

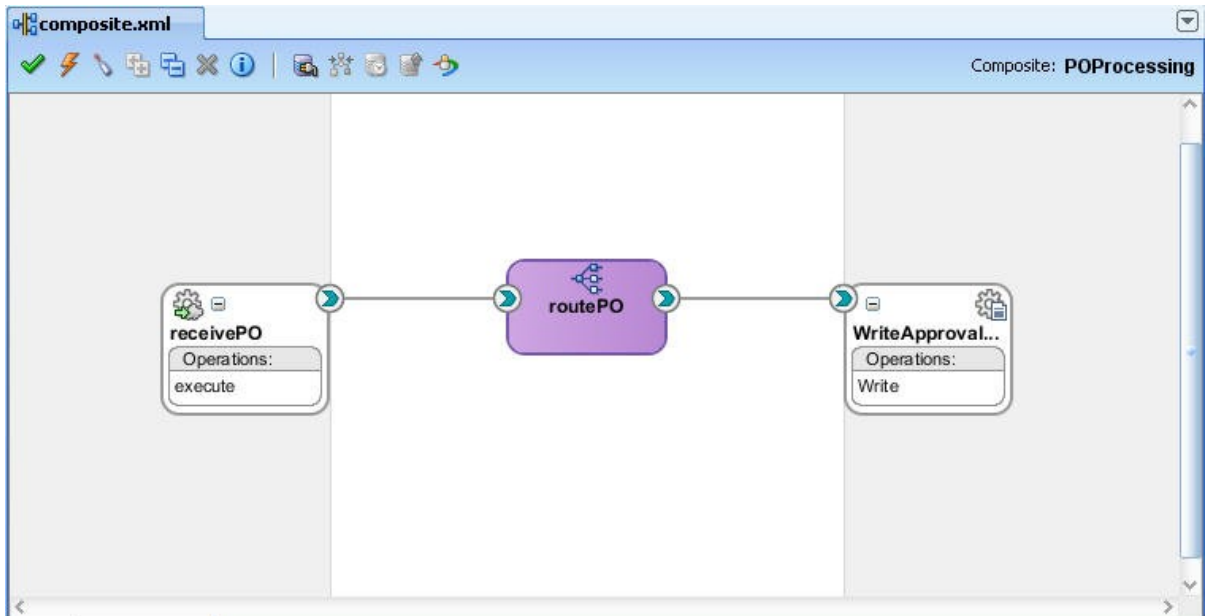
BPEL Orchestration

[4.1 Introduction..... 1](#)
[4.2 Designing the flow..... 2](#)
[4.3 Invoking the CreditCardStatus service..... 2](#)
[4.4 Designing the BPEL approval process..... 8](#)
[4.5 Modifying the Mediator component..... 18](#)
[4.6 Deploying the application..... 25](#)
[4.7 Testing the application..... 26](#)

4.1 Introduction

Note: To start this exercise:
 Be sure you have followed the instructions on the wiki at:
<http://wikis.sun.com/display/OTNVirtualDevDay/Hands+on+Lab+-+VirtualBox+users>
 under the heading **Preparing for the hands-on Lab.**
 You should have a directory: /home/oracle/setup/POProcessing
 Start JDeveloper and import the project from the /home/oracle/setup/POProcessing folder.

You are starting with a credit card validation composite service already deployed and a 'POProcessing' composite that looks like this:



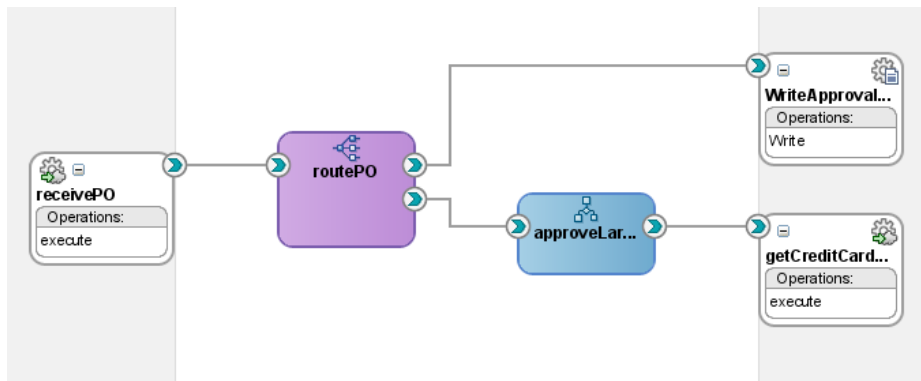
This composite receives a purchase order and uses a mediator component to route the order to a file. There are no validation or approval steps.

When receiving large orders (greater than \$1000) we want to be more cautious and:

- Validate the customer's credit card
- Automatically accept or reject the order based on the credit card status

The tool for orchestrating these interactions is the BPEL process manager. The overall flow of the application uses the credit validation service and the po processing service created earlier.

Once completed, your POProcessing composite will look like this:

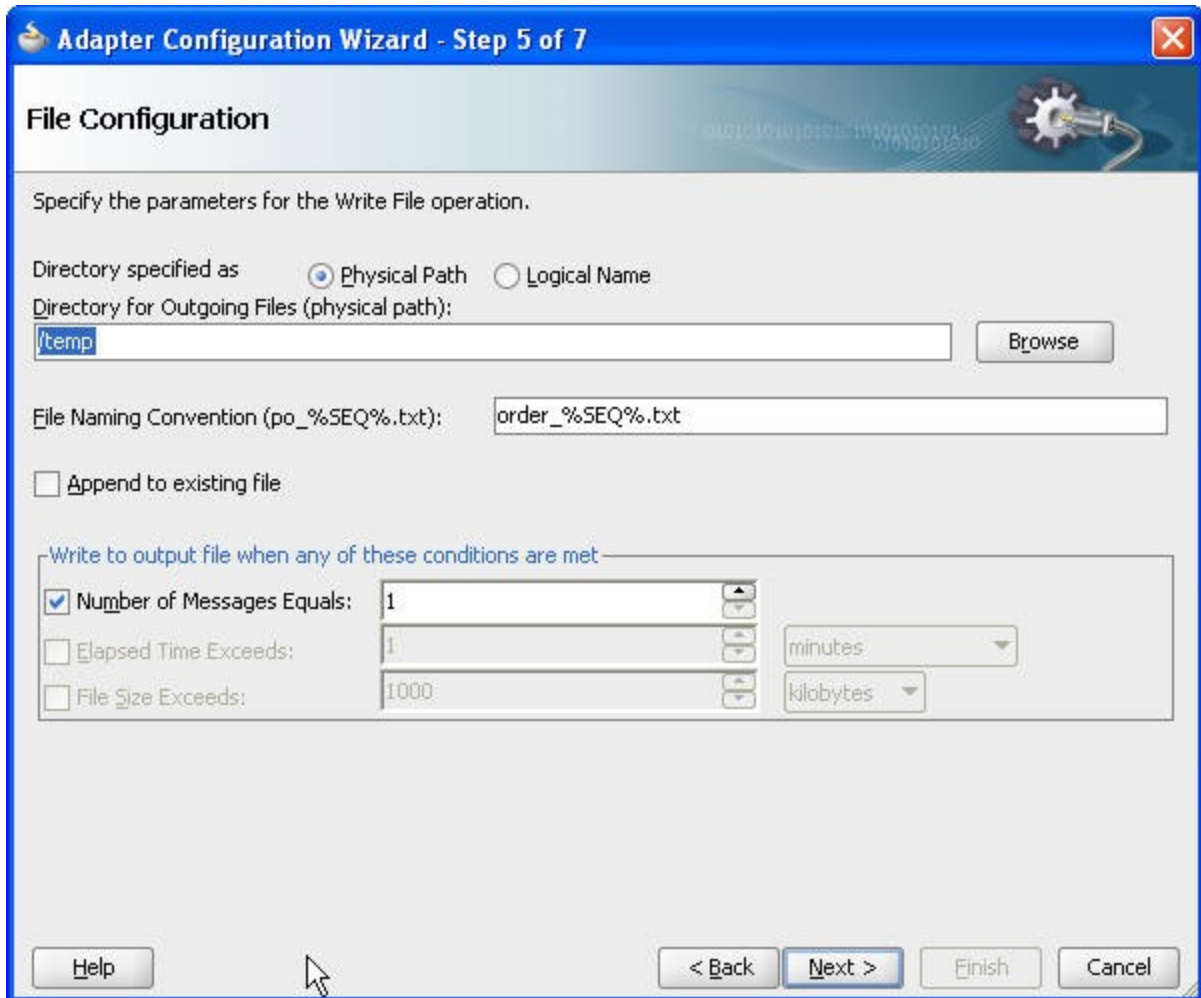


4.2 Designing the flow

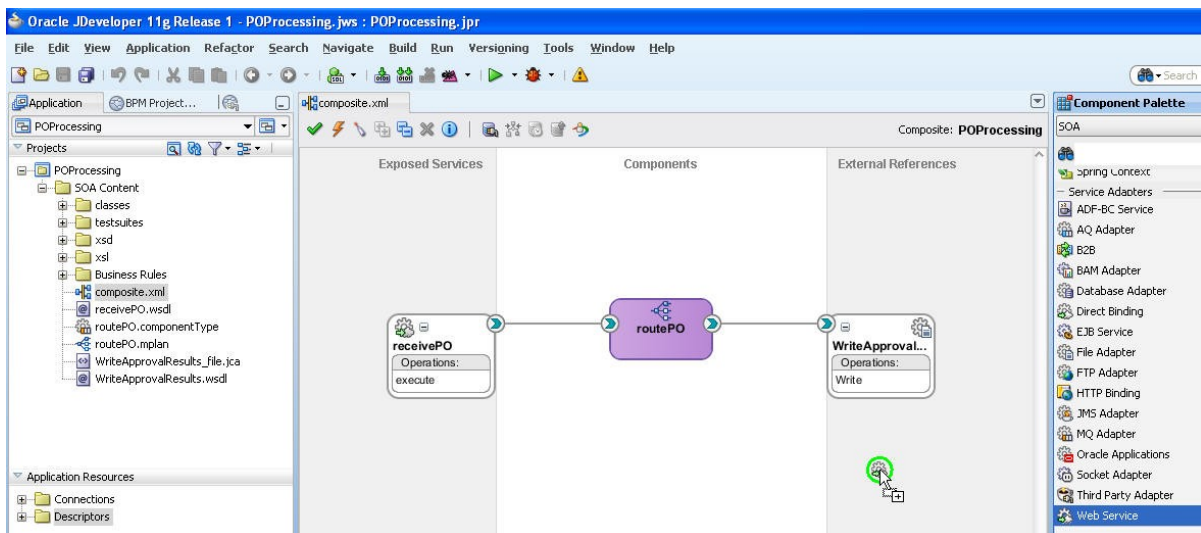
You will modify POProcessing with the changes outlined above. Start by creating a reference to the service you created in chapter 2, the credit card validation service, using SOAP. Then create the BPEL process to orchestrate the call to the credit card validation service and based on the returned value set the return status on the order. Finally, you wire the BPEL process to the Mediator and add a routing rule so that it is called only when the order total is over \$1000.

4.3 Invoking the CreditCardStatus service

1. If it is not already open, open the POProcessing application in JDeveloper.
2. Make sure there are no other files open and then open composite.xml for POProcessing. This eliminates any possible confusion as every composite has a composite.xml file.
3. Start your server and make sure your credit card validation service (created in 002-POProcessing-02-DBAdapter.doc) is running.
4. Double-click on the WriteApproval (FileAdapter) .
5. Use the **Next** button to go to step 5 of the adapter wizard and change the file location to an appropriate location for your server. This should be a directory that has permissions to write a file.



