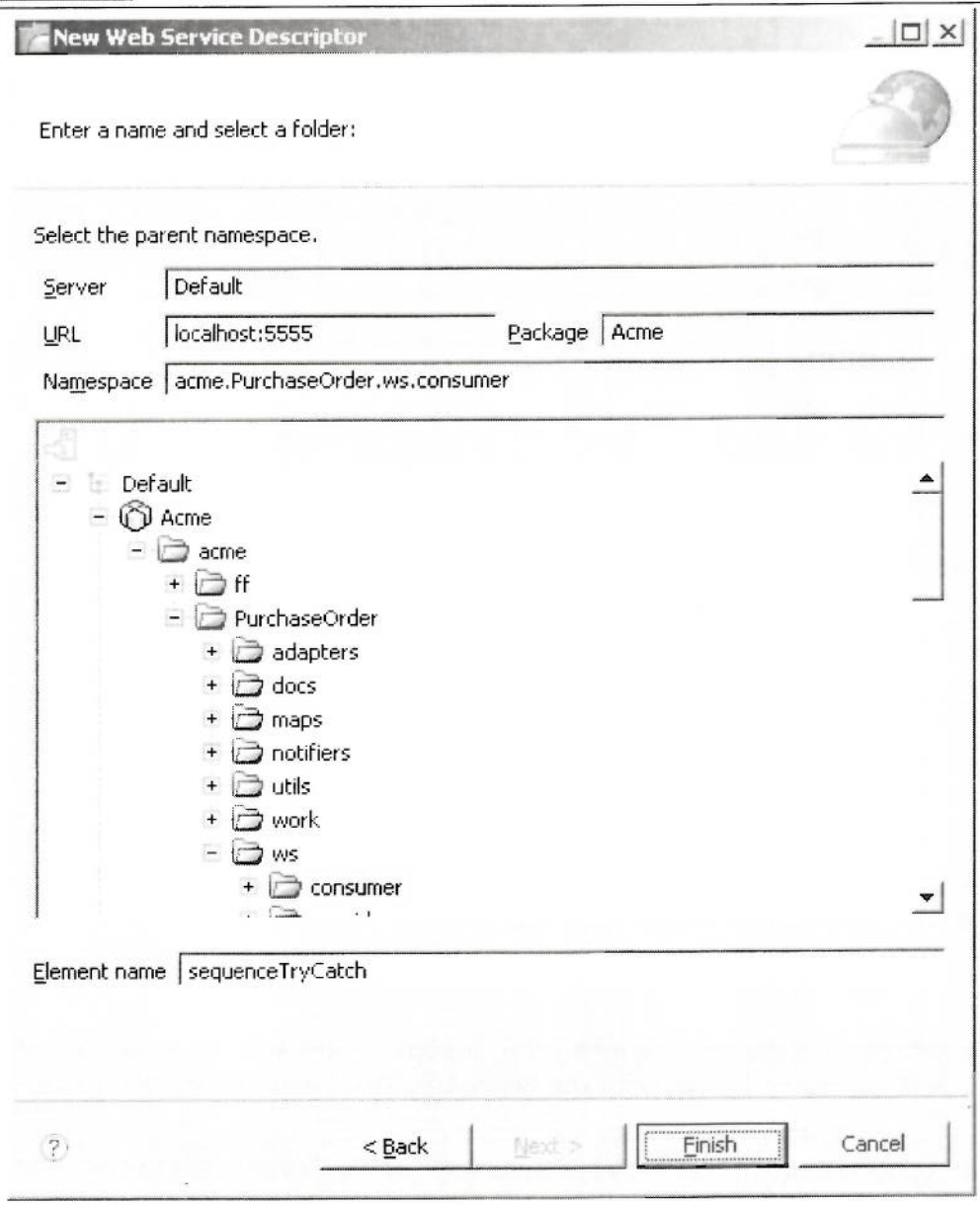
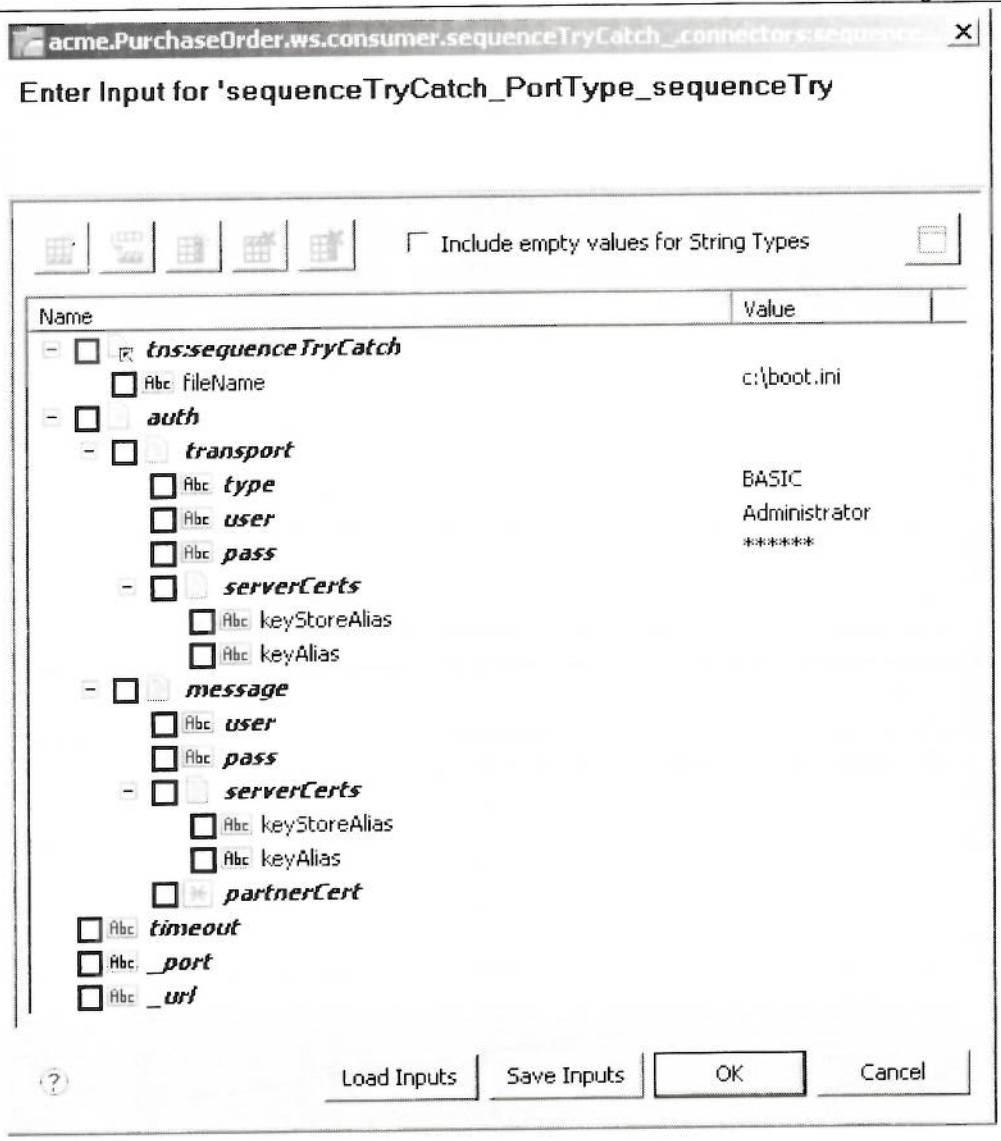
2. When the new **Provider WSD** appears in the editor, in the **Properties** panel, highlight the **WSDL URL** property value and copy it to the clipboard - you will need this value in one of the next steps.



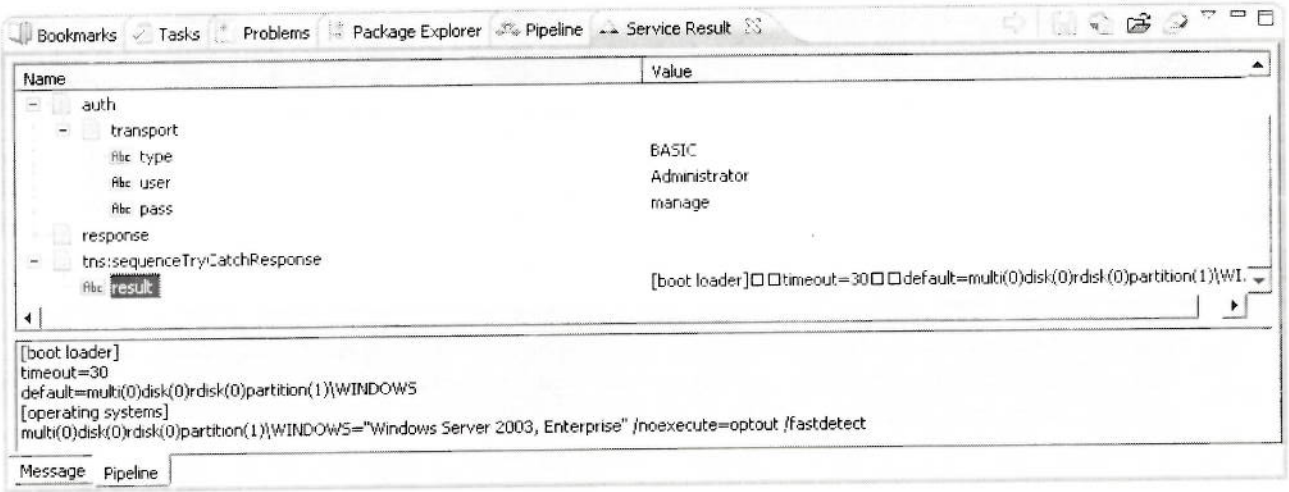
3. Open Internet Explorer, paste the **WSDL URL** in the IE's address field and hit return. The WSDL for the **sequenceTryCatch** web service should appear. This is the URL that consumers of the web service must use to access its WSDL.
4. Now create another Web Service Descriptor in the . **acme.PurchaseOrder.ws.consumer**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



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



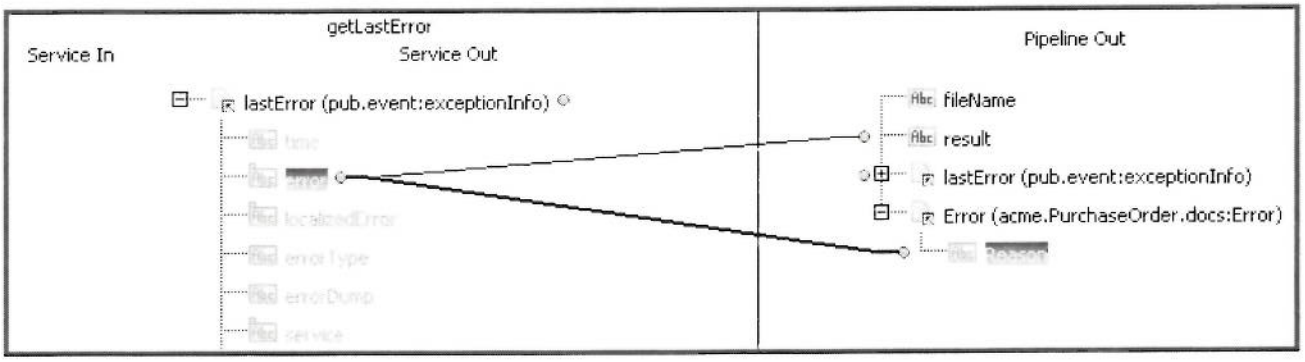
7. Review the results in Service Result view.



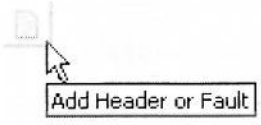
8. Now create a new Document Type called **acme.PurchaseOrder.docs:Error**. Add one string field to the **Error** document type and name it **Reason.**

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

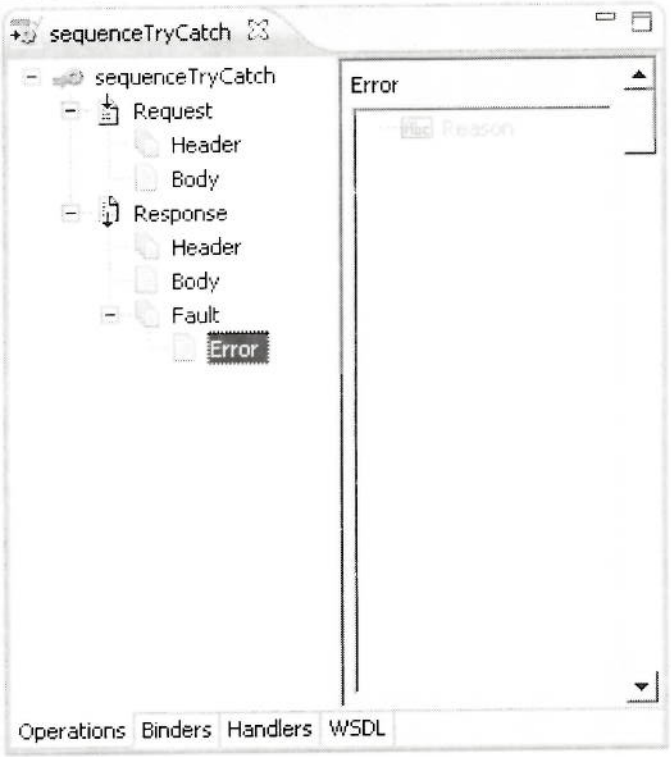
   a. In the service, open the **Tree** tab. Expand the service and select the **pub.flow:getLastError** step. Click on the **Pipeline** tab, select the **Pipeline Out** section, right click in it and select **Insert ➡ Document Reference**, then navigate to the **acme.PurchaseOrder.docs:Error** document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



