Practices for Lesson 6: Introduction to BPEL

Practices for Lesson 6: Overview

Practices Overview

In this practice, you learn how to use the following BPEL activities:

- Receive and Reply
- Invoke
- Assign
- Define Partners

Practices Overview

In this practice, you learn how to use the following BPEL activities:

- Receive and Reply
- Invoke
- Assign
- Define Partners

Overview

In this practice, you learn how to use the following BPEL activities:

Practice 6-1: Creating a New Application

Overview

In this practice, you create a composite application that includes a simple BPEL process. This process accepts input and calls the credit card validation (`CCValidate`) project that you built earlier.

Assumptions

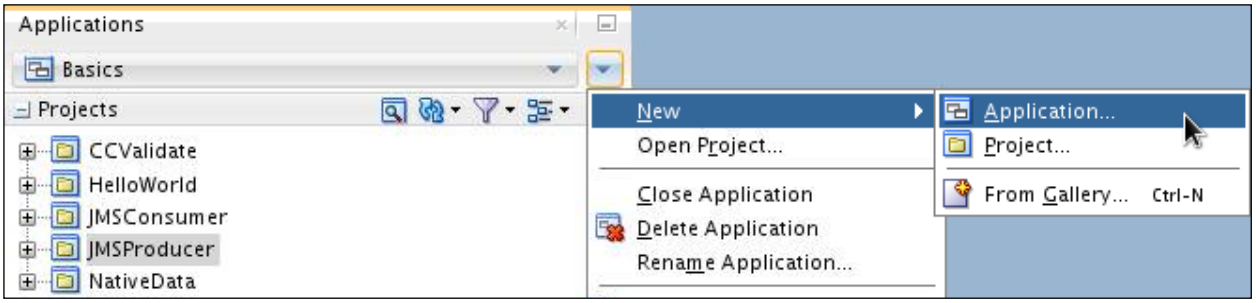
This practice assumes the following:

- The JDeveloper IDE is open.
- You have successfully built, deployed, and tested the project (`CCValidate`) in Practice 5.

Tasks

Creating a New Application and Project

1. In this section, you create a new workspace named BPELProjects, and then create a composite application in this workspace.
2. In the JDeveloper Application Navigator, select New > Application from the **Applications** menu.



3. Use the instructions in the following table to create the application workspace with a SOA project that contains a composite application. Accept the default values for all fields that are not specified in the “Choices or Values” column.

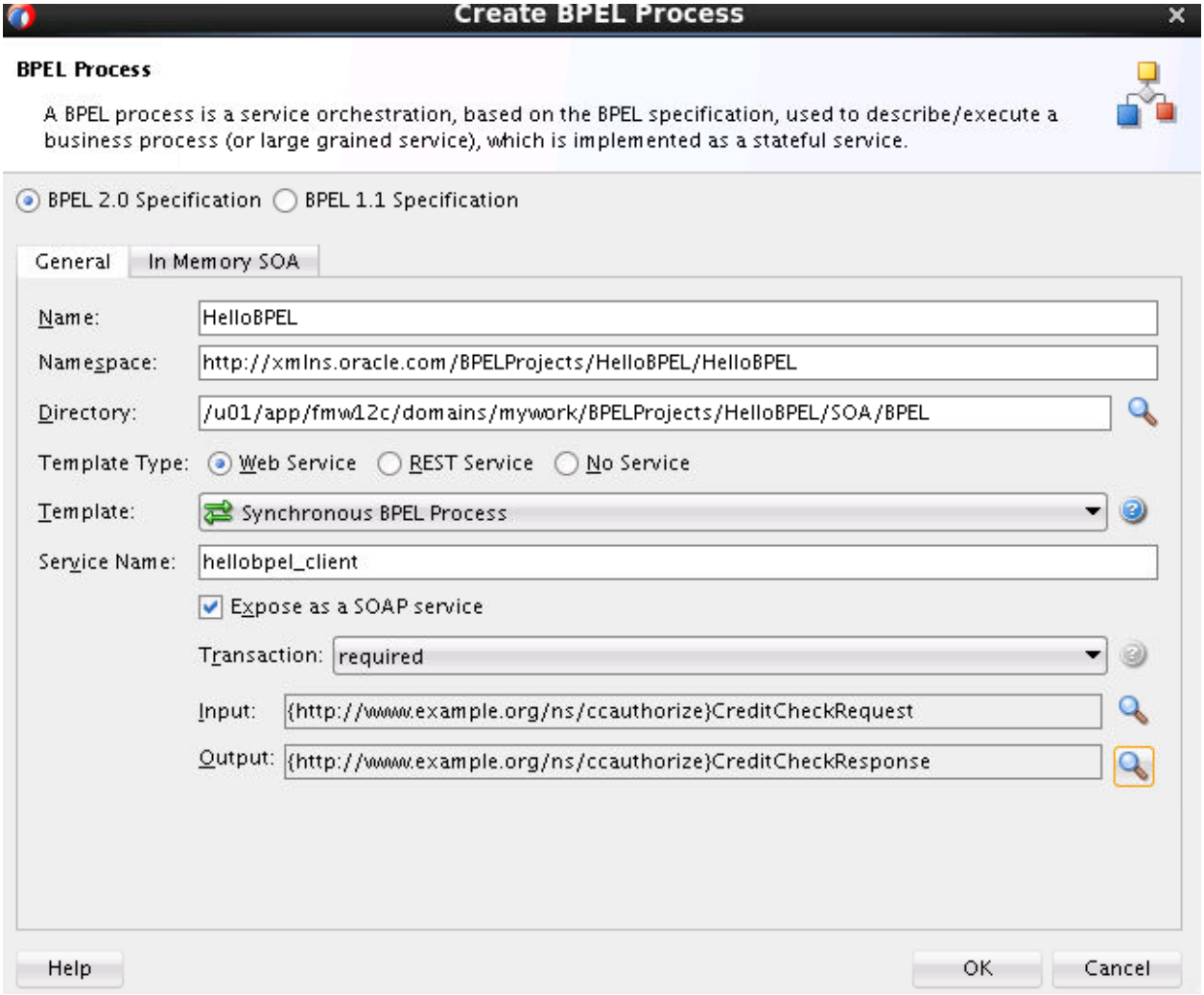
Window	Choices or Values
New Gallery	Application Template: SOA Application Click OK.
Name your application	Application Name: `BPELProjects` Click Next.
Project Name	Project Name: `HelloBPEL` Click Next.
Project SOA Settings	Composite Template: Composite with BPEL Process Click Finish.

The Create BPEL Process wizard appears.

4. Complete the steps in the following table to create the BPEL Process:

Window	Choices or Values
BPEL Process	Name: `HelloBPEL` Template: Synchronous BPEL Process Template Type: Web Service Make sure “Expose as a SOAP service” is selected Click the Browse icon to the right of Input. The Type Chooser opens.
Type Chooser	Click the Import Schema File icon.
Import Schema File	With File System selected, navigate to `/home/oracle/labs/files/xsd`. Select `creditcheck.xsd`. Click OK.
Localize Files	Click OK.
Type Chooser	Select the message element CreditCheckRequest. Click OK.
Create BPEL Process	Click the Browse icon to the right of Output.
Type Chooser	Select the message element CreditCheckResponse. Click OK.

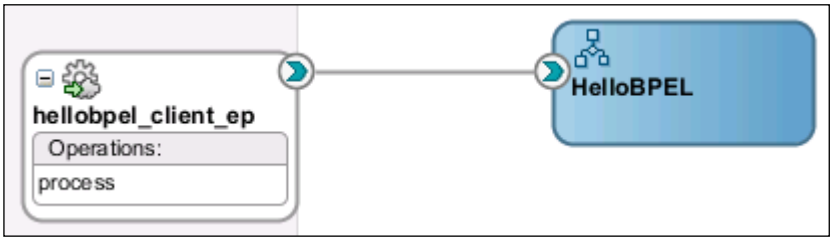
5. Compare your work to the following screenshot:



6. Click OK.

   The composite application design window is updated and the BPEL editor is opened.

7. Click the composite overview (HelloBPEL) tab and verify your composite.



8. Add and configure a SOAP web service to the composite as an external reference.
   a. Drag a SOAP web service to the External Reference swimlane.

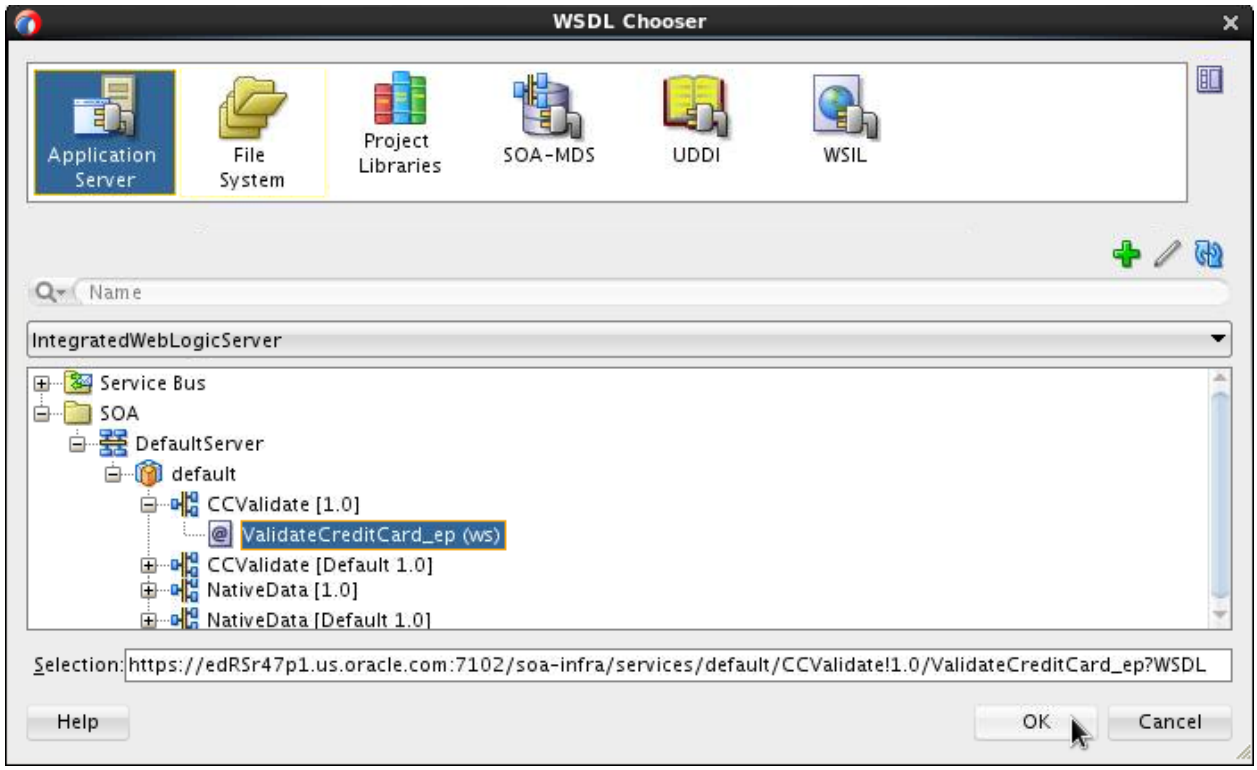
      The Create Web Service window opens.

   b. Name the service `ValidateCC`.
   c. Define the WSDL URL.

d. Click the Find Existing WSDLs button.

The WSDL Chooser opens.

e. With Application Server selected, navigate to SOA > DefaultServer > default > CCValidate [1.0] > ValidateCreditCard_ep (ws).



f. Click OK to accept the choice and close the WSDL Chooser.

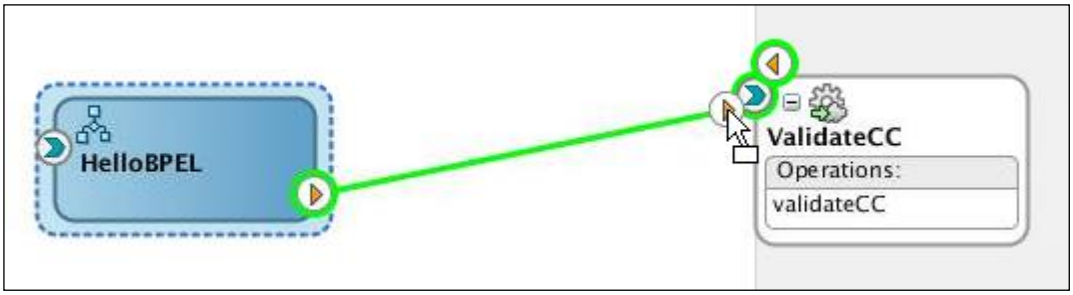
g. Click OK to close the Create Web Service window.

The Localize Files window opens.

h. Click OK.

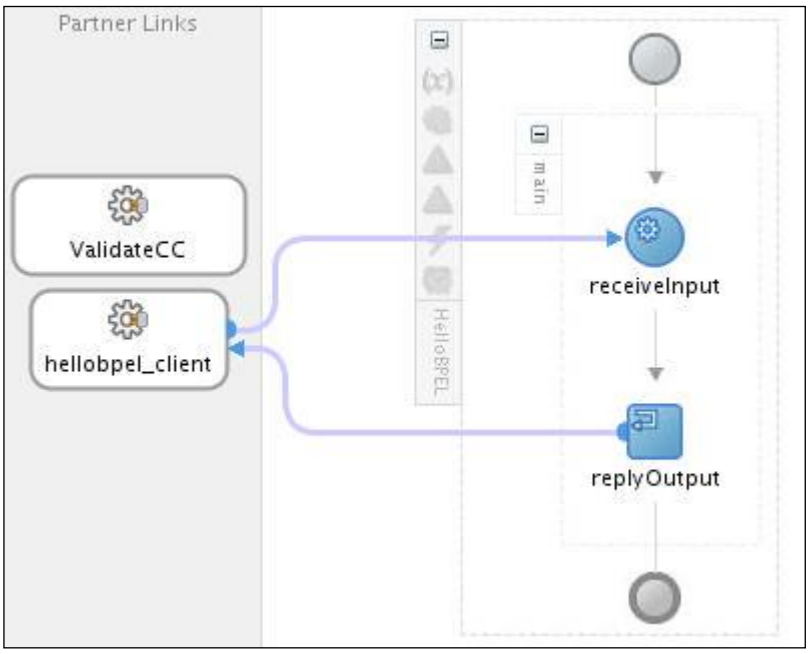
i. Save your work.

9. Create a wire from the BPEL Process to the new SOAP web service.



10. Double-click the HelloBPEL.bpel icon and the HelloBPEL tab opens.

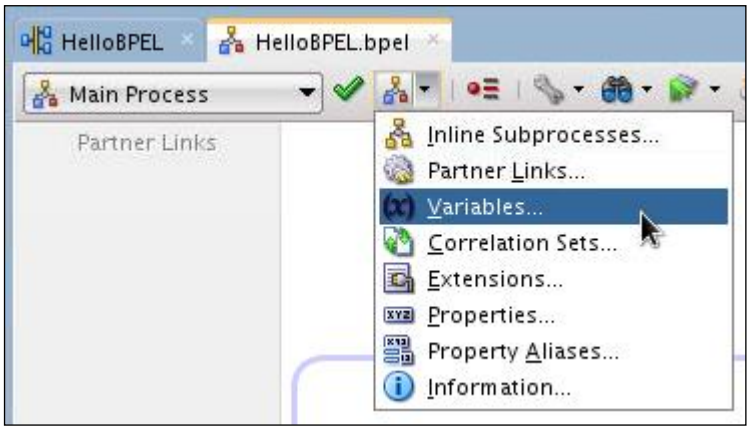
    The business process is created with the Receive and Reply activities.



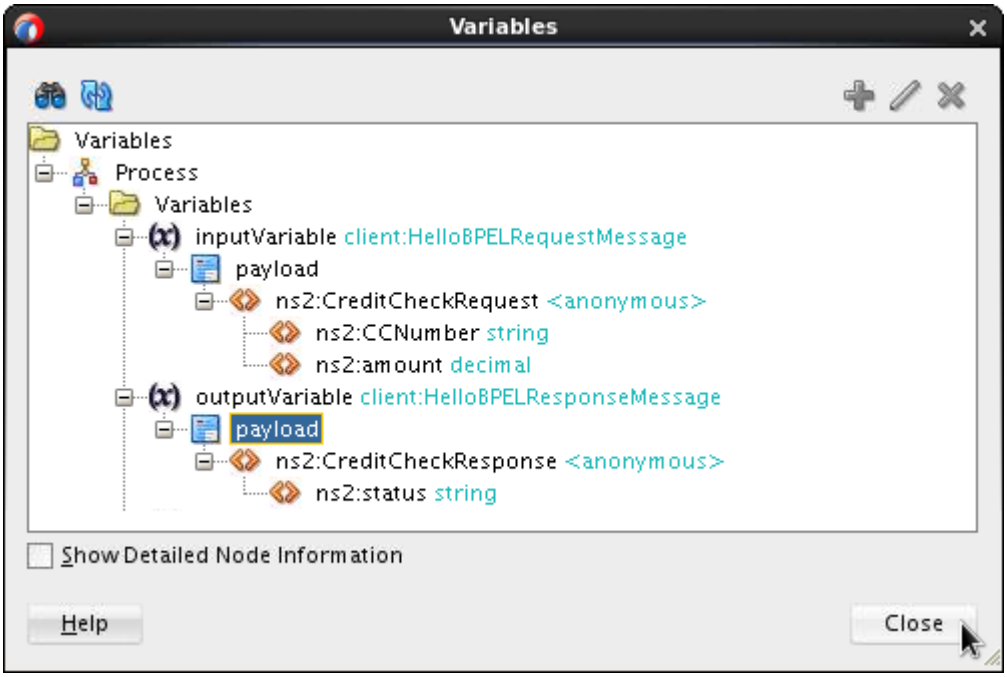
11. Optional Step: Pause to check your understanding.

    If you are new to BPEL, this is a good time to pause and check your understanding of the elements of the business process that are already in place.

    a. From the BPEL editor menu, select Property Structure > Variables.



The process input and output variables are shown.

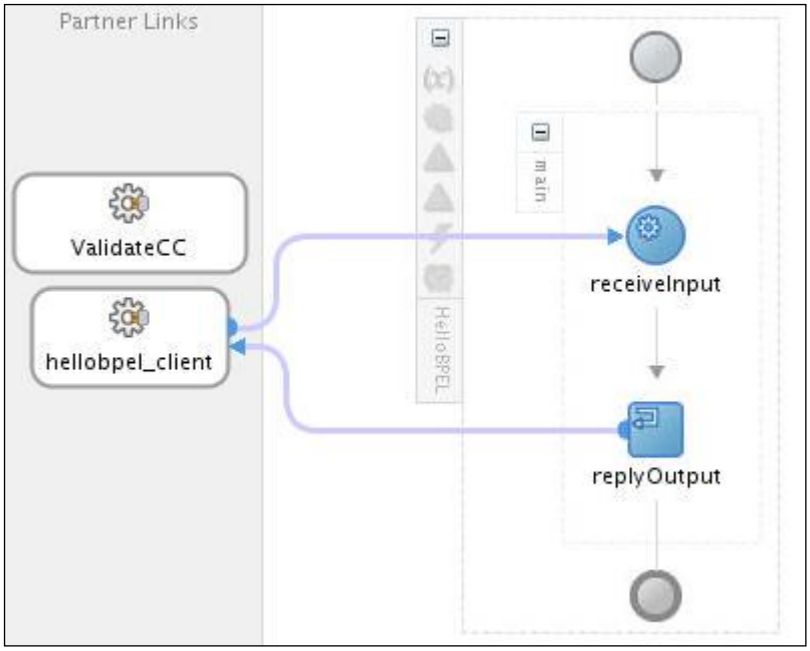


b. Consider the answer to the following questions:

- What action did you take to create them?
- Why were both the input and output variables created?

c. Close the variables pane.

d. Identify the Service Interface and the Activities in the following screenshot:

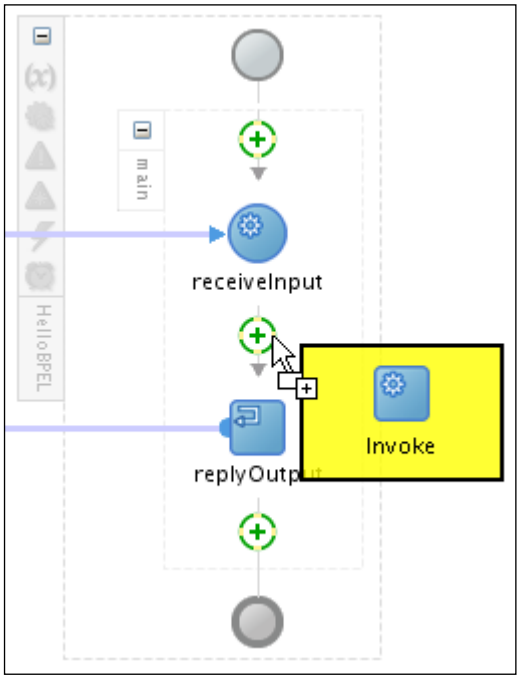


e. If you are comfortable with the vocabulary and concepts that you have just reviewed, continue building your BPEL process.

12. Create and configure an Invoke activity in the business process.

    This activity calls the `ValidateCC` project that was deployed in Practice 5.

    a. Add an Invoke activity to the process between the Receive and Reply activities.



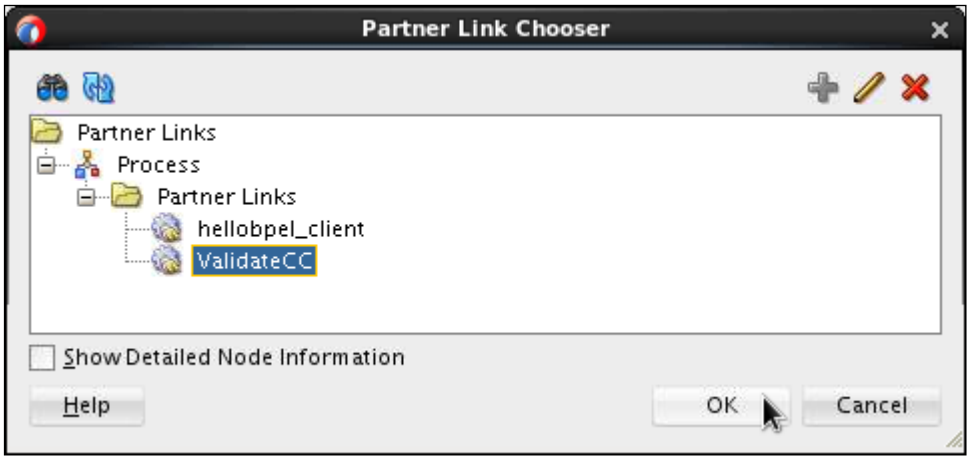
    b. Right-click the Invoke activity and select Edit.

    c. Rename the activity `Invoke_ValidateCC`.

    d. Click the Browse icon to the right of the Partner Link field.