6. Drag-and-drop a Web Service activity into the **External References** swim lane.



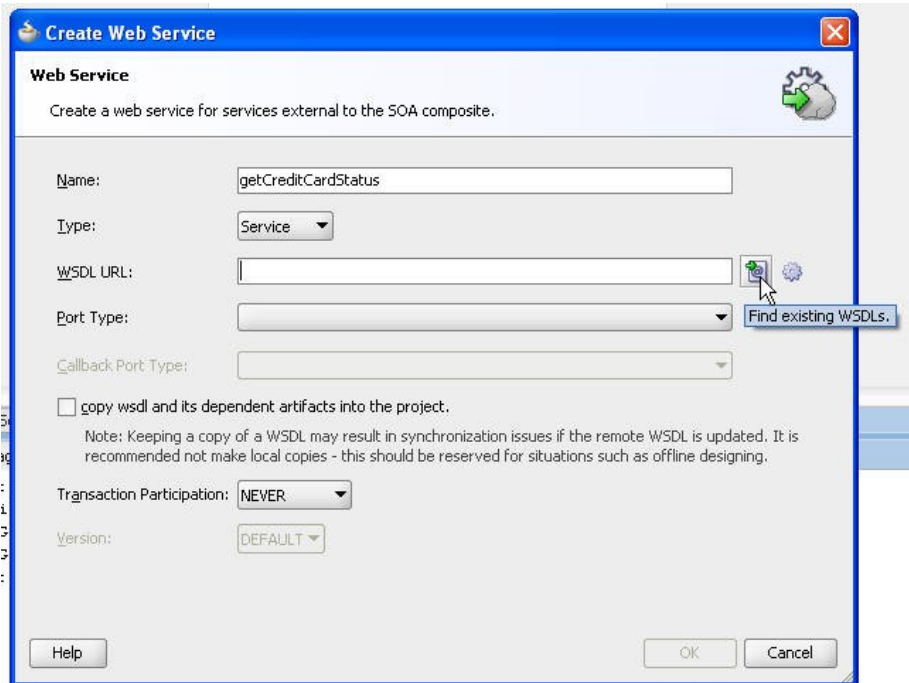
7. Set the following fields:

- **Name:** getCreditCardStatus

- **WSDL File:** The server must be running and the composite deployed

Navigate to the service using the Resource Browser as follows:

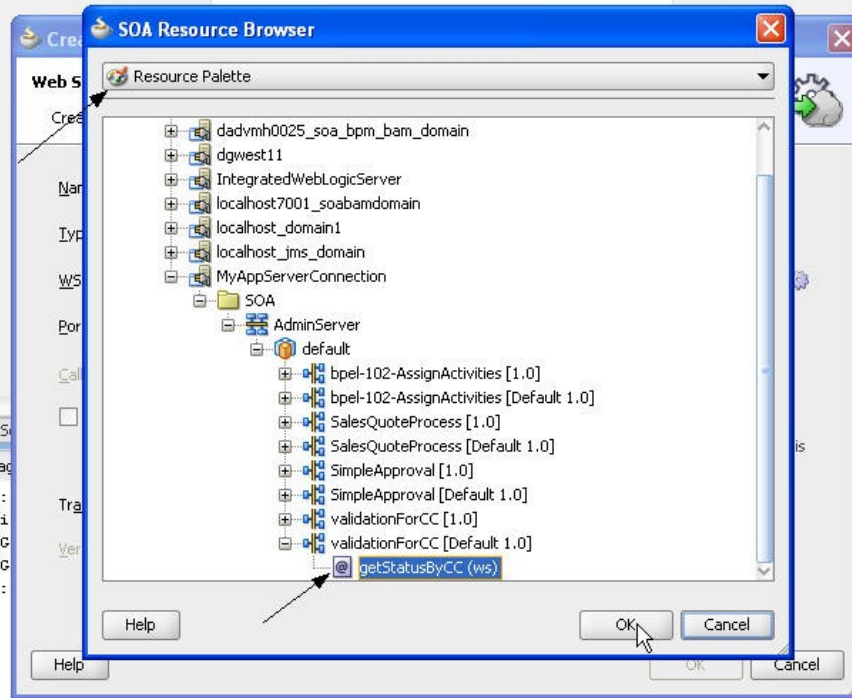
- Click the **Find Existing WSDLs** button next to the WSDL field



Select **Resource Palette** from the drop-down at the top of the **Resource Browser**

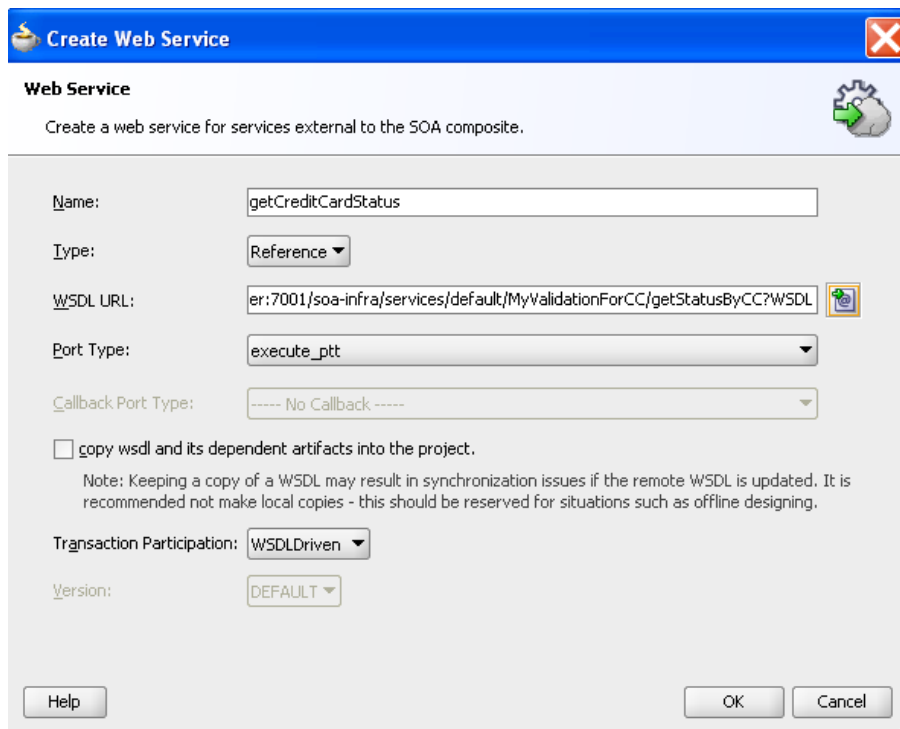
Navigate to the *validationForCC* service

Select *getStatusByCC* operation and click **OK**

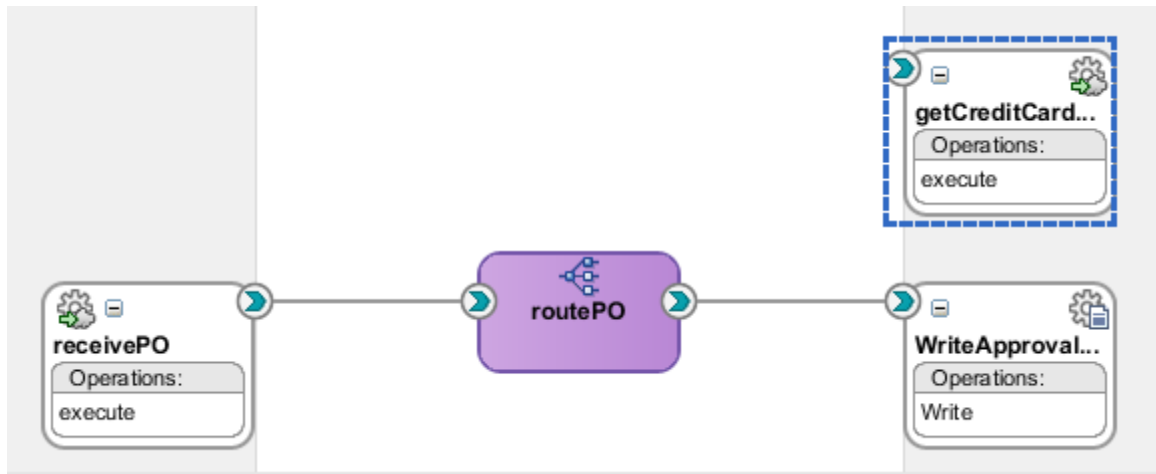


Alternatively, you can simply paste the URL to the WSDL into the field. The URL to the WSDL file can be obtained from the validationForCC test page in EM that we visited earlier. Be sure to remove any version numbering.

- **Port Type:** Once you have the WSDL URL, press the Tab key to move to the next field. The **Port Type** will be updated automatically based on the WSDL.

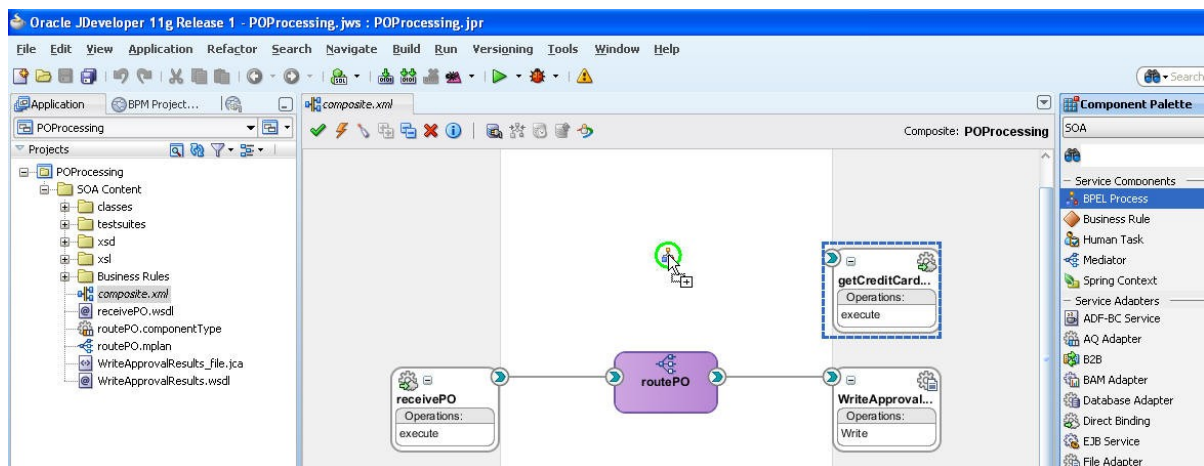


8. Click **OK**.



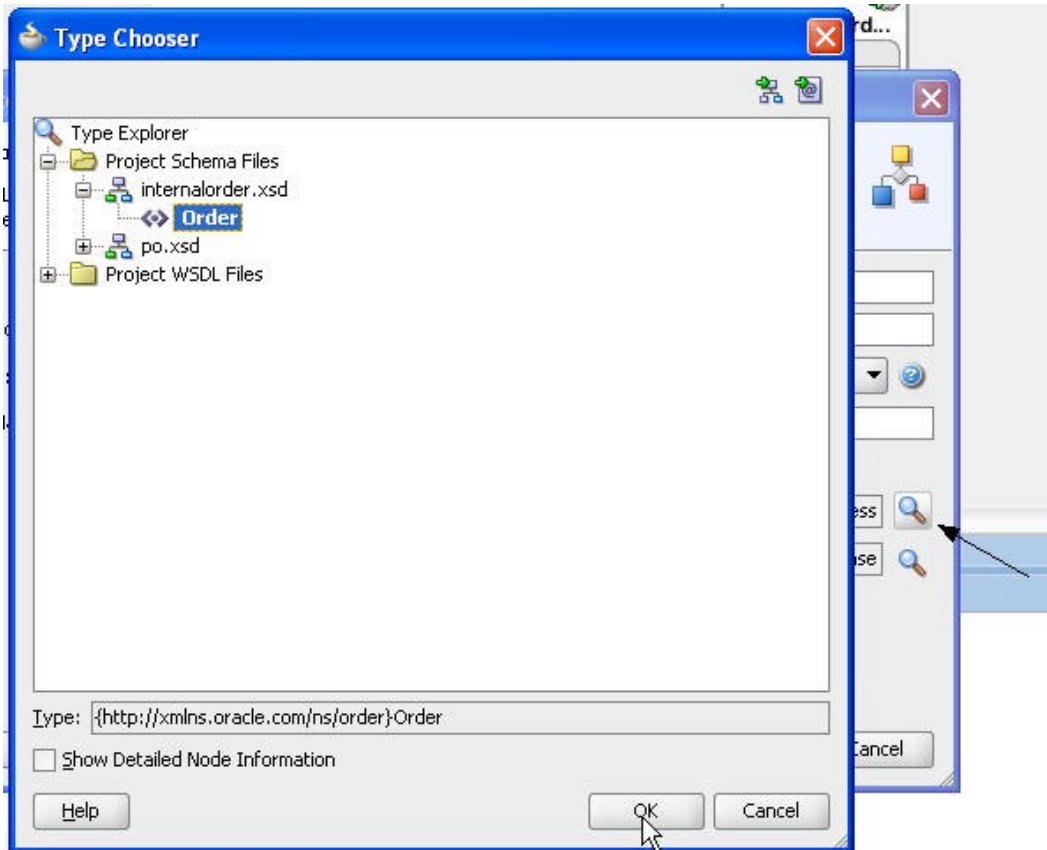
Now create a BPEL process and you can begin the orchestration for large order processing. The input and output to the BPEL process is in the internal order format of the order.