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

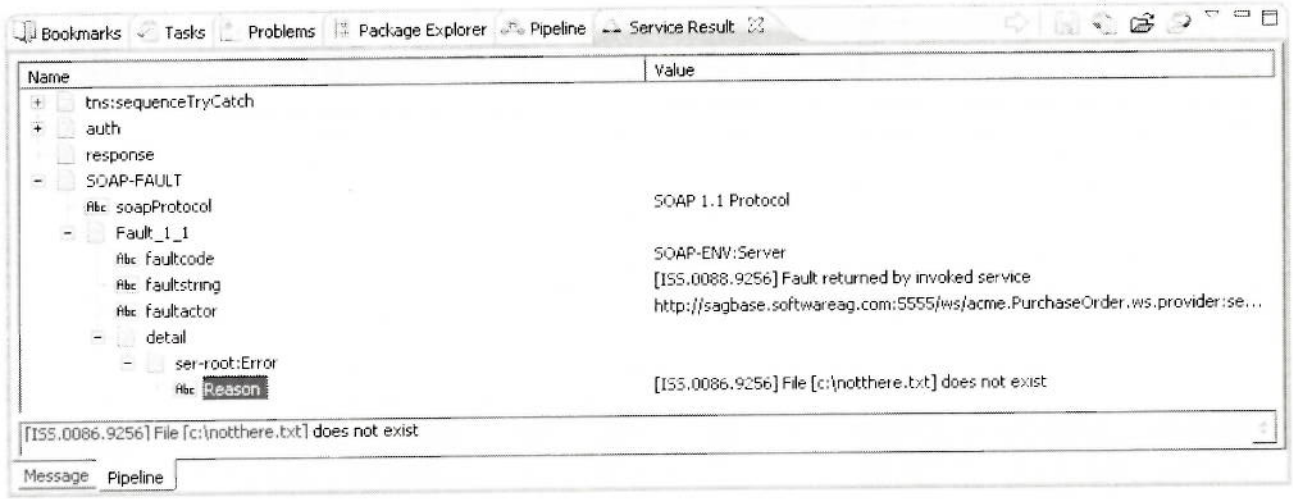


Note: This button is in the top icon bar of designer. In the popup dialog, navigate to the **acme.PurchaseOrder.docs:Error** document type, select it, and click the **OK** button. Your sequenceTryCatch provider should look like the following:



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

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



Check Your Understanding

1. When would you create a Provider WSD when a Consumer WSD?
2. How and when are WSC's created? After you have the Provider WSD.
3. Can you have more than one custom SOAP Fault Document? Yes, because the fault object is a document list

This page intentionally left blank

Exercise 16:
Broker Pub/Sub

Overview

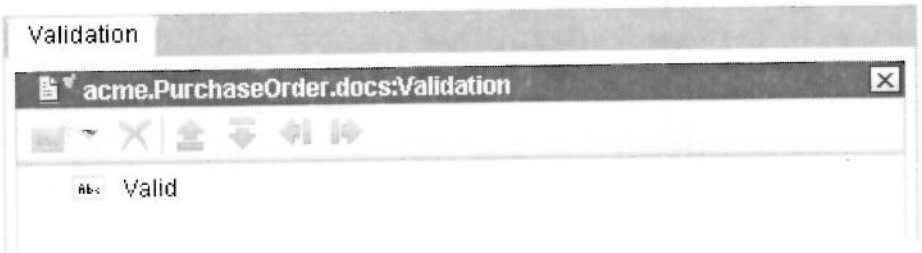
In this exercise, you will create a document type, a handling service and a subscribing Broker trigger. Then you create a service to publish the document.

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

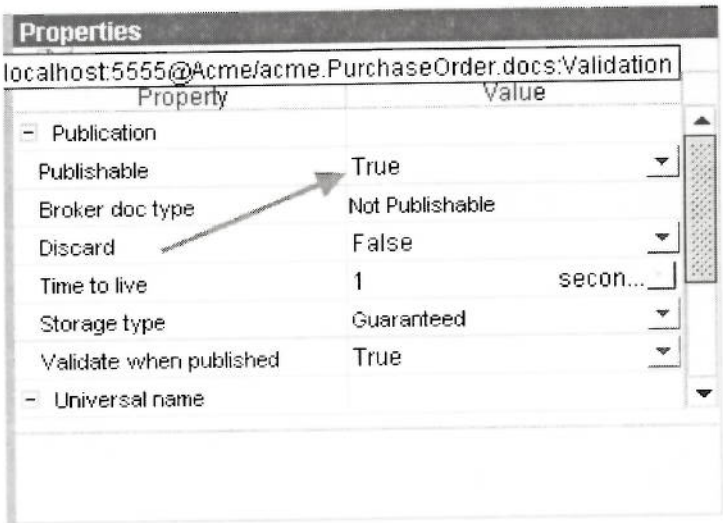
Steps

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

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

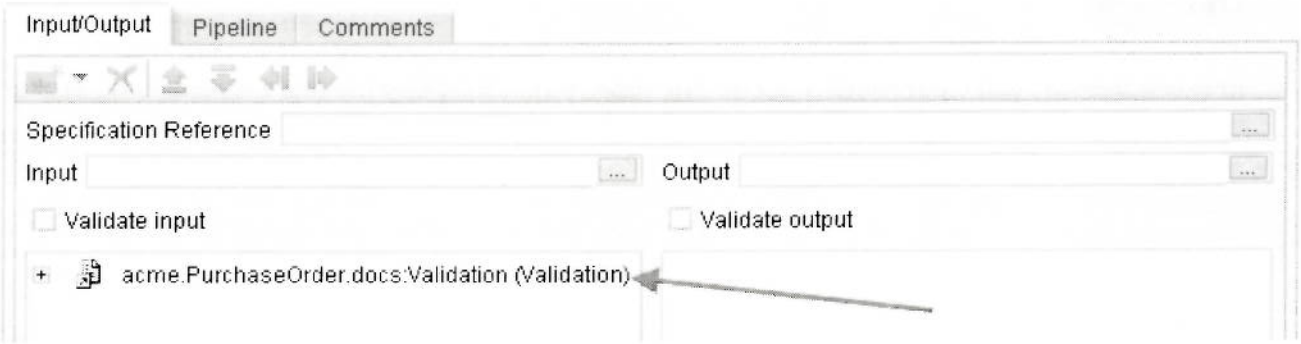


2. Make this document **publishable** to the Broker by setting the **Publishable** property for the document to **true**.

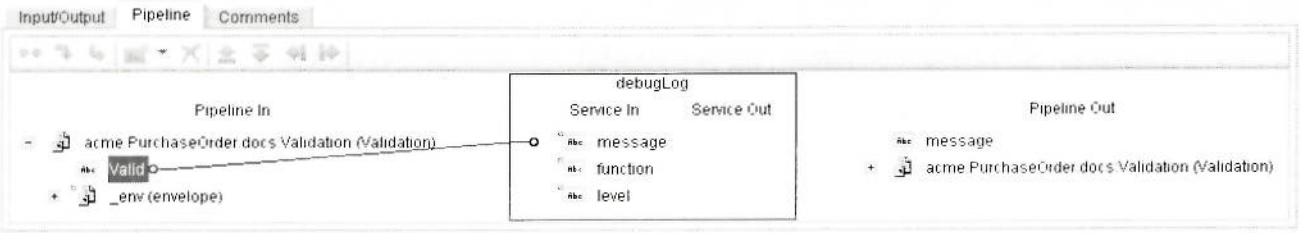


Note: if you do not see the publishable property, make sure you are viewing the properties of the Validation document itself.Open the Documemt Type in the editor and then double-click the editor's title bar of the document type to see its properties.

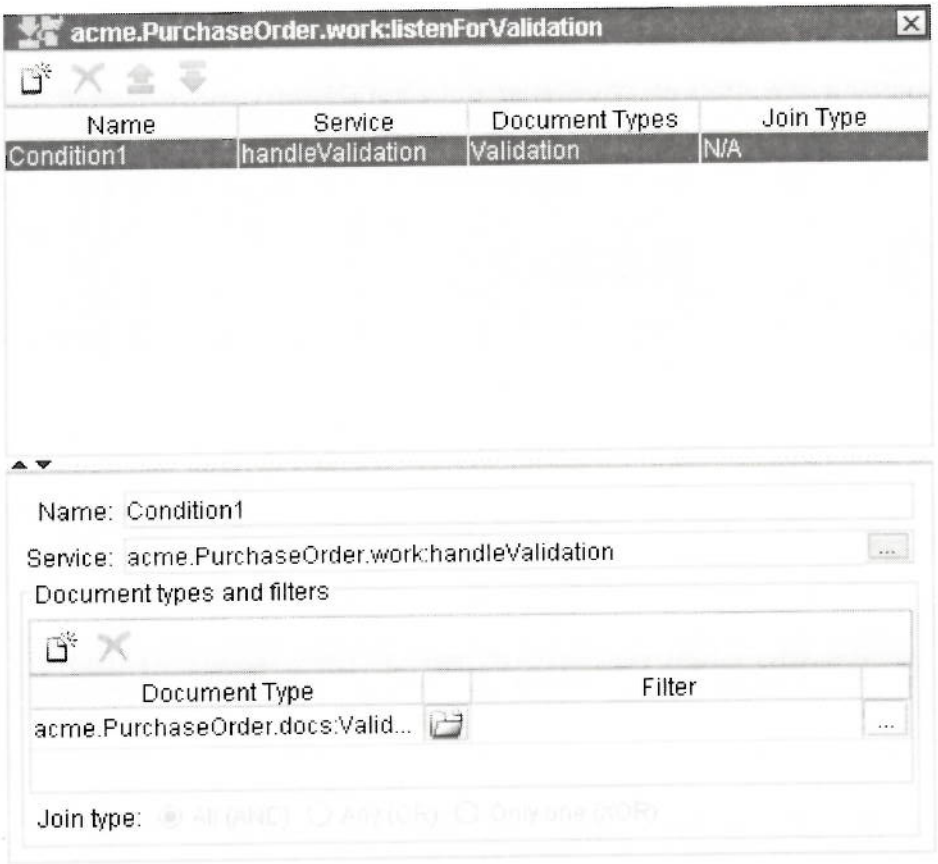
3. Create a new Flow service **acme.PurchaseOrder.work:handleValidation**. Set the input of this service to be a document reference to your document **acme.PurchaseOrder.docs:Validation**. Set the name of the document reference to be the fully-qualified name of the document type: **acme.PurchaseOrder.docs:Validation.**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



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

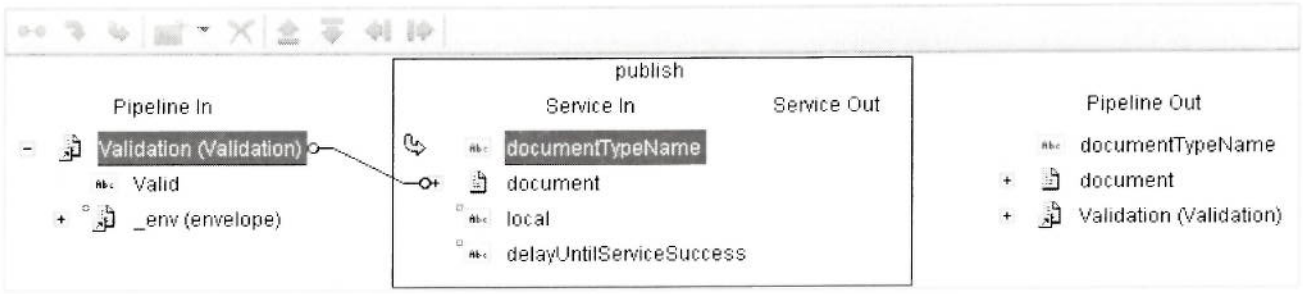


5. In the **acme.PurchaseOrder.work** folder, create a new Broker/Local Trigger, name it **listenForValidation**. Configure the trigger as follows:
   a. Name = Condition1
   b. Service = acme.PurchaseOrder.work:handleValidation (you may use copy and paste here as well)
   c. Document type = acme.PurchaseOrder.docs:Validation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

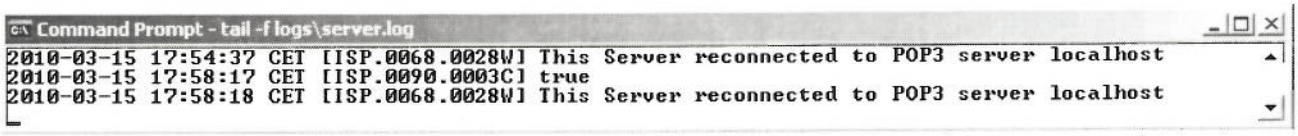


6. Save the trigger. If you want to test you work so far, double-click the **acme.PurchaseOrder.docs:Validation** document in the Navigation view and click the run button. Enter some message int the **Valid** field, leave all other fields unchanged and click next. Then select "Publish to the Broker"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

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



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



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

Check Your Understanding

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

Exercise 17:
JMS Pub/Sub

Overview

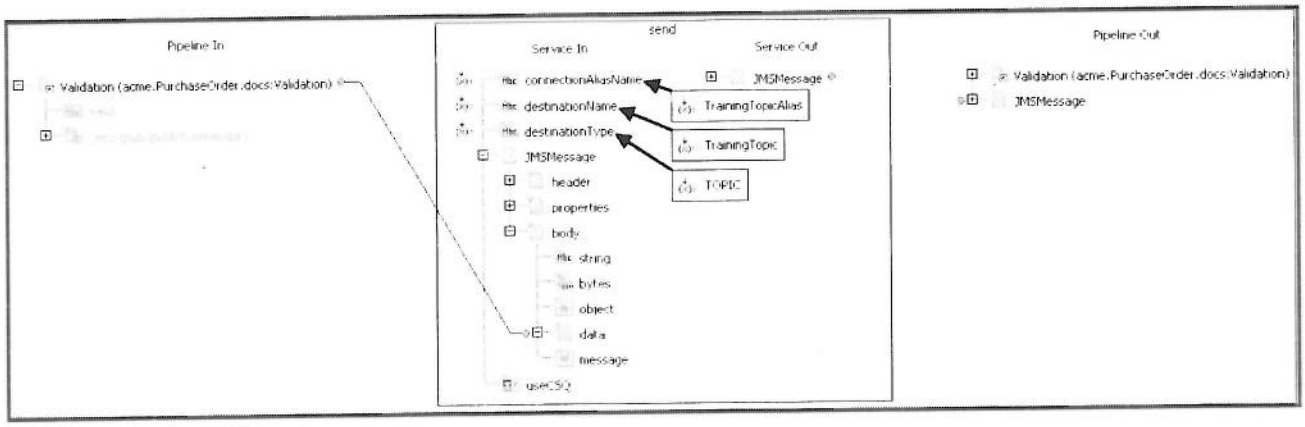
In this exercise, you will create a service to publish an existing document via an existing JMS Topic. Then you create a handling service and a subscribing JMS trigger to receive the document instance.