    The Partner Link Chooser dialog box opens.

    e. Select the Partner Link ValidateCC.



    f. Click OK.

    You are returned to the Invoke configuration window.

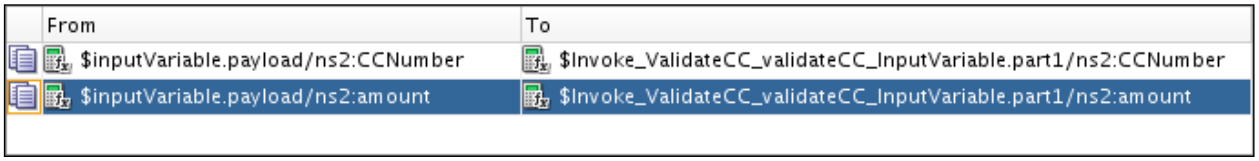
    g. Click the Add (green plus) icon to create an input variable.

    h. Click OK to accept the default parameters.

    i. Click the Output tab.

j. Click the Add icon to create an output variable.
k. Click OK to accept the default parameters.
l. Click OK to complete the Invoke configuration.

13. Create and configure an Assign activity to populate the input variable for the ValidateCC service.
    a. Add an Assign activity above `Invoke_ValidateCC`.
    b. Name the activity `Assign_ValidateCC`.
    c. Right-click Assign and select Edit.
    d. Map the `CCNumber` node of `inputVariable` to the `CCNumber` node of `Invoke_ValidateCC_validatecc_InputVariable`.
    e. Map the `amount` node of `inputVariable` to the `amount` node of `Invoke_ValidateCC_validatecc_InputVariable`.



    f. Click OK.

14. Add an Assign to map the ValidateCC service output to the process response message.
    a. Add an Assign below the Invoke.
    b. Right-click Assign and select Edit.
    c. Name the activity `Assign_Response`.
    d. Map the `status` node of `Invoke_ValidateCC_validatecc_OutputVariable` to the `status` node of `outputVariable`.
    e. Verify your work and click OK.

    f. Save your work.

Practice 6-2: Deploying and Testing the Application

Overview

In this practice, you deploy and test the HelloBPEL composite application.

Assumptions

This practice assumes that you have completed all previous steps in this practice successfully and that JDeveloper is open.

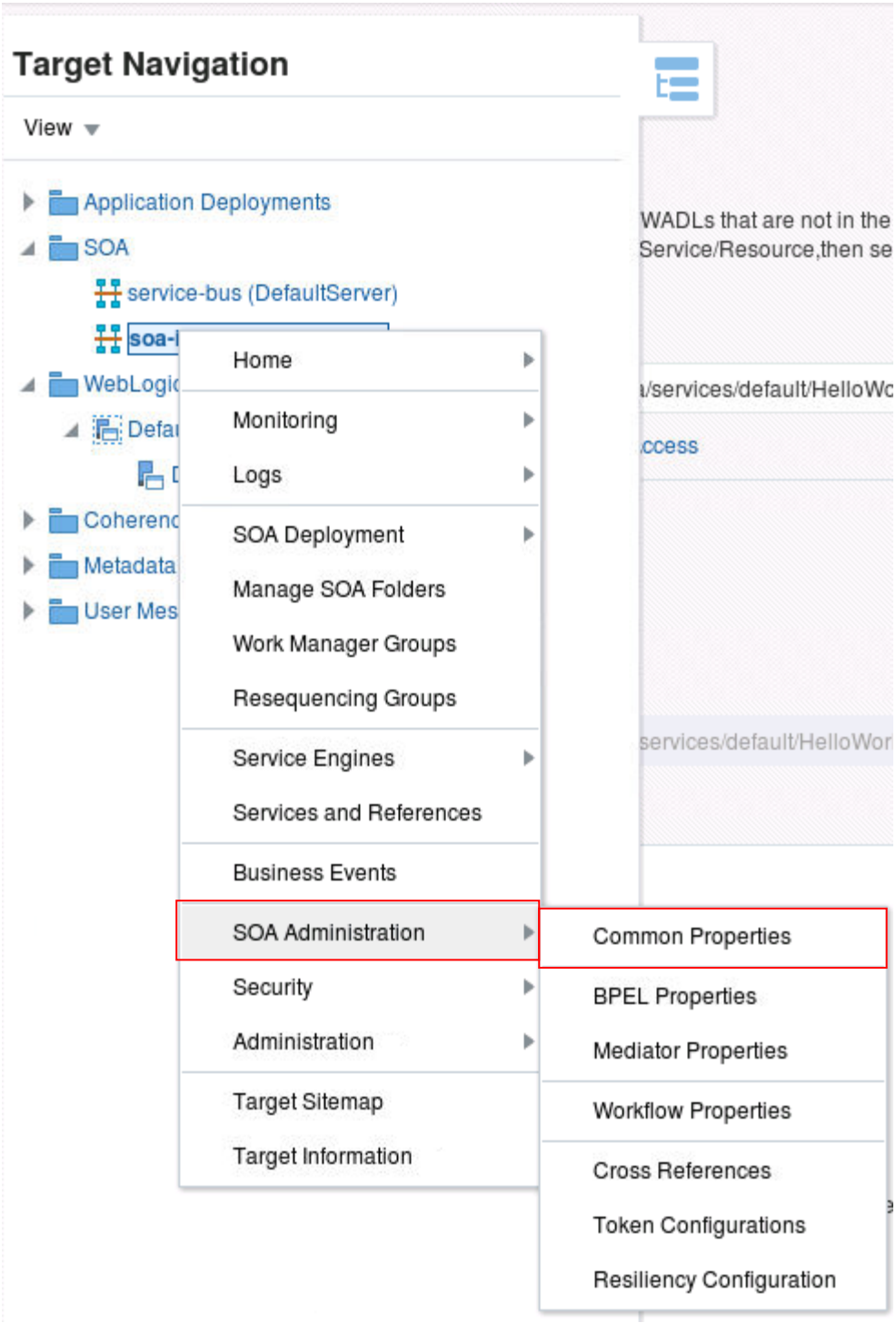
Tasks

1. In the JDeveloper Application Navigator window, deploy the HelloBPEL composite application to IntegratedWebLogicServer.
   - Remember to select the “Overwrite any existing composites with the same revision ID” check box.
   - Monitor the “Deployment – Log” window to ensure that deployment is successful.
2. Update the Audit Level to Development.

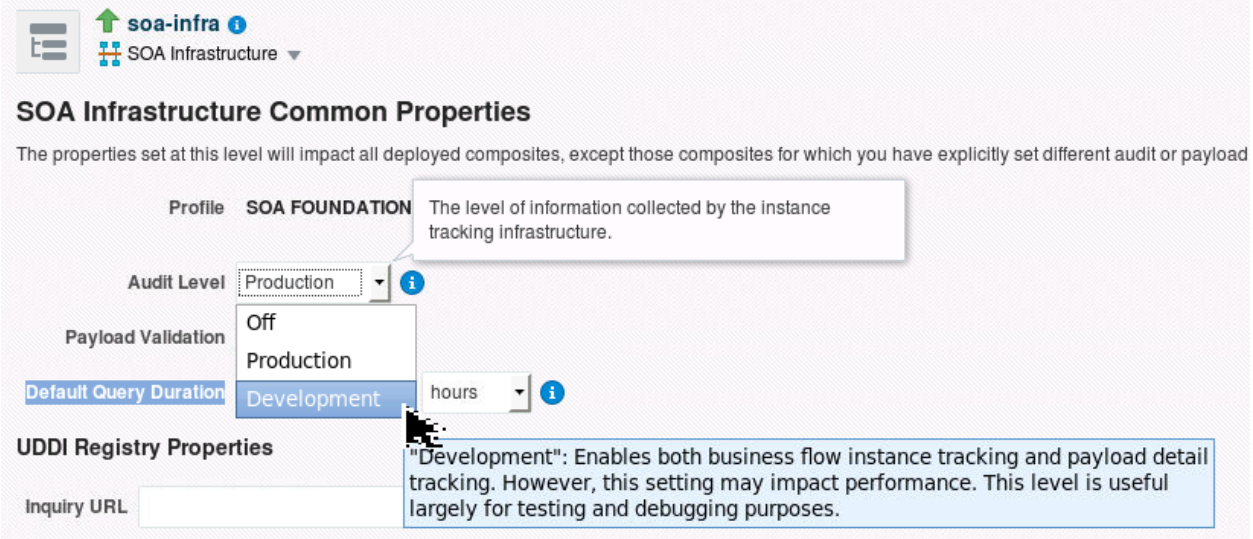
   Note: This change causes more detail to be visible in the Enterprise Manager test tool, which may be of value when debugging this and the subsequent practices.

   a. In a web browser window, access Enterprise Manager at http://localhost:7101/em.

b. In the Target Navigation pane, right-click soa-infra and select SOA Administration > Common Properties.



c. In the Common Properties pane, use the drop-down list to update the Audit Level to Development.

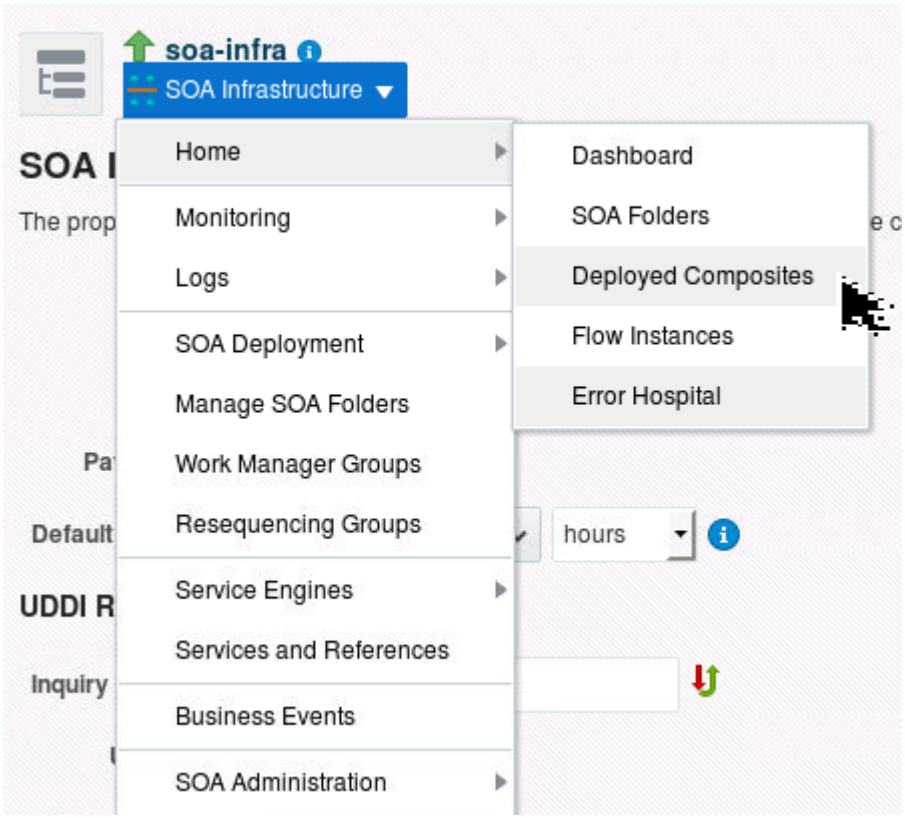


d. Click Apply.

e. In the Confirmation dialog box, click Yes.

3. Test the project.

a. From the SOA Infrastucture pull-down, select Home > Deployed Composites.



b. Click the HelloBPEL [1.0] link.

c. On the HelloBPEL [1.0] home page, click Test.

d. On the Test Web Service page, scroll down to the Input Arguments section of the Request tab and enter the following values:
   - CCNumber: 1234-1234-1234-1234
   - Amount: 1000

e. Click Test Web Service.

   The Response tab is displayed.

f. Verify that the status value returned is `VALID`.

4. Perform a second test to display the INVALID response.
   a. On the Request tab, modify the CCNumber value to be `4321-4321-4321-4321`.
   b. Leave the amount at `1000`.
   c. Click Test Web Service.
   d. On the Response tab page, verify that the status string value returned is `INVALID`.

Points to Consider

This test and the outcome are identical to the test for the CCValidate project that you built earlier. We could have defined the Database adapter as a partner link in this process. It seems like an extra step to have called the composite application from this process.

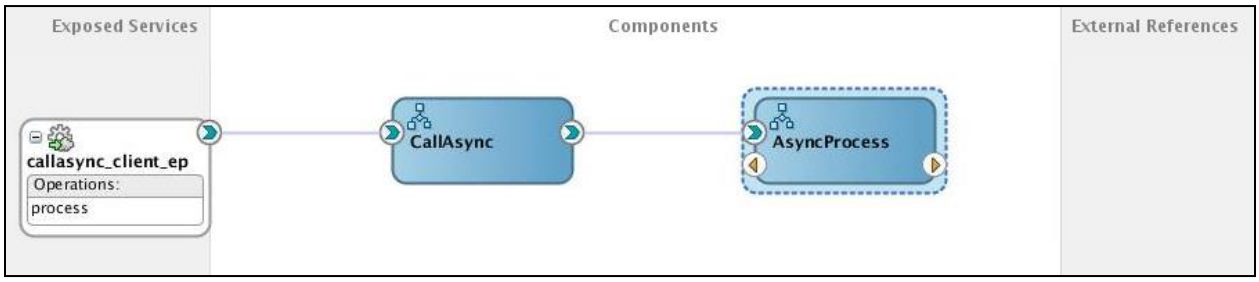
- When would it make sense to incorporate credit card validation directly into a business process?
- When would this architecture (calling one application from another) make sense?

Practices for Lesson 7: More BPEL Activities

Practices for Lesson 7: Overview

Practices Overview

This practice demonstrates calling an asynchronous service. To do this, you build a composite application with two BPEL processes. The first of these is an asynchronous BPEL process. The process uses the While and Wait activities to loop for a specified amount of time. The time is specified by a value that is passed in the incoming message. The second process calls the asynchronous service, and specifies how long that process should loop. The calling process then uses a Pick activity with both an onMessage and an onAlarm branch. The onAlarm branch specifies how long to wait for a response. If the called process returns a message before onAlarm is activated, the onMessage branch executes. If not, the onAlarm branch is triggered instead.



Before You Begin

The path references in this and other practices include XML namespace prefixes (for example, the `ns2` in **`ns2:`**`product`). If you have built each of the components in the order prescribed in the instructions, your namespace references will match. If you have built the components in a different order, or have deleted, and then re-created the components, your namespace prefixes may not match what is listed in these instructions. In that case, *use the namespace references in your project. Do **NOT** change those references to match the book.*

Best Practices

In this practice, you build a synchronous BPEL process, which calls an asynchronous BPEL process. This is not a common design pattern in real-world implementations. It suits our goals for this practice very well, but it would not be a desirable choice for most applications outside the classroom.

Practice 7-1: Creating a New Project

Overview

In this practice, you create a new composite application that includes a BPEL project.

Assumptions

This practice assumes that the JDeveloper IDE is open.

Tasks

Creating a New Project

In this section, you create a new composite application in the BPELProjects workspace.

1. Create a composite application.
   a. In the JDeveloper Application Navigator Project menu, select New > Project.
      The Project wizard opens.
   b. Complete the steps in the following table:

Window	Choices or Values
New Gallery	SOA Project Click OK.
Name your project	Project Name: `AsyncDemo` Click Next.
Project SOA Settings	Composite Template: Composite With BPEL Process Click Finish.

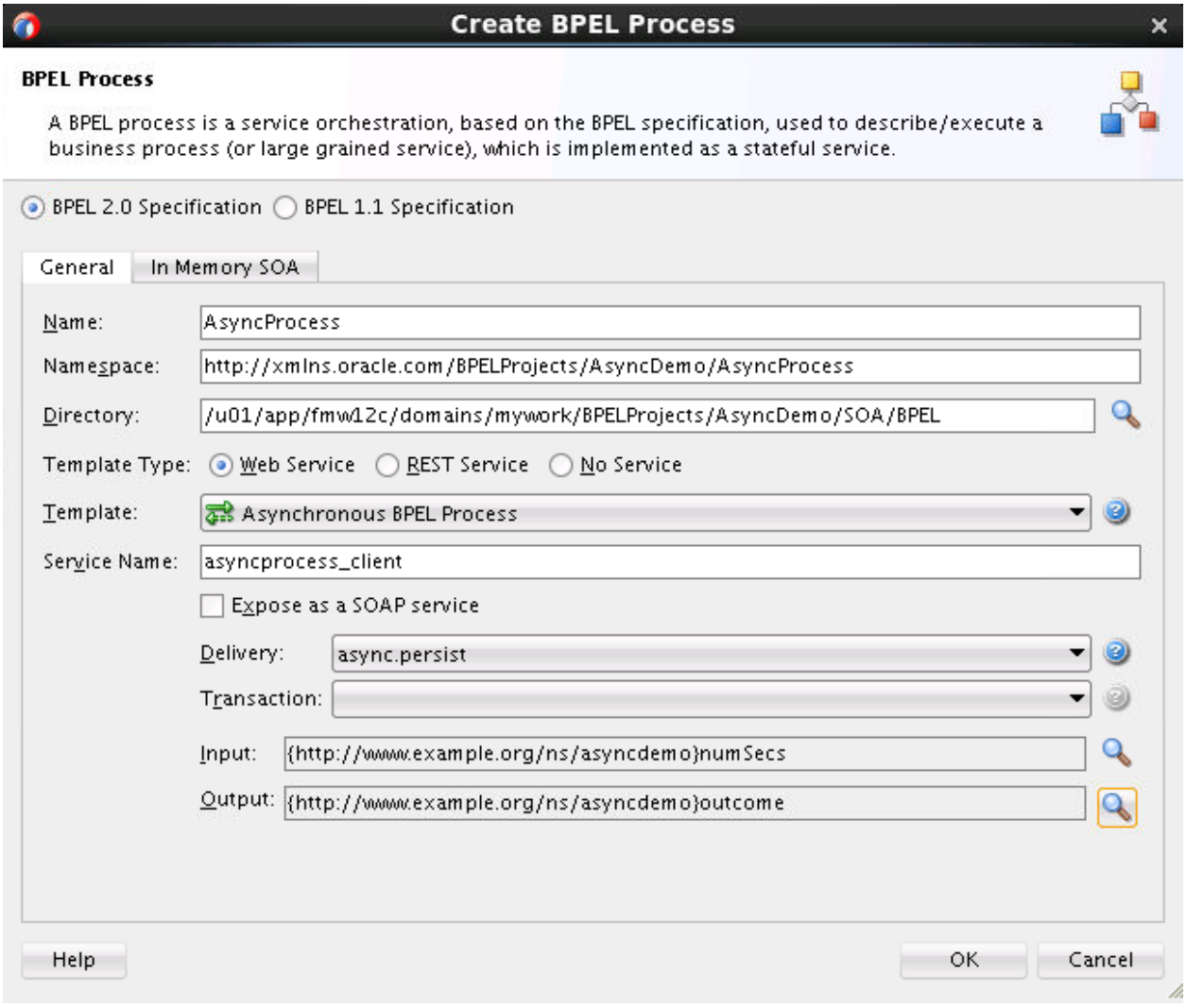
   The Create BPEL Process wizard opens.

2. Use the instructions in the following table to configure the BPEL process:

Step	Window	Choices or Values
a.	BPEL Process	Name: `AsyncProcess` Template: Asynchronous BPEL Process (default) Deselect the “Expose as a SOAP Service” check box. Click the Browse Input Elements icon. The Type Chooser opens.
b.	Type Chooser	Click the Import Schema File icon.
c.	Import Schema file	Select File System. Navigate to `/home/oracle/labs/files/xsd`. Select `asyncDemo.xsd`. Click OK.

Step	Window	Choices or Values
d.	Localize Files	Click OK.
e.	Type Chooser	Select the message element `numSecs`. Click OK.
f.	Create BPEL Process	Click the Browse Output Elements icon to the right of Output.
g.	Type Chooser	Select the `asyncDemo.xsd` message element `outcome`. Click OK.

3. Compare your work to the following screenshot and click OK:



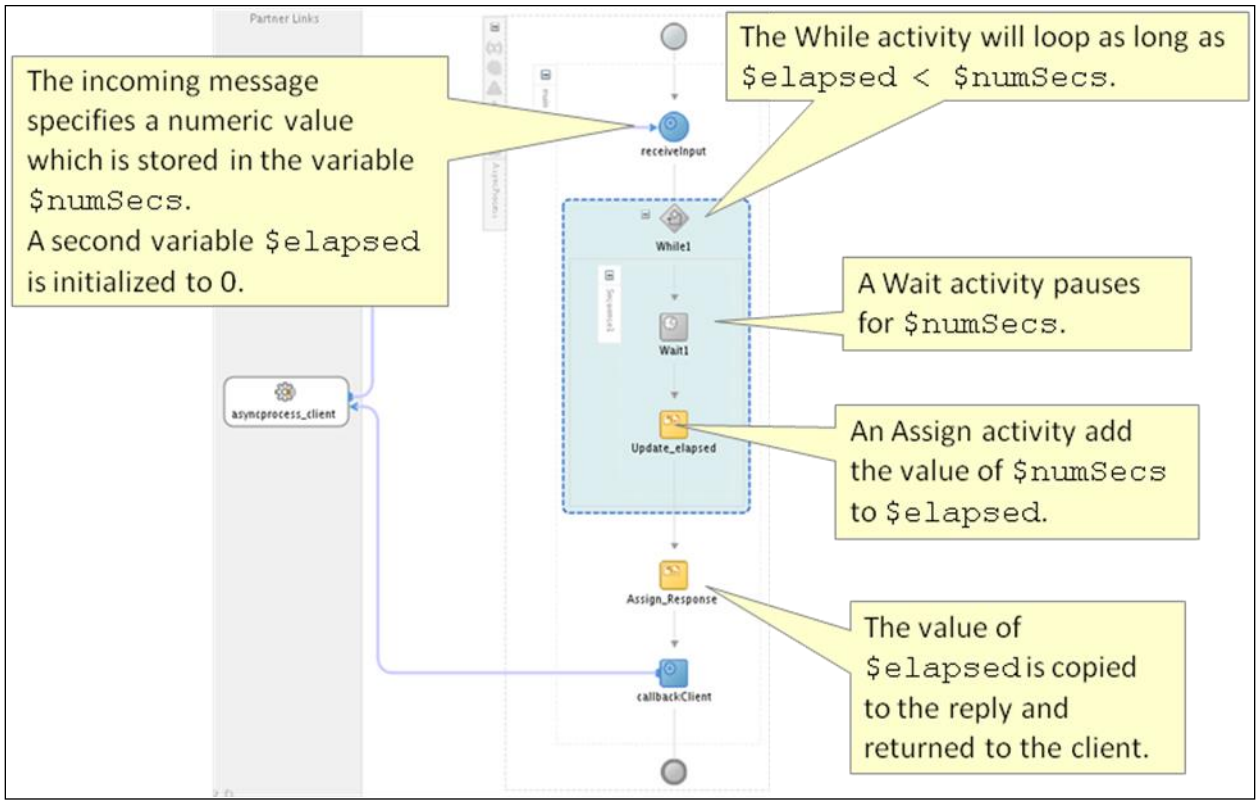
The BPEL Editor opens.