9. Drag-and-drop a BPEL Process component on to the **Components** swim lane.



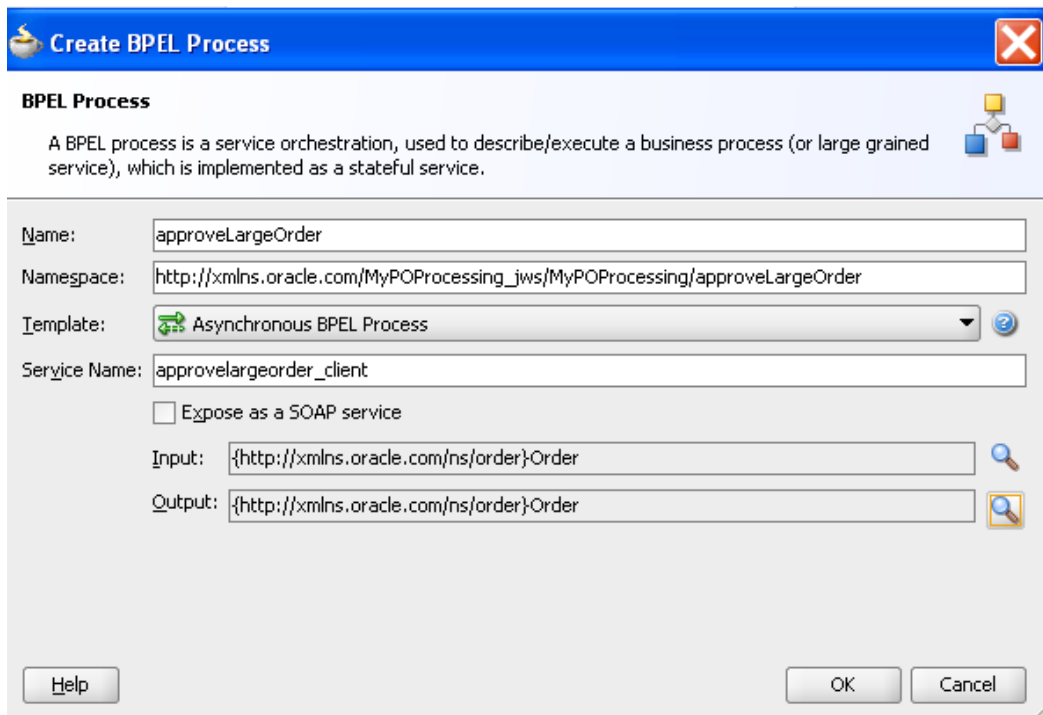
10. In the **Create BPEL Process** dialog, specify the following settings:

- **Name:** approveLargeOrder
- **Template:** Asynchronous BPEL Process
- **Service Name:** approveLargeOrder_client
- **Expose as a SOAP Service:** Unchecked
- **Input:** Click the flashlight icon, expand **Project Schema Files > internalorder.xsd** and select **Order**

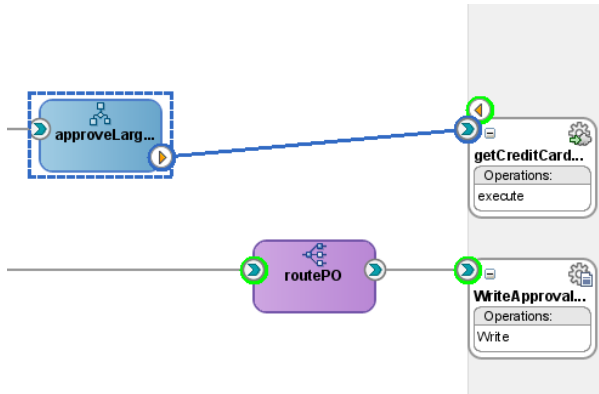


- **Output:** Use the **Order** type, like you did for **Input**

The WSDL for this service is created automatically using this information. Note that the input and output types specified here go in to the WSDL for this service.



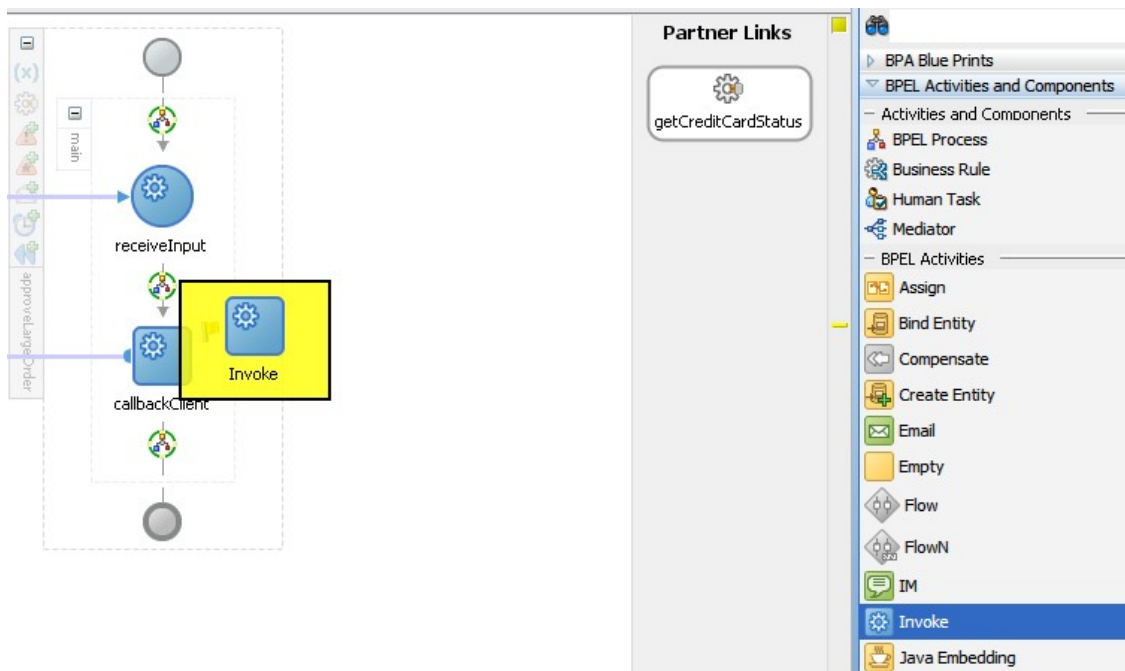
11. Click **OK**.
12. Wire the BPEL process to the **getCreditCardStatus** service. Wiring in the composite automatically creates a partner link reference inside the BPEL process.



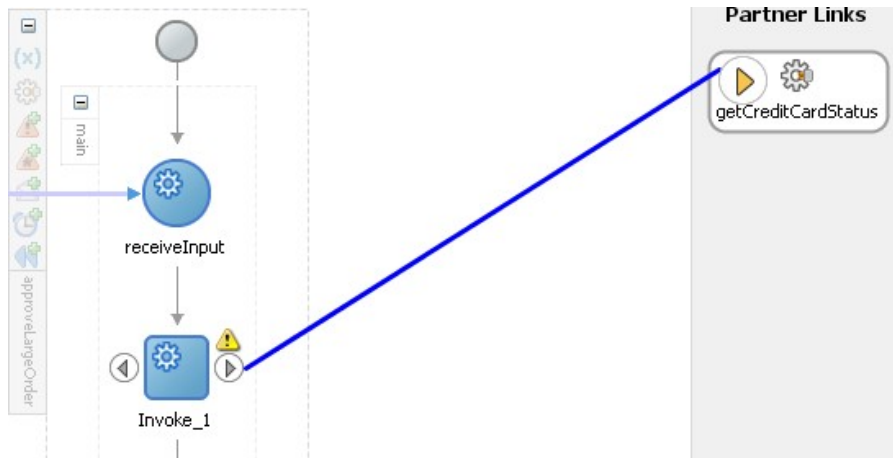
4.4 Designing the BPEL approval process

Next you'll build a simple BPEL process that calls the external **getCreditCardStatus** service.

1. Double-click the BPEL component to open the BPEL editor.
 Notice the **getCreditCardStatus** partnerlink already in the **References** swim lane because you wired it in the composite. The editors keep the references in sync between the BPEL process and composite.xml.
13. Drag-and-drop an **Invoke** activity from the Component Palette to an insertion point under the **receiveInput** activity.



- Drag the wire from the Invoke activity to the **getCreditCardStatus**. This tells your BPEL process to invoke that service.



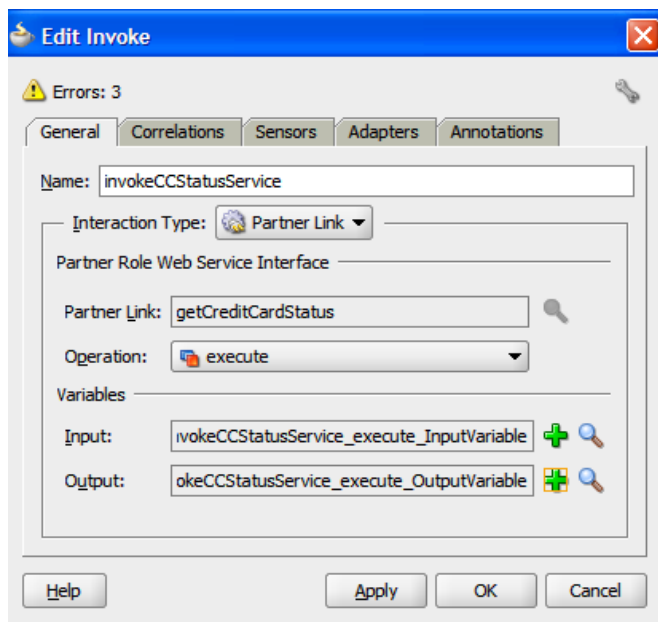
- In the **Edit Invoke** dialog that opens, specify the following:

- Name:** invokeCCStatusService
- Input Variable:** Click the green plus icon and then click **OK** to create a new global variable, accepting the default name and type.

The variable designated for the input will contain the data that will be sent to the service when it is invoked. It is automatically created with the correct type expected by the service.

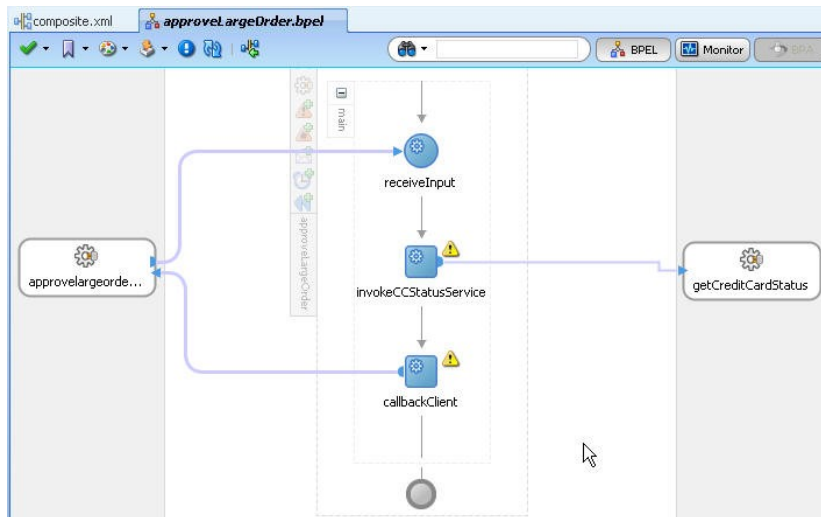
- Output Variable:** Click the green plus icon and then click **OK** to create a new global variable, accepting the default name and type.

This variable contains the data that will be returned by the service, or the output of the service. It is automatically created with the correct type.



- Click **OK**.

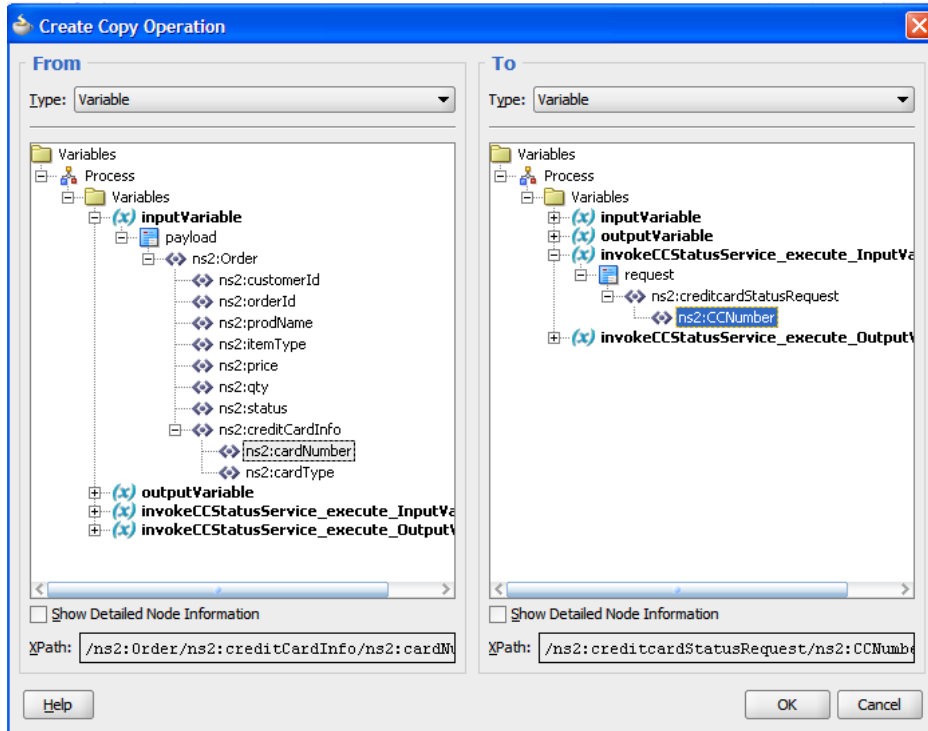
17. Your BPEL process looks like this so far (you will fix the warnings in the upcoming steps):



18. You have created the variables that will be used when interacting with the **getCreditCardStatus** service, but they haven't been populated. The output variable will be populated when the service returns a result, but you need to populate the input variable yourself.