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

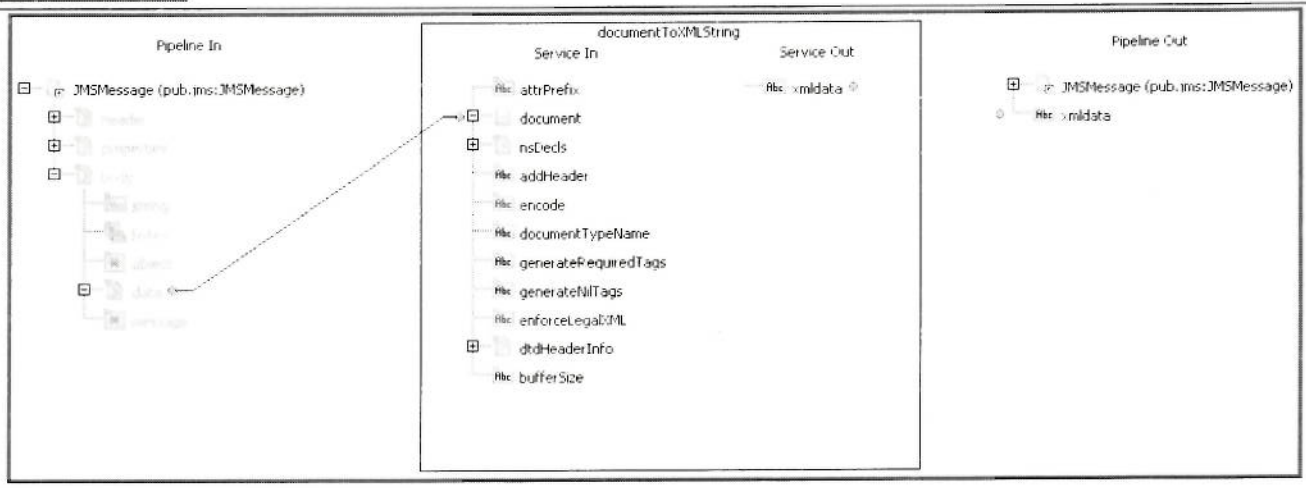
Steps

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

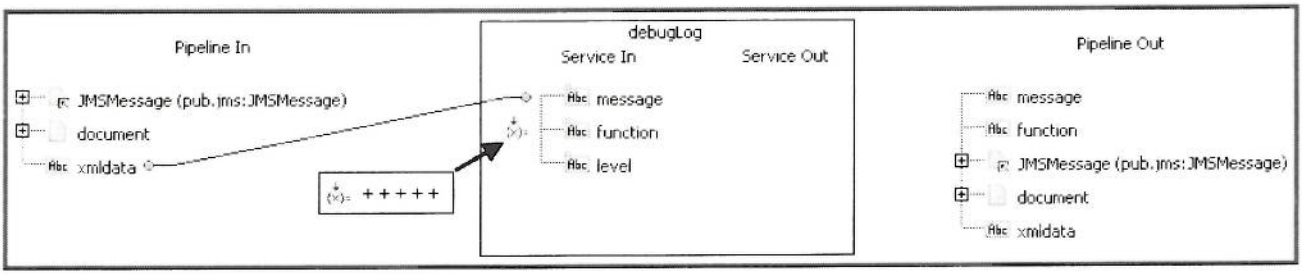
   When you are done editing your service, save the **publishValidationJMS** service.



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



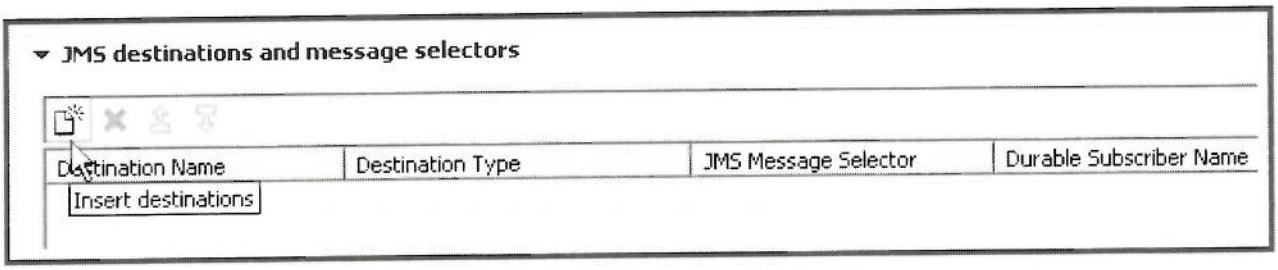
5. Next add an invocation of **pub.flow:debugLog** to the service and map **xmldata** to **message**. As our eyecatcher set the value of **function** to "+ + + + + ". Of course, you do not enter the Quotes.



Your finished service should look like this:

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

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

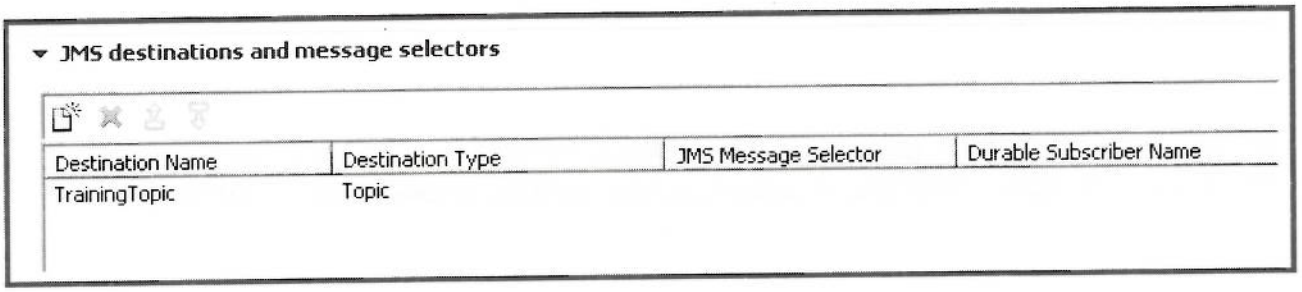


Then enter the following:

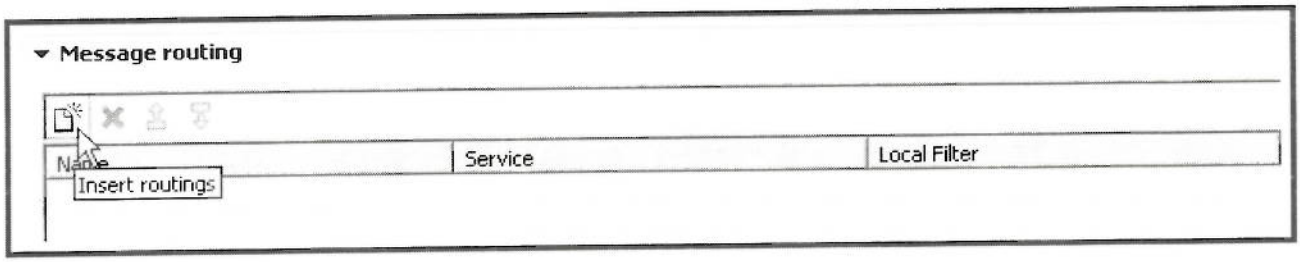
a. Destination name = TrainingTopic. If this is not in dropdown, you have to create it by selecting the "create new destination" button.

b. Destination Type = Topic

Leave all other fields empty



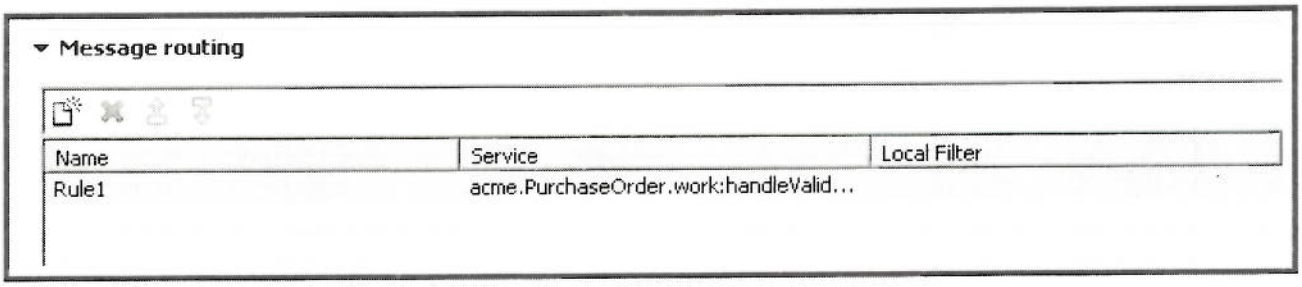
8. Configure the trigger **Message routing** by clicking the **Insert routings** button.



Then enter the following:

a. Name = Rule1

b. Service = acme.PurchaseOrder.work:handleValidationJMS



Then save your trigger.

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

Check Your Understanding

1. What is a topic versus a queue?

a topic allows consumes to subscribe to all the messages a topic may have while a queue is more point to point message where one consume may receive a message and oth consumes on not.

Exercise 18:
Create Adapter Services

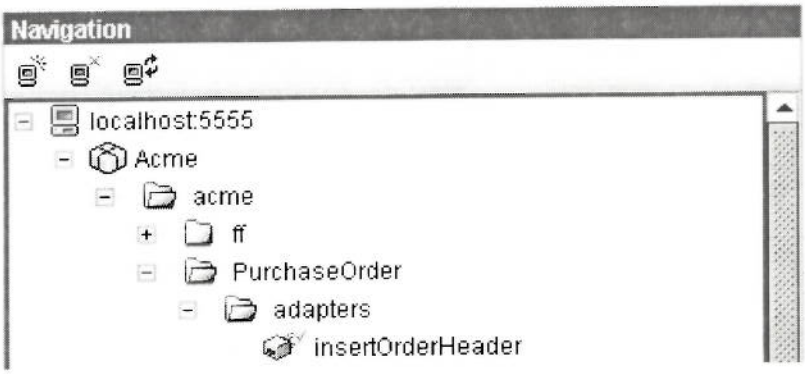
Overview

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

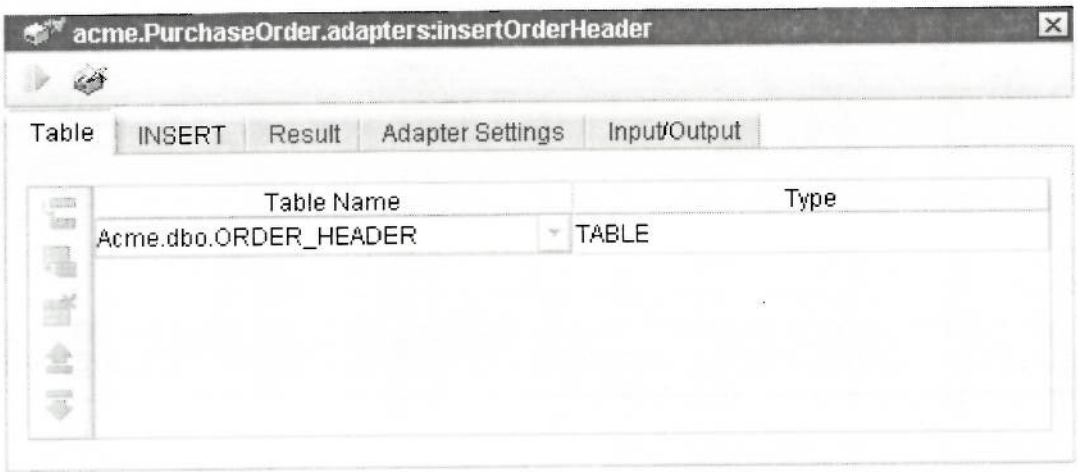
This exercise will be done with Developer.

Steps

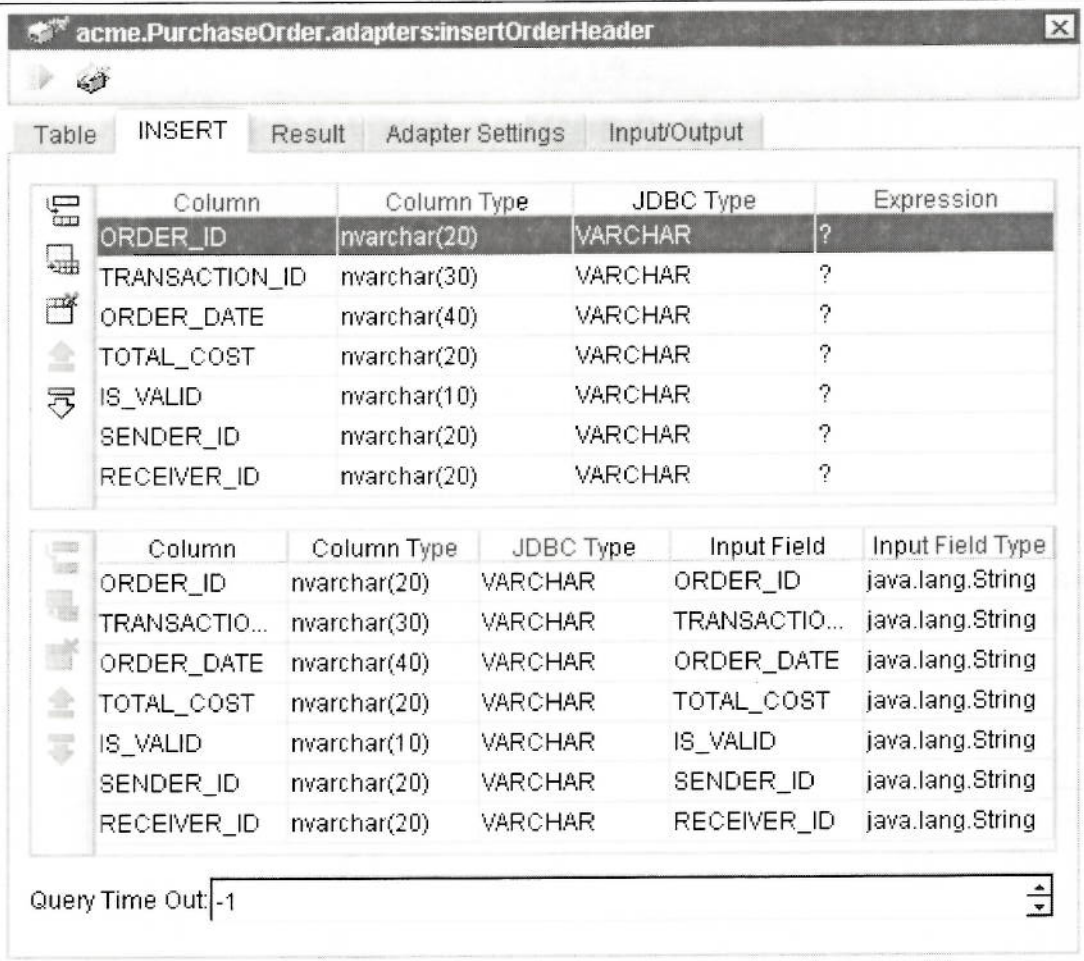
1. Start Developer and create a new Adapter Service in the **acme.PurchaseOrder.adapters** folder. Specify this will be an adapter service, of type **JDBC Adapter**. Then choose the existing **commonSupport.adapters.acmeAdapter** as Adapter Connection Name. Finally select the **InsertSQL** as a template,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



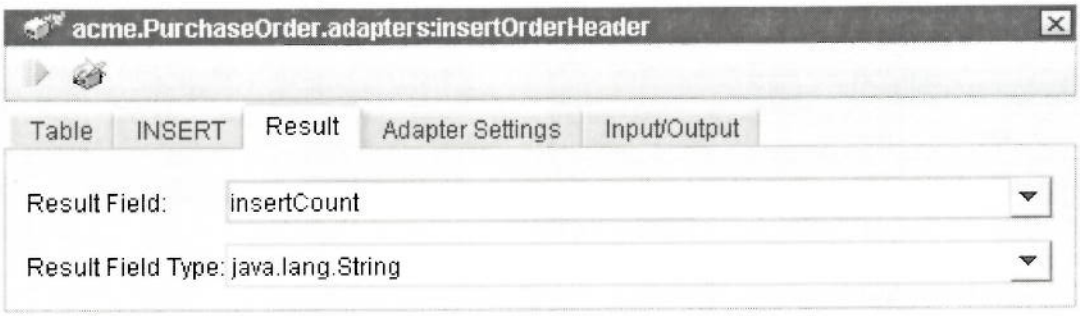
   a. On the Table tab, select **Acme.dbo.ORDER_HEADER**.