4. Save your work.

Practice 7-2: Configuring the Asynchronous Service

Overview

In this practice, you create an asynchronous service that includes a BPEL process. The process uses the While and Wait activities to loop for a specified amount of time. The time is specified by a value that is passed in the incoming message.



Assumptions

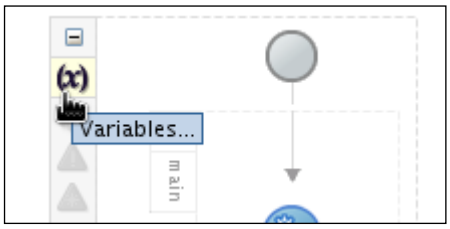
This practice assumes that JDeveloper is open, and that you have completed Practice 7-1 successfully.

Tasks

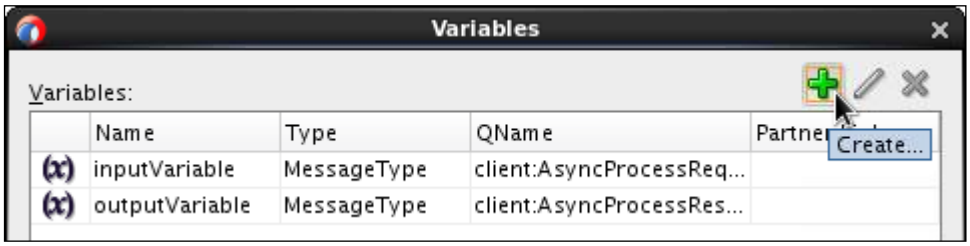
Creating a Counter Variable

The counter variable counts the number of iterations of the While loop.

1. Create and initialize a variable named `elapsed` of type `int`.
   a. In the main process scope, click the Variables icon.

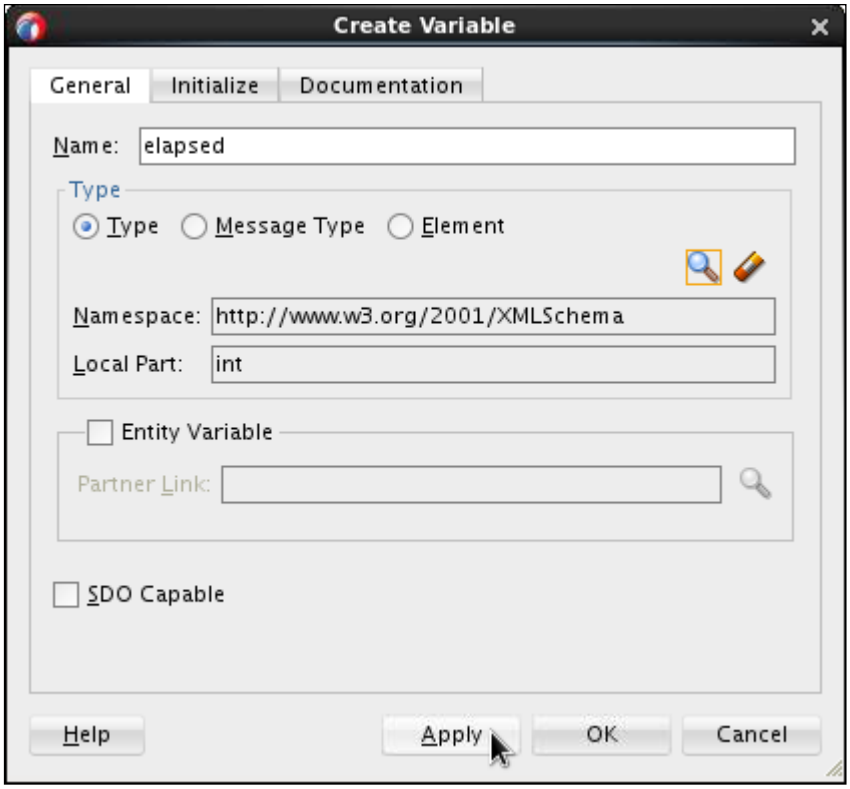


   b. In the Variables dialog box, click the Create icon.



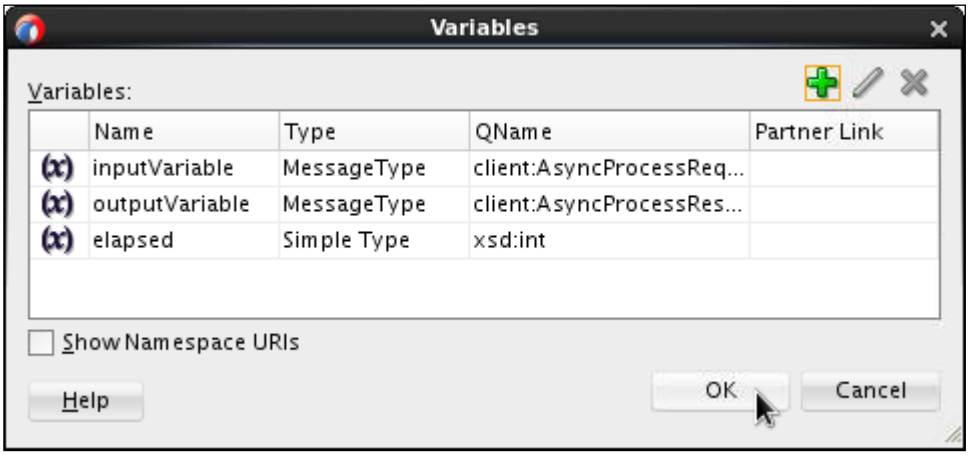
   The Create Variable dialog box is displayed.

   c. To define the variable, perform the following steps:
      1) Enter `elapsed` in the Name field.
      2) Select the Type option.
      3) Click the Browse icon next to the Type option to open the Type Chooser window.
      4) In the Type Chooser window, select the “XML Schema Simple Types” > int entry.
      5) Click OK.



      6) Click Apply. (Do not click OK.)

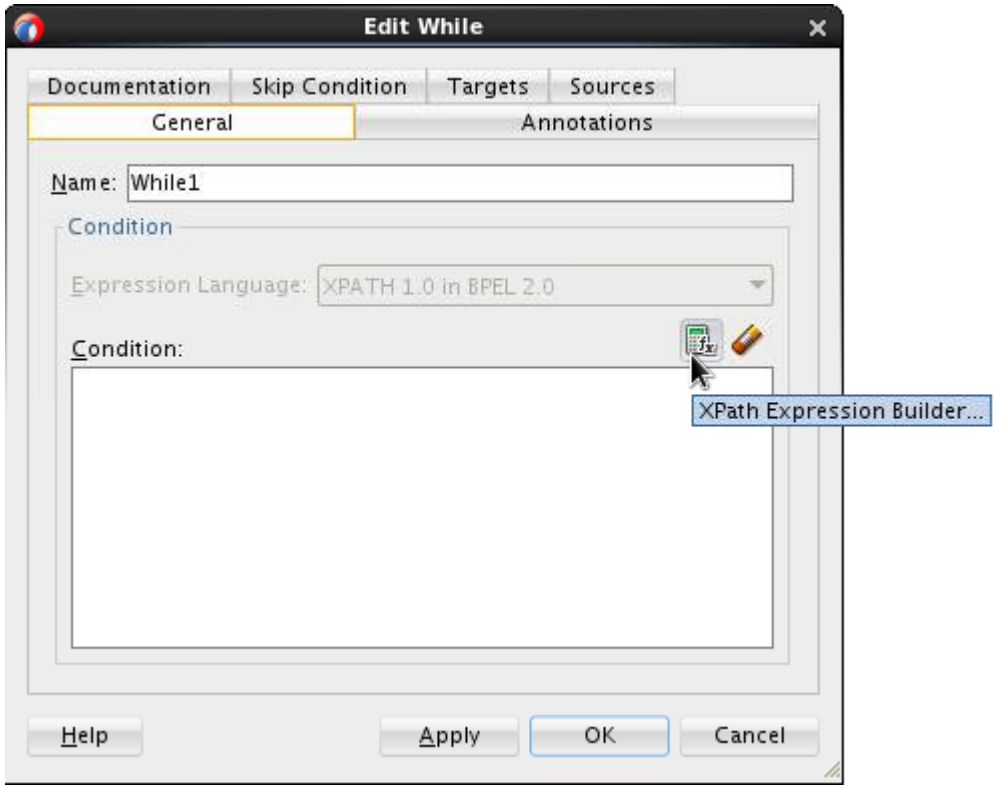
d. To define the variable, perform the following steps:
   1) Click the Initialize tab.
   2) Select Data Source: Expression
   3) In the Expression pane, enter the number `0`.
   4) Click OK to close the Create Variable dialog box.



e. Verify your work and click OK to close the Variables dialog box.

Creating a While Loop

2. Create and configure a While activity so that the loop continues when `$elapsed < $inputVariable`.
   a. Add a While activity after the receiveInput activity.
   b. Right-click the While icon and select Edit.
   c. On the While > General tabbed page, click the XPath Expression Builder icon.



The Expression Builder window opens.

d. Select the BPEL variable `elapsed`.

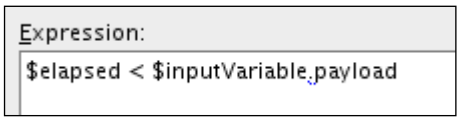
e. Click Insert Into Expression.

f. In the Expression pane, enter "<" (excluding the quotation marks).

g. Select the BPEL variable `inputVariable.payload`.

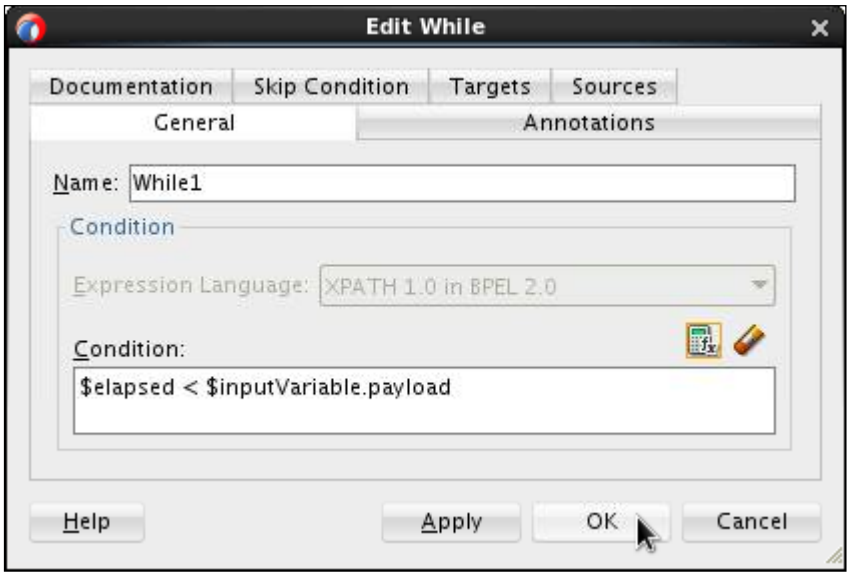
h. Click Insert Into Expression.

i. Verify your work and click OK.



The Expression Builder window closes.

j. Verify the While loop and click OK.

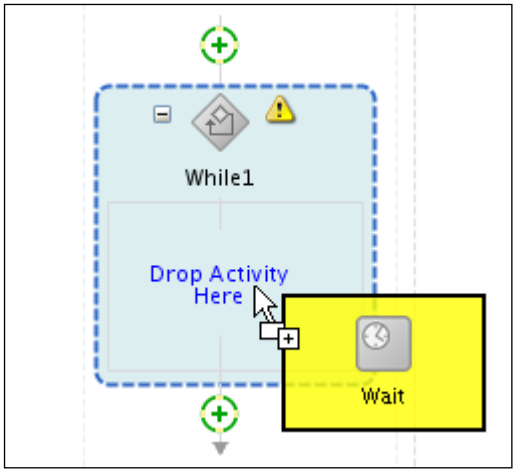


k. Save your work.

Creating a Wait Activity

3. Create and configure a Wait activity.

a. Drag a Wait activity into the While activity.



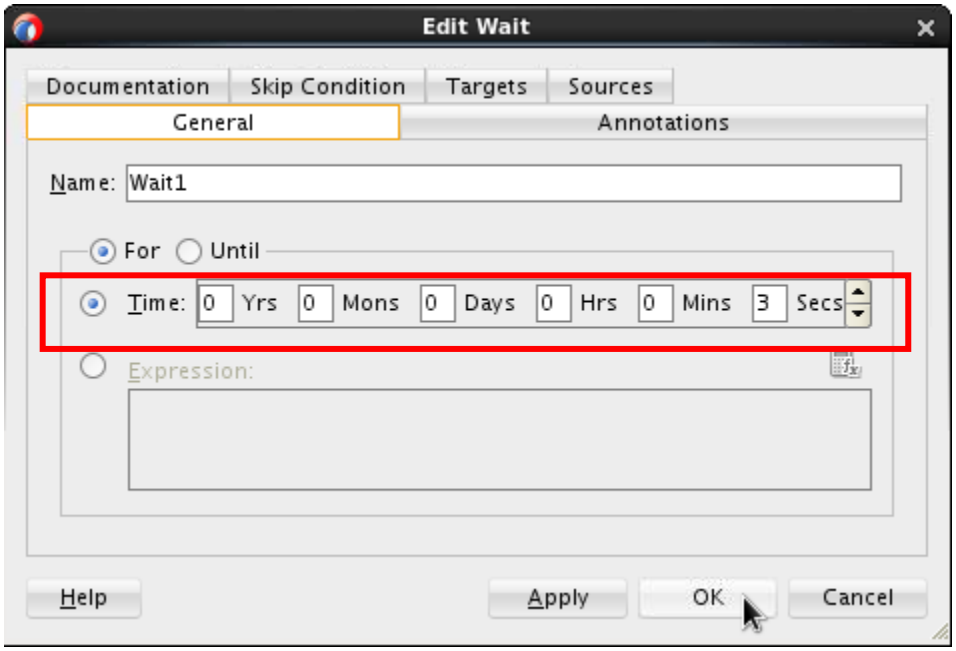
The Wait activity collapses.

b. Right-click the Wait activity and select Edit.

The Wait properties dialog box is displayed.

c. Enter a value of 3 in the Secs field.

d. Verify your work and click OK.

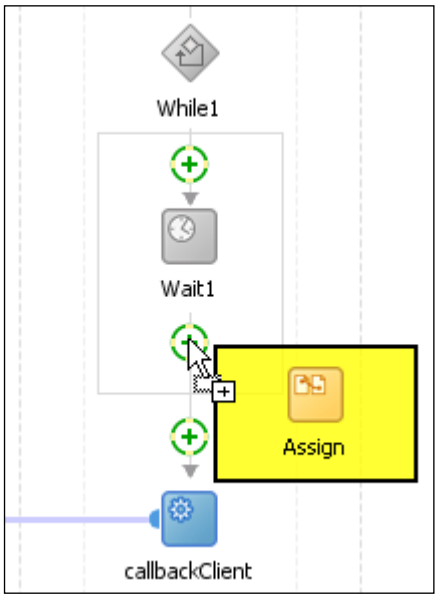


The Wait properties dialog box closes.

Updating Variables with Each Iteration of the Loop

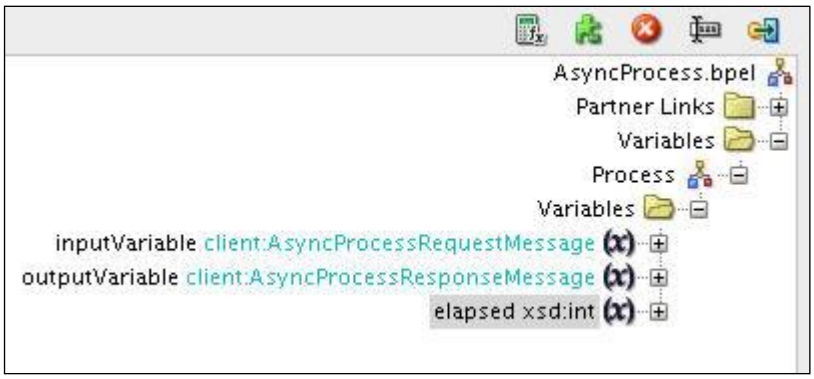
In this step, you update the value of the counter `$elapsed`.

4. Create and configure an Assign activity as the last step in the While sequence.
   a. Add an Assign activity within the While activity (after the Wait activity).



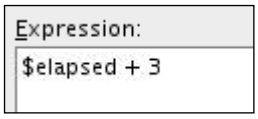
   b. Rename the Assign activity Update_elapsed.
   c. Right-click Update_elapsed and select Edit.

d. Drag the Expression Builder icon onto the `$elapsed` variable.



The Expression Builder is displayed.

e. In the Expression pane, add the text `$elapsed + 3`.



f. Verify your work and click OK.

The Expression Builder is closed.

g. Verify the Assign activity and click OK to close the Assign editor.

5. Save your work.
6. Assign a value to the response message.
   a. Add an Assign activity after the While activity.
   b. Name the activity Assign_Response.
   c. Use the Expression Builder to assign the XPath expression:

```
concat('total of ', string($elapsed), ' seconds elapsed.')
```

   to `$outputVariable.payload`.

   d. Verify and save your work.

Practice 7-3: Creating a BPEL Process to Make an Async Call

In this practice, you build another BPEL process. The second process calls the asynchronous service, and specifies how long the process should loop. The calling process then uses a Pick activity with both an onMessage and an onAlarm branch. The onAlarm branch specifies how long to wait for a response. If the called process returns a message before onAlarm is activated, the onMessage branch executes. If not, the onAlarm branch is triggered instead.

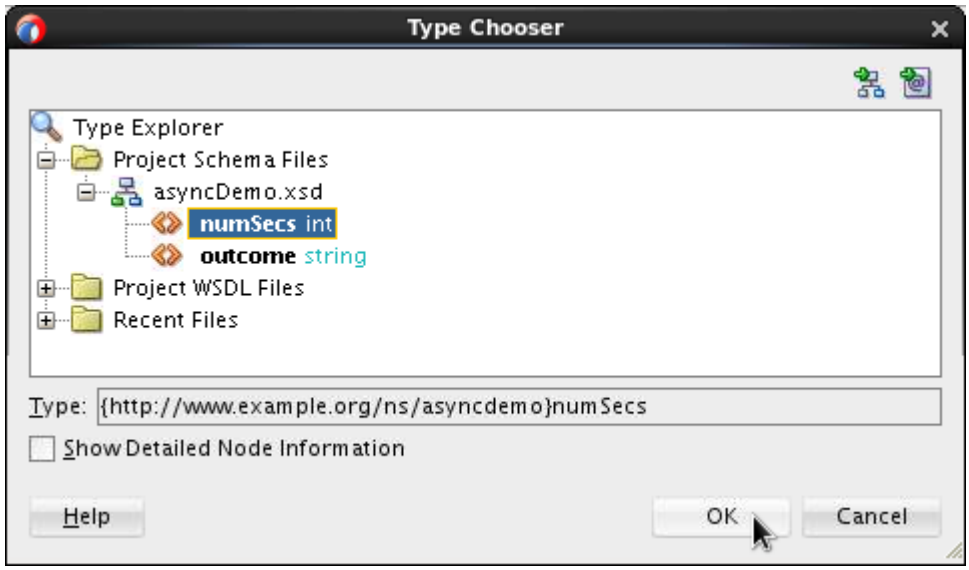
Assumptions

This practice assumes that JDeveloper is open, and that you have completed Practice 7-2 successfully.

Tasks

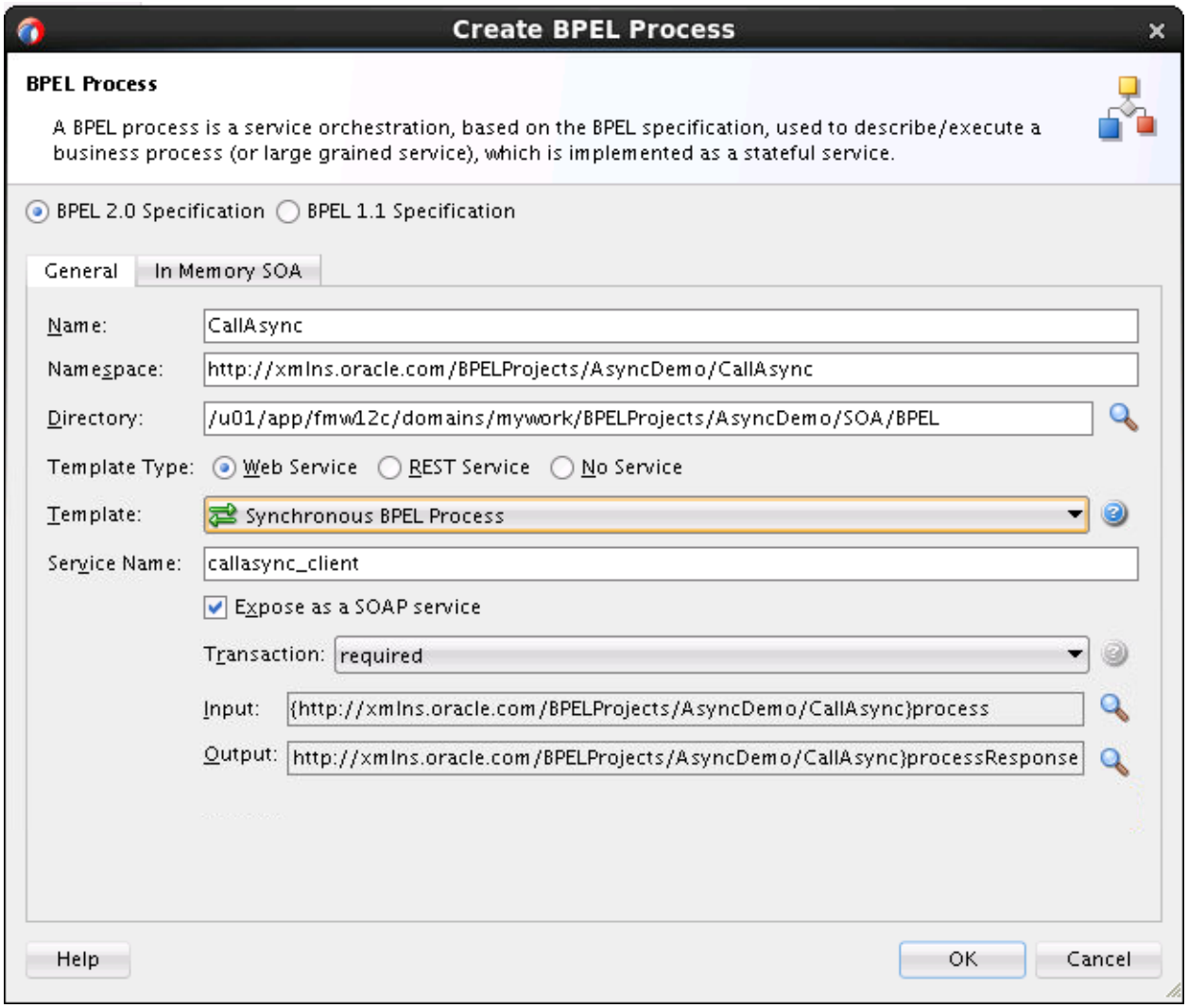
Adding a BPEL Process to the Application

1. Create and configure a BPEL service component.
   a. In JDeveloper, click the composite overview tab.
   b. Add a BPEL process to the Components swimlane.
      The Create BPEL Process dialog box is displayed.
   c. Name the process `CallAsync`.
   d. Select the Synchronous BPEL template.
   e. To define the input, click the Browse Input Elements button.
   f. In the Type Chooser, navigate to the `numSecs` element of `asyncDemo.xsd`.