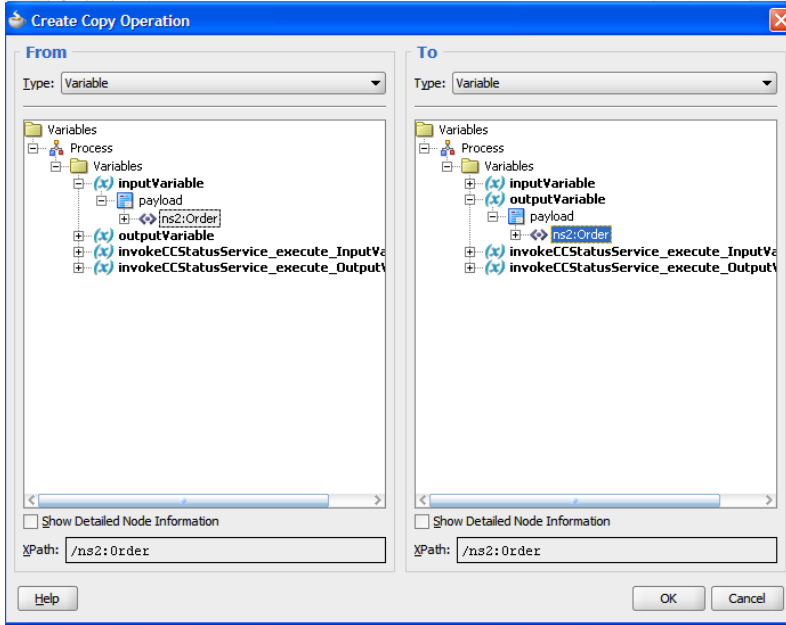
In BPEL, you use an **Assign** activity to assign data to a variable. In this case, you want to assign the credit card number that was passed into the **POProcessing** service to the **getCreditCardStatus** service.

19. Drag-and-drop an **Assign** activity above your **Invoke** activity.
20. Double-click the **Assign** activity to edit it.
21. Click the **General** tab and change the name to **assignCCNumber**.
22. Click the **Copy Operation** tab.
23. Click the green plus icon and select **Copy Operation** to open the **Create Copy Operation** dialog, and specify the following.
 - In the From side, select **Variables > Process > Variables > inputVariable > payload > Order > creditCardInfo > cardNumber**
 - In the To side, select **Variables > Process > Variables > invokeCCStatusService_execute_InputVariable > request > creditcardStatusRequest > CCNumber**

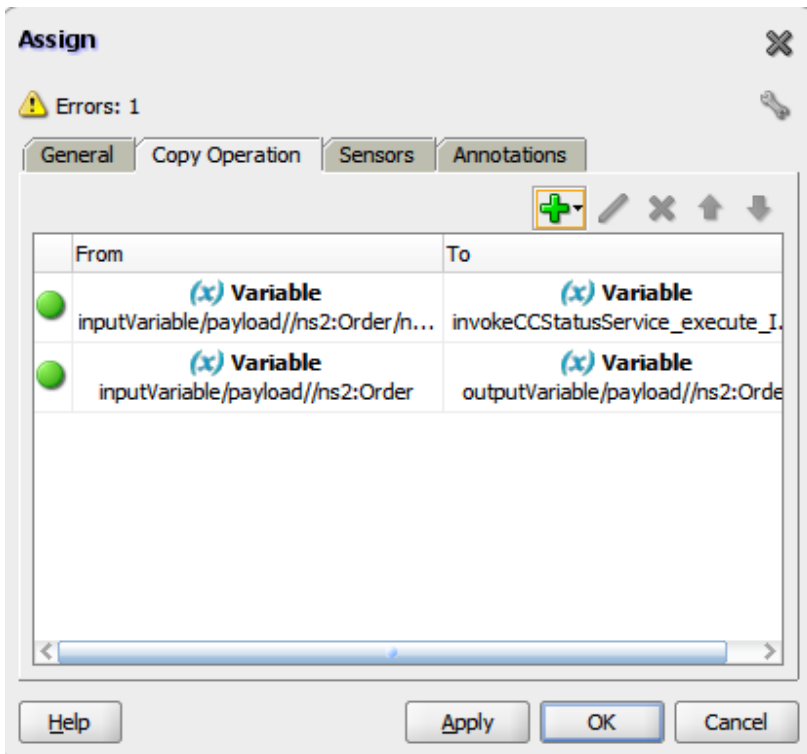


24. Click **OK**.
25. Back in the Assign dialog, add a second copy operation by click the green plus icon and selecting **Copy Operation**, and specify the following.
 - In the From side, select **Variables > Process > Variables > inputVariable > payload > Order**
 - In the **To** side, select **Variables > Process > Variables > outputVariable > payload > Order**

You are doing this because the BPEL process returns **outputVariable** when it finishes. You will return the input data, as well as some updates which will be made later in the BPEL process.

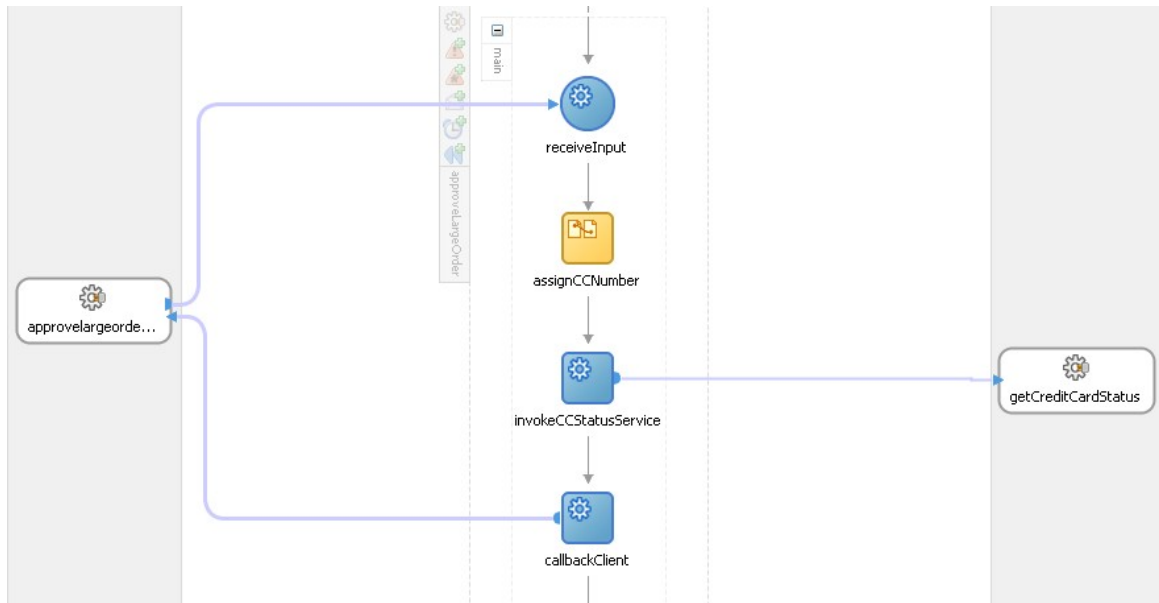


26. The **Assign** dialog now looks like this:



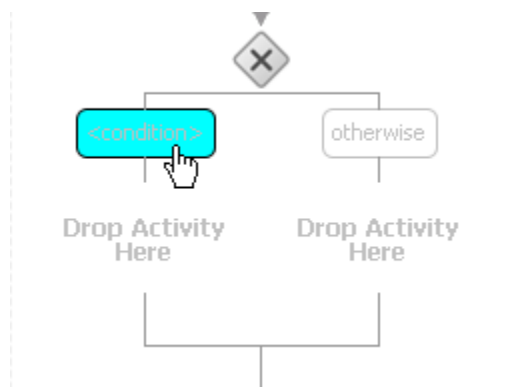
27. Click **OK** to return to the BPEL process.

28. Click the green check button in the upper left of the BPEL process to validate the process. All the warnings should go away. It now looks like this:

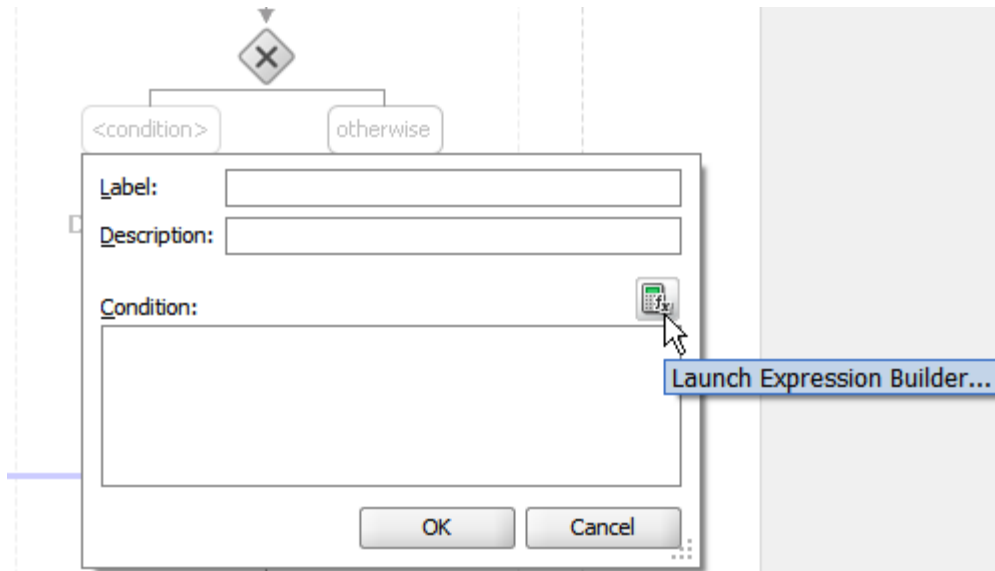


The input variable to **getCreditCardStatus** is now populated. The Invoke activity will pass that data to the service. The next step is to process the return data from the service, the output. The returned value can be VALID or INVALID. You want a **Switch** statement so you can do something different for each case.

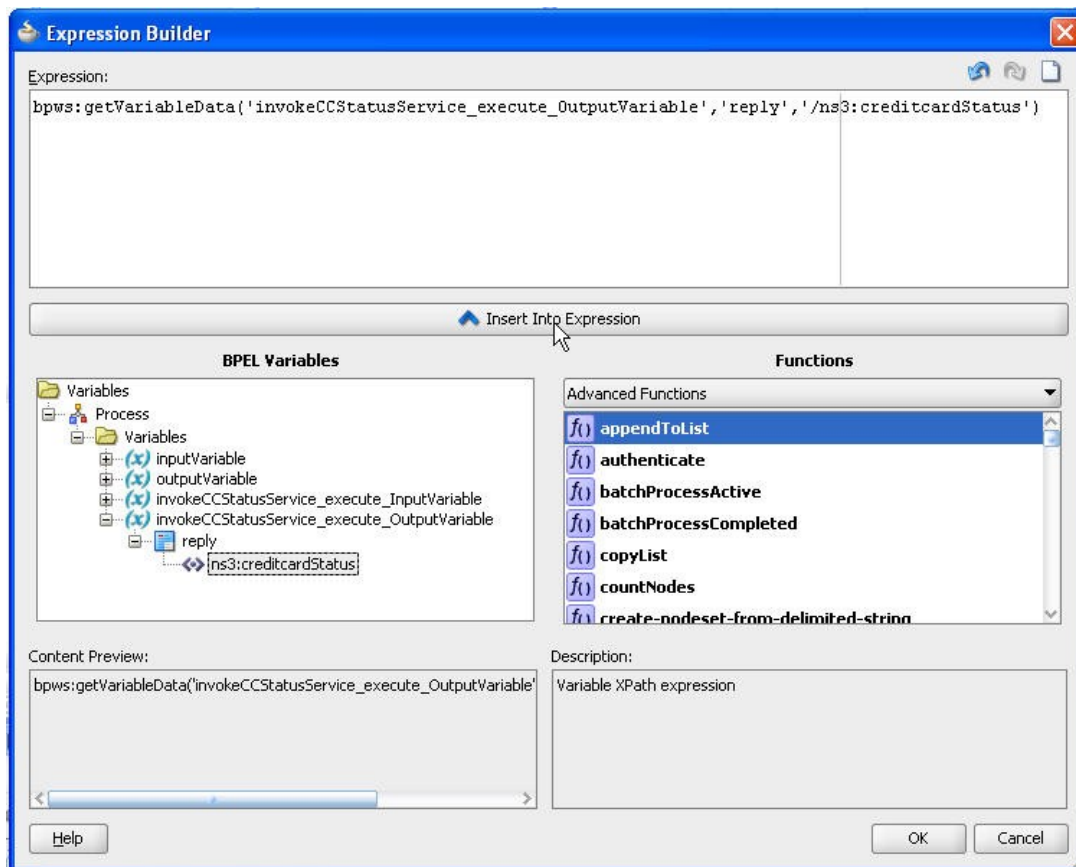
29. Drag-and-drop a Switch activity underneath **invokeCCStatusService**.
30. Double-click the name of the **Switch** underneath the icon (which is probably something like **Switch_1**) and rename it to **EvaluateCCStatus**. Note: You can also double-click the Switch icon and change the name in the subsequent dialog, but if you double-click the text itself you can change the name in-place.
31. Expand the Switch by clicking the small plus icon next to it.
32. Click the **View Condition Expression** button.