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

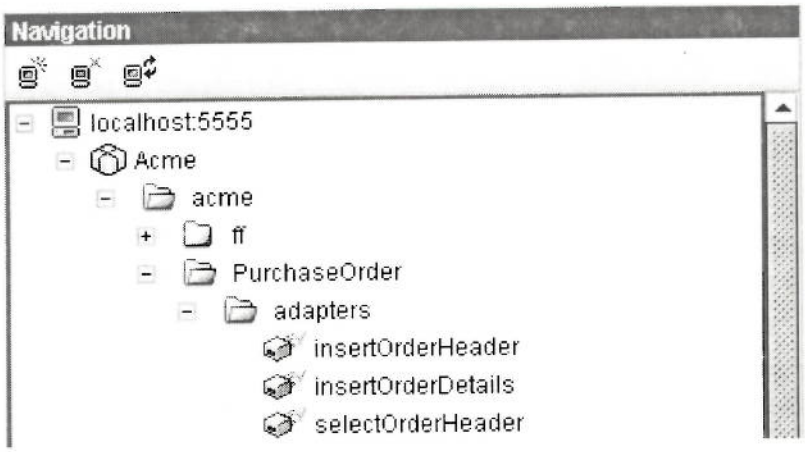


c. On the Result tab, set Result Field to be **insertCount** and Result Field Type to be **java.lang.String**.

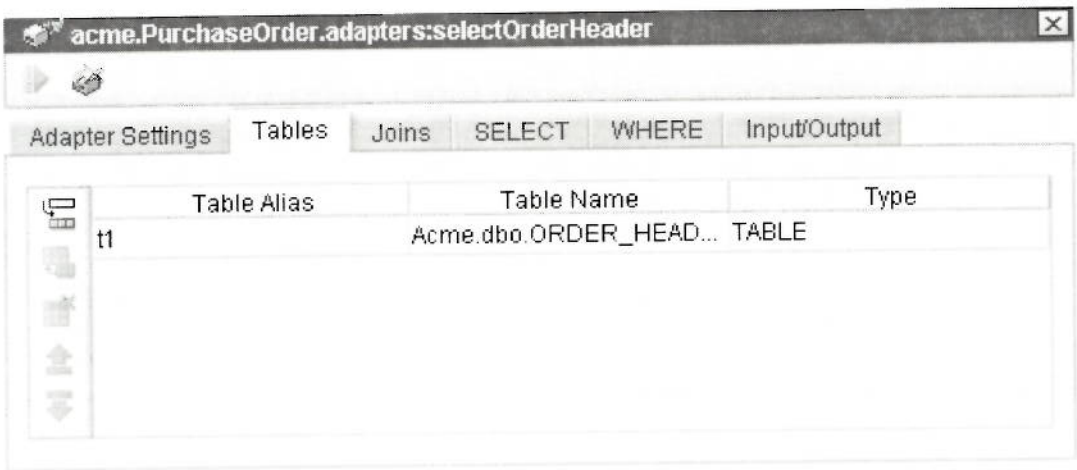


2. Save and run the **insertOrderHeader** service. Insert any data (but make sure to insert for every field) and confirm that the **insertCount** returns with a value of 1. Do not provide overrideCredentials or a $connectionName.

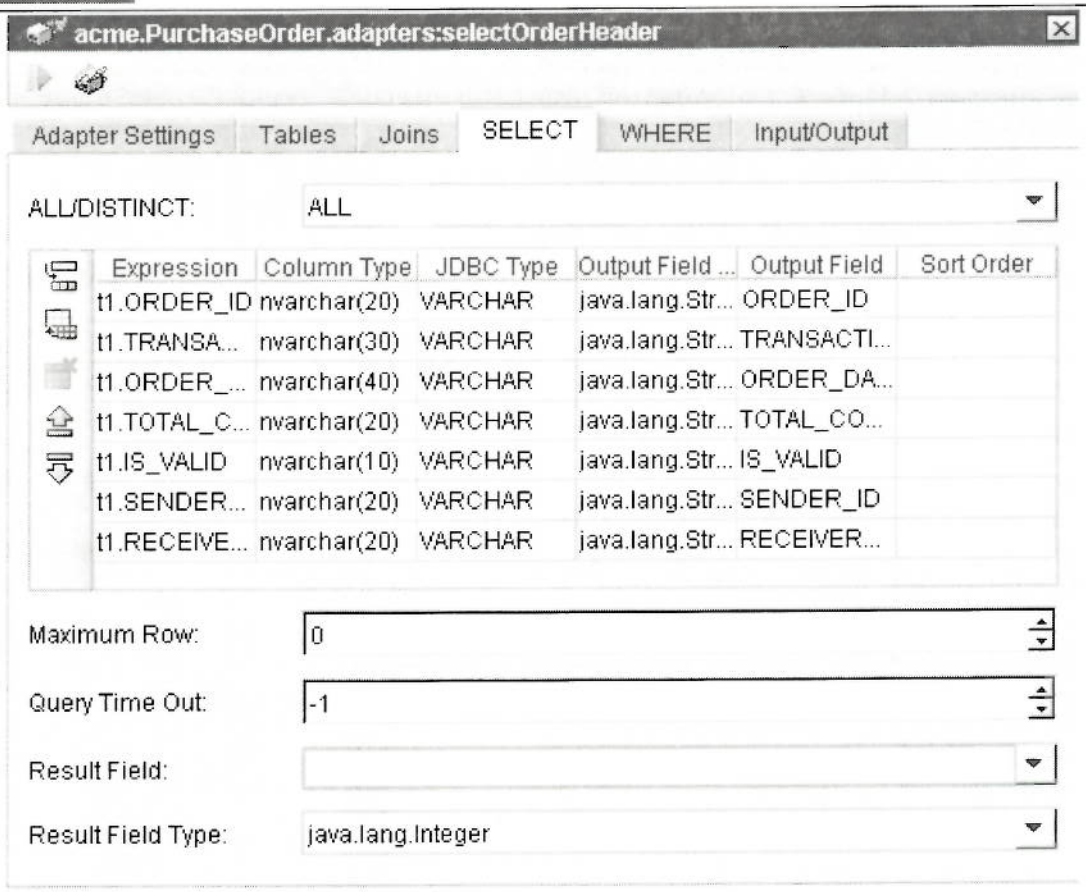
3. Like you did in step 1, create another Adapter Service in the **acme.PurchaseOrder.adapters** folder. Specify this will be an **adapter service**, of type **JDBC Adapter**, using **commonSupport.adapters.acmeAdapter**, using the **InsertSQL** template, and name it **insertOrderDetails**.
   a. On the Table tab, select **Acme.dbo.ORDER_DETAILS**.
   b. On the Insert tab, click the **Fill in all rows** button.
   c. On the Result tab, set Result Field to be **insertCount** and Result Field Type to be **java.lang.String**.
4. Save and run the **insertOrderDetails** service. Insert any data (but make sure to insert for every field) and confirm that the **insertCount** returns with a value of 1. Do not provide overrideCredentials or a $connectionName.
5. Create a new Adapter Service in the **acme.PurchaseOrder.adapters** folder. Specify this will be an **adapter service**, of type **JDBC Adapter**, using **commonSupport.adapters.acmeAdapter**, using the **SelectSQL** template, and name it **selectOrderHeader**.



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

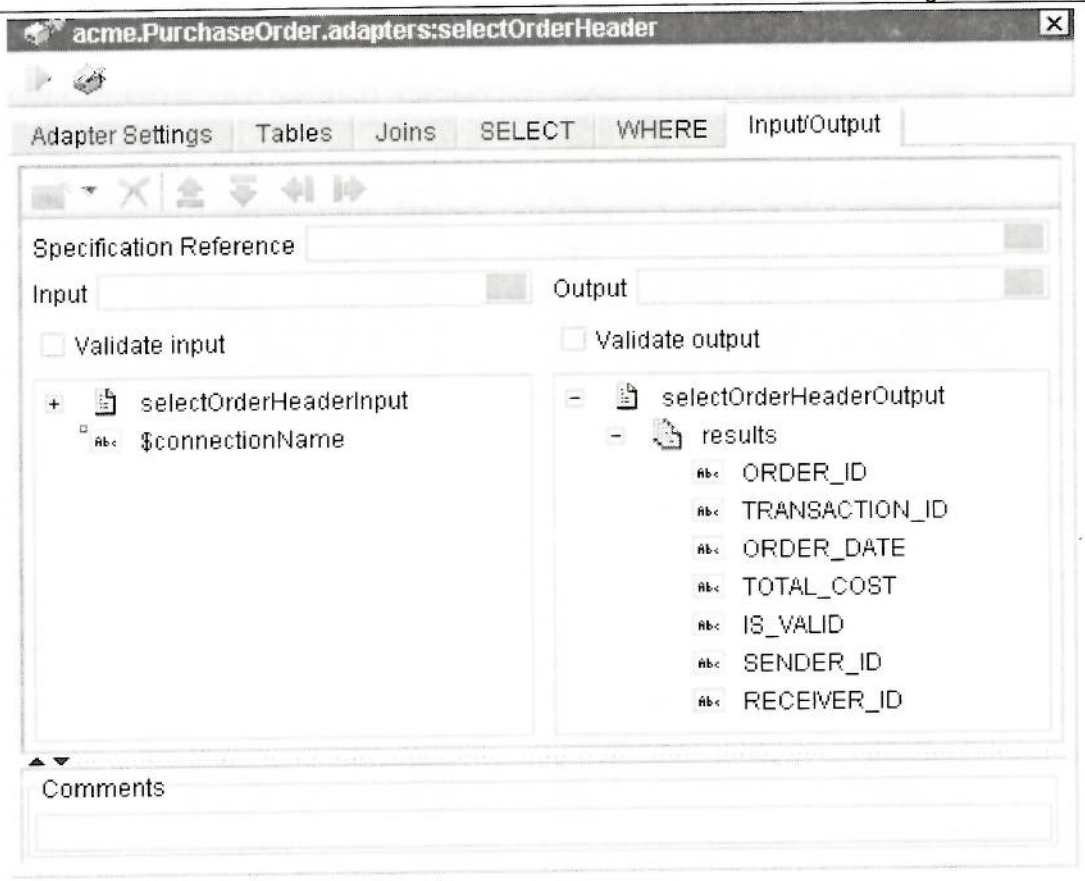


   b. Skip the Joins tab
   c. On the Select tab, specify **ALL**, and click **Fill in all rows**

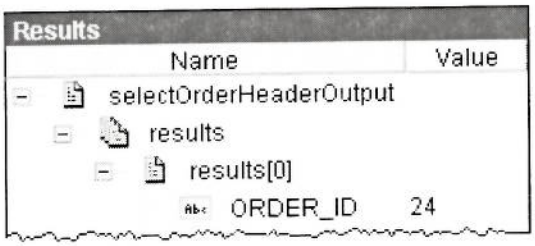


d. Skip the Where tab

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

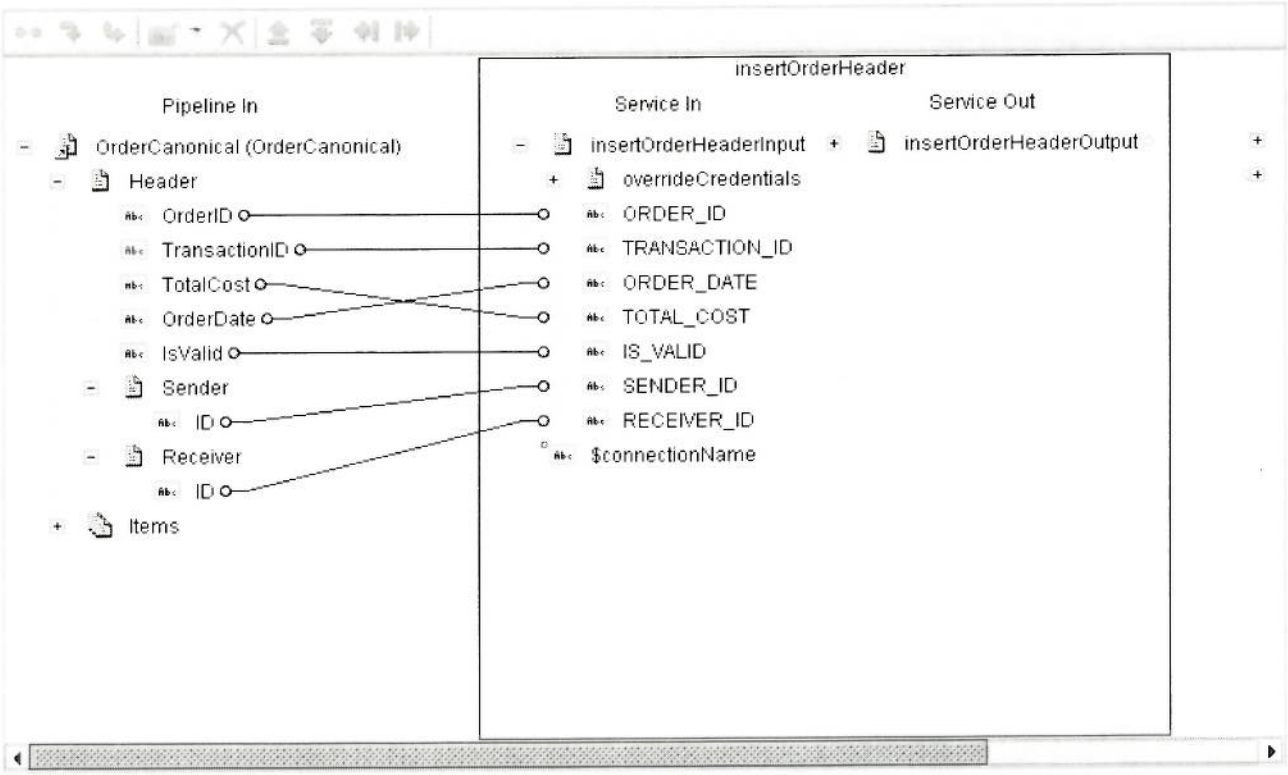


6. Save and run the **selectOrderHeader** service and confirm that the database table was populated with your data from Step 2.