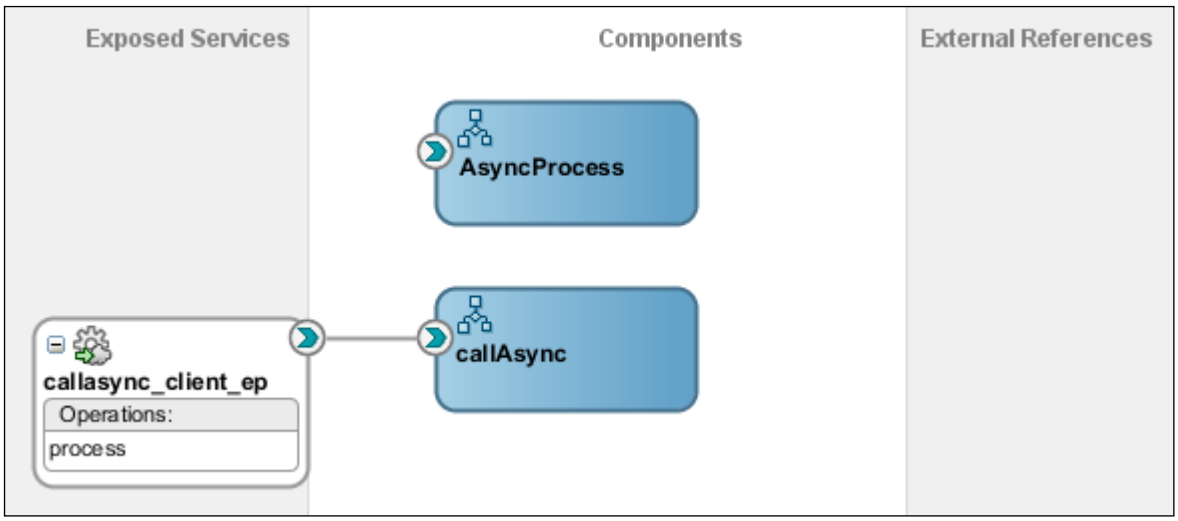


   g. Click the Browse Output Elements icon.
   h. In the Type Chooser, navigate to the `outcome` element of `asyncDemo.xsd`.

i. Verify your work and click OK.



2. Verify your composite application and save your work.

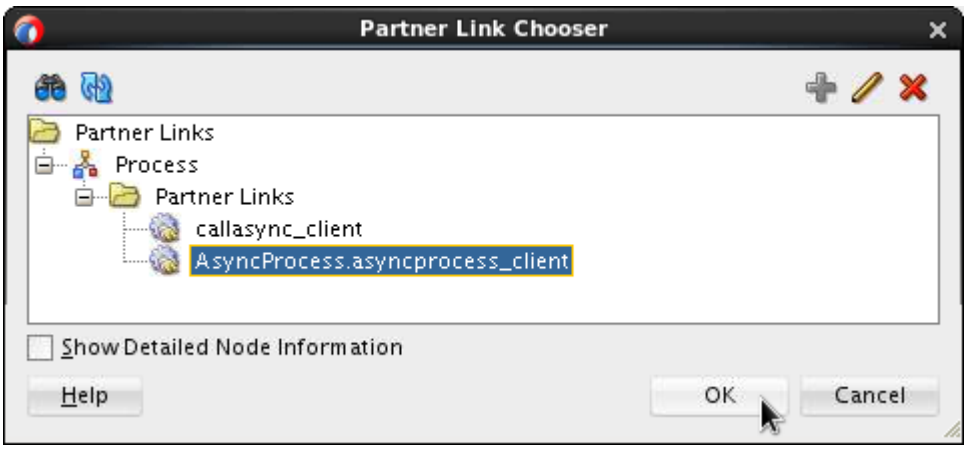


3. Wire callAsync to AsyncProcess.

Invoking the Asynchronous Service

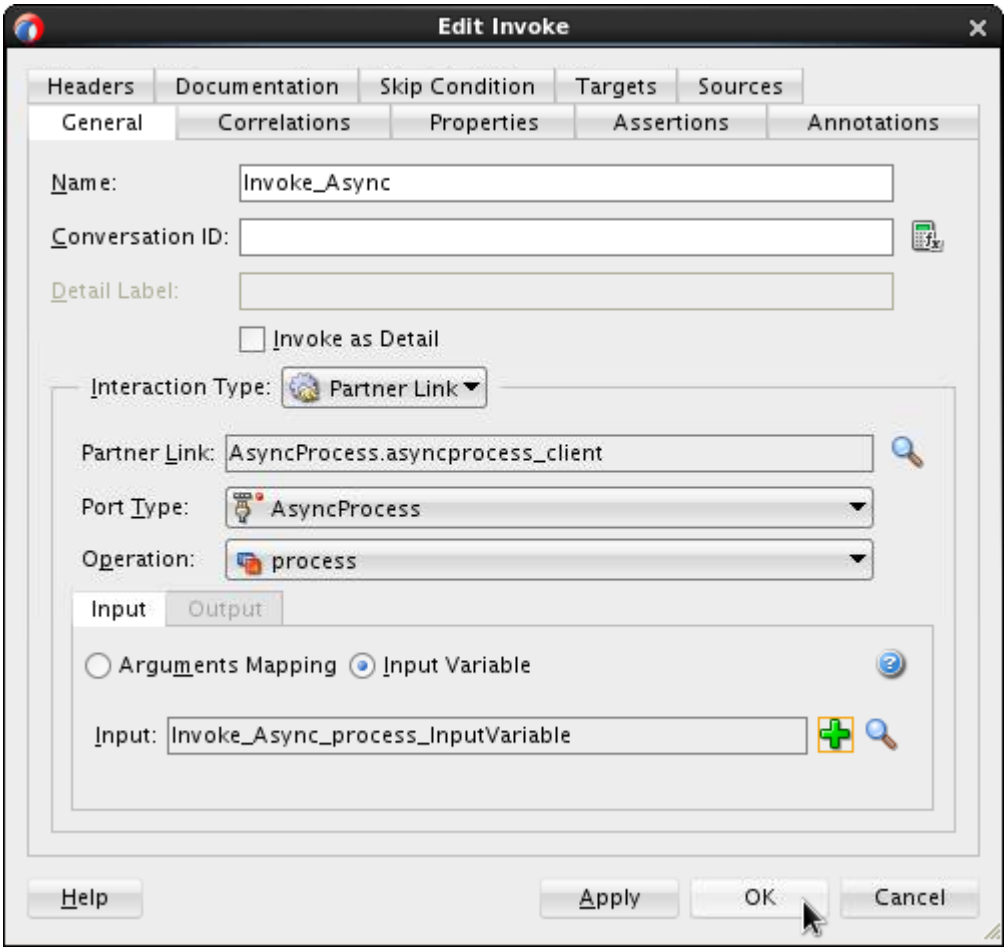
In this step, you create an Invoke activity that calls the web service.

4. Create and configure an Invoke activity.
   a. Right-click the BPEL component callAsync and select Edit.
   b. Add an Invoke activity to the process below the Receive activity.
   c. Right-click the Invoke activity and select Edit.
   d. Name the activity `Invoke_Async`.
   e. Click the Browse Partner Links button and select the partner link `AsyncProcess.asyncprocess_client`.



   f. Click OK.
   g. Create an Input variable. (Accept the default name.)

h. Verify your work and click OK.

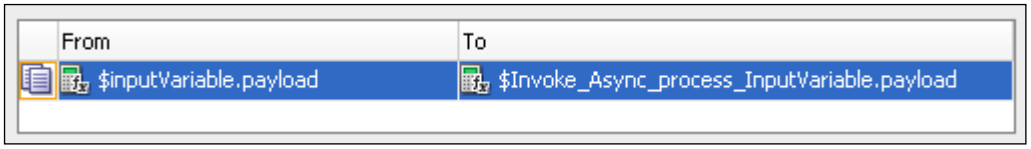


i. Save your work.

Populating the Input Variable for the Service Invocation

In this step, you create an Assign activity to populate the Input variable for the web service invocation.

5. Create and configure the `Assign_Async` activity.
   a. Add an Assign activity above `Invoke_Async`.
   b. Name the activity `Assign_Async`.
   c. Right-click `Assign_Async` and select Edit.
      The Edit Assign window opens.
   d. Map variable `inputVariable.payload` to `$Invoke_Async_process_InputVariable.payload`.



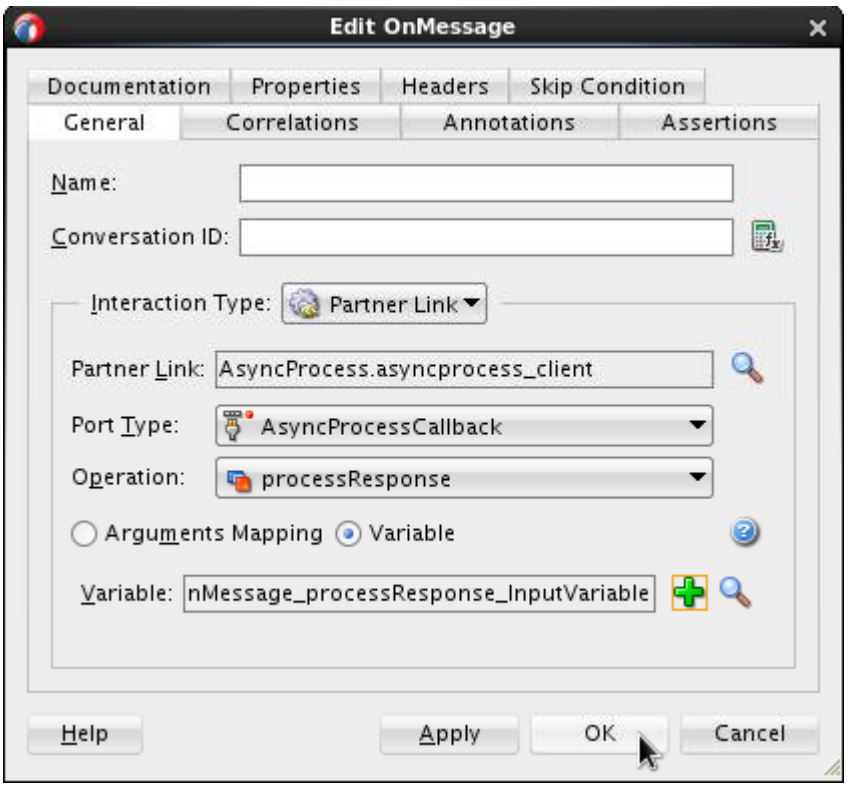
   e. Click OK.

f. Save your work.

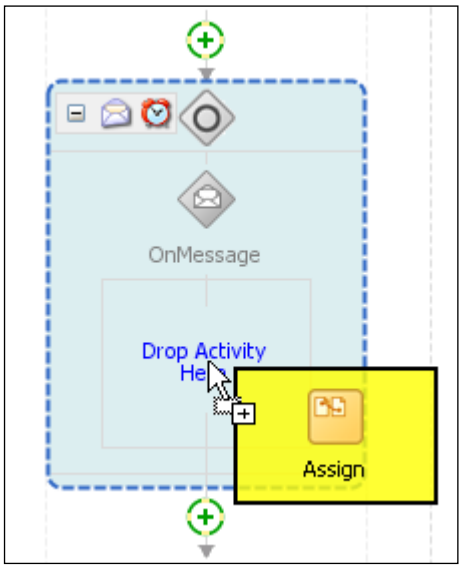
Processing the Service Response

In this section, you create and configure a Pick activity. This activity waits for a specified amount of time to receive the callback from the async service.

6. Create a Pick Activity.
   a. Add a Pick activity to the BPEL process after the Invoke_Async activity.
   b. Right-click the new Pick activity and select Edit.
      The Pick settings window opens.
   c. Rename the activity `Pick_Operation`.
   d. Click OK.
7. Configure the `onMessage` branch.
   a. Right-click the `onMessage` branch icon and select Edit.
   b. Browse to the partner link AsyncProcess.asyncprocess_client.
   c. Auto-create a variable.
   d. Verify your work and click OK.



8. Create and configure `Assign_Message`.
   a. Expand the `onMessage` branch of the `Pick_Operation` activity.
   b. Add an Assign activity named `Assign_Message` to the branch.

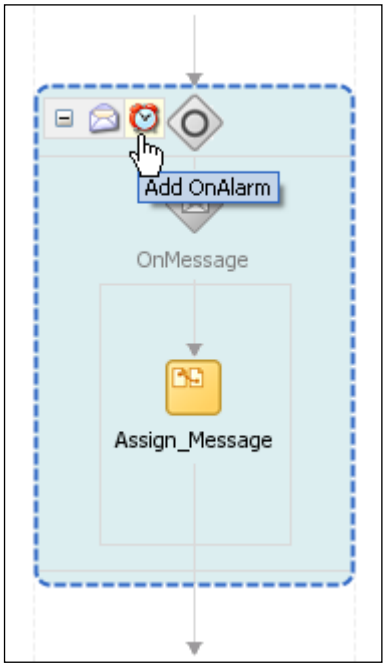


   c. Right-click `Assign_Message` and select Edit.
   d. Map `$OnMessage_processResponse_InputVariable.payload` to `outputVariable.payload`.

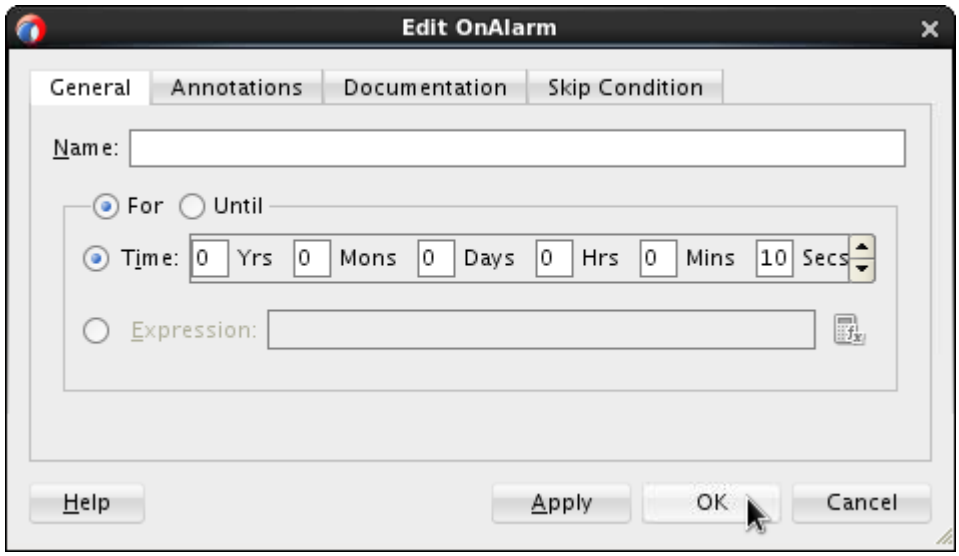
      The Edit Assign window displays the following:

   e. Verify your work and click OK.

9. Create and configure an `onAlarm` branch.
   a. Click the Pick activity.
   b. Click the Add OnAlarm button.



   An on Alarm branch is added to the Pick activity.
   c. Right-click the onAlarm branch and select Edit.
   The onAlarm properties dialog box is displayed.
   d. Configure the timeout to 10 seconds.

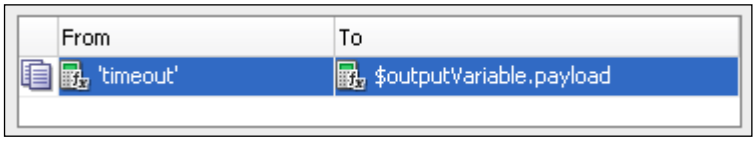


   e. Verify your work and click OK.
10. Assign the text “timeout” to the process output variable.
   Refer to step 7, if needed.
   a. Add an Assign activity to the onAlarm branch of the Pick activity.

b. Name the activity `Assign_Timeout`.

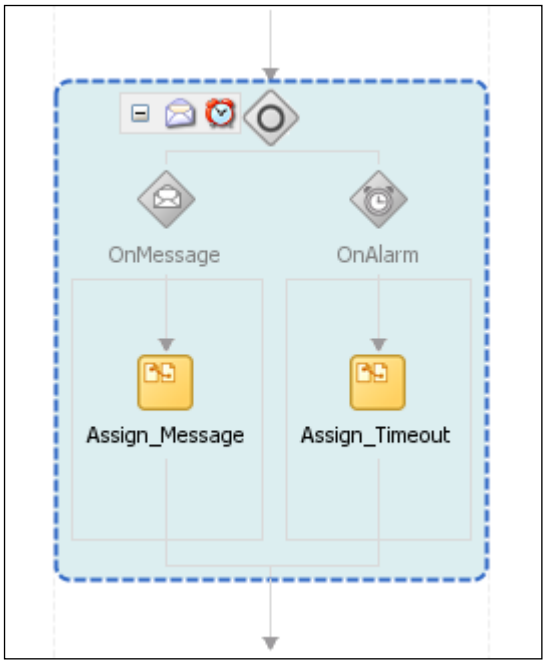
c. Configure `Assign_Timeout` with a copy operation that stores the string literal `'timeout'` in the variable `$outputVariable.payload`.

d. Verify the copy operation and click OK.



The Assign Editor dialog box is closed.

11. Verify the Pick activity and save your work.



Practice 7-4 Deploying and Testing Your Work

Overview

In this practice, you deploy and test the completed `AsyncDemo` composite application.

Assumptions

This practice assumes the following:

- You have completed all previous practices successfully.
- JDeveloper is open.
- The WebLogic Application Server is running.

Tasks

Deploying the Application

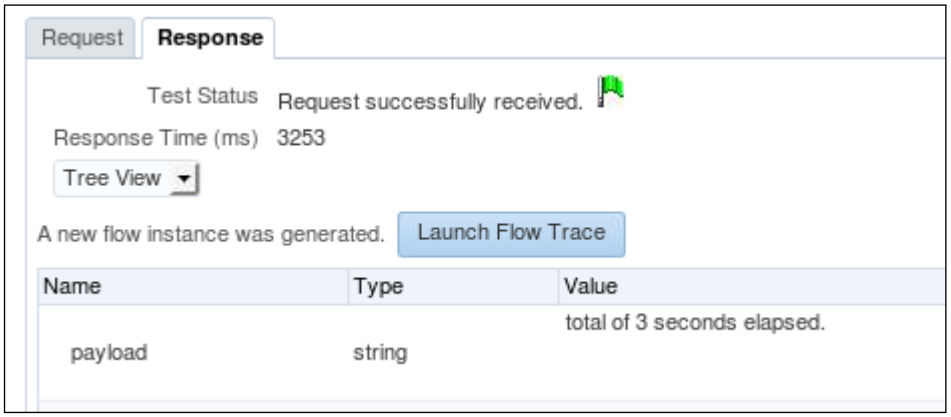
1. In the JDeveloper Application Navigator window, deploy the `AsyncDemo` composite application to IntegratedWebLogicServer.
   - Remember to select the “Overwrite any existing composites with the same revision ID” check box.
   - Monitor the “Deployment – Log” window to ensure that deployment is successful.

Testing the Application

2. Set up the test.
   a. Log in to Oracle Enterprise Manager (http://localhost:7101/em).
   b. In the Target Navigation pane, expand the SOA > soa-infra node, then right-click and select Home > Deployed Composites. When it displays, click the “AsyncDemo [1.0]” link.
   c. On the “AsyncDemo [1.0]” dashboard page, click Test.
   d. On the Test Web Service page, scroll down to the Input Arguments section of the Request tab.
   e. On the Request tab, enter a value of `2` for the payload.

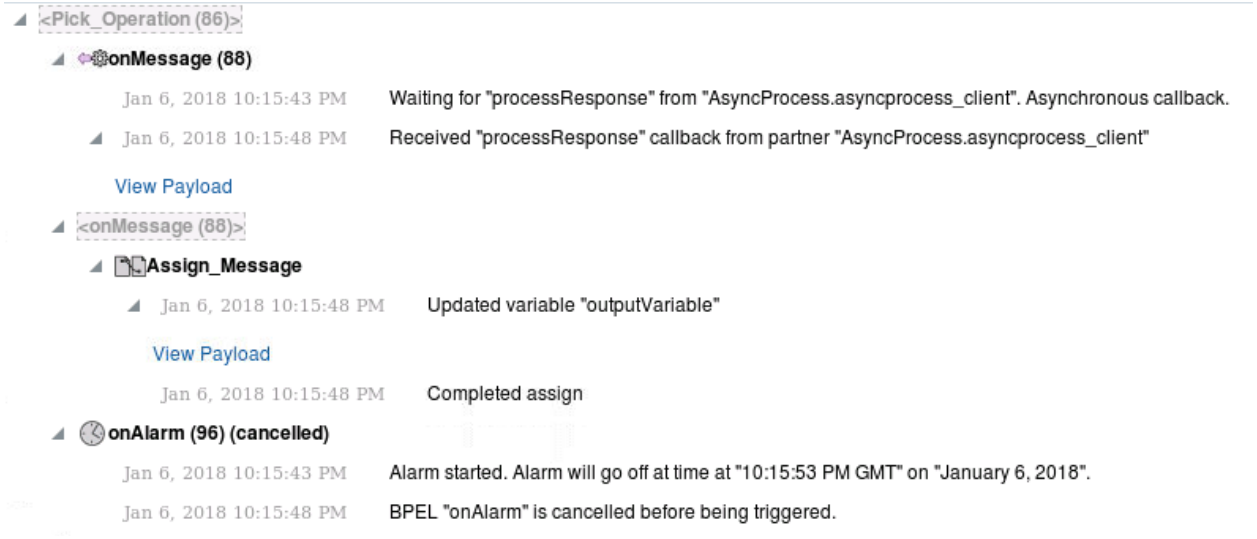
3. Execute the test.
   a. Click Test Web Service.

      After a brief delay, the message “total of 3 seconds elapsed” is displayed on the Response tab.