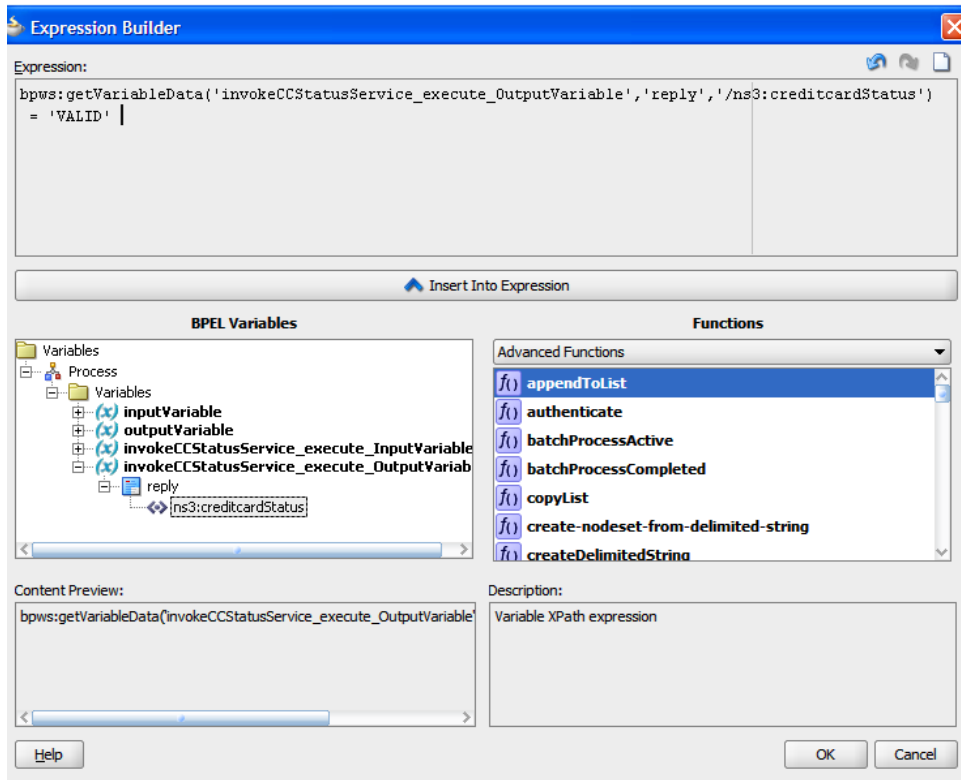
33. Click the **XPath Expression Builder** button.



34. In the **BPEL Variables** field, expand **Variables > Process > Variables > invokeCCStatusService_execute_OutputVariable > reply** and select **creditCardStatus**.
35. Click the **Insert Into Expression** button (it's the wide button under the **Expression** field).



36. Put your cursor in the **Expression** field and at the end and add: **= 'VALID'**



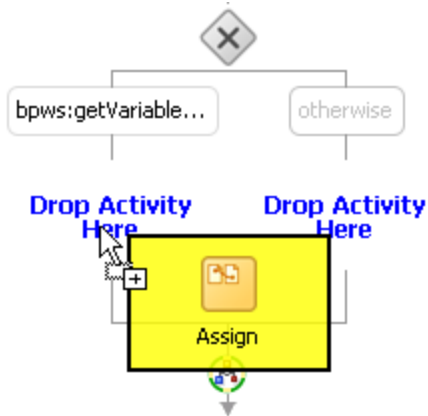
37. Click **OK**.

38. Be sure to click **OK** on the switch condition expression popup box.

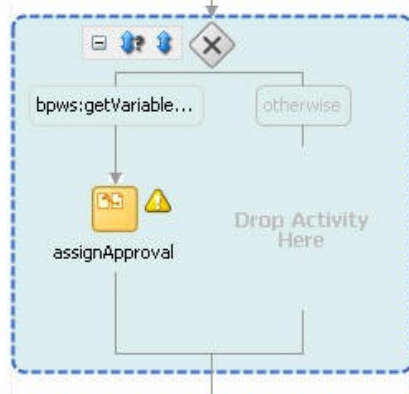


If that condition is true, then BPEL will execute any activities in the **<case>** part of the switch. If not, any activities in the **<otherwise>** section will be executed.

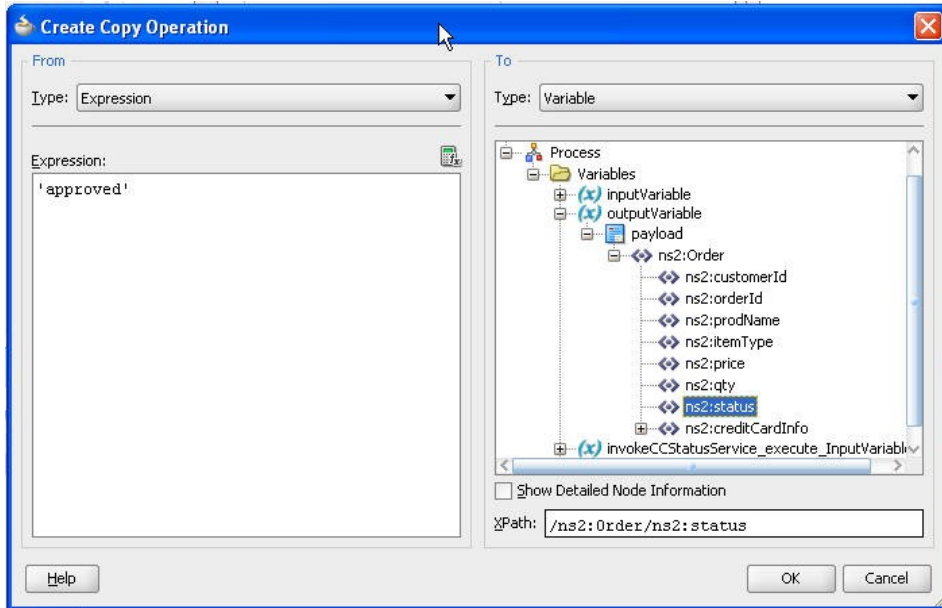
39. Drag-and-drop an **Assign** activity into the **<case>** section of the Switch.



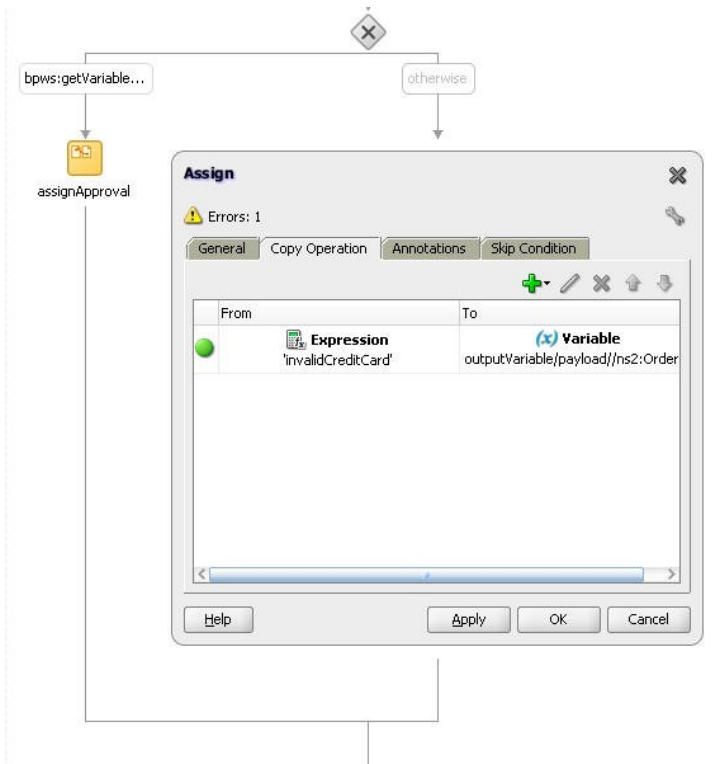
40. Double-click the name of the **Assign** (which will be something like **Assign_2**) and rename it to **assignApproval**.



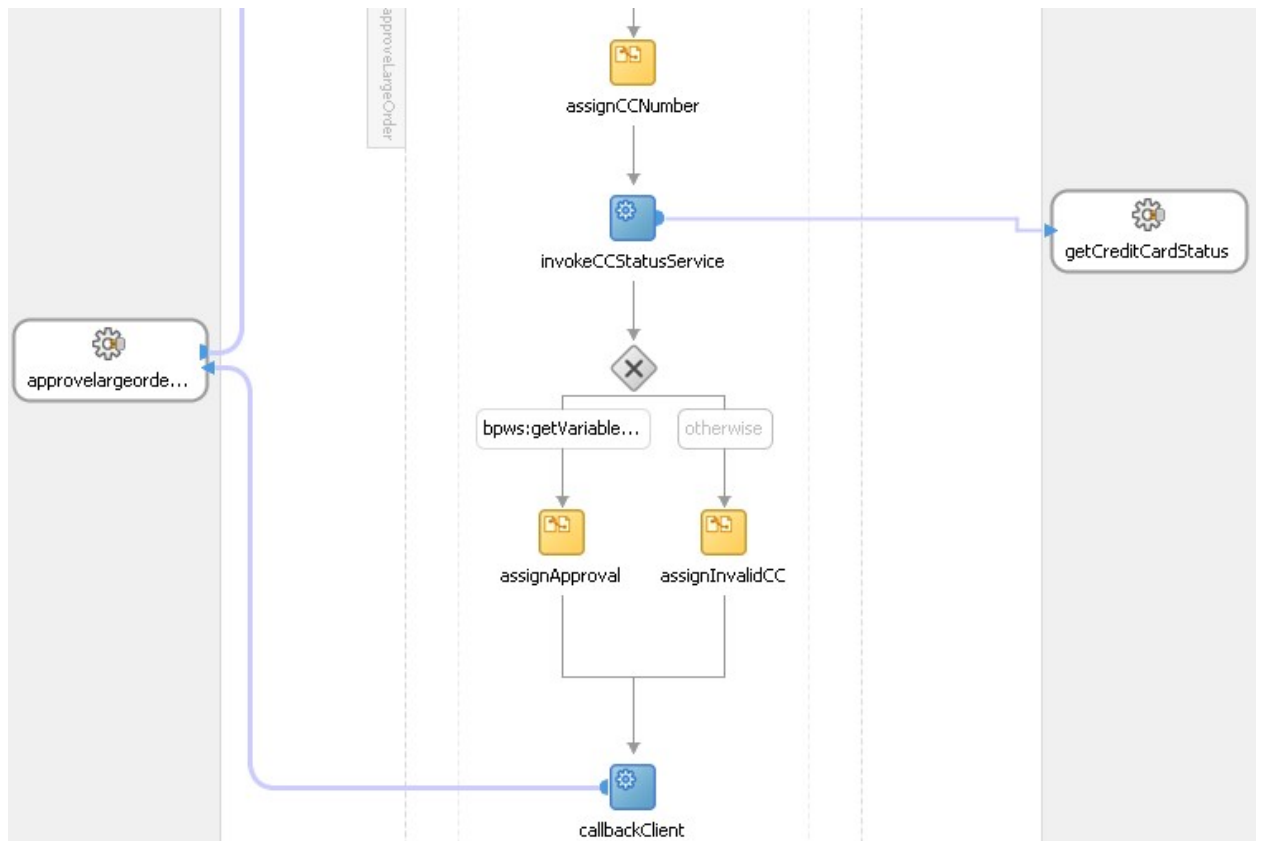
41. Double-click the **Assign** icon to open the **Assign** dialog.
42. Click the green plus icon and add a new copy operation.
43. In the **From** section, change the **Type** poplist to **Expression**.
44. In the **Expression** field, type: *'approved'* (with single quotes)
45. In the **To** section, select **Variables > Process > Variables > outputVariable > payload > Order > status**.



46. Click **OK**.
47. Drag-and-drop an Assign activity into the <otherwise> section of the Switch.
48. Rename it to **assignInvalidCC**.
49. In the same way you just did, assign the value **'invalidCreditCard'** to the **status** field of the **outputVariable** variable.



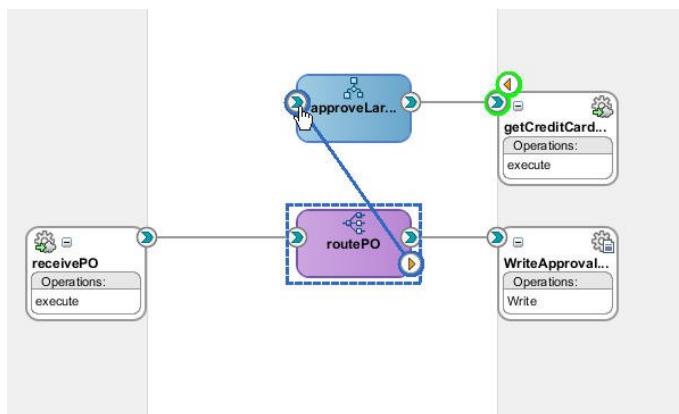
50. At the top of BPEL designer, click the green check mark to validate your process. Any yellow flags you had should disappear and you should not have any warning messages.