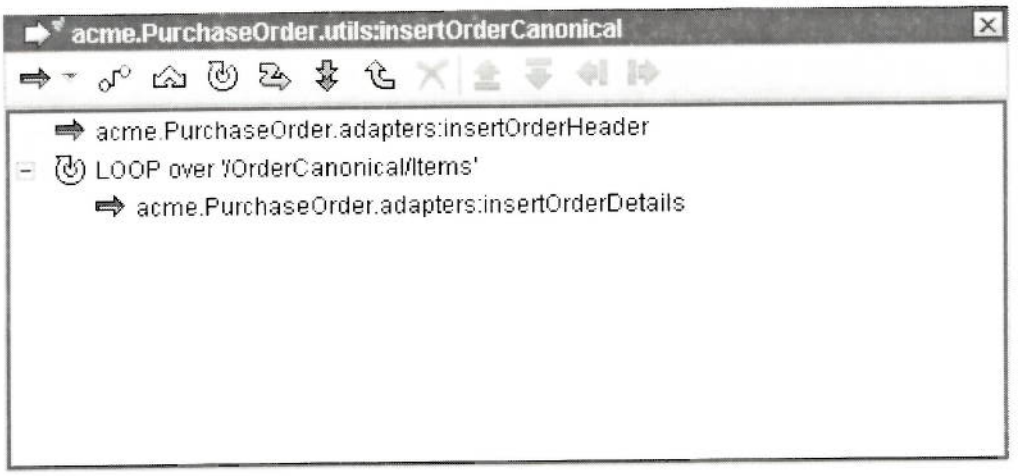
7. Create another Adapter Service in the **acme.PurchaseOrder.adapters** folder. Specify this will be an **adapter service** which is of type **JDBC Adapter** and is using **commonSupport.adapters.acmeAdapter.** Then choose the **SelectSQL** template and name the Adapter Service **selectOrderDetails**.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

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



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

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



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

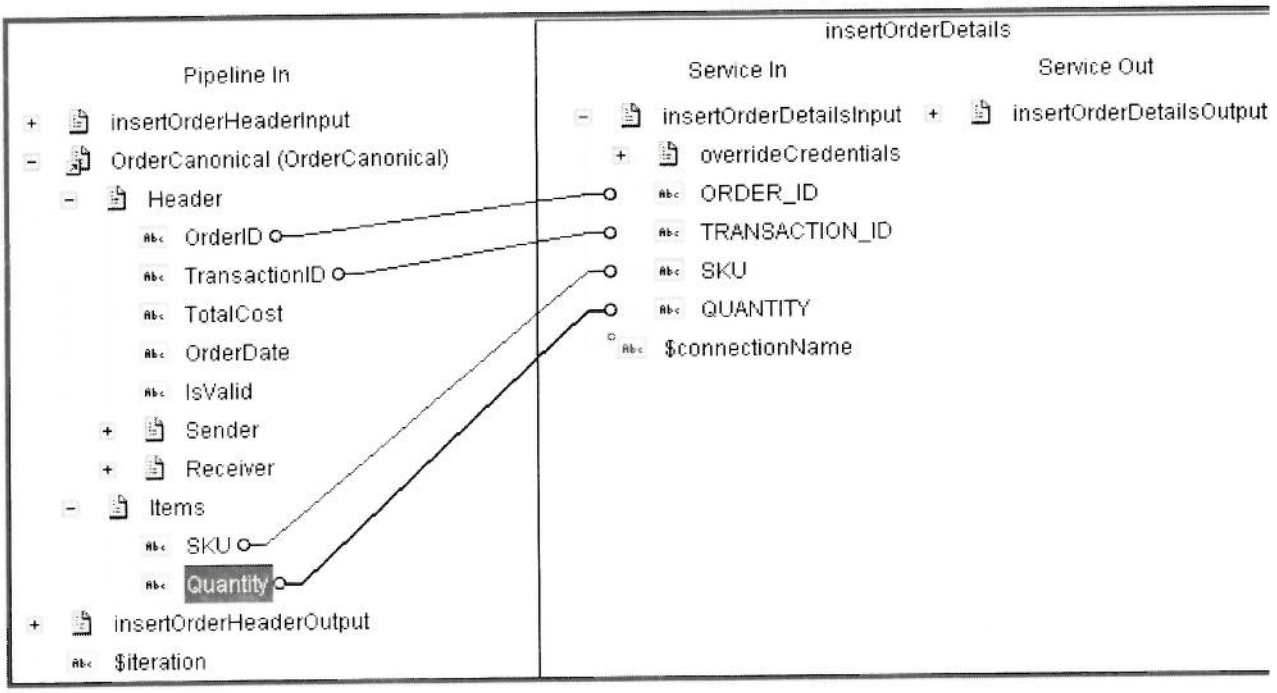
Then map as follows:

i. OrderCanonical/Header/OrderID to insertOrderDetailsInput/ORDER_ID

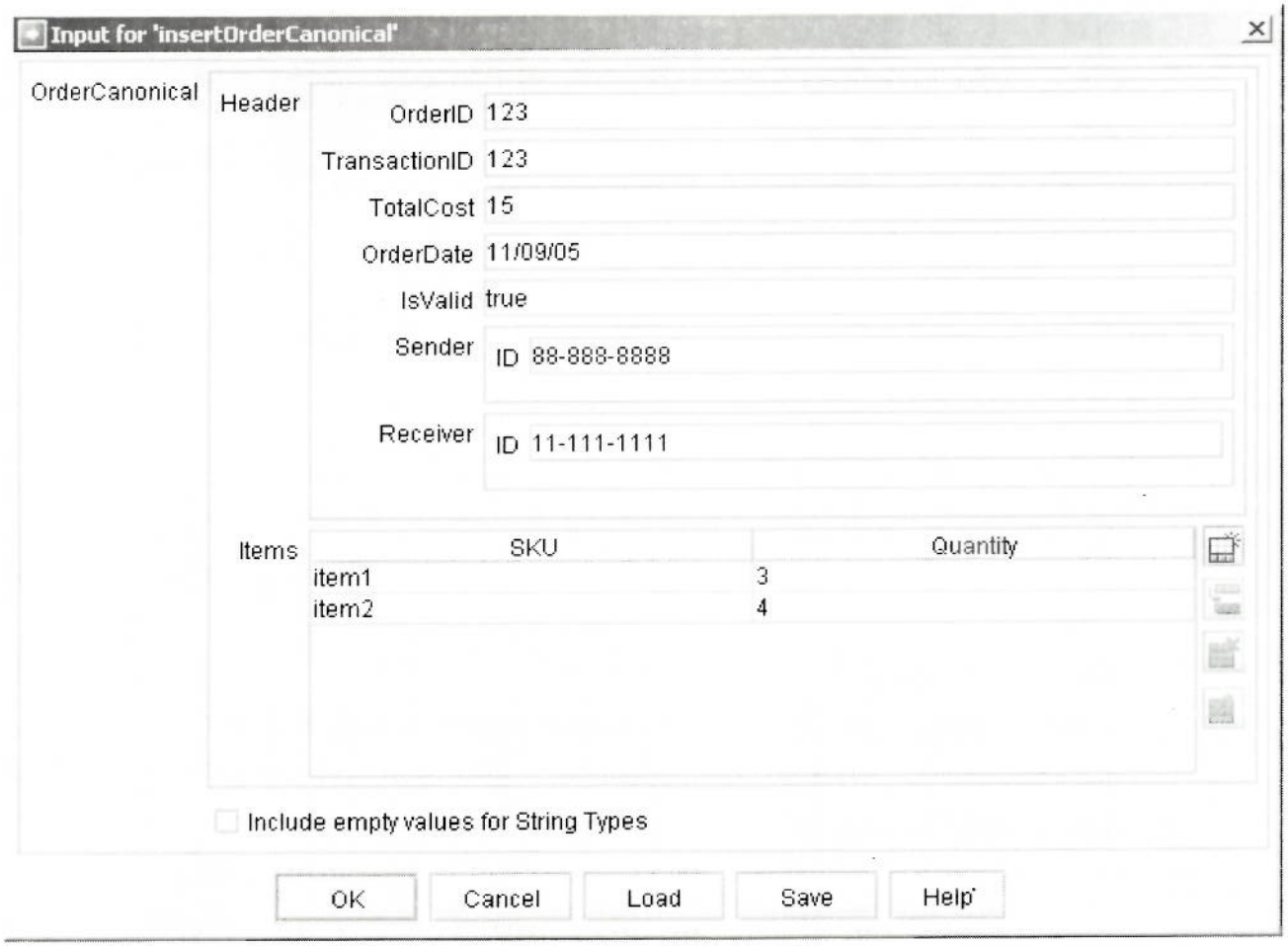
ii. OrderCanonical/Header/TransactionID to insertOrderDetailsInput/TRANSACTION_ID

iii. OrderCanonical/Items/SKU to insertOrderDetailsInput/SKU

iv. OrderCanonical/Items/Quantity to insertOrderDetailsInput/Quantity



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



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

Check Your Understanding

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

① Configuring Adapter Connection within IS

② Because OrderCanonical document had a set of Items so we wanted the mapping ~~the loop~~ but the insert adapter service that it one document instead of an array of document. By placing the label on the Loop, it makes the mapping seems like one document instead of a Document List in which adapter is not looking for.

Exercise 19:
Adapter Notifications

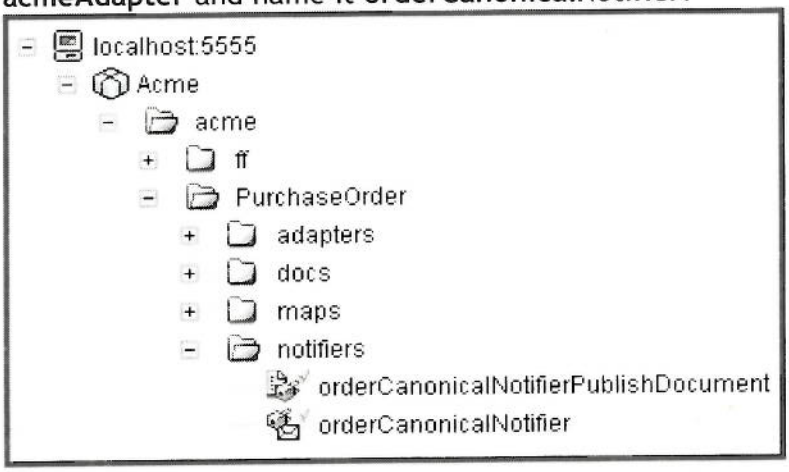
Overview

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

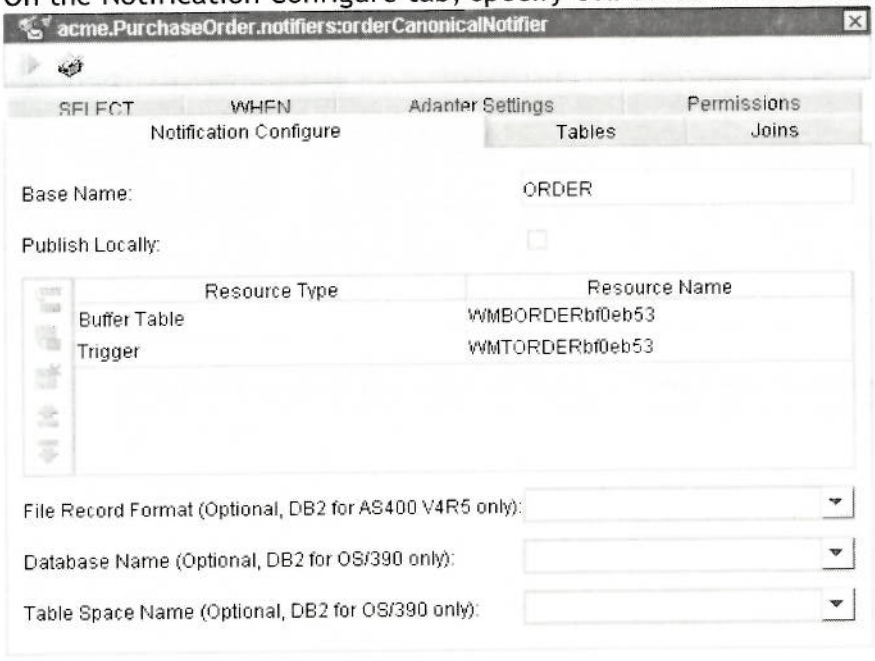
This exercise will also be done using Developer.

Steps

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



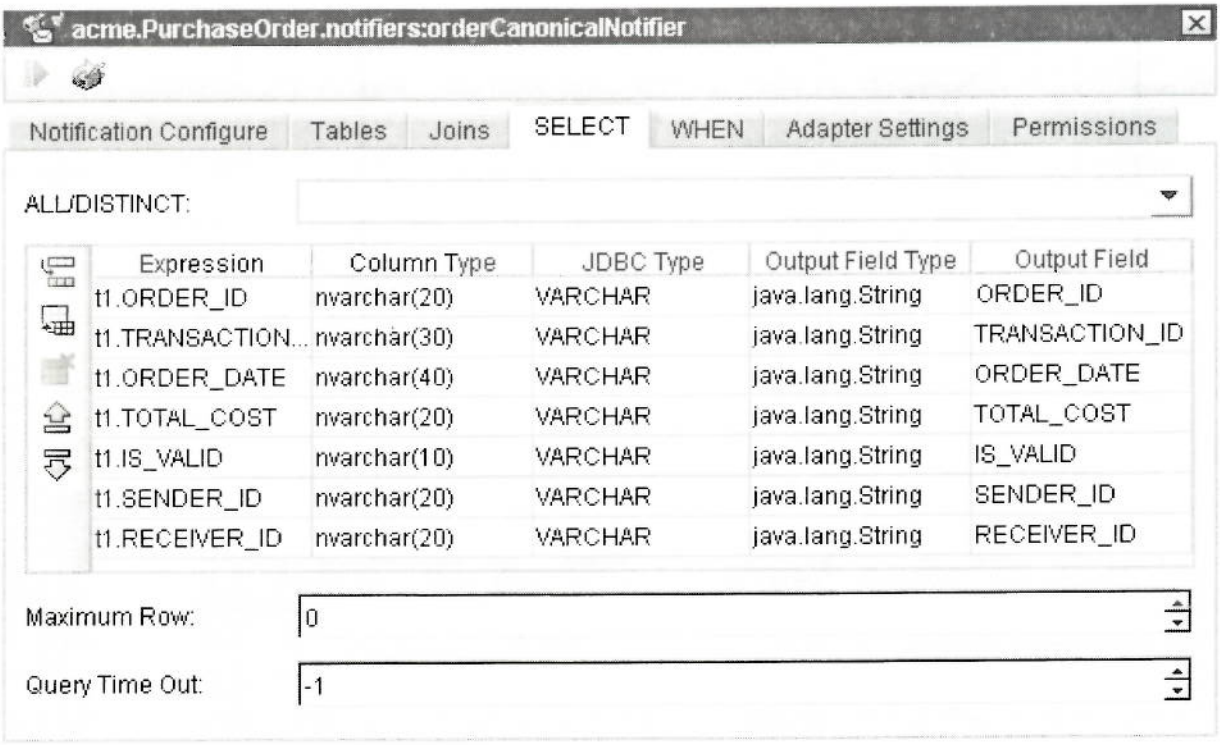
   a. On the Notification Configure tab, specify **ORDER** as the **base name.**



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

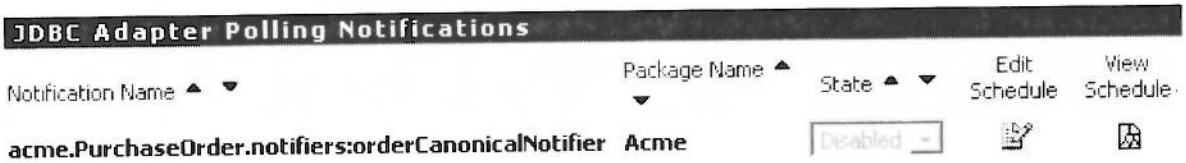
Skip the Joins tab.