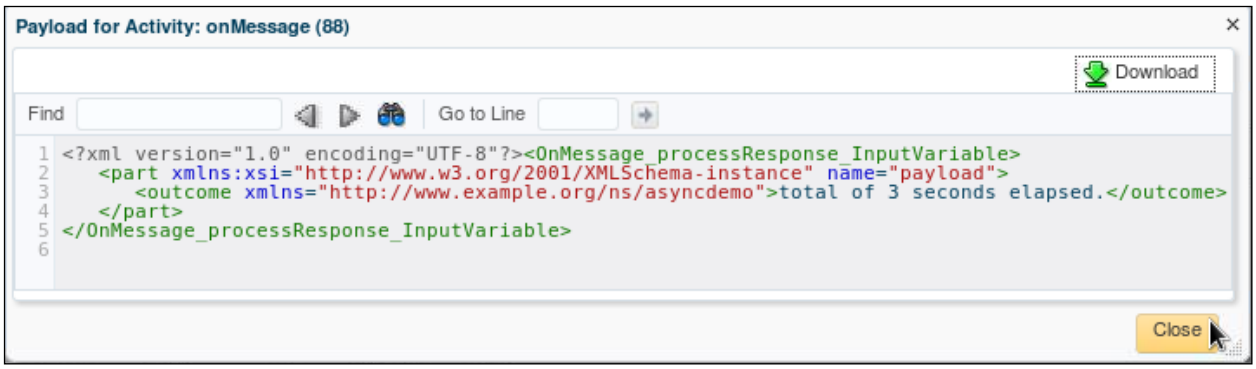
   b. On the Response tab, click the Launch Flow Trace button.
   c. Select the CallAsync process.
   d. Expand the Pick_Operation and each of its subnodes.

      You see the onMessage node receive the callback and the Assign_message activity update the output variable accordingly. The onAlarm node is cancelled because a message was received before the alarm was triggered.



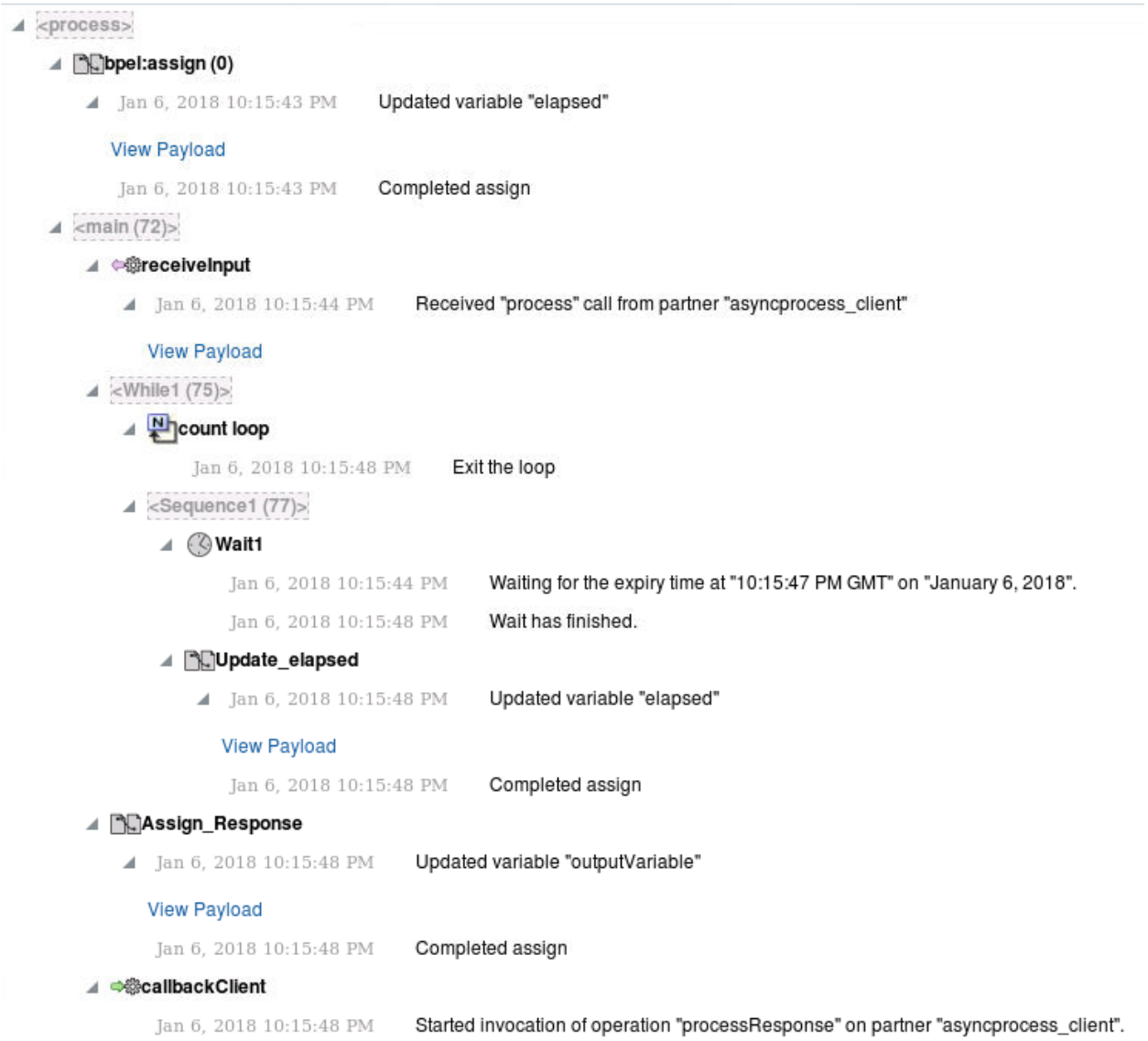
e. Click View Payload in onMessage.

The callback message that you defined for AsyncProcess is displayed.



f. Navigate to the Flow Trace for AsyncProcess.

g. Expand the While and Wait nodes.

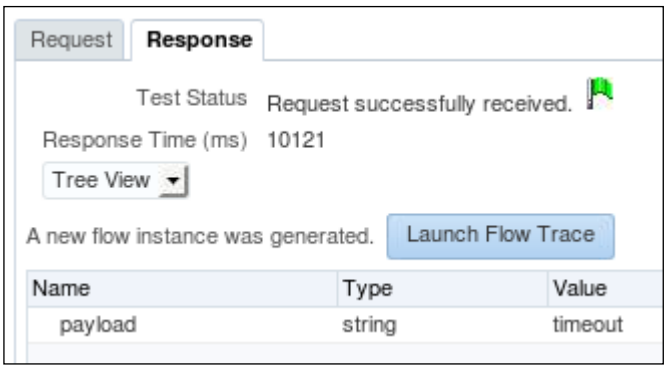


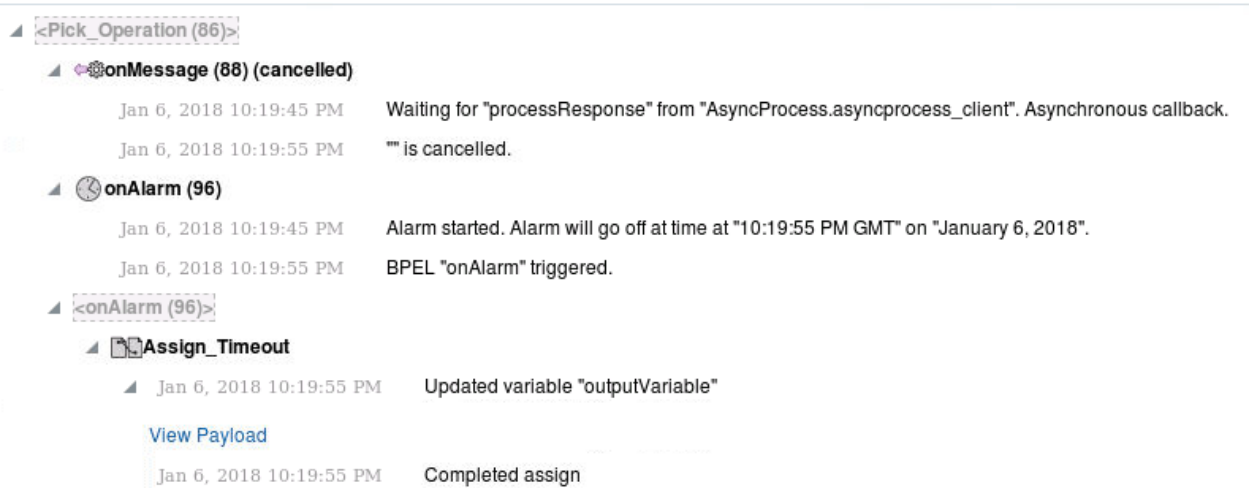
h. Close the Flow Trace window.

4. Refer to step 2 to run the test again. This time, supply the value `12` as the input.
5. Refer to step 3 to explore the test results.

   The results for the CallAsync process are different than in the previous test.

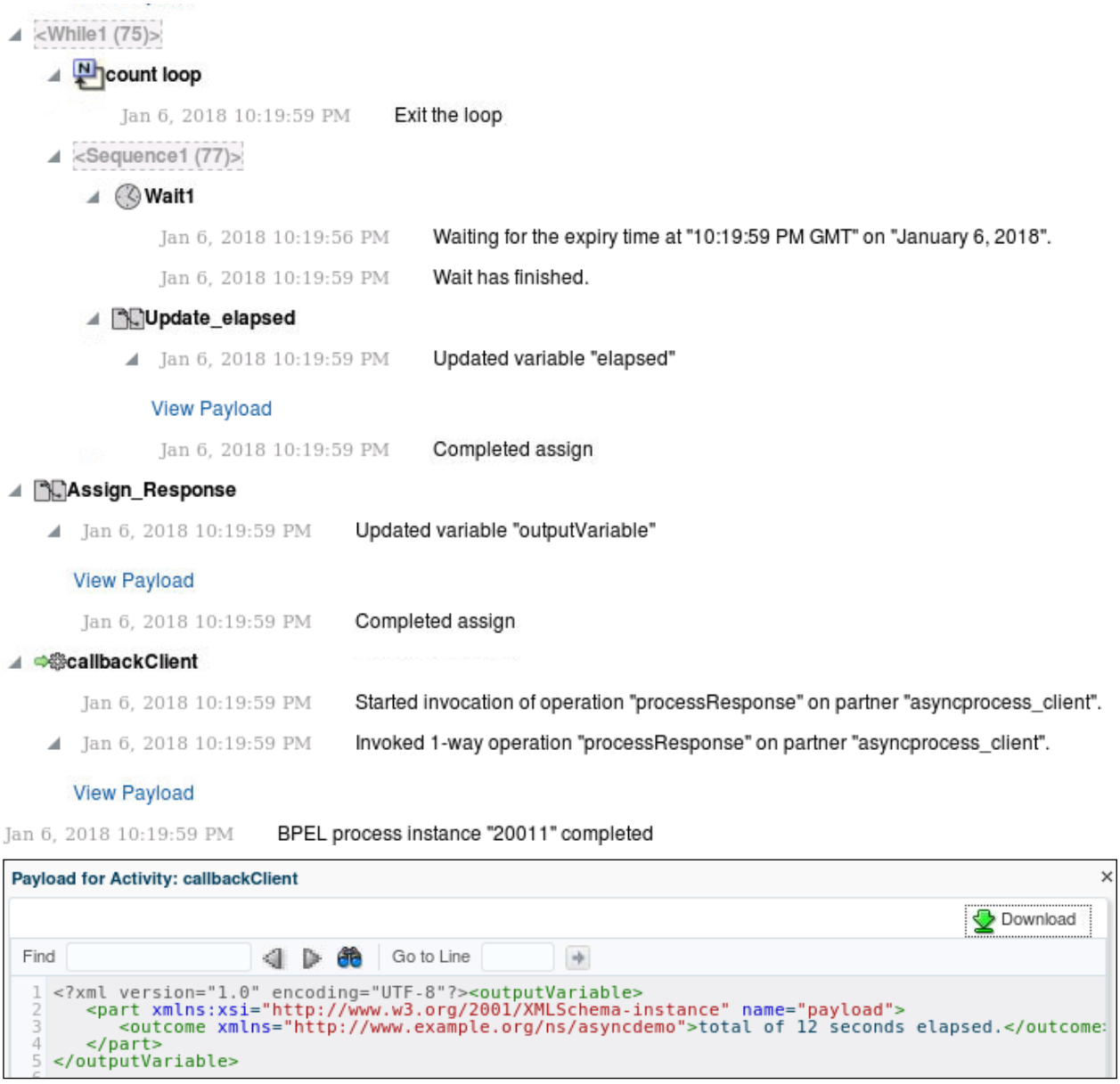
   - Is this expected? Why or why not?





   Except for the value in the outcome, the results for AsyncProcess seem to be the same as the previous test.

   - Is this expected? Why or why not?
   - What happened to the callback message?



6. Close the Flow Trace and minimize your browser window.

Practices for Lesson 8: Handling Faults in Composite Applications

Practices for Lesson 8: Overview

Practices Overview

In this practice, you first add a supplied project (BookingSystem) to your BPELProjects application, and then deploy the project. You then create and configure a second project (Enroll) that interacts with BookingSystem, and include a fault-handling configuration that is invoked if the interaction with BookingSystem results in a fault condition.

The `BookingSystem` project provides a simple mechanism to simulate a training enrollment booking system. It receives enrollment-related messages from `Enroll` project. In most cases, it responds with positive confirmations, but provided with certain data values, it returns a fault. You deliberately trigger this behavior to test the fault-handling configuration that you have built in the `Enroll` project.

Practice 8-1: Opening and Deploying the `BookingSystem` Project

Overview

In this practice, you add the supplied `BookingSystem` project to your BPELProjects application, and then deploy it.

Assumptions

This practice makes no assumptions.

Tasks

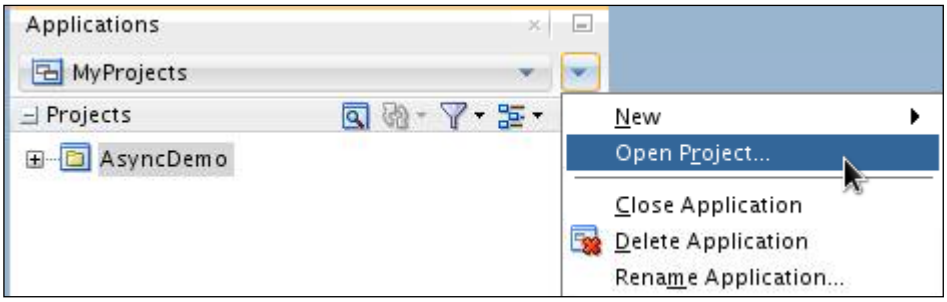
1. Stop the Integrated WebLogic Server and close JDeveloper.

   Note: This is not strictly necessary, but “bouncing” the server and JDeveloper periodically is a good idea. This is a good time to do it.

2. Open a terminal window and issue the following commands to copy three additional projects into application BPELProjects:

   ```
   cd /u01/app/fmw12c/domains/mywork/BPELProjects
   cp -R /home/oracle/labs/apps/projects/BookingSystem .
   cp -R /home/oracle/labs/apps/projects/CallGreeting .
   cp -R /home/oracle/labs/apps/projects/SayHello .
   ```

3. Reopen JDeveloper and restart the Integrated WebLogic Server.
4. Verify that the BPELProjects application is selected.
5. From the Application Navigator menu, select Open Project.



6. Navigate to `/u01/app/fmw12c/domains/mywork/BPELProjects/BookingSystem/BookingSystem.jpr`. If prompted to upgrade the project, click select Yes and OK.
7. Deploy the project to IntegratedWebLogicServer.

   Note: Remember to select the “Overwrite any existing composites with the same revision ID” check box.

Practice 8-2: Creating and Configuring the Enroll Project

Overview

In this practice, you configure the BPEL process that calls the booking system to process the enrollment and payment information. The configuration includes the creation and configuration of fault-handling routines.

Assumptions

This practice assumes the following:

- JDeveloper is open.
- Practice 8-1 has been completed successfully.

Tasks

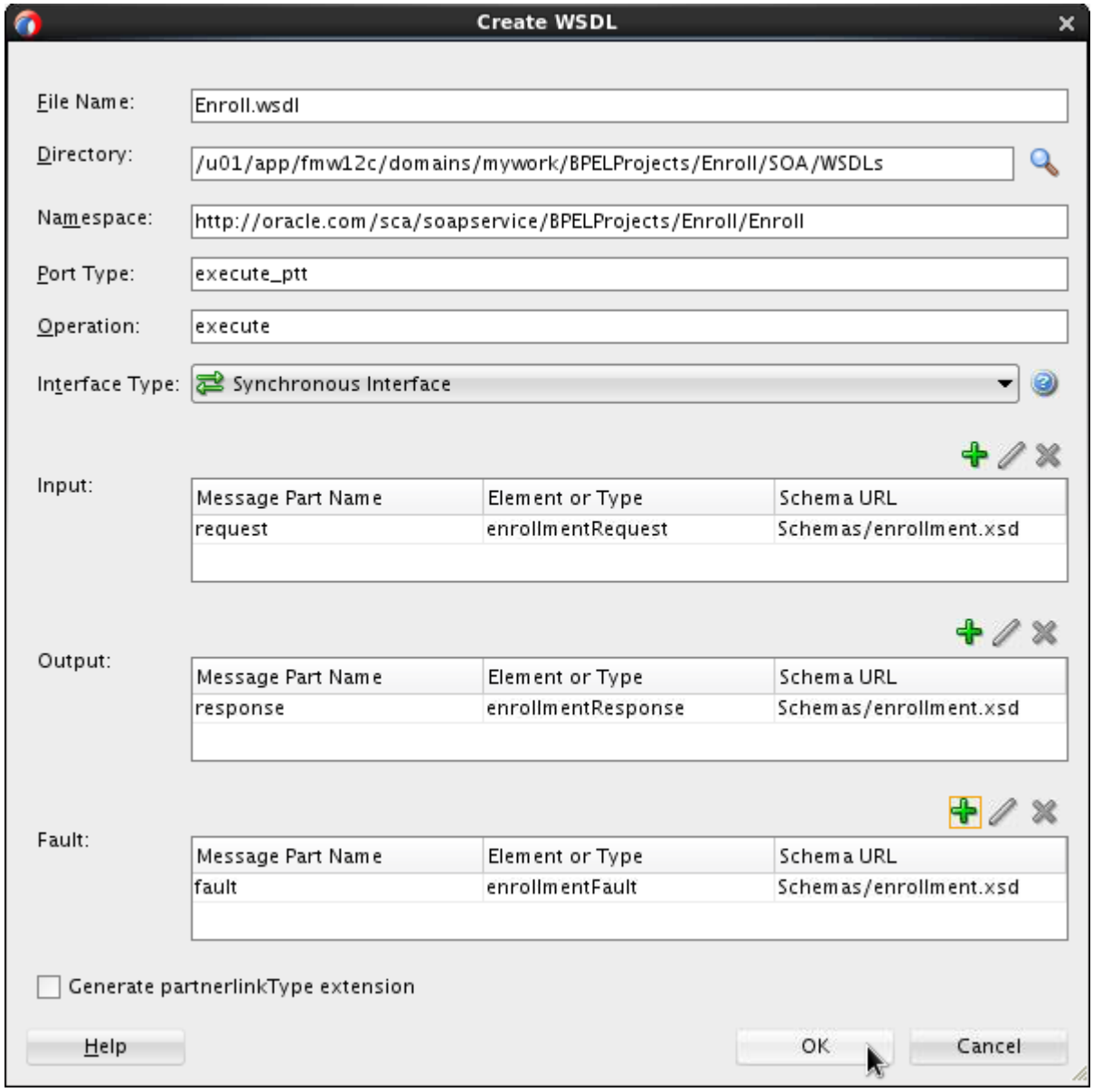
Creating a Composite Application

In this section, you create a new composite application. You define an exposed service and an external reference, and add a BPEL service component.

1. In the BPELProjects application, create a new empty SOA project named `Enroll`.

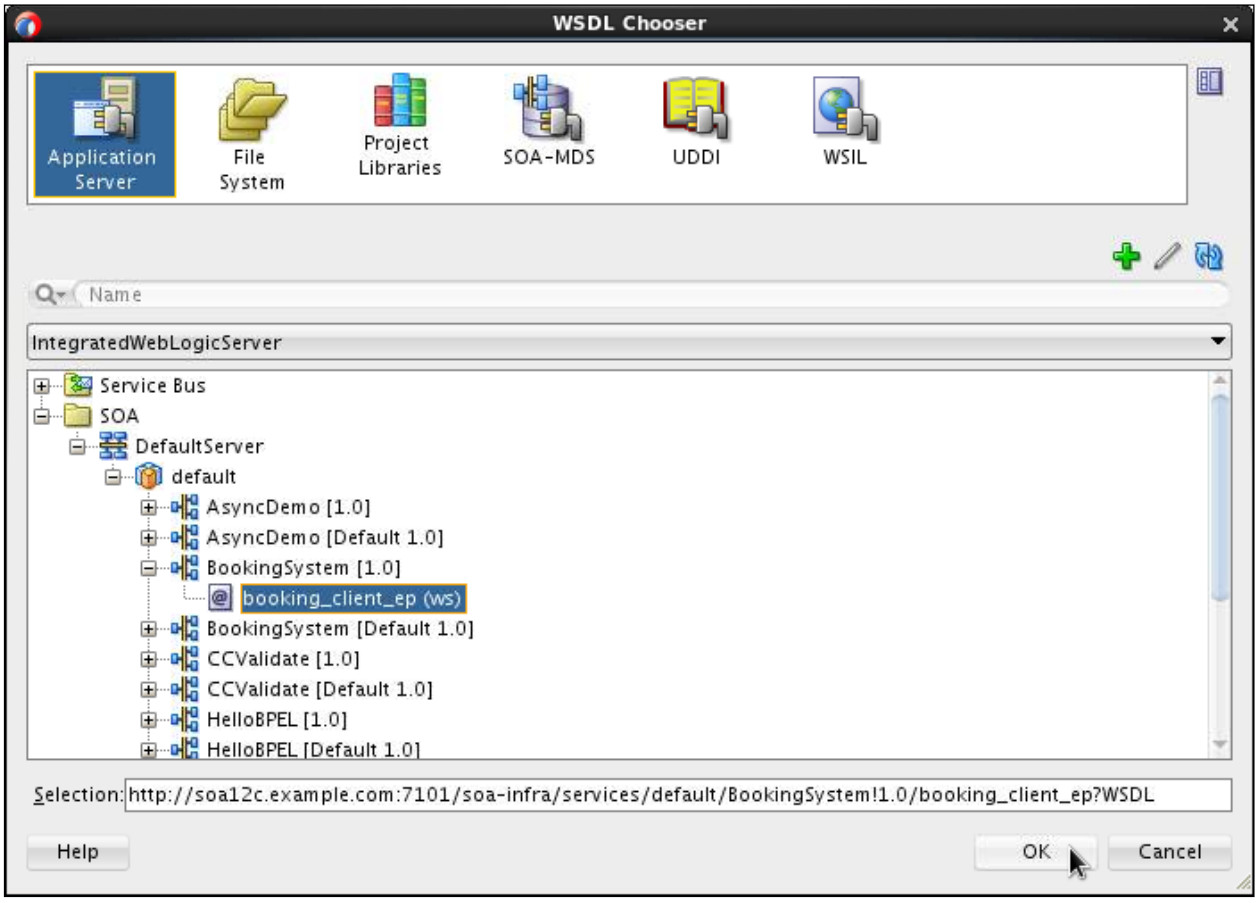
   The `composite.xml` file is created and opened for editing.
2. Create and configure an exposed service.
   a. Add a SOAP web service to the Exposed Service swimlane.
   b. Provide the name `Enroll`.
   c. Generate the WSDL from a schema.
   d. Select the Synchronous Interface Type.
   e. Define the Input:
      1) Add a new message part.
      2) Name the message part `request`.
      3) To specify the URL, import the file /home/oracle/labs/files/xsd/enrollment.xsd.
      4) Copy the file to the current project.
      5) Select the `enrollmentRequest` element.
   f. Define the Output:
      1) Add a new message part.
      2) Name the message part `response`.
      3) To specify the URL, browse to `enrollment.xsd`. Select the element `enrollmentResponse`.
   g. Repeat the previous step to define the fault message part by using the element `enrollmentFault` from the `enrollment.xsd` file.

h. Verify and save your work.