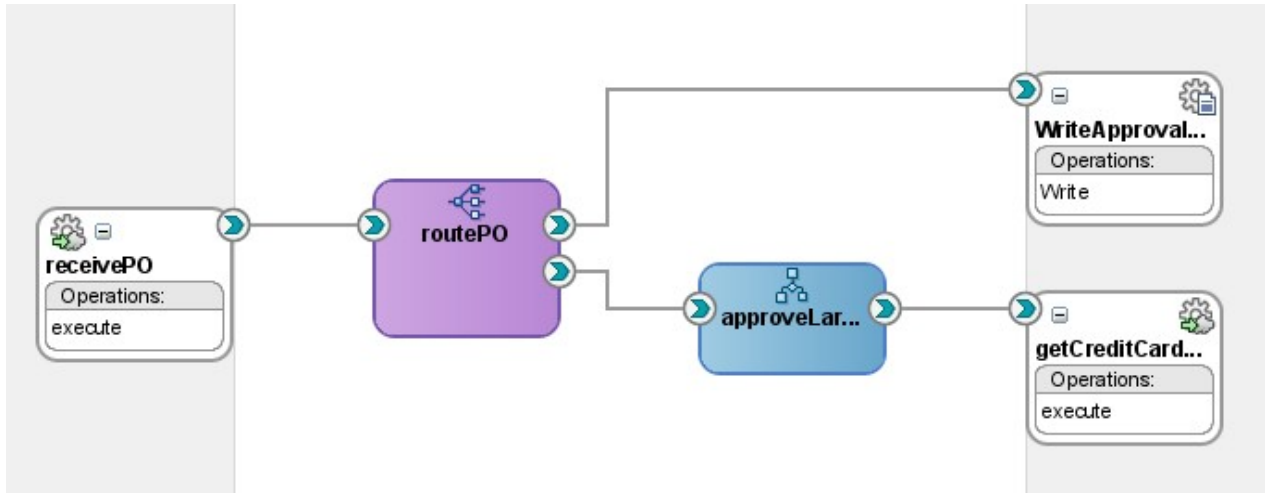
51. Save the BPEL process and close the window to return to the composite.

4.5 Modifying the Mediator component

1. Wire the Mediator to the BPEL process.



52. Your composite now looks like this:

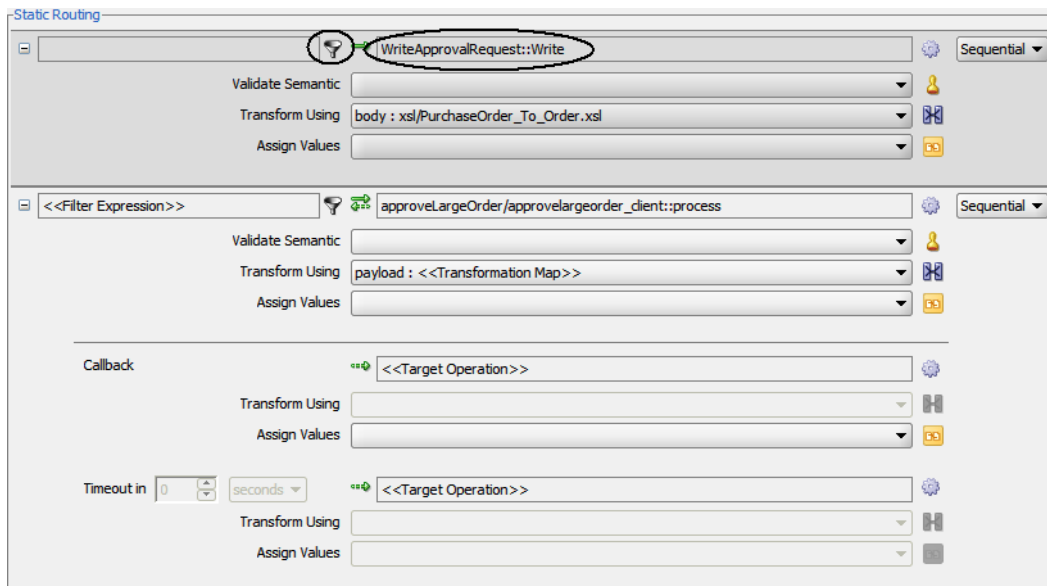


53. Now the Mediator is routing requests to both the **WriteApprovalResults** service and the **approveLargeOrder** BPEL process. Sometimes this is what you want a Mediator service to do, but in this case a new order should either be automatically approved or have to be approved by going through the **approveLargeOrder** process.

If you recall, orders under \$1,000 should be approved automatically while orders greater than or equal to \$1,000 need to go through an approval process. The Mediator is capable of creating a content-based routing service to enable this kind of processing.

Double-click the **routePO** Mediator component to open the Mediator editor. (You can double-click on the tabname **routePO.mplan** to get the full view of the configuration).

54. Click on the filter icon, called **Invoke Expression Builder** which looks like a funnel, for the **WriteApprovalResults::Write** request operation.



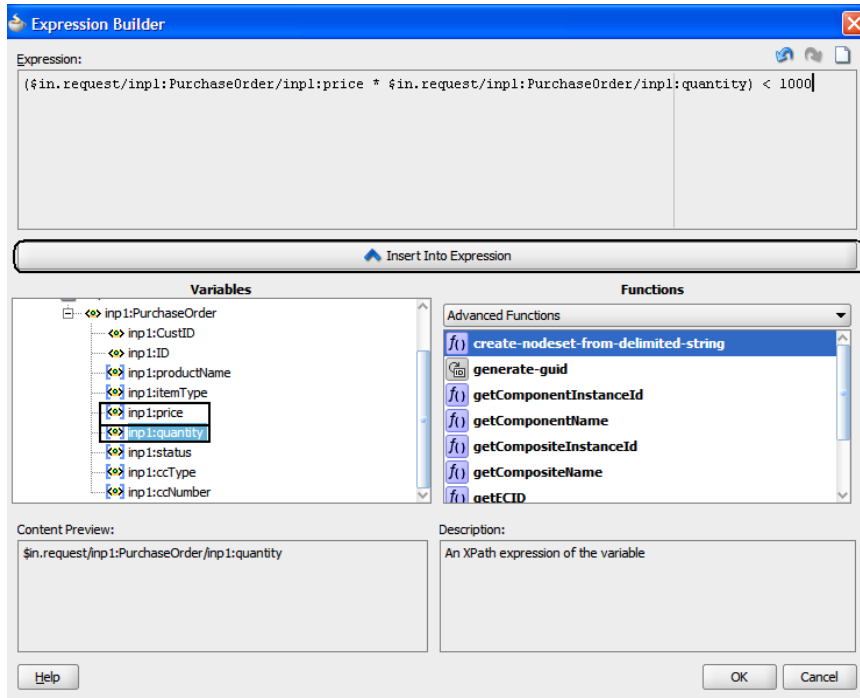
55. In the **Expression Builder** dialog, build up the following expression (don't copy this text into your expression but use the Variables frame to select the variables):

```

($in.request/inp1:PurchaseOrder/inp1:price *
 $in.request/inp1:PurchaseOrder/inp1:quantity) < 1000
  
```

Note: The namespaces (e.g., **inp1**) may be different for you.

Hint: Expand the nodes in the **Variables** section to find the field you want and either double-click or press the **Insert Into Expression** button to add them to the expression.

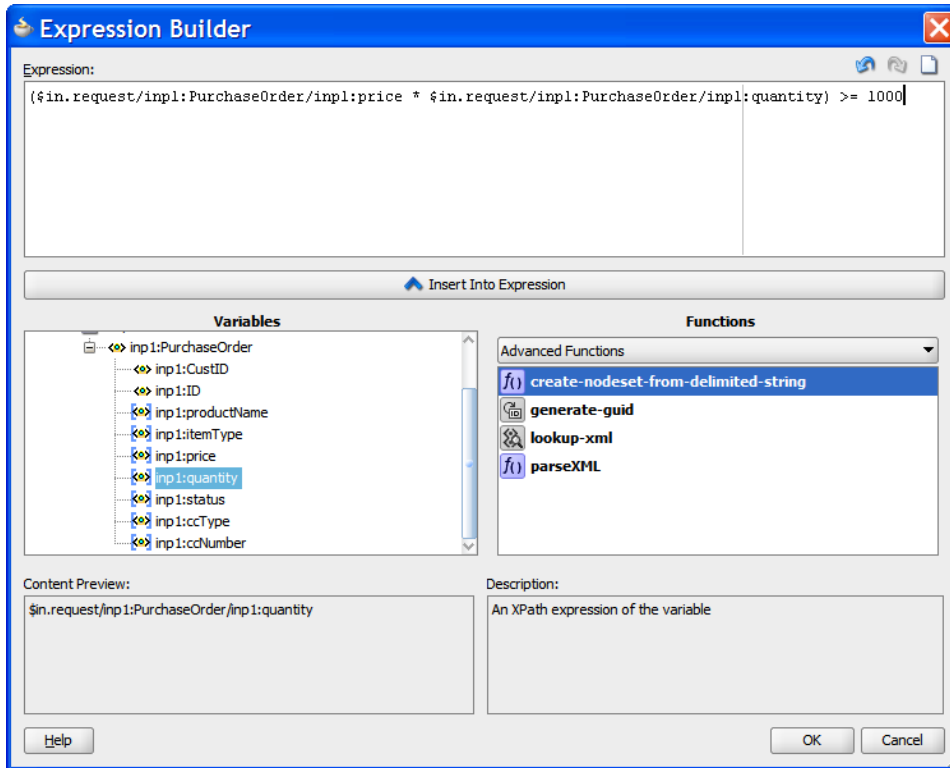


56. Click **OK**.

57. Similarly, click the filter icon for the request invocation of **approveLargeOrder/client::process**

58. Add the following expression:

```
($in.request/inpl:PurchaseOrder/inpl:price *
$in.request/inpl:PurchaseOrder/inpl:quantity) >= 1000
```



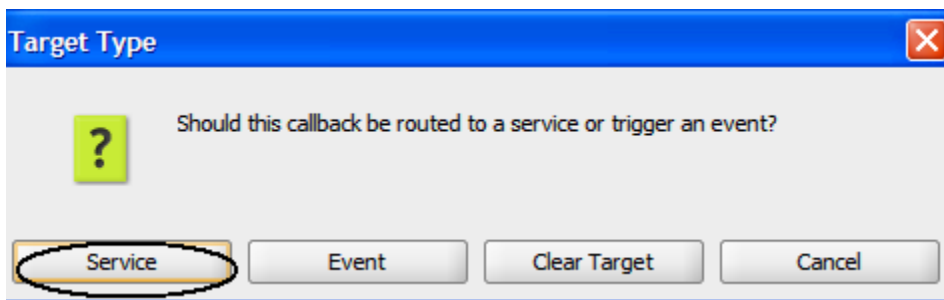
59. Click **OK**.

60. You also need to set the callback of the asynchronous BPEL process to call the file adapter service so that in the case of the large order processing, the order is still written to a file for the fulfillment archive.

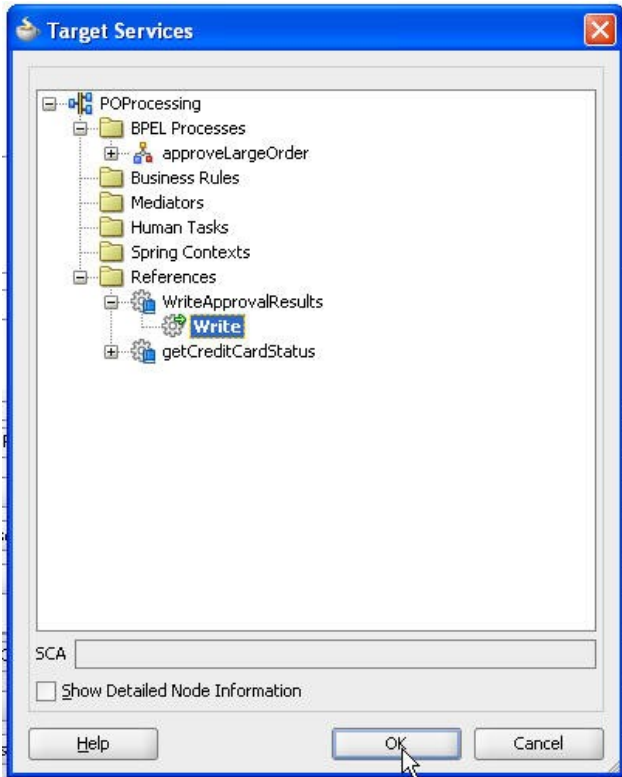
Click the cog icon next to the **Target Operation** field in the callback section.



61. In the **Target Type** dialog, click the **Service** button.



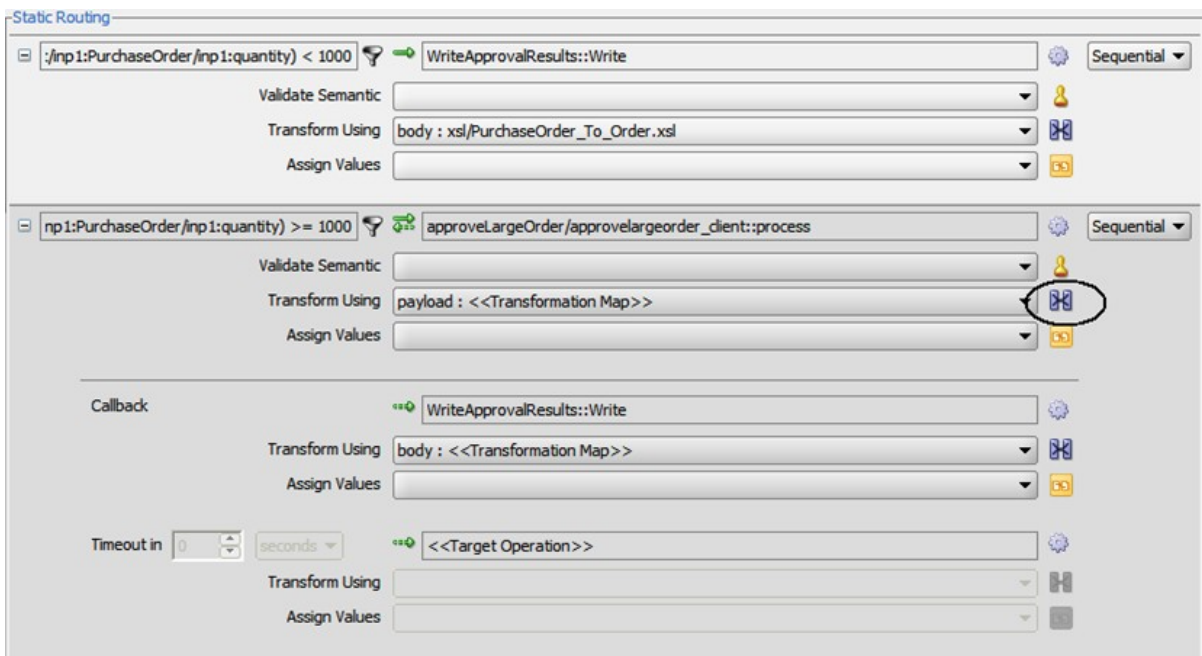
62. In the **Target Services** dialog, select **POProcessing > References > WriteApprovalResults > Write**.