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



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

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

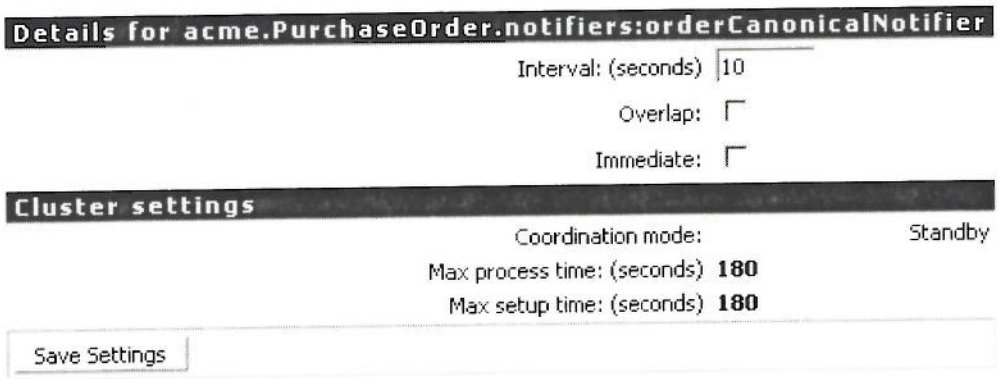


3. Edit your notification schedule by clicking the **Edit Schedule** icon for your notification service. Specify the following parameters and then select **Save Settings**:

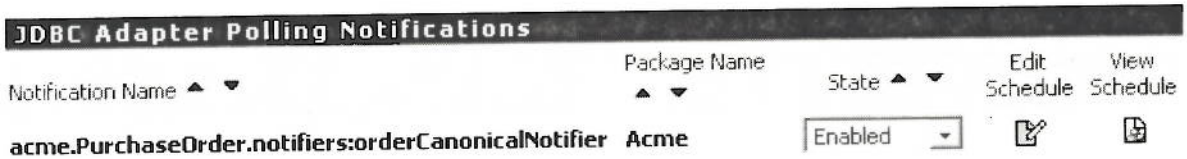
   a. Interval = 10

   b. Overlap = *unchecked*

   c. Immediate = *unchecked*



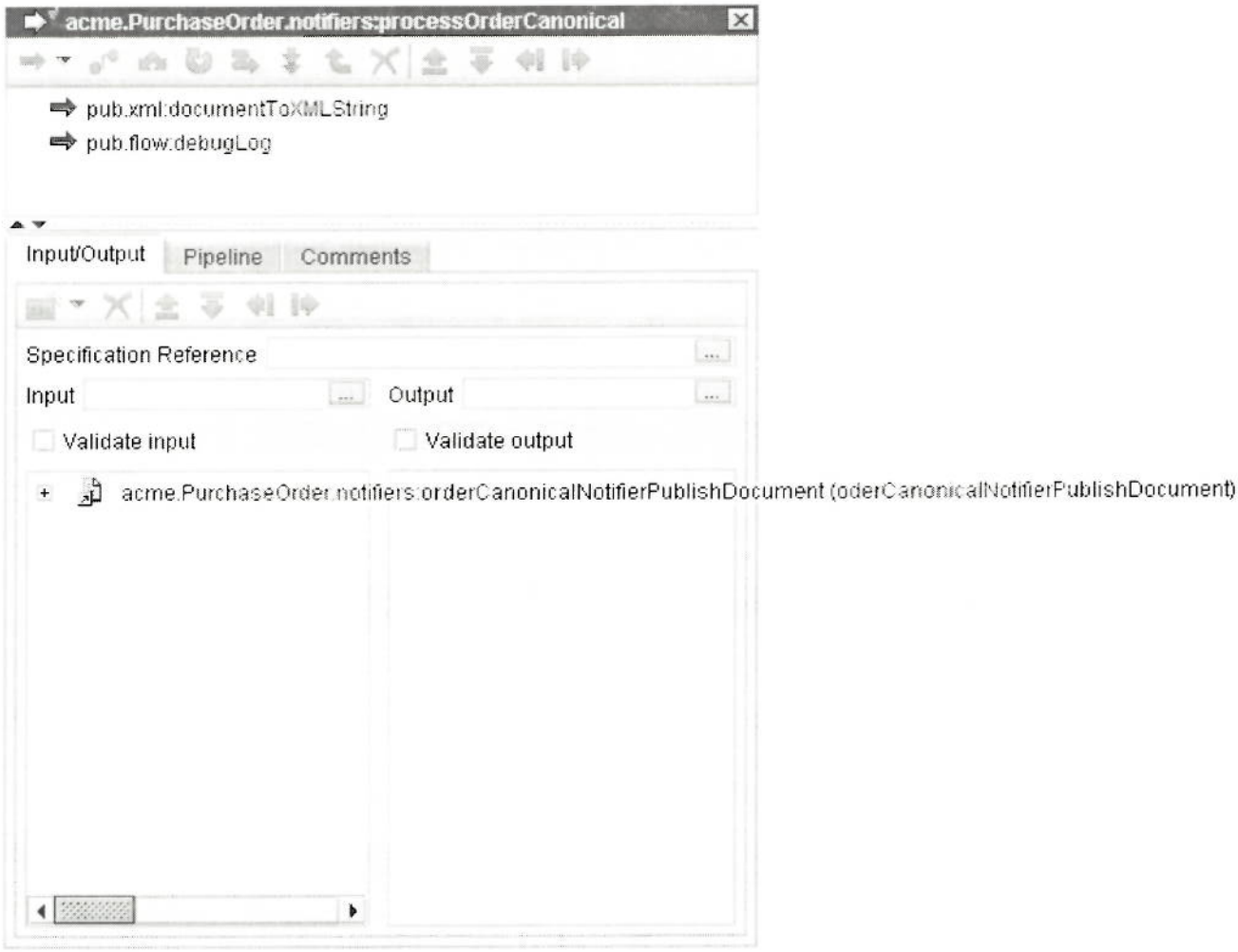
4. Enable the notification schedule.

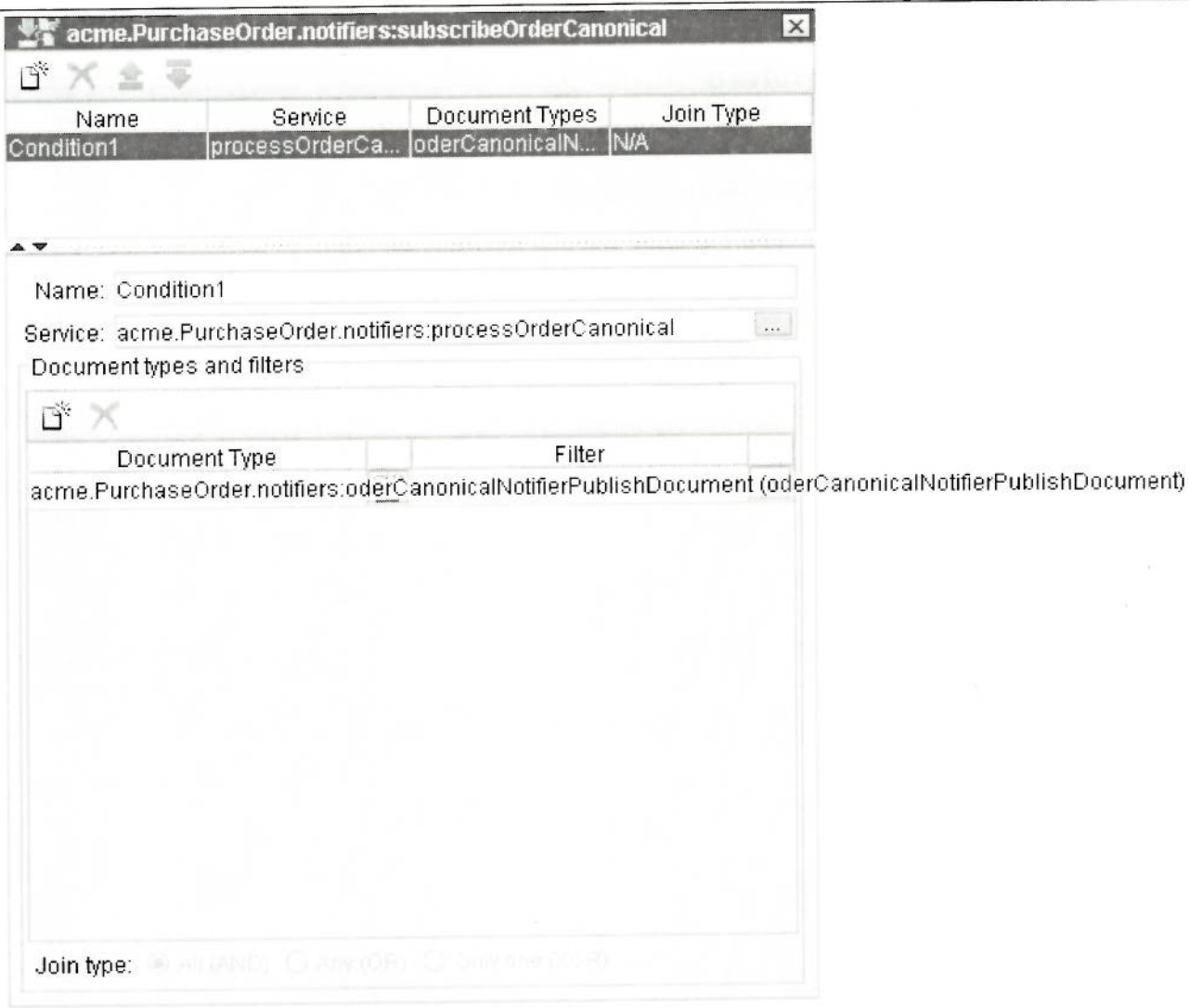


5. As in "**Exercise 16: Broker Pub/Sub**" create a handling Service called "**acme. Purchase Order.notifiers: processOrderCanonical**" and a Trigger called "**acme. PurchaseOrder. notifiers:subscribeOrderCanonical**". Equivalent to Exercise 16 the service receives a document reference to the "**acme.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

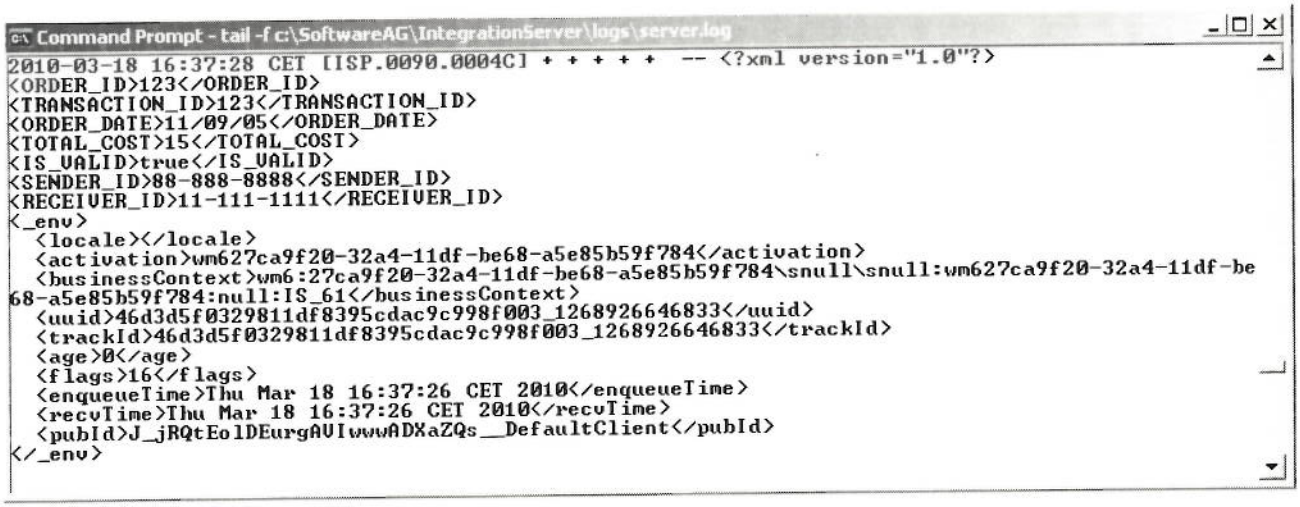
   Use the **documentToXMLString** to convert the document to an XML representation and the **debugLog** service to display the XML document.