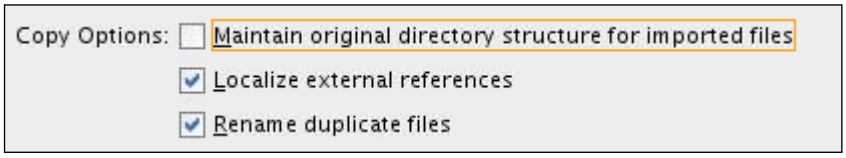


3. Create and configure an external reference.
   a. Add a SOAP web service to the External Reference swimlane.
   b. Name the reference `BookingService`.
   c. To specify the WSDL URL, click the Find Existing WSDLs icon.

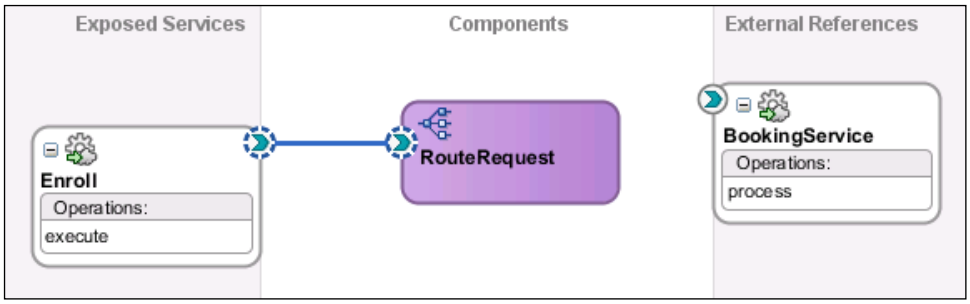
d. In the WSDL Chooser, select Application Server. Navigate to the deployed application `BookingSystem`. Select `booking_client_ep` and click OK.



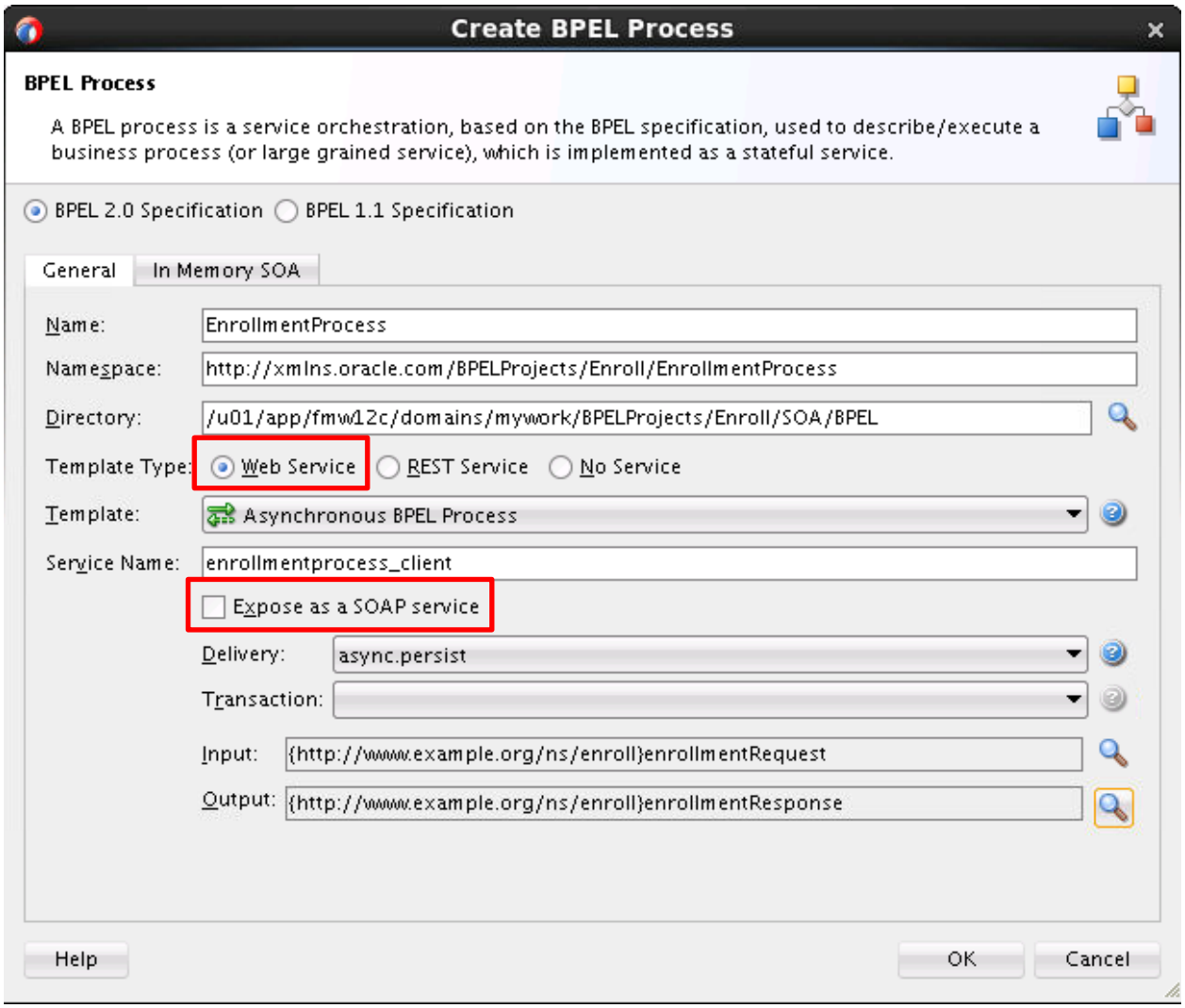
e. In the Localize Files dialog box, click OK.



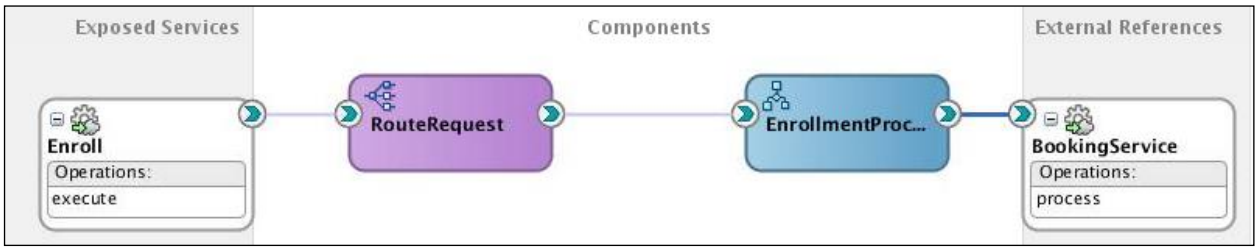
4. Create a Mediator component.
   a. Add a Mediator to the Components swimlane.
   b. Name the mediator `RouteRequest`.
   c. Click OK to accept the Define Interface Later option.
   d. Add a wire from the Exposed Service Enroll to the Mediator RouteRequest.



5. Create and configure a BPEL service component.
   a. Add a BPEL process to the Components swimlane.
   b. Name the process `EnrollmentProcess`.
   c. Select the Synchronous BPEL template.
   d. Deselect the “Expose as a SOAP service” check box. Ensure Web Servcie radio button is selected.
   e. To define the input, use the Browse button to navigate to the `enrollmentRequest` element of `enrollment.xsd`.
   f. To define the output, browse to the `enrollmentResponse` element of `enrollment.xsd`.
   g. Verify and save your work.



6. Configure additional wires in `composite.xml`.
   a. Create a wire from the Mediator RouteRequest to the BPEL process `EnrollmentProcess`.
   b. Create a wire from the BPEL process `EnrollmentProcess` to the external reference `BookingService`.
   c. Verify and save your work.



Practice 8-3: Configuring the EnrollmentProcess Process

Overview

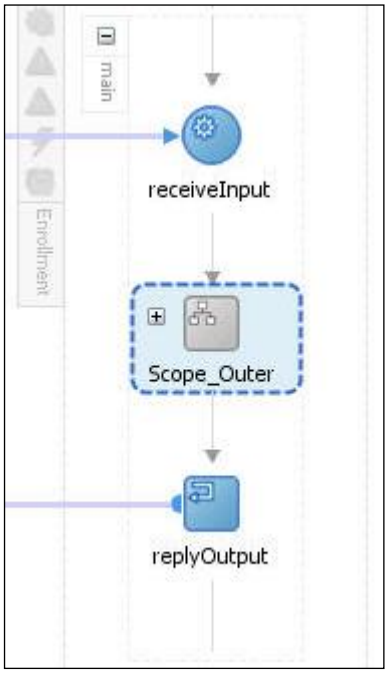
In this practice, you create and configure the activities that comprise the BPEL Process.

Assumptions

This practice assumes that you have completed Practice 8-2 successfully.

Tasks

1. Right-click the BPEL process EnrollmentProcess and select Edit.
   The model editor opens.
2. Create and configure a Scope activity.
   a. Add a Scope activity between the receiveInput and replyOutput activities.
   b. Name the Scope `Scope_Outer`.

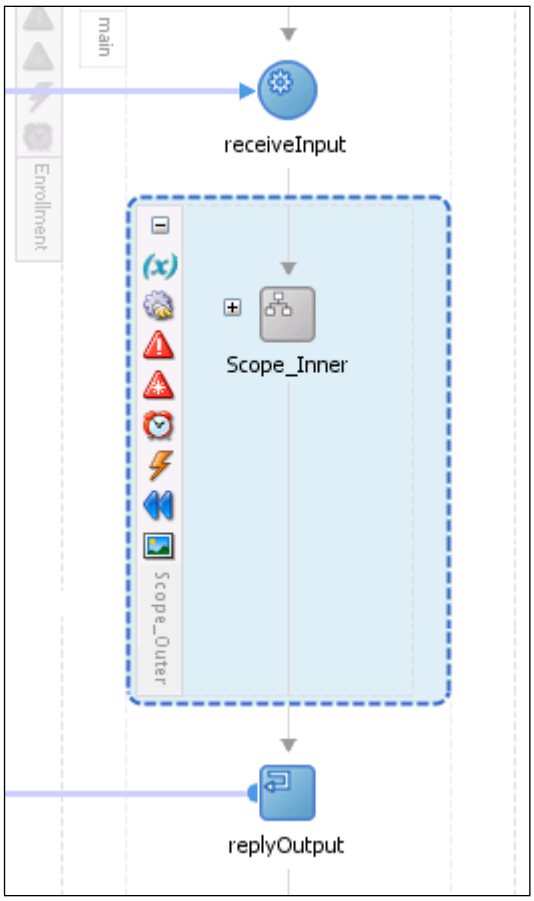


Defining Enrollment Invocation

In this section, you define the activities that are directly related to the invocation of the enrollment request.

3. Create and configure an inner Scope.
   a. Expand `Scope_Outer`.
   b. Add a Scope within `Scope_Outer`.

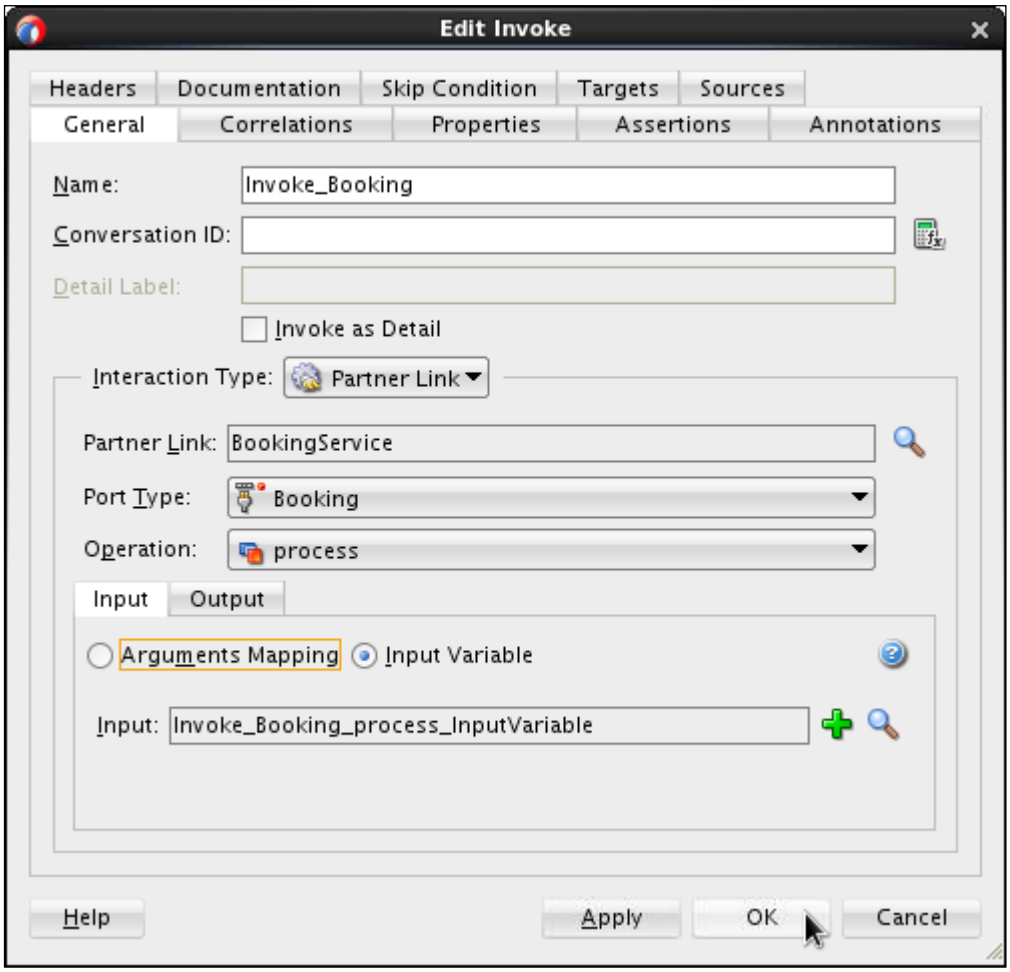
c. Name the new Scope `Scope_Inner`.



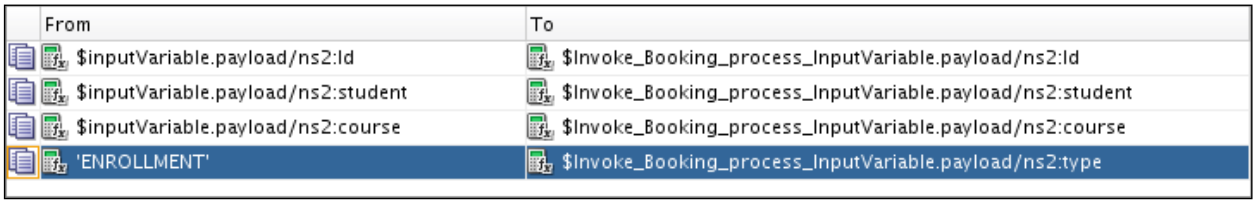
4. Create and configure an Invoke activity.
   a. Expand `Scope_Inner`.
   b. Add an Invoke activity to `Scope_Inner`.
   c. Right-click the Invoke activity and select Edit.
   d. Name the Invoke `Invoke_Booking`.
   e. To specify the partner link, use the Browse icon to navigate to BookingService.
   f. Create a new Input variable.
   g. Create a new Output variable.

   Note: Within a scope, you have the option of creating either global or local variables. You opt to use global variables in this practice.

h. Verify and save your work.



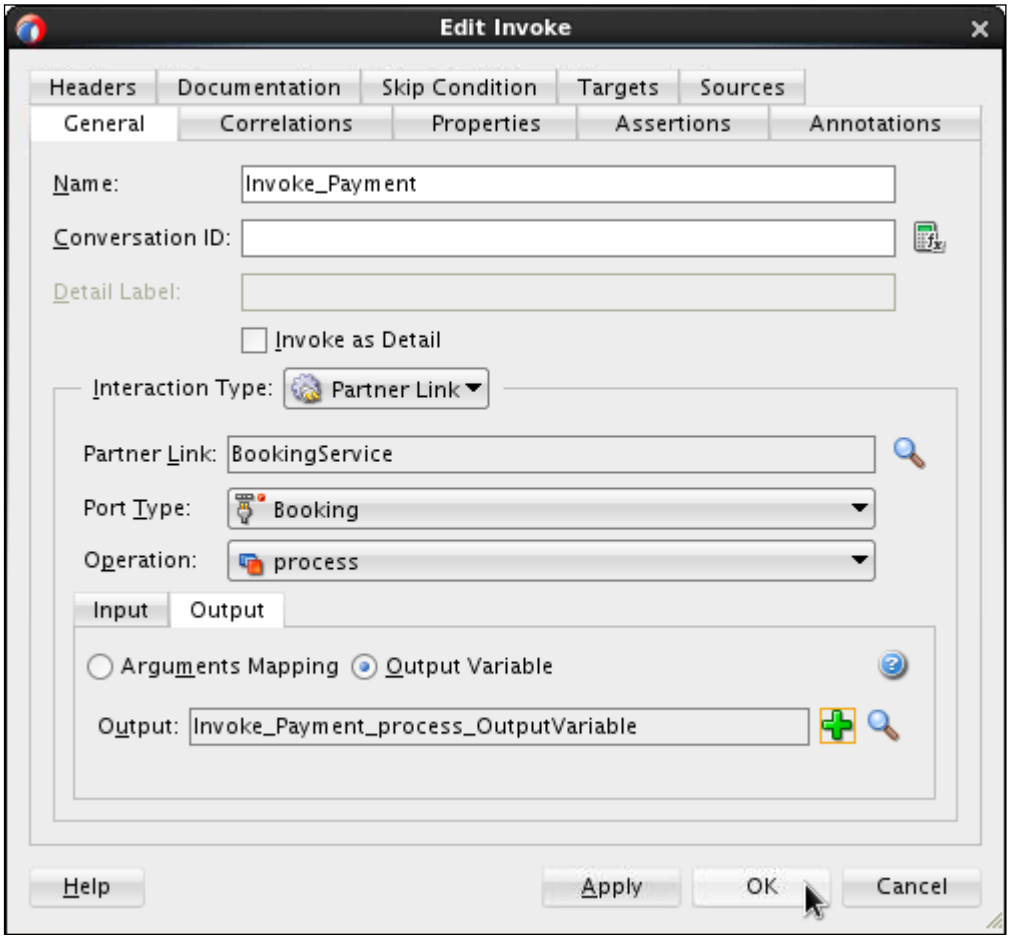
5. Create and configure an Assign activity.
   a. Add an Assign activity just above `Invoke_Booking` (inside `Scope_Inner`).
   b. Name the activity `setBooking`.
   c. Right-click the new activity and select Edit.
   d. Map the `inputVariable` elements `Id`, `student`, and `course` to their corresponding elements in the `Invoke_Booking_process_InputVariable`.
   e. Use Expression Builder to map the text `'ENROLLMENT'` to the `type` node of `Invoke_Booking_process_InputVariable`.
   f. Verify and save your work.



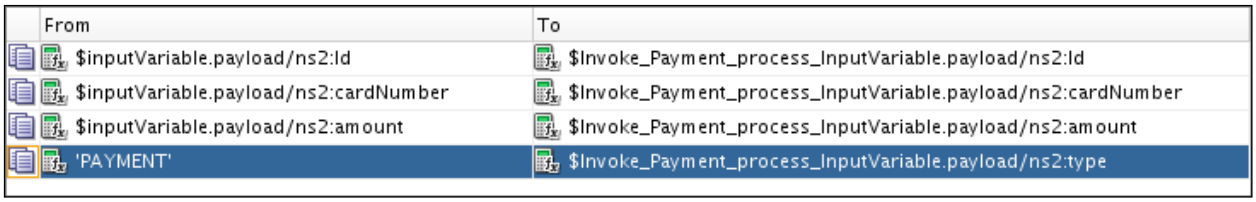
Defining Payment Invocation

In this section, you create and configure the activities that are directly related to the invocation of the payment processing.

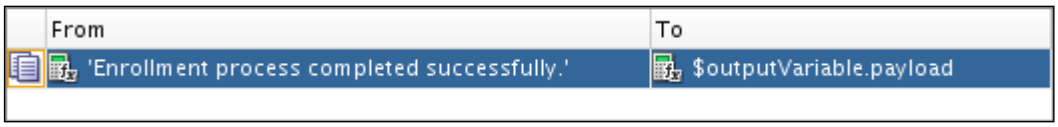
6. Create and configure an Invoke activity.
   a. Collapse `Scope_Inner`.
   b. Add an Invoke activity to `Scope_Outer` immediately following `Scope_Inner`.
   c. Right-click the new Invoke activity and select Edit.
   d. Name the activity `Invoke_Payment`.
   e. To specify the partner link, use the Browse icon to navigate to BookingService.
   f. Create a new Input variable.
   g. Create a new Output variable.
   h. Verify and save your work.



7. Create and configure an Assign activity.
   a. Add an Assign activity just above `Invoke_Payment`.
   b. Name the activity `setPayment`.
   c. Right-click the new activity and select Edit.
   d. Map the `inputVariable` elements `Id`, `cardNumber`, and `amount` to their corresponding elements in `Invoke_Payment_process_InputVariable`.
   e. Use Expression Builder to map the text `'PAYMENT'` to the `Invoke_Payment_process_InputVariable type` field.
   f. Verify, and then save your work.