63. Click **OK**.

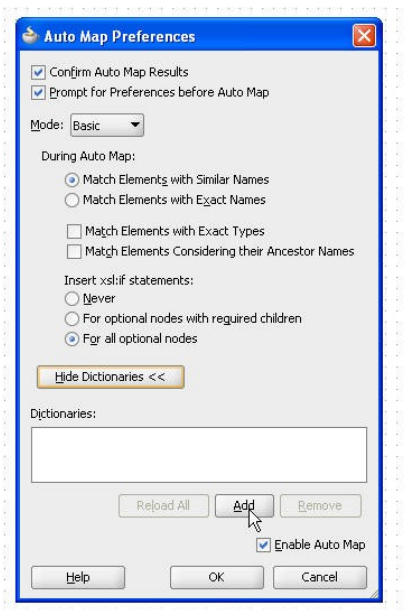
A transformation needs to be added for the operations. It's the same as the transformation done earlier, but the namespaces are different so a new transformation will need to be created.

64. Click the transformation icon in the request section.



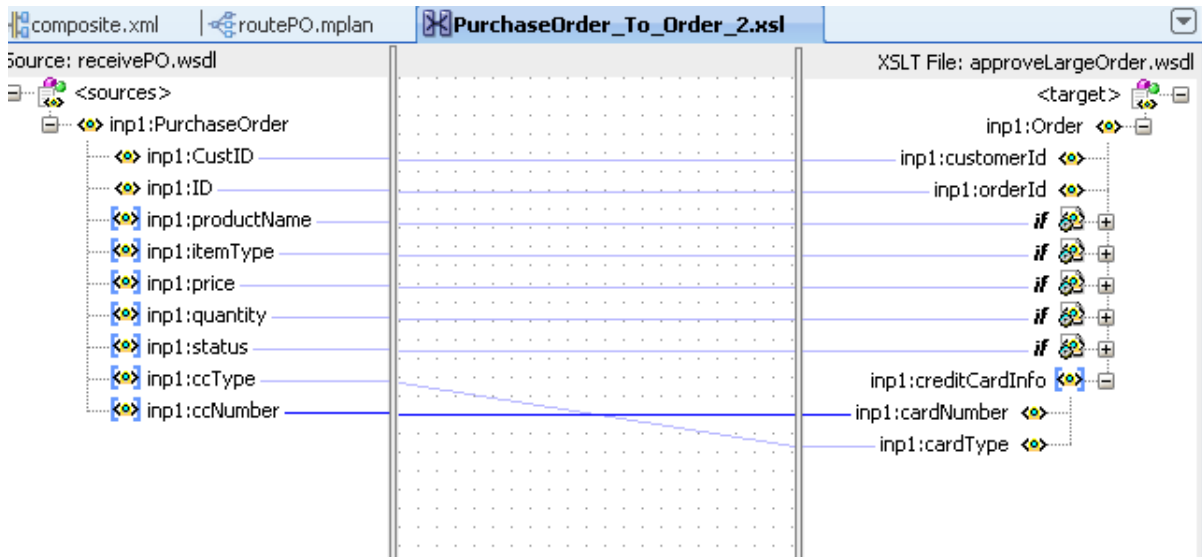
65. Select **Create New Mapper File** and click **OK**.

66. Drag a wire from **PurchaseOrder** in the source to **Order** in target.
67. To help in the mapping, you are going to leverage a dictionary created by the business analysts and that lists common synonyms in use across your data objects. (e.g., “qty” is sometimes used instead of “quantity”, some departments use “ID” instead of “orderId”, etc.).
 1. In the **Auto Map Preferences** dialog, uncheck **Match Elements Considering their Ancestor Names**
 2. Click on **Show Dictionaries**.

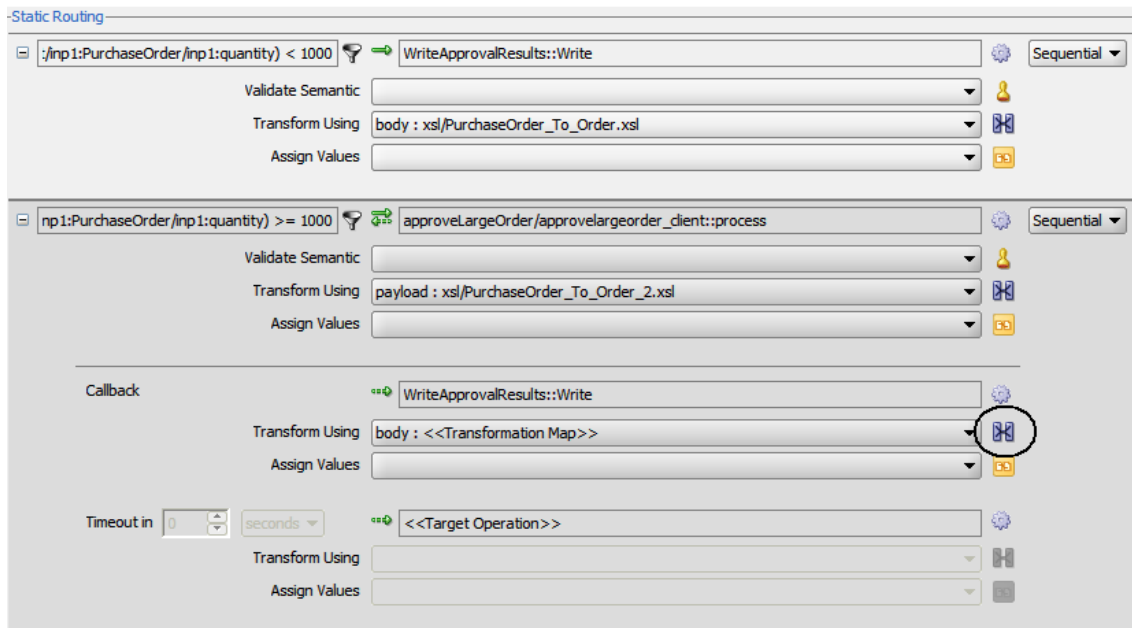


3. Click **Add**.
4. Browse for `C:\Oracle_SOA\Resources\po\schemas\po2internalorders-dictionary.xml`.
68. Click **Open**

The resulting transformation looks like this:

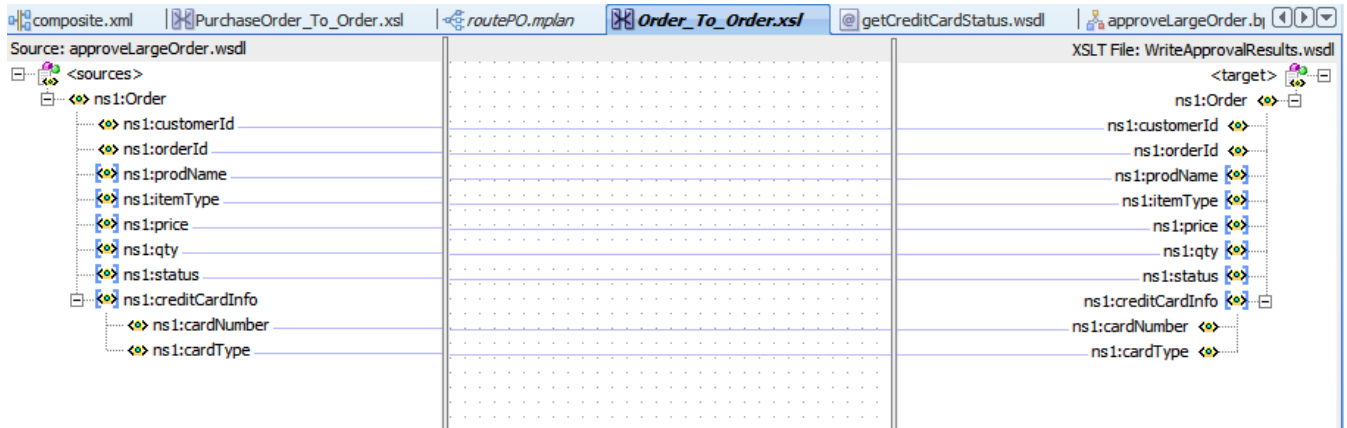


69. Save and close the mapper to return to the Mediator editor.
70. You must also add a transformation for the callback. Click the transformation icon in the callback section.

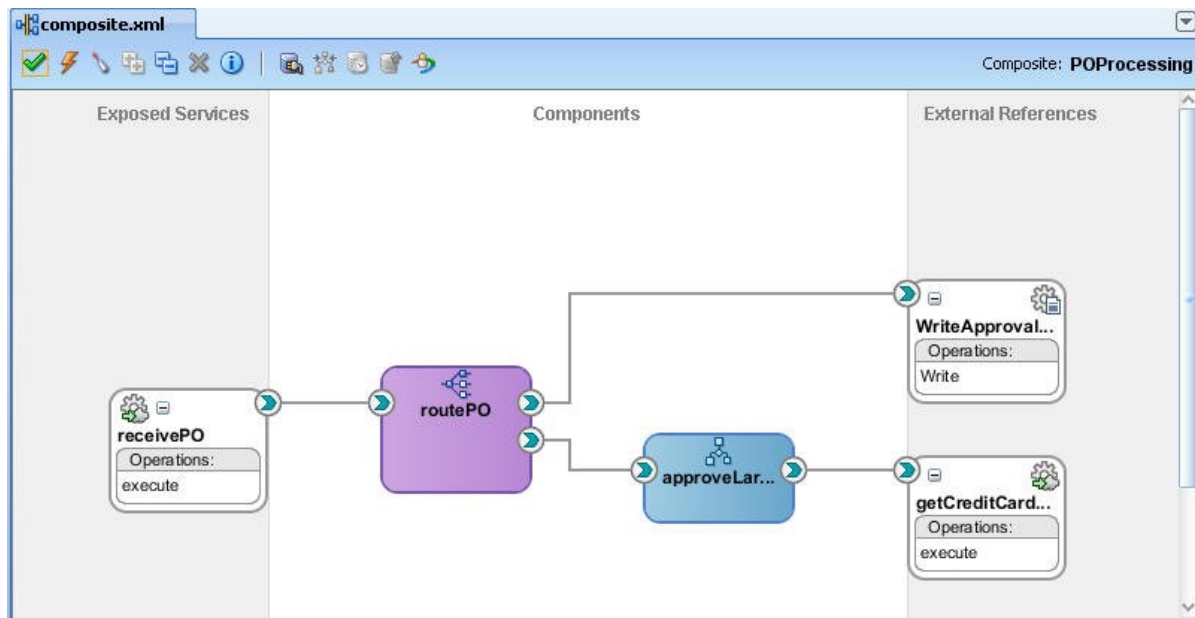


71. Select **Create New Mapper File** and click **OK**.
72. Drag a wire from **Order** in the source to **Order** in the target.
73. In the **Auto Map Preferences** dialog, click **OK**. Click **OK** again.