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





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



Check Your Understanding

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

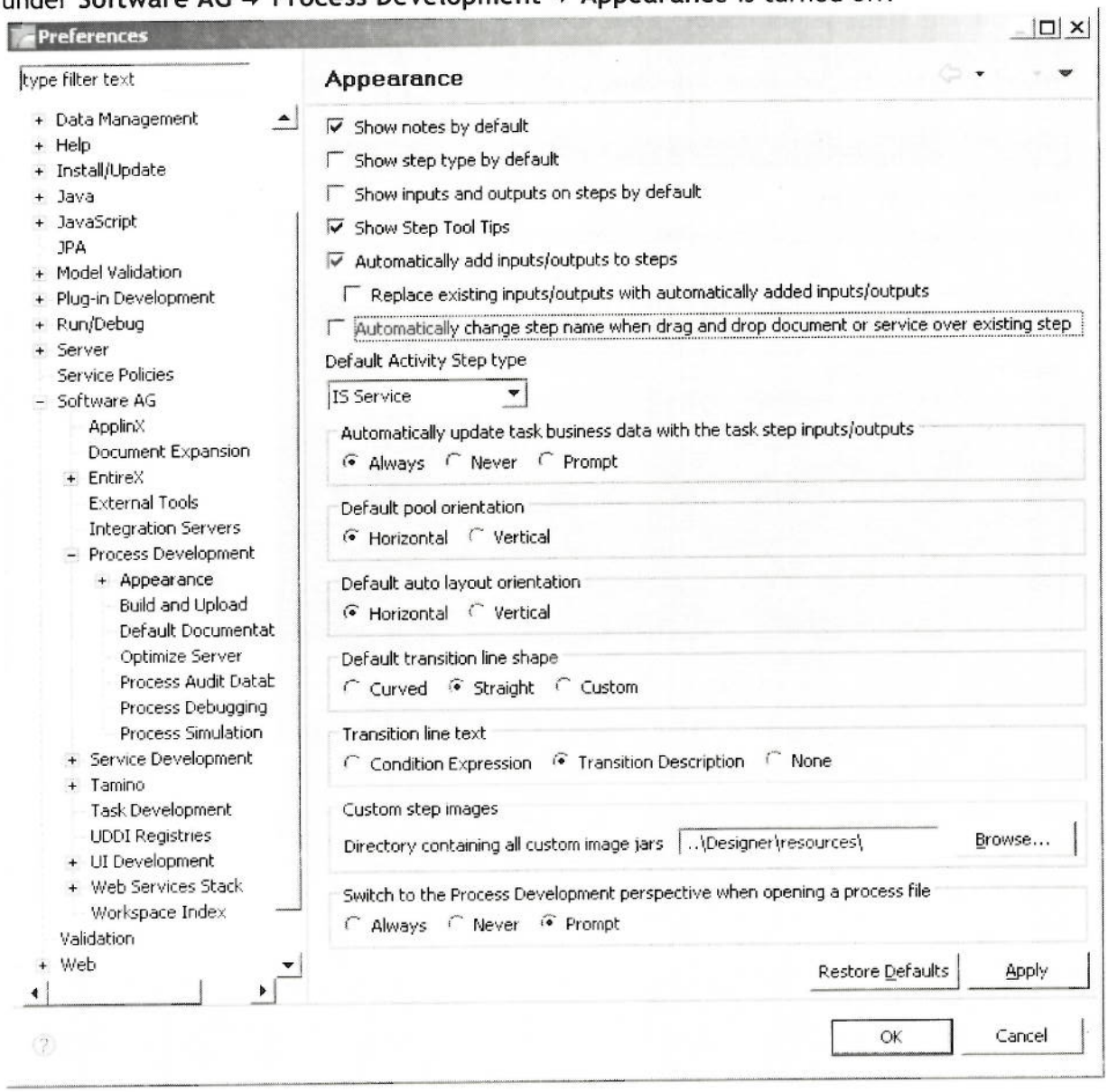
Exercise 20:
Use Services In a Business Process

Overview

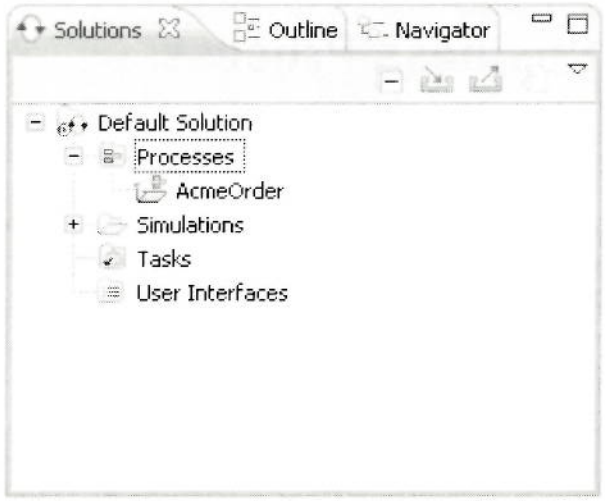
Use Designer to modify an existing business process to subscribe to a starting document, include services from previous exercises, and incorporate an error handling service. Build and update the model, and then test your process in Designer.

Steps

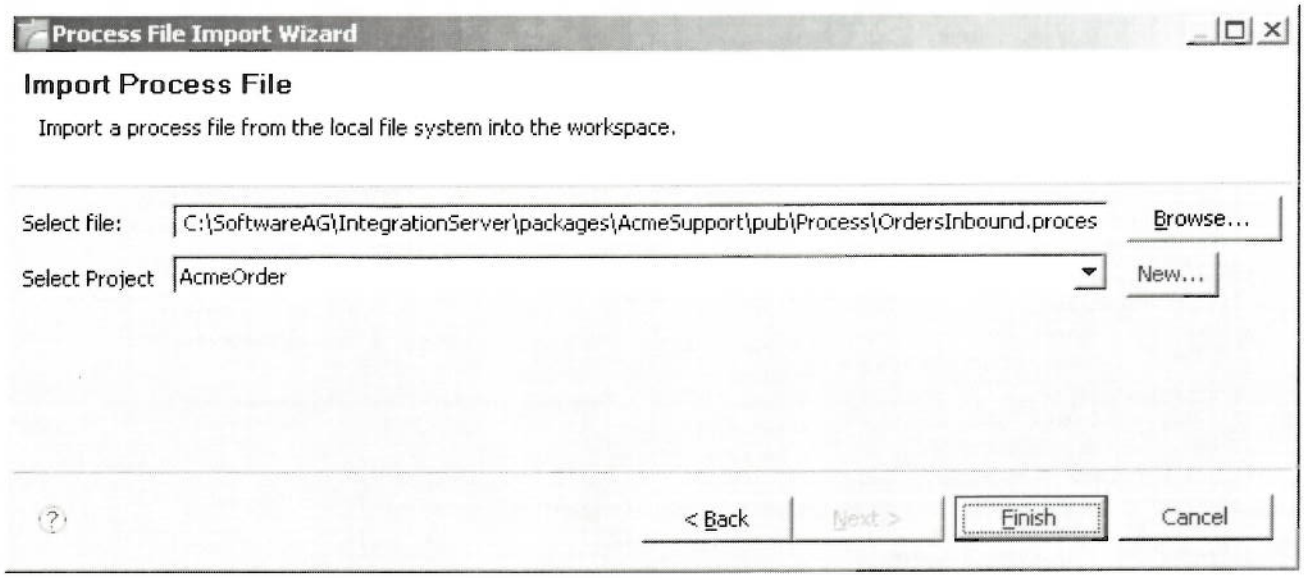
1. Open Designer and select the Process Development Perspective. Open the preferences view under **"Window" ➡ "Preferences"** and verify that the Property **"Automatically change step name when drag and drop document or service over existing step"** which can be found under **Software AG ➡ Process Development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



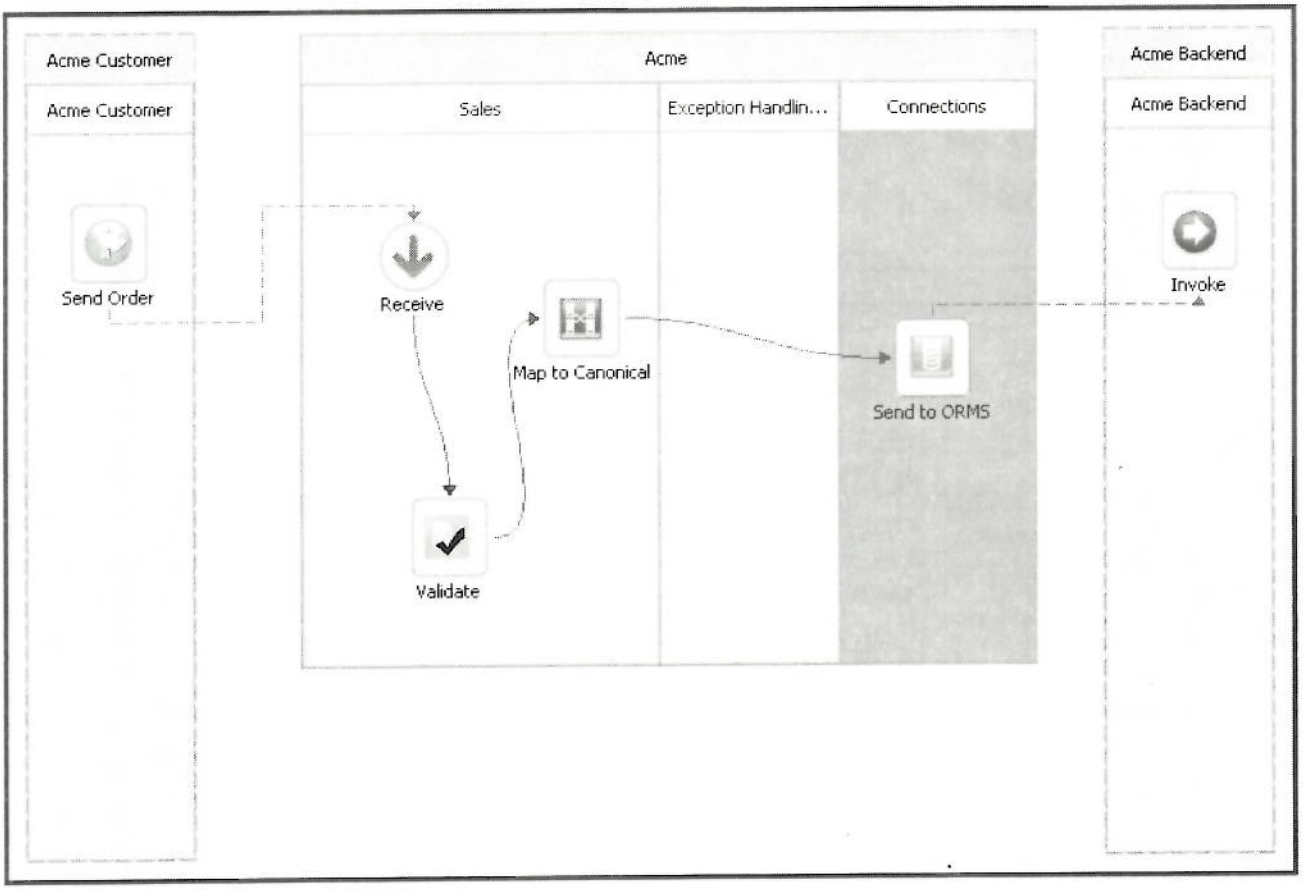
2. Using the File menu **create** a Process Project called AcmeOrder.



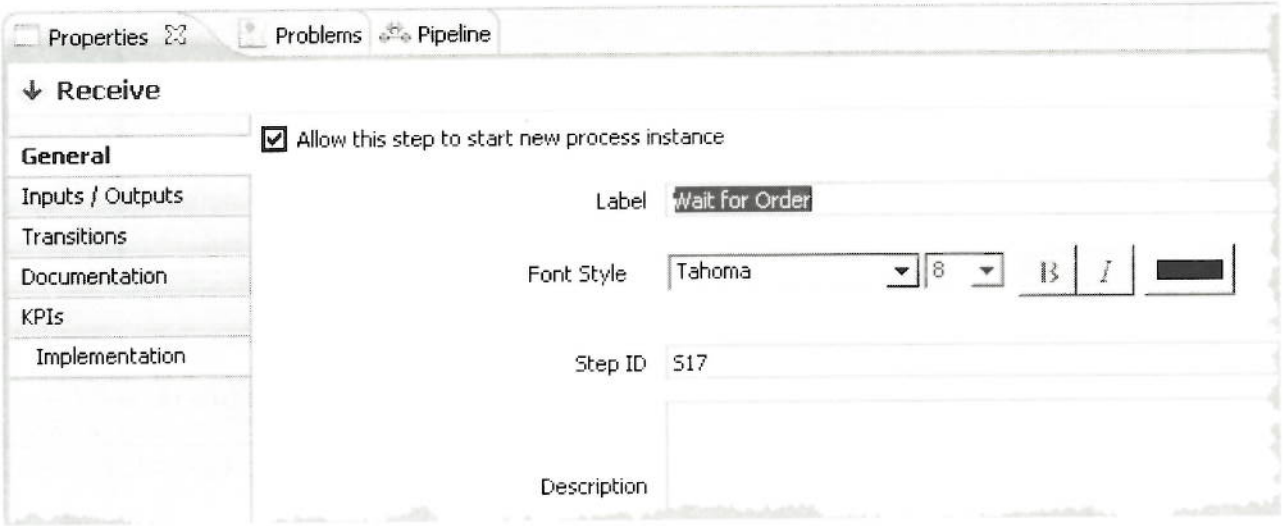
3. From the File Menu select **Import** from an **import source** of **SoftwareAG ➡ Process File. Click Next** and choose the directory **...\IntegrationServer\packages\AcmeSupport\pub\ Process**. Select the file **OrdersInbound.process** and the project AcmeOrder.