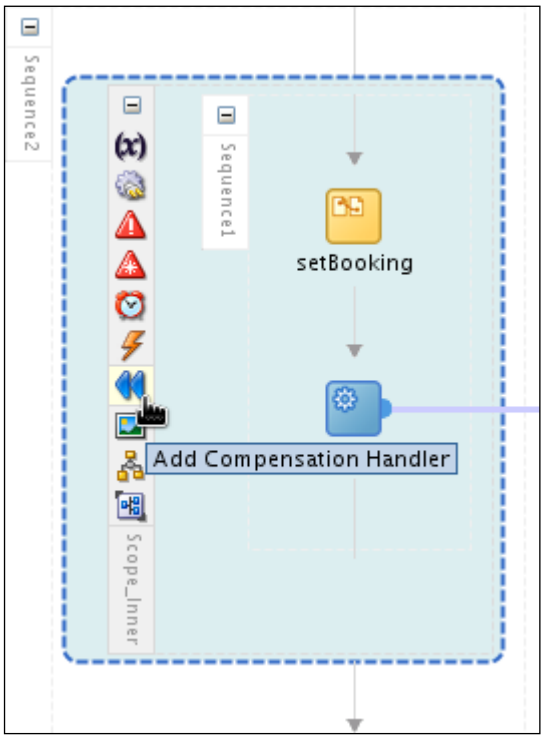
8. Create and configure an Assign activity.
   a. Add an Assign activity just below `Invoke_Payment`.
   b. Name the activity `setSuccess`.
   c. Right-click the new activity and select Edit.
   d. Use Expression Builder to map the text `'Enrollment process completed successfully'` to the `enrollmentResponse` node of `outputVariable`.
   e. Verify, and then save your work.



Adding Compensation Handling to `Scope_Inner`

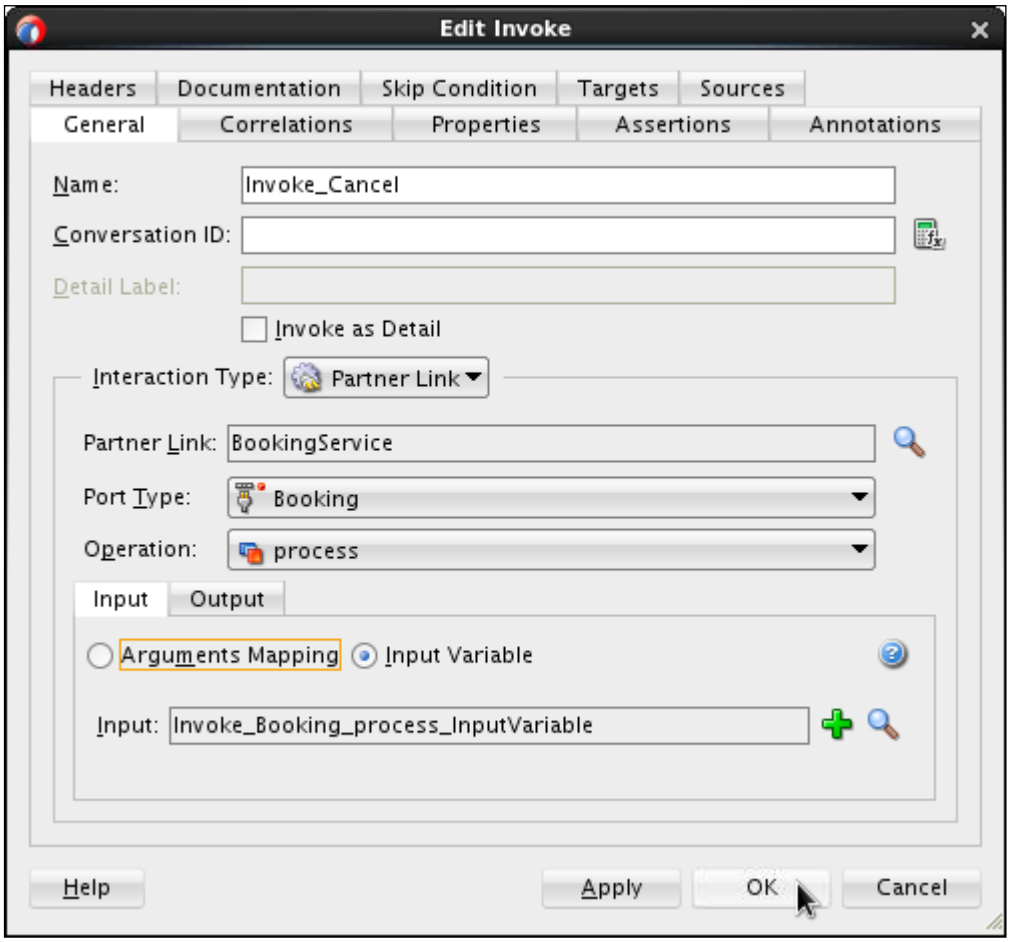
At this point, you have created and configured the “happy path,” where the enrollment and payment information are processed successfully. Recall, however, that the practice scenario describes the possibility of a fault being raised in the payment processing, and the need for handling that fault. Specifically, by the time payment processing fails, space will already have been reserved in the class for the enrollment. In this section, you configure the compensation handler to send an offsetting enrollment cancellation that will free up the reserved seat.

9. Create and configure a compensation handler on `Scope_Inner`.
   a. Expand `Scope_Inner`.
   b. On the left edge of the Scope, click the Add Compensation Handler icon as shown in the following diagram:

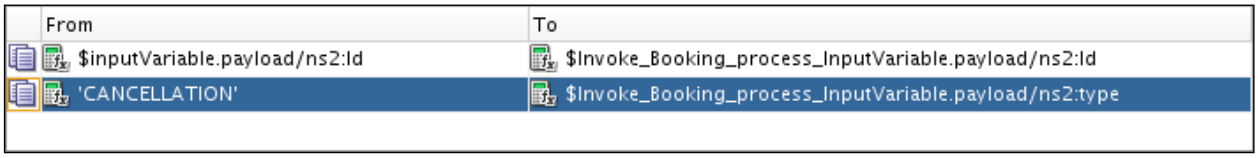


10. Create and configure an Invoke activity.
   a. Add an Invoke activity to the compensation handler.
   b. Right-click the new Invoke and select Edit.
   c. Name the activity `Invoke_Cancel`.
   d. To specify the partner link, use the Browse icon to navigate to BookingService.
   e. Use the Browse icon to *reuse* `Invoke_Booking_process_InputVariable`.
   f. Use the Browse icon to *reuse* the `Invoke_Booking_process_OutputVariable`.

g. Verify and save your work.

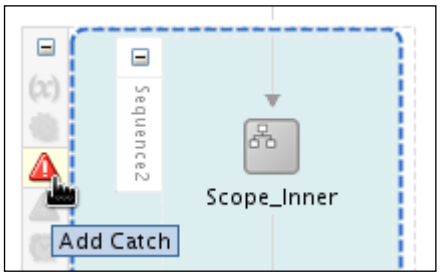


11. Create and configure an Assign activity.
    a. Add an Assign activity just above `Invoke_Cancel`.
    b. Name the activity `setCancel`.
    c. Right-click the new activity and select Edit.
    d. Map the `inputVariable` element ID to the corresponding element in `Invoke_Booking_process_InputVariable`.
    e. Use Expression Builder to map the text `'CANCELLATION'` to the `Invoke_Booking_process_InputVariable type` field.
    f. Verify and save your work.



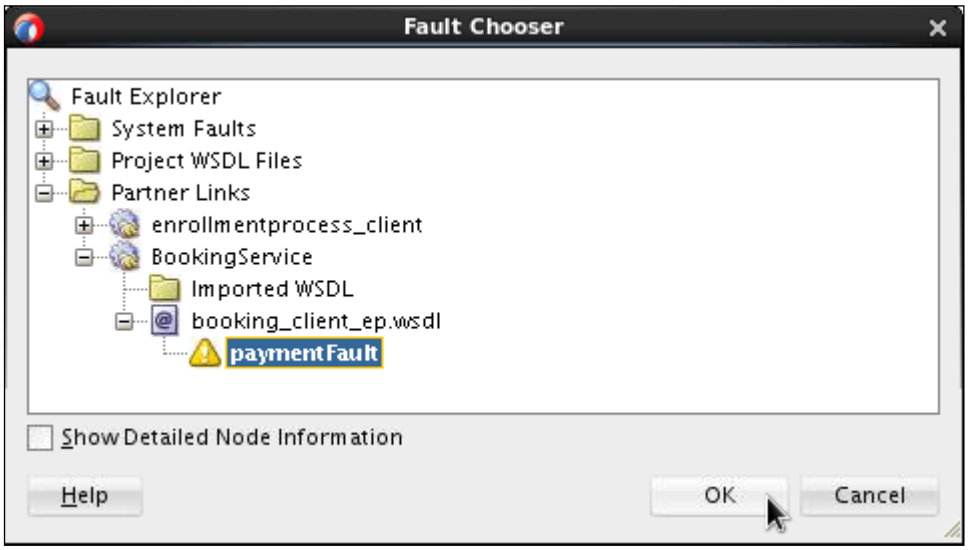
Adding Error Handling to `Scope_Outer`

12. Create and configure a Catch branch.
    a. Collapse `Scope_Inner`.
    b. On the left edge of `Scope_Outer`, right-click the Add Catch icon and select Edit.



The Edit Catch window is displayed.

    c. Use the Browse icon to navigate to Partner Links > BookingService > booking_client_ep.wsdl > `paymentFault`.



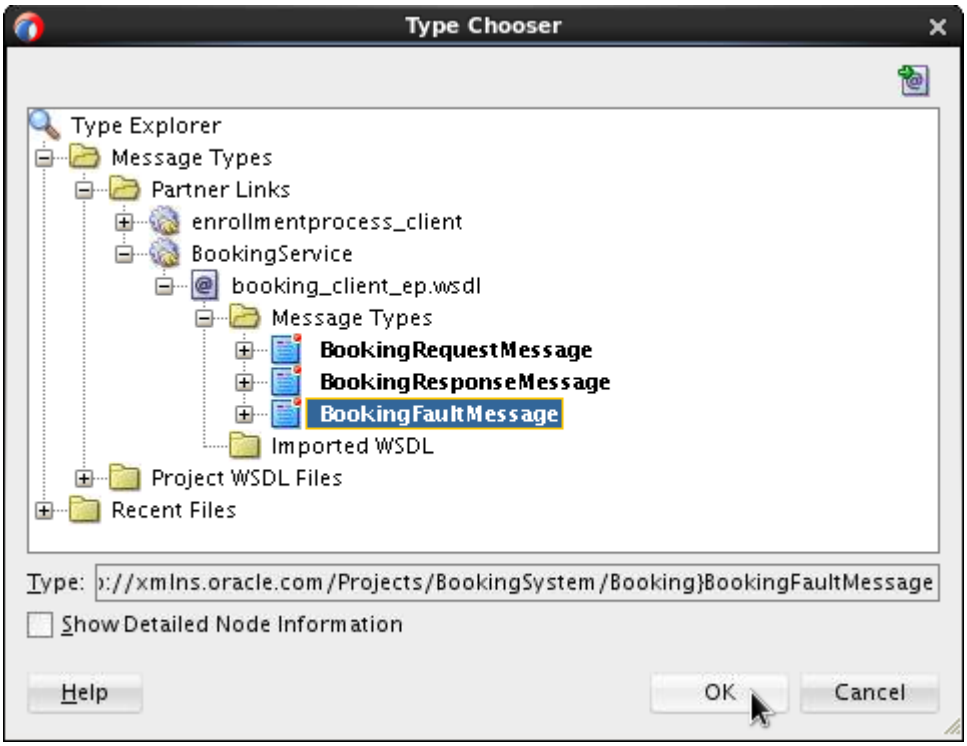
This specifies which fault to catch. The fault is thrown by the BookingService application, and is described in its `.wsdl` document.

    d. Specify `faultVariable` in the Fault Variable field.

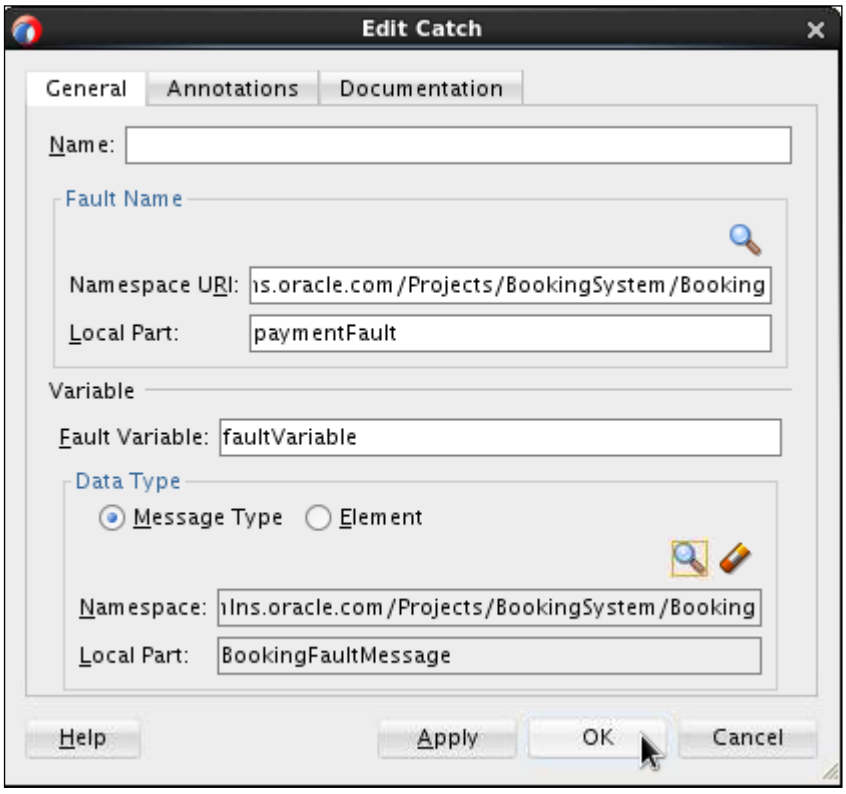
This is the variable that holds the fault message.

    e. Select the Message Type radio button.

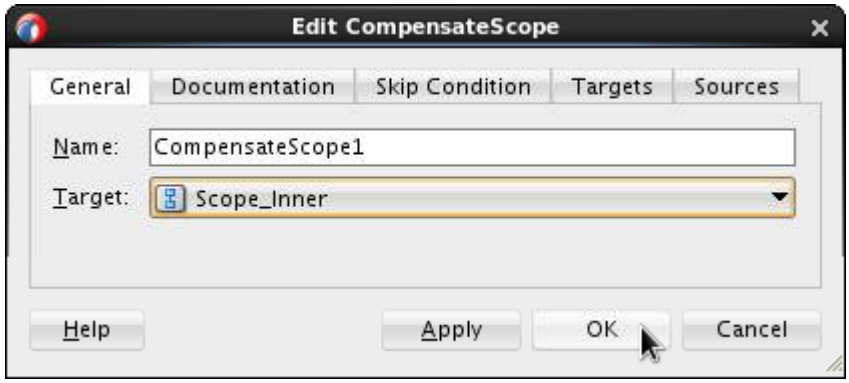
f. Use the Browse icon to navigate to the partner link `BookingService` file `booking_client_ep.wsdl` and specify the Message Type `BookingFaultMessage`.



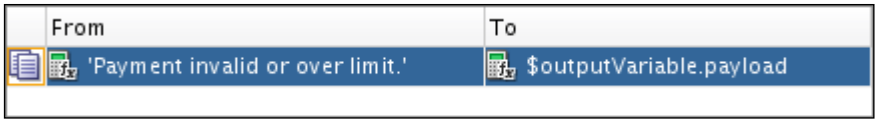
g. Verify and save your work.



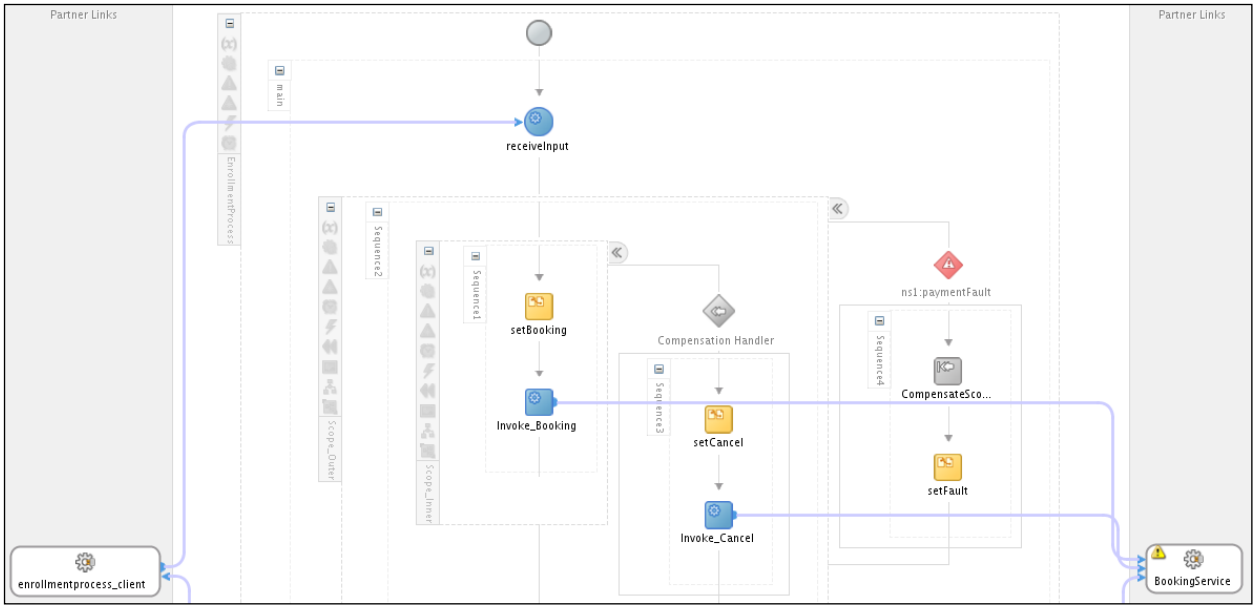
13. Create and configure a CompensateScope activity.
    a. Add a CompensateScope activity within Catch.
    b. Right-click the activity and select Edit.
    c. From the Target drop-down list, select `Scope_Inner`.
    d. Verify and save your work.



14. Create and configure an Assign activity.
    a. Add an Assign activity just below CompensateScope.
    b. Name the activity `setFault`.
    c. Right-click the new activity and select Edit.
    d. Use Expression Builder to map the text `'Payment invalid or over limit.'` to the `outputVariable enrollmentResponse` field.
    e. Verify and save your work.



15. Compare your BPEL process to the following partial image. (The fault-handling components are shown here. The “successful” path is not shown in its entirety.)



Completing the Configuration of Mediator

Configuration of the BPEL process is complete. The only step that remains is to configure the routing rules in the Mediator.

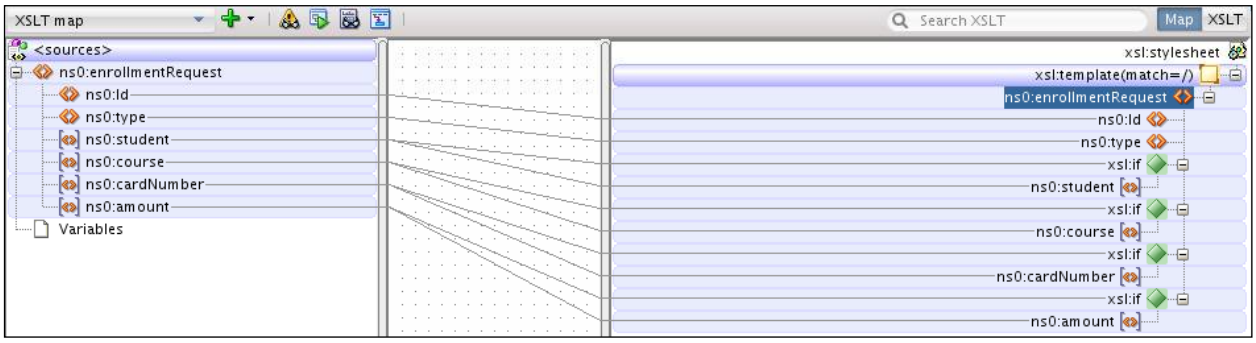
16. Create and configure the transformation for the enrollment request.
    a. In the composite overview window, right-click the Mediator `RouteRequest` and select Edit.
    b. To the right of the first Transform Using field, click the Mapper file icon.
    c. Create a new XSLT mapper file.
    d. In the mapper, map `enrollmentRequest` to `enrollmentRequest`.

       Note: This mapping passes the data unchanged from the Mediator to the BPEL Process. This is possible because both the components specify the same input message in their `wsdl` files.

       The AutoMapper dialog box opens.
    e. Click OK to accept all defaults.

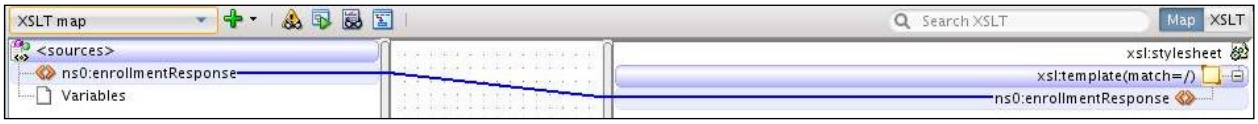
       Each of the leaf nodes is mapped from source to target.

f. Verify and save your work.

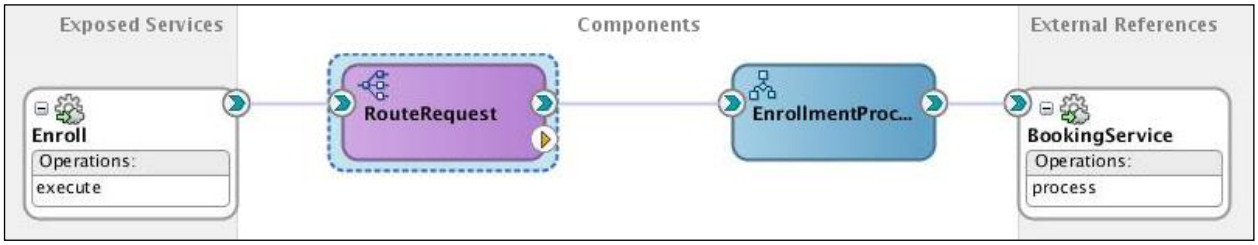


g. Close the editor.

17. Create and configure the transformation for the enrollment reply.
    a. In the Synchronous Reply section, to the right of the Transform Using field, click the Mapper file icon. Note, you may have to scroll the RouteRequest `.mplan` window down to see the Synchronous Reply
    b. Create a new mapper file.
    c. In the mapper, map `enrollmentResponse` to `enrollmentResponse`.
    d. Verify and save your work.



18. Close the `.mplan` and `.bpel` file editors.
19. In the `compsite.xml` editor, verify your composite application.



Practice 8-4: Deploying and Testing the Application

Overview

In this practice, you deploy the `Enroll` application and test the fault-handling configuration.

Assumptions

This practice assumes that you have completed practices 8-1 through 8-3 successfully.