The resulting transformation looks like this:

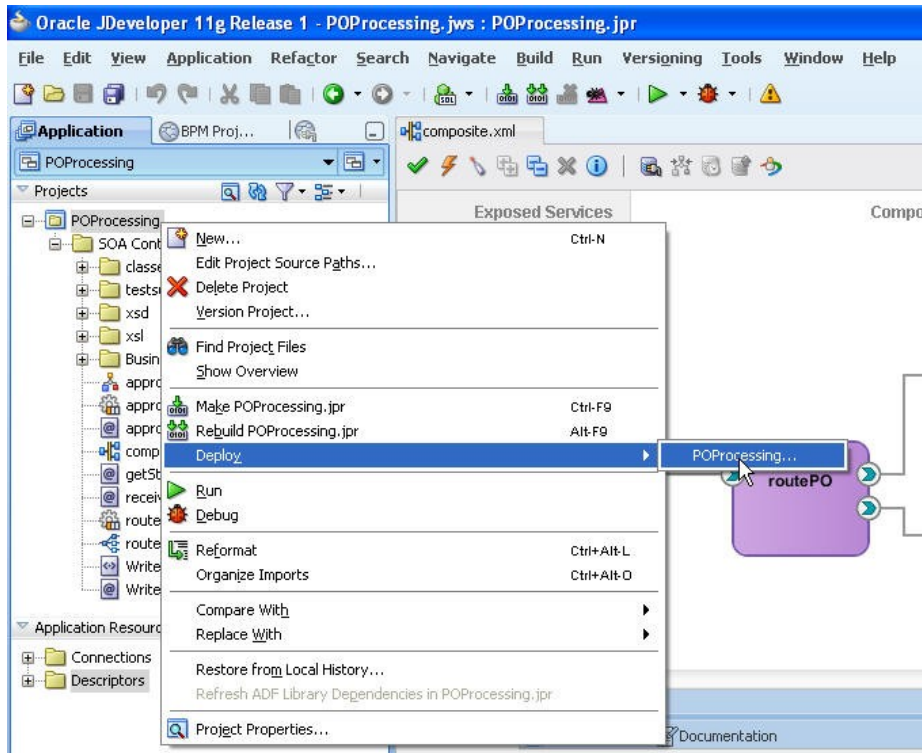


74. Save and close the mapper.
75. Save and close the Mediator editor to return to the composite.
76. Choose File > Save All from the menu or the toolbar just to make sure everything is saved.
77. Use the green checkmark icon to validate the composite.



4.6 Deploying the application

Deploy the application in the same way as before using the **Deploy** command on the Project Menu. If you need to deploy POProcessing more than once, you must either choose a new version number or select **Overwrite any existing composite** when deploying. If you do overwrite the previous composite, the existing instances for the version become stale and can no longer be viewed. So please choose a new version number.



4.7 Testing the application

1. After deploying, open the EM console (<http://<servername>:7001/em>).
2. Click on the **POProcessing** application and then open the Test page.



3. Click **XML View**.
4. In the previous chapter you submitted a small order which created an order file directly. This time you'll create a large order which the Mediator will route to the BPEL approval process.
5. Open the following file in a text editor: `C:\Oracle_SOA\Resources\po\input\po-large-iPodx30.xml`

6. Copy the entire contents and paste them into the large text field in your browser:

Input Arguments

XML View ▼

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body xmlns:ns1="http://xmlns.oracle.com/ns/order">
    <ns1:PurchaseOrder>
      <ns1:CustID>1111</ns1:CustID>
      <ns1:ID>2222</ns1:ID>
      <ns1:productName>iPod shuffle</ns1:productName>
      <ns1:itemType>Electronics</ns1:itemType>
      <ns1:price>145</ns1:price>
      <ns1:quantity>30</ns1:quantity>
      <ns1:status>Initial</ns1:status>
      <ns1:ccType>Mastercard</ns1:ccType>
      <ns1:ccNumber>1234-1234-1234-1234</ns1:ccNumber>
    </ns1:PurchaseOrder>
  </soap:Body>
</soap:Envelope>
```

[Test Web Service](#)

7. Click **Test Web Service**.

78. The **Test Result** screen won't have any response because this is a one-way invocation with no reply or callback.

Request **Response**

Test Status **Passed**

Response Time (ms) 318

XML View ▼

[Launch Message Flow Trace](#)

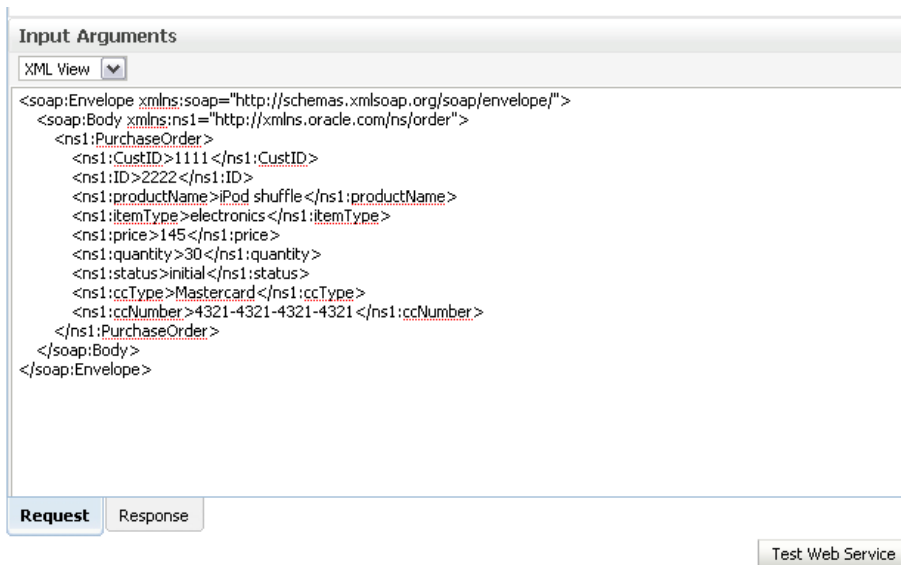
The web service invocation was successful. However, there was no response to the invocation from the server.

However a new `order_n.txt` file will have been created in `/tmp` (or the directory that you define in the WriteApprovalResults component). You can open it in a text editor and view the results. (On the sever, change directories to `/tmp`. “`cd /tmp`”, use “`ls`” to see the list of files).

79. Notice that the value of `<status>` on line 8 has been set to **approved**.

```
[oracle@soabpm-vm tmp]# more order_1.txt
<?xml version="1.0" encoding="UTF-8" ?><ns1:Order xmlns:ns1="http://xmlns.oracle.com/ns/order" xmlns="http://xmlns.oracle.com/ns/order">
  <ns1:customerId>1111</ns1:customerId>
  <ns1:orderId>2222</ns1:orderId>
  <ns1:prodName>iPod shuffle</ns1:prodName>
  <ns1:itemType>Electronics</ns1:itemType>
  <ns1:price>145</ns1:price>
  <ns1:qty>30</ns1:qty>
  <ns1:status>approved</ns1:status>
  <ns1:creditCardInfo>
    <ns1:cardNumber>1234-1234-1234-1234</ns1:cardNumber>
    <ns1:cardType>Mastercard</ns1:cardType>
  </ns1:creditCardInfo>
</ns1:Order>
[oracle@soabpm-vm tmp]#
```

80. Click the Request tab in the browser to return to the request page.
81. Re-run the application using the same input data, but this time change the credit card number to **4321-4321-4321-4321** which represents an invalid credit card.



82. Click **Test Web Service** to run the application.
83. Open the new order file in `/tmp` and notice what the status is this time.

```
[oracle@soabpm-vm tmp]$ more order_2.txt
<?xml version="1.0" encoding="UTF-8" ?><ns1:Order xmlns:ns1="http://xmlns.oracle.com/ns/order" xmlns="http://xmlns.oracle.com/ns/order">
  <ns1:customerId>1111</ns1:customerId>
  <ns1:orderId>2222</ns1:orderId>
  <ns1:prodName>iPod shuffle</ns1:prodName>
  <ns1:itemType>Electronics</ns1:itemType>
  <ns1:price>145</ns1:price>
  <ns1:qty>30</ns1:qty>
  <ns1:status>invalidCreditCard</ns1:status>
  <ns1:creditCardInfo>
    <ns1:cardNumber>4321-4321-4321-4321</ns1:cardNumber>
    <ns1:cardType>Mastercard</ns1:cardType>
  </ns1:creditCardInfo>
</ns1:Order>
[oracle@soabpm-vm tmp]$ _
```

84. This is the result of the `<switch>` statement in your BPEL process.
85. Click **Launch Message Flow Trace** to see the details of the message flow for this instance of your composite.



Alternatively, click on the application. In the **Last 5 Instances** section click on the most recent instance to see the execution flow. Notice that the flow trace shows you the flow through both composites. You can see the composite instance value on the far right.

Trace

Click a component instance to see its detailed audit trail.

Show Instance IDs

Instance	Type	Usage	State	Time	Composite Instance
receivePO	Web Service	Service	Completed	Oct 7, 2010 2:52:48 PM	POProcessing of 90005
routePO	Mediator Component		Completed	Oct 7, 2010 2:52:48 PM	POProcessing of 90005
WriteApprovalResults	JCA Adapter	Reference	Completed	Oct 7, 2010 2:52:48 PM	POProcessing of 90005
approveLargeOrder	BPEL Component		Completed	Oct 7, 2010 2:52:48 PM	POProcessing of 90005
getCreditCardStatus	Web Service(Local Invocatio	Reference	Completed	Oct 7, 2010 2:52:48 PM	POProcessing of 90005
getStatusByCC	Web Service(Local Invocatio	Service	Completed	Oct 7, 2010 2:52:48 PM	validationForCC of 90006
RouteRequest	Mediator Component		Completed	Oct 7, 2010 2:52:48 PM	validationForCC of 90006
getCreditValidation	JCA Adapter	Reference	Completed	Oct 7, 2010 2:52:48 PM	validationForCC of 90006

86. You can click the approveLargeOrder link to look at the BPEL process instance. You can click the various activities to see their results.

Instance	Type
receivePO	Web Service
routePO	Mediator Component
WriteApprovalResults	JCA Adapter
approveLargeOrder	BPEL Component
getCreditCardStatus	Web Service(Local Invocatio
getStatusByCC	Web Service(Local Invocatio
RouteRequest	Mediator Component
getCreditValidation	JCA Adapter

An interesting one to click is the getCreditCardStatus invoke payload.

Flow Trace > Instance of approveLargeOrder

Instance of approveLargeOrder

This page shows BPEL process instance details.

Audit Trail | Flow | Sensor Values | Faults

Expand a payload node to view the details.

```

<process>
  <scope name="main">
    receiveInput
      Oct 7, 2010 2:52:48 PM Received "inputVariable" call from partner "approveLargeOrder_client"
    assignCCNumber
      Oct 7, 2010 2:52:48 PM Updated variable "invokeCCStatusService_execute_InputVariable"
      <payload>
      Oct 7, 2010 2:52:48 PM Updated variable "outputVariable"
      <payload>
      Oct 7, 2010 2:52:48 PM Completed assign
    invokeCCStatusService
      Oct 7, 2010 2:52:48 PM Invoked 2-way operation "execute" on partner "getCreditCardStatus".
      <payload>
        <messages>
          <invokeCCStatusService_execute_InputVariable>
            <part name="request">
              <creditcardStatusRequest>
                <CCNumber>4321-4321-4321-4321</CCNumber>
              </creditcardStatusRequest>
            </part>
          </invokeCCStatusService_execute_InputVariable>
          <invokeCCStatusService_execute_OutputVariable>
            <part name="reply">
              <inp1:creditcardStatus>INVALID</inp1:creditcardStatus>
            </part>
          </invokeCCStatusService_execute_OutputVariable>
        </messages>
      </payload>
    <scope name="EvaluateCCStatus">
      switchNode (105)
        Oct 7, 2010 2:52:48 PM Switch otherwise is selected.
        Oct 7, 2010 2:52:48 PM Switch otherwise is selected.
      assignInvalidCC
        Oct 7, 2010 2:52:48 PM Updated variable "outputVariable"
        <payload>
        Oct 7, 2010 2:52:48 PM Completed assign
      callbackClient
  </scope>
  </scope>
</process>

```

87. When you click the **getCreditCardStatus** invoke payload, it's a synchronous (request-response) call, so you see both the input to the service (i.e., what you're passing to the service) and the output (i.e., what you're getting back from the service).

In the screenshots, below, the input to the service is the credit card number **4321-4321-4321-4321** and the output returned is **INVALID**.

 **invokeCCStatusService**

Mar 3, 2009 2:13:58 PM Invoked 2-way operation "execute" on partner "getCreditCardStatus".

<payload>

```
<messages>
  <invokeCCStatusService_execute_InputVariable>
    <partname="request">
      <creditcardStatusRequest>
        <CCNumber>4321-4321-4321-4321</CCNumber>
      </creditcardStatusRequest>
    </part>
  </invokeCCStatusService_execute_InputVariable>
  <invokeCCStatusService_execute_OutputVariable>
    <partname="reply">
      <inp1:creditcardStatus>INVALID</inp1:creditcardStatus>
    </part>
  </invokeCCStatusService_execute_OutputVariable>
</messages>
```

88. Back in the main screen of the EM console, run the Test page again and this time use the small order, which can be found in C:\Oracle_SOA\Resources\po\input\po-small-Headsetx1.xml

Note that the BPEL process does not get invoked and instead you only get a new order file generated in /tmp (or the directory that you defined in the file adapter component).

```
[oracle@soabpm-vm tmp] $ more order_3.txt
<?xml version="1.0" encoding="UTF-8" ?><inp1:Order xmlns:inp1="http://xmlns.oracle.com/ns/order" xmlns="http://xmlns.oracle.com/ns/order">
  <inp1:customerId>1111</inp1:customerId>
  <inp1:orderId>2121</inp1:orderId>
  <inp1:prodName>Bluetooth Headset</inp1:prodName>
  <inp1:itemType>Electronics</inp1:itemType>
  <inp1:price>49.99</inp1:price>
  <inp1:qty>1</inp1:qty>
  <inp1:status>Initial</inp1:status>
  <inp1:creditCardInfo>
    <inp1:cardNumber>8765-8765-8765-8765</inp1:cardNumber>
    <inp1:cardType>Mastercard</inp1:cardType>
  </inp1:creditCardInfo>
</inp1:Order>
```