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

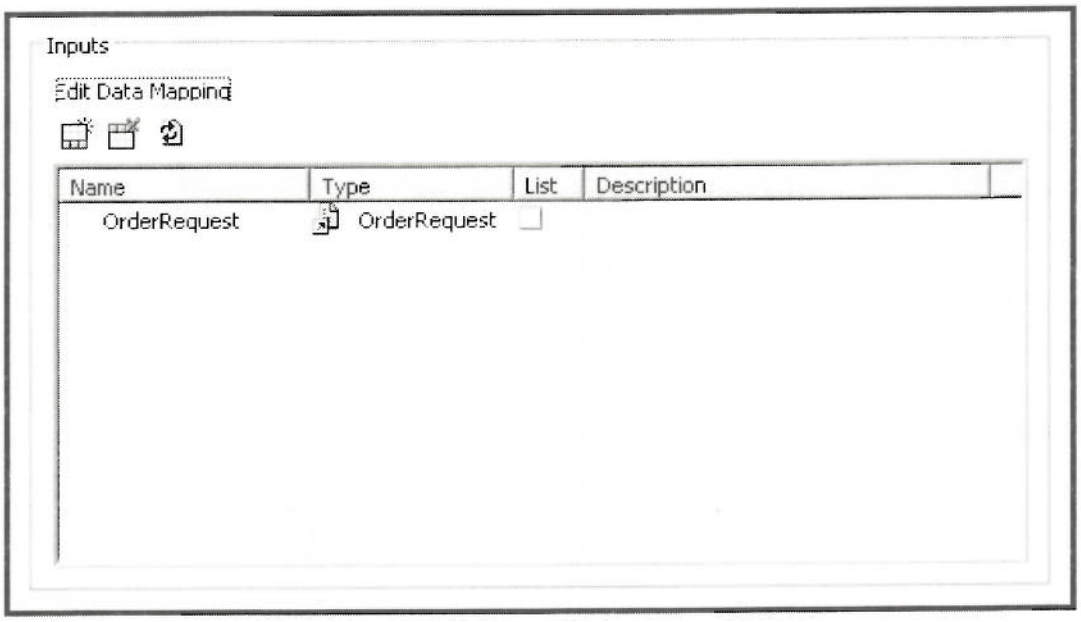


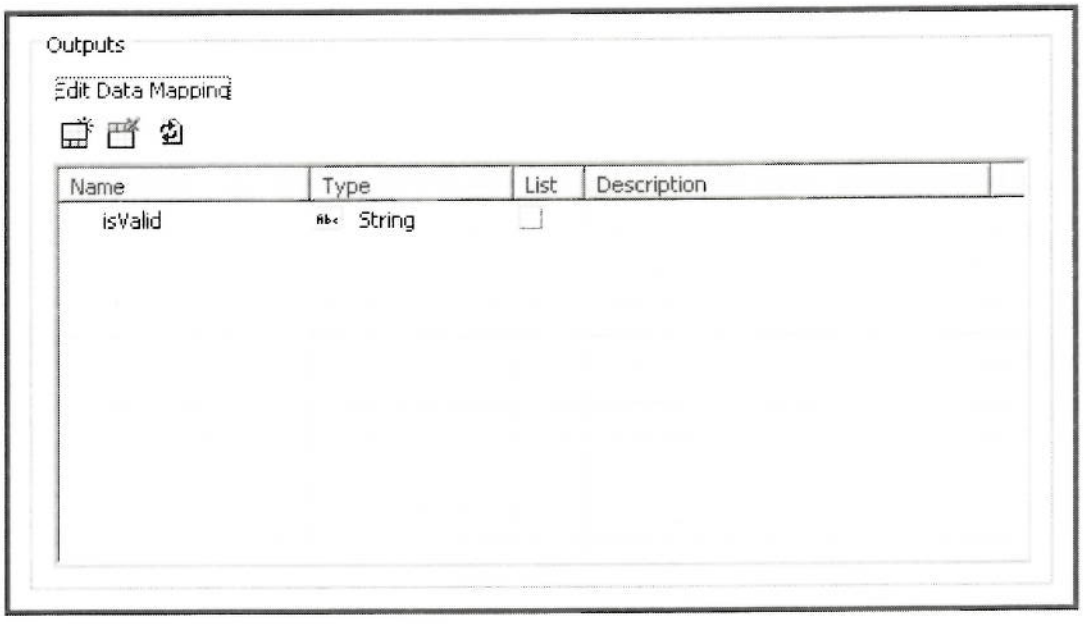
5. Modify the **Receive** step in the process. First, change the name of the step to **Wait for Order**.



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

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

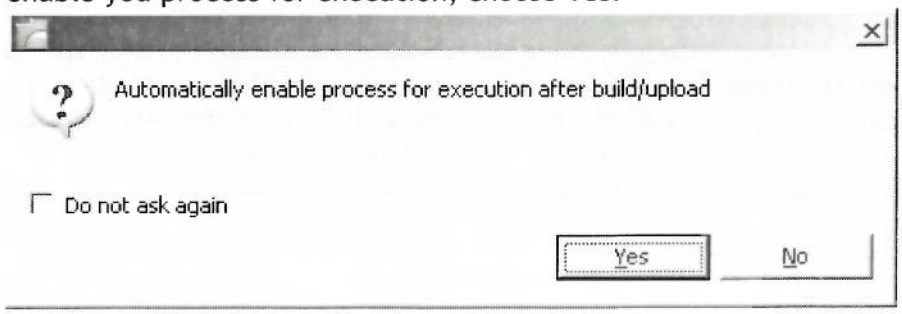




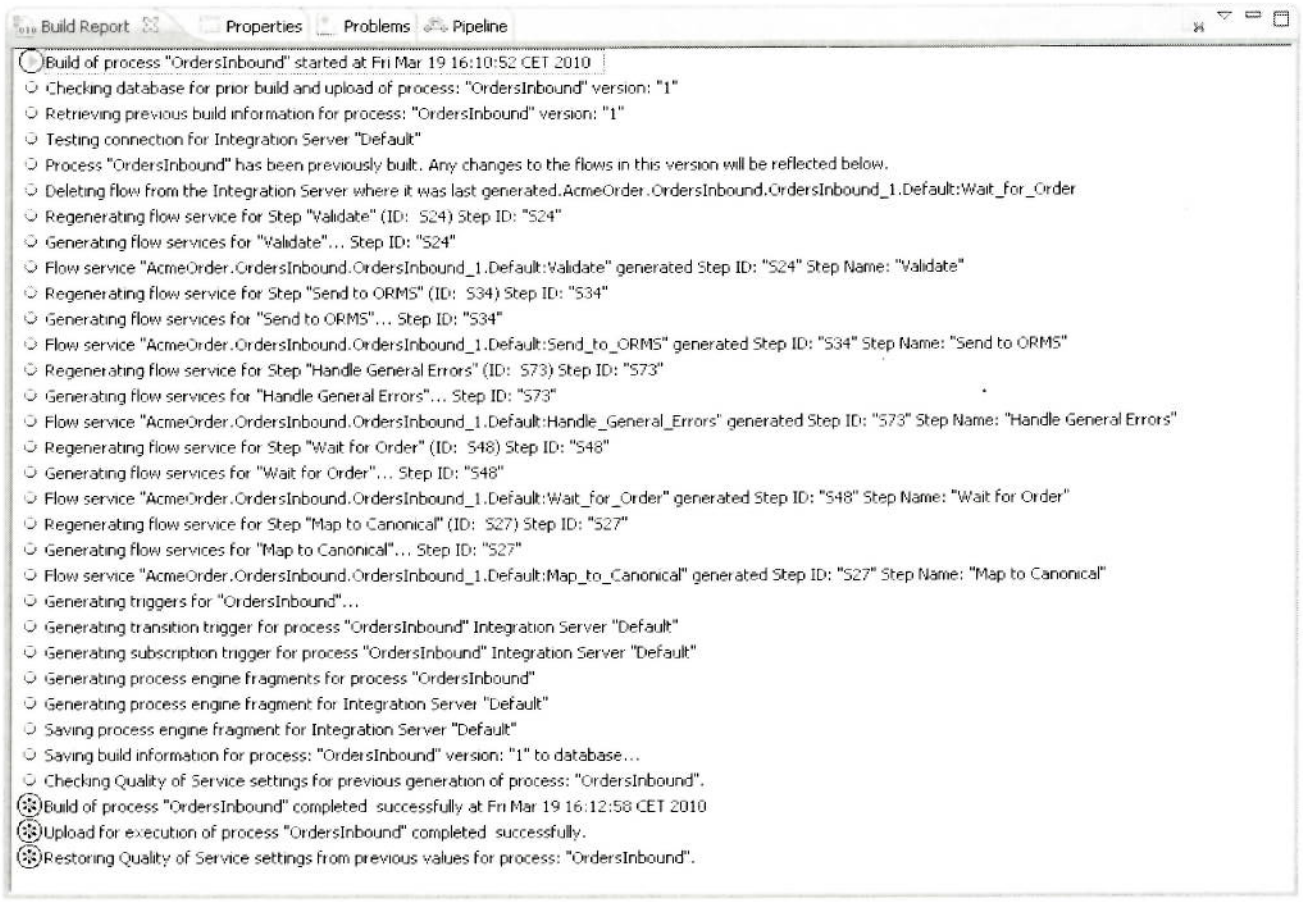
7. Modify the **Map to Canonical** step as you did for the Validate step, but this time call the service **acme.PurchaseOrder.maps:orderRequestToCanonical**. You can set the service also by dragging the Service from the Package explorer view to the "**Map to Canonical**" step. Bring in the Inputs/Outputs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

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

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



Your build report should look like the following:



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

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

Check Your Understanding

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

This page intentionally left blank

Check Your Understanding: Anwers to the Questions

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

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

Exercise 2: Packages and Folders

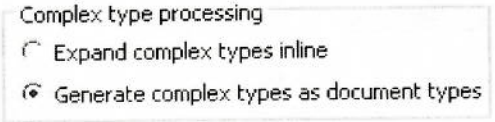
1. If you place the folders in the wrong parent folder, how could you correct it? **You can drag and drop folders with the mouse.**
2. Why is a consistent folder structure in all packages important? **This makes it easier for new people on a project to understand the inner workings of a package.**

Exercise 3: Create a Service

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

Exercise 4: Document Types

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



on its default value "Generate ... as document types". The generated Document type docType_PartnerRoleDescription is reused in several places inside the OrderRequest document, making maintenance of it much easier.

2. What is the benefit of using a schema for import over using a DTD? **Both DTD and XML schemas allow the same functionality. So there is not much of a difference. But DTD's are written in their own language, while schemas are written using XML syntax and can be processed with the standard XML toolset. Furthermore DTD's are more oriented towards text based documents (books) while schemas are oriented towards data based documents (orders, invoices).**

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

Exercise 5: Flow Services - BRANCH

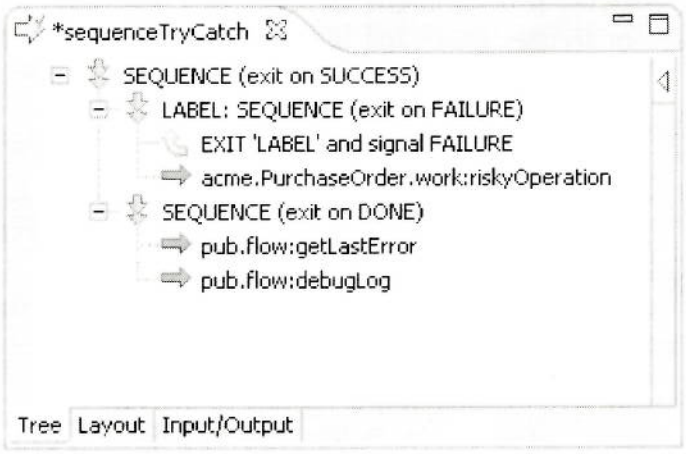
4. When are regular expressions useful in branch? **When you want to perform an operation on a set of similar input values.**
5. Can you combine a switch variable with Evaluate labels=True? **No, this wouldn't make sense.**
6. What are the special test values that can be used as labels in a branch statement?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

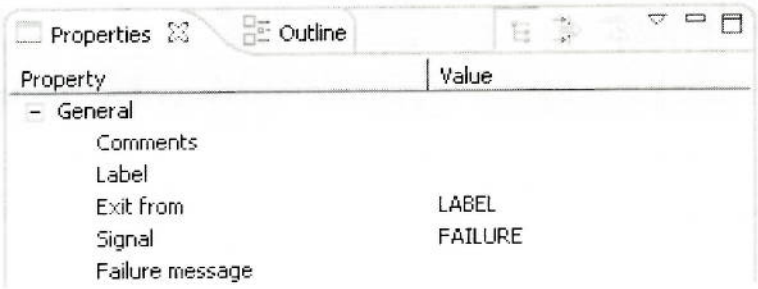
Exercise 6: Building Flow Services - LOOP

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

Exercise 7: Building Flow Services – SEQUENCE

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



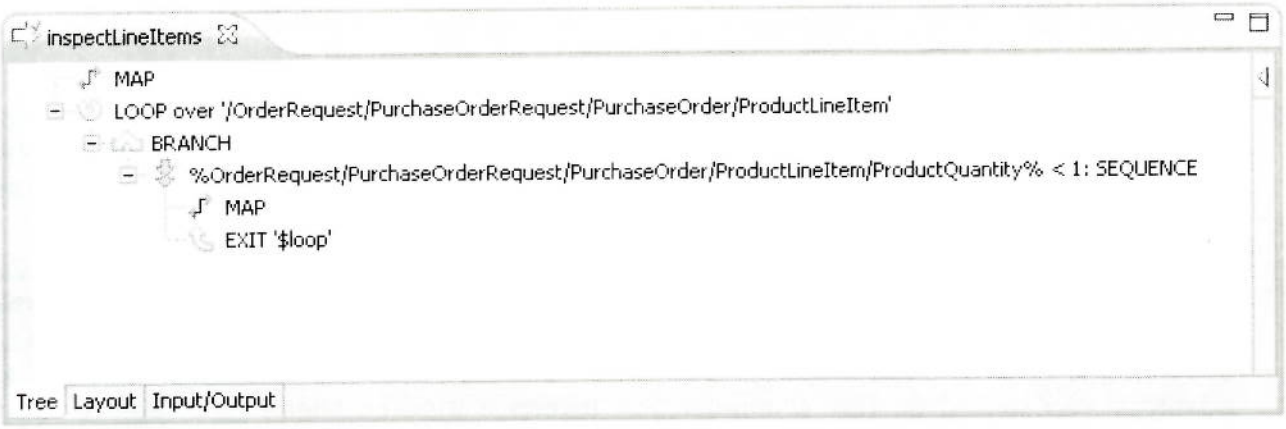


2. What happens if the riskyOperation service works (doesn't fail) ? **Only the 1st and the 2nd SEQUENCE statements get executed. The content of the file is read into memory and is returned to the caller.**

Exercise 8: Validation Service

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

Extra credit solution:



Exercise 9: Mapping Service

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

Exercise 10: Create a Java Service

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

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

IDataCursor cursor = pipeline.getCursor();

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

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

// compute the value to be returned

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

// store the return value in the pipeline

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

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

cursor.destroy();
```

2. Is the service thread safe? What would you have to do if not? **Yes, it is. Because it's not using any shared state and it's not calling any method that's not thread safe. Otherwise you would have to add the appropriate synchronization primitives to protect such shared state.**
3. How could the cursor handling be improved? **The cursor should be destroyed before we start the computation of the result (The line calling string.endsWith()). To store the results in the pipeline another cursor should be created using the getCursor() method of the pipeline object. The reason for this is, to have the cursors allocated for as short as possible. You should do this whenever you invoke a method that might require some time to compute its result. In the present example the overhead caused by cursor management outweighs the benefits of a shorter cursor lifetime.**

Exercise 11: Monitoring Services

1. Why is it necessary to create remote server aliases? **By design, MWS will communicate only with one instance of Integration server. When resubmitting a service invocation, MWS tells this Integration server where it wants the service instance to be scheduled for execution. To do so, MWS is sending the name of the remote server alias that should be used to resolve the final execution server.**
2. Under what circumstances would it be acceptable to resubmit a service? Why? **Those circumstances depend only on the service execution to be resubmitted. If a failed service had already executed half of its statements, then those statements may have caused some state changes where it may not be viable to do those changes again**

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

Exercise 12: Invoking Services

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

Exercise 13: Create a Flat File Schema

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

Exercise 14: Create a Flat File Dictionary

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

Exercise 15: Web Service Descriptors and Custom Faults

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

Exercise 16: Broker Pub/Sub

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