Tasks

Deploying the `Enroll` Composite Application

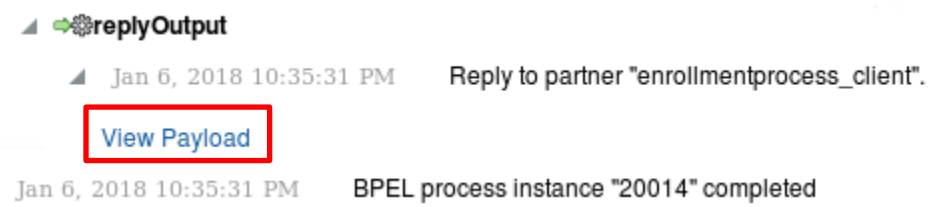
1. Prepare for the test by deploying the `Enroll` application to IntegratedWebLogicServer.

   Note: Remember to select the “Overwrite any existing composites with the same revision ID” check box.

Testing the Composite Application

2. Open Oracle Enterprise Manager (http://localhost:7101/em).
   a. In the Target Navigation pane, expand the SOA node and right click soa-infra and select Home > Deployed Composites.
   b. Click the Enroll [1.0] link.
   c. On the Enroll [1.0] dashboard page, click Test.
   d. On the Test Web Service page, navigate to the Request tab. Use the Browse button to replace the default message with the contents of the file `/home/oracle/labs/files/xml_in/enrollment_input.xml`.
   e. Click Test Web Service.

      The Response tab is displayed.
   f. Click Launch Flow Trace.
   g. Click Enrollment Process.
   h. Under the replyOutput node, click View Payload.

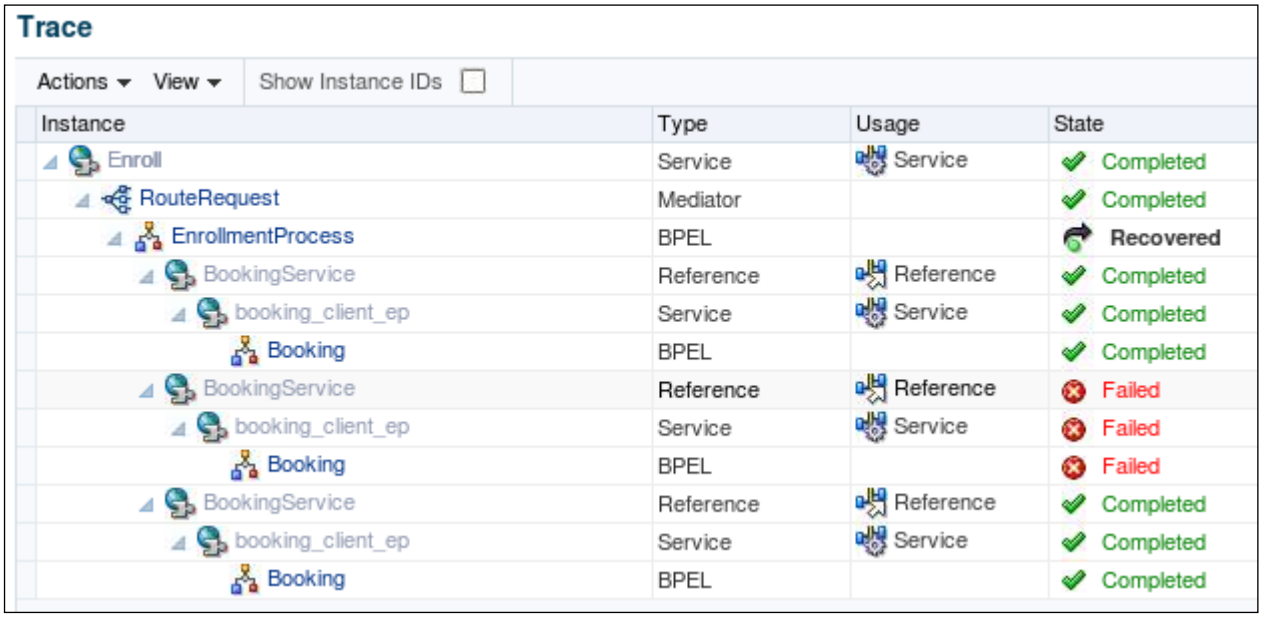


   i. Verify that the status value returned is “Enrollment process completed successfully.”

```
<?xml version="1.0" encoding="UTF-8"?><outputVariable>
   <part xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="payload">
      <enrollmentResponse xmlns="http://www.example.org/ns/enroll">Enrollment process completed successfully.</enrollmentResponse>
   </part>
</outputVariable>
```

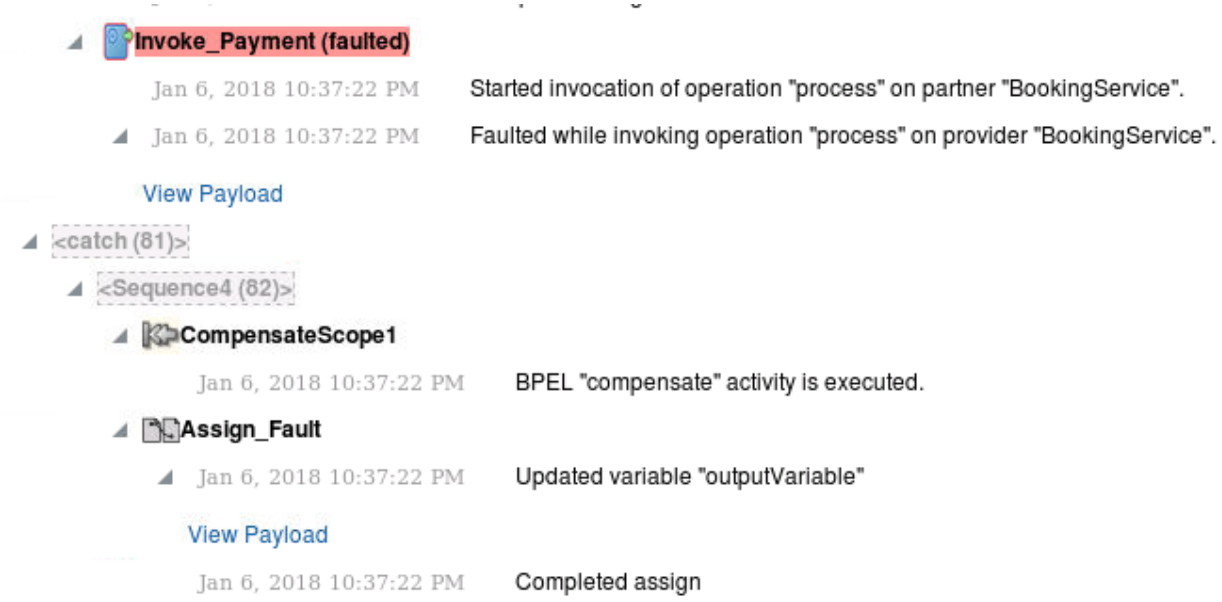
   j. Close the Flow Trace windows.

3. To perform a second test (and generate a fault), click the Request tab.
   a. On the Request tab page, modify the amount to be any number greater than 5000.
   b. Click Test Web Service.
   c. On the Response tab page, click Launch Flow Trace.



   d. Click EnrollmentProcess.

   The Flow Trace shows that the Invoke faulted and that execution passed to CompensateScope.



e. View the payload for Invoke_Payment. The response message matches what is configured.

```
<?xml version="1.0" encoding="UTF-8"?><messages>
   <input>
<Invoke_Payment_process_InputVariable>
         <part xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="payload">
            <enrollmentRequest xmlns="http://www.example.org/ns/enroll">
               <Id>10</Id>
               <type>PAYMENT</type>
               <cardNumber>9090-8080-7070-6060</cardNumber>
               <amount>5100</amount>
            </enrollmentRequest>
         </part>
      </Invoke_Payment_process_InputVariable>
   </input>
   <fault>
<bpelFault>
         <faultType>1</faultType>
         <paymentFault xmlns="http://xmlns.oracle.com/Projects/BookingSystem/Booking">
            <part name="payload">
               <enrollmentFault xmlns="http://www.example.org/ns/enroll">Payment invalid or excessive.</enrollmentFault>
            </part>
         </paymentFault>
      </bpelFault>
   </fault>
   <faultType>
<message>1</message>
   </faultType>
</messages>

```

f. Close the Flow Trace windows and minimize your browser.

Practice 8-5: Using the Fault Policy Wizard

Overview

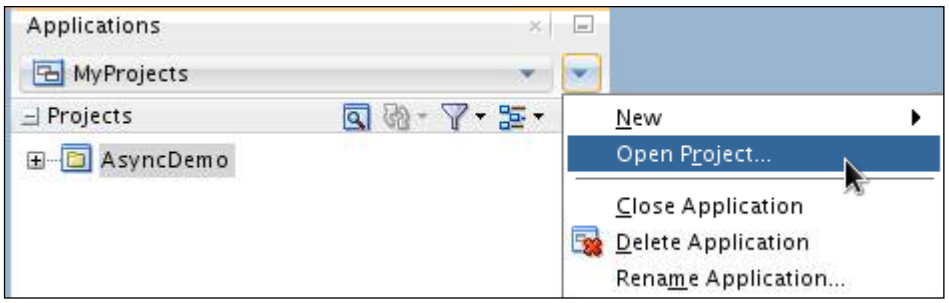
In this practice, you design a fault policy with the Fault Policy wizard and associate the fault policy with the fault policy binding file.

Assumptions

This practice makes no assumptions.

Tasks

1. Open and deploy SayHello and CallGreeting.
   a. Verify that the BPELProjects application is selected.
   b. From the Application Navigator menu, select Open Project.



   c. Navigate to `/u01/app/fmw12c/domains/mywork/BPELProjects/ SayHello/SayHello.jpr`. Note: If prompted to upgrade the project, click select Yes and OK.
   d. Repeat steps b and c to open the CallGreeting project.
   e. Deploy SayHello to IntegratedWebLogicServer.

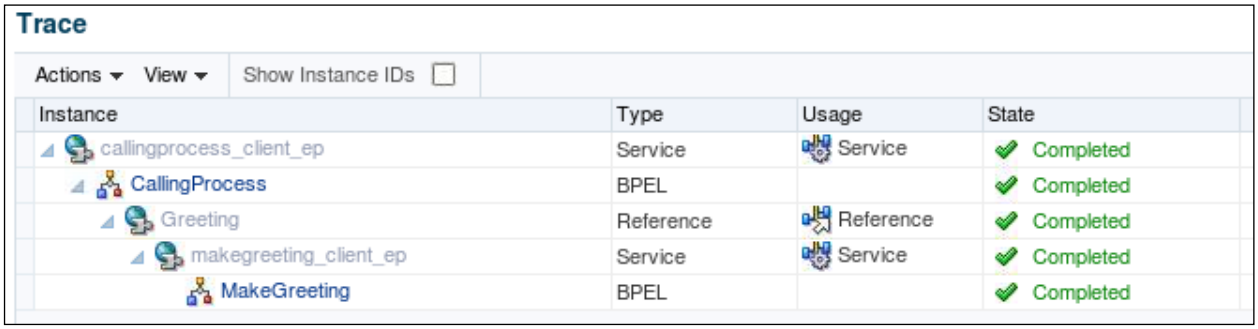
      Remember to select the “Overwrite any existing composites with the same revision ID” check box.
   f. Repeat step e to deploy the CallGreeting project.
2. Test the applications.

   Note: The SayHello project is the classic “Hello World” project. It concatenates the string “Hello” and an input string that is received as part of the project invocation. The CallGreeting [asynchronous] project accepts an input string that you provide, and then passes that string when it invokes SayHello. In the following test, you verify that the applications run without error and do not [yet] throw any faults.
   a. Open the Oracle Enterprise Manager page (http://localhost:7101/em).
   b. In the Target Navigation pane, expand the SOA node and right-click soa-infra and select Home > Deployed Composites.
   c. Click the “CallGreeting [1.0]” link.
   d. On the “CallGreeting [1.0]” dashboard page, click Test.
   e. On the Test Web Service page, scroll down to the Input Arguments section to the Request tab.

f. Provide your first name (or any other text you like) as input.

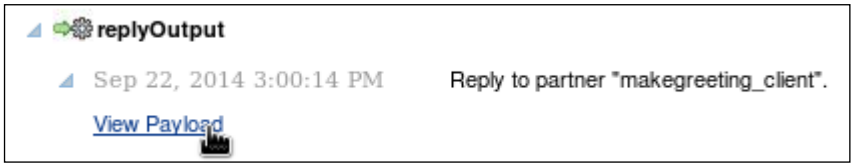
g. Click Test Web Service.

h. On the Response tab, click the Launch Flow Trace link. Verify that each step has completed without error.

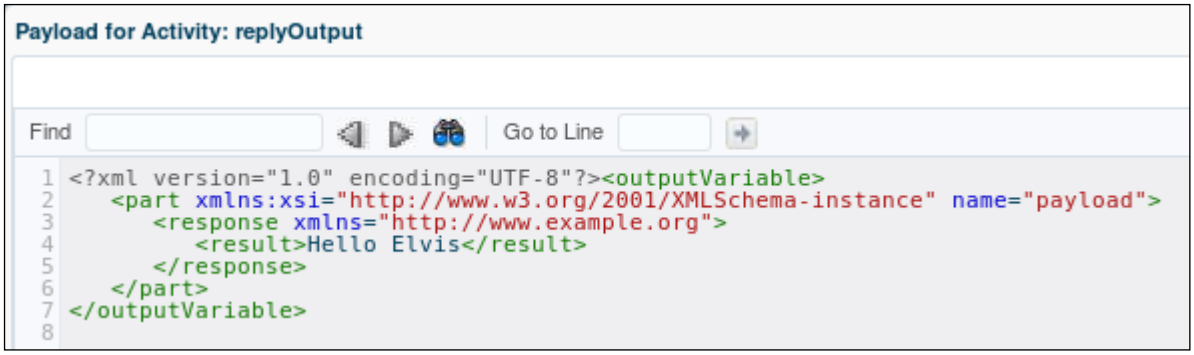


i. In the Trace, click MakeGreeting.

j. In the Audit Trail, click ViewPayload.



k. Verify that the payload includes the concatenated text.



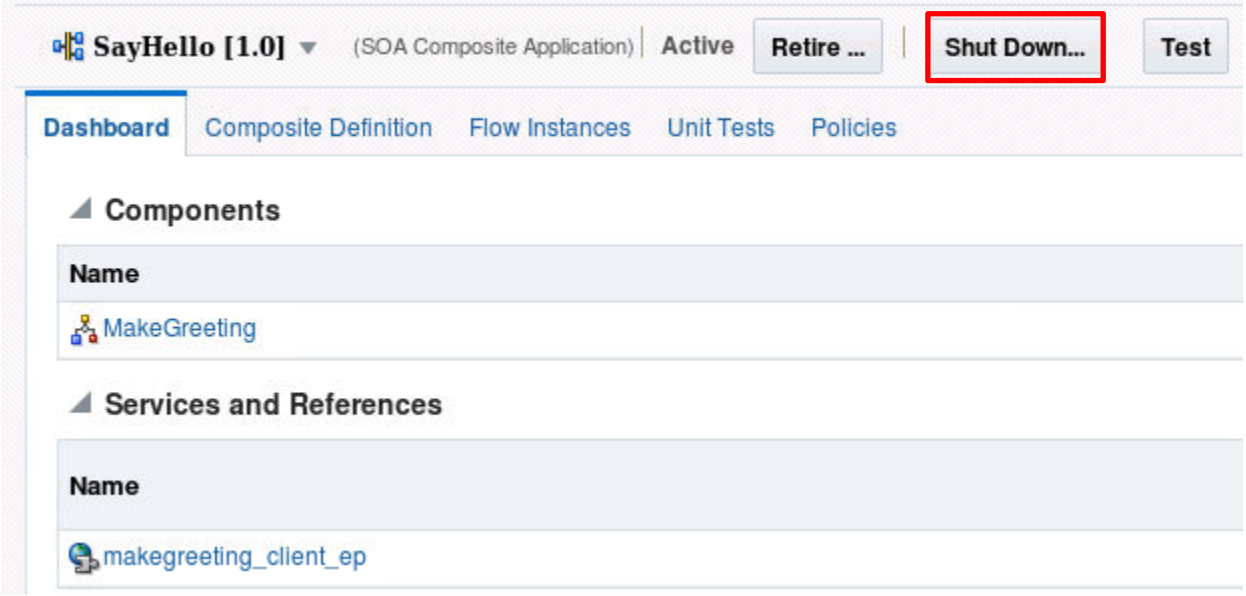
Generating a Remote Fault

3. Shut down the SayHello project.

a. In the Enterprise Manager target navigation pane, expand SOA and right-click soa-infra and select Home > Deployed Composites.

b. Select the SayHello project.

c. In the SayHello dashboard, click Shut Down.



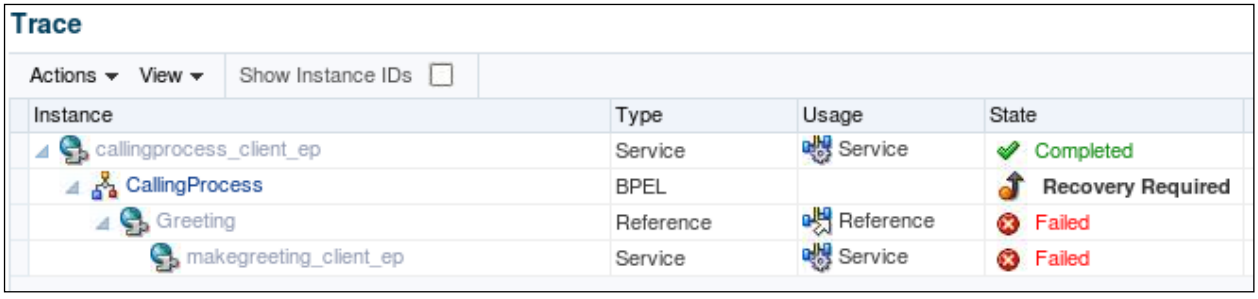
d. In the Confirmation dialog box, click Yes.

The project is shut down.

4. Generate a fault condition.

a. Repeat the preceding task 2, steps b–f to test the CallGreeting composite.

b. On the Response tab, click the Launch Flow Trace link. Note the state of each step of the instance.



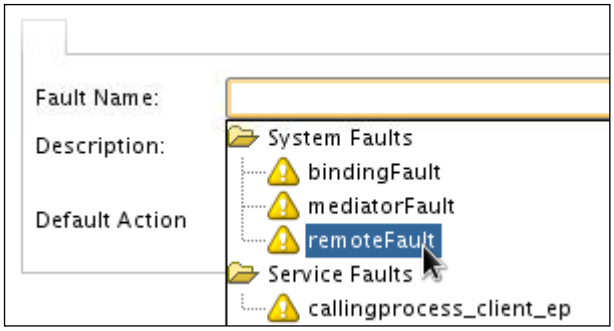
c. Above the Trace pane, on the Faults tab, notice the fault information.

Creating the Fault Policy

In this section, you use the wizard to create a Fault Policy.

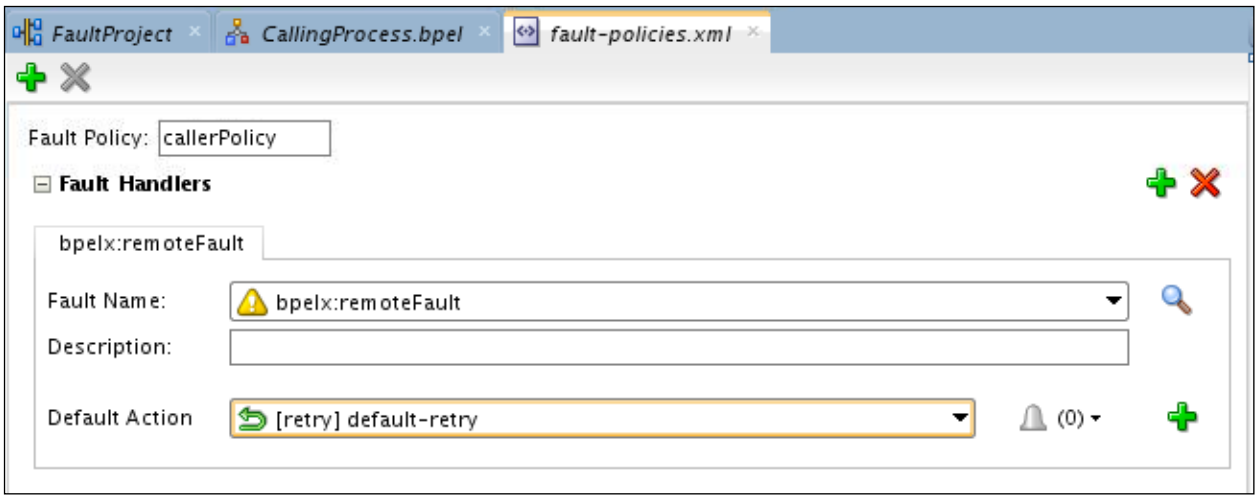
5. In JDeveloper, open the composite overview for the CallGreeting project.
6. Create a fault policy document.
   a. To open the wizard, select File > New> From Gallery.
      The New Gallery dialog box opens.
   b. In the Categories pane, select SOA Tier > Faults.
   c. In the Items pane, select Fault Policy Document and click OK.

   The Fault Policies editor opens.
7. Define a fault policy.
   a. Name the policy `callerPolicy`.
   b. Select remoteFault from the Fault Name drop-down list.



c. Select [retry] default-retry from the Default Action drop-down list.

This defines what actions the policy will take when it is activated.



8. Edit the properties of the retry action.

a. Click the **Actions** tab and highlight **default-retry**.

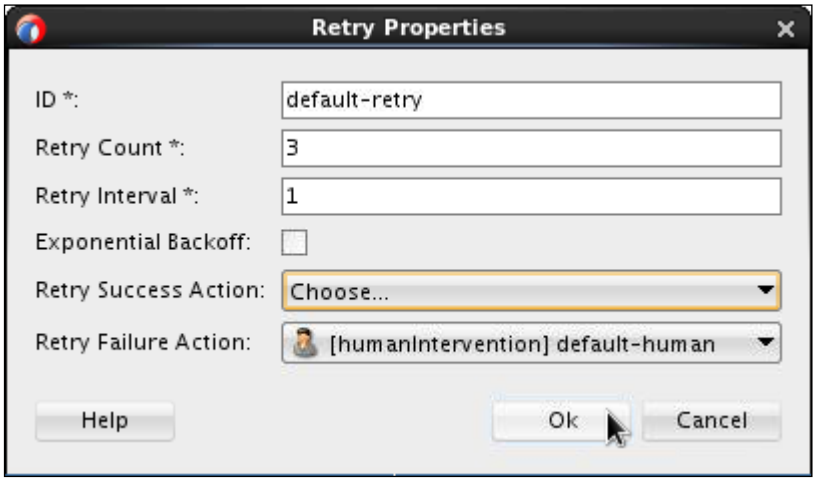
b. Click the Pencil icon to edit the action.

1) Retry Count: 3

2) Retry Interval: 1

3) Retry Failure Action: [humanIntervention] default-human

c. Verify your work and click OK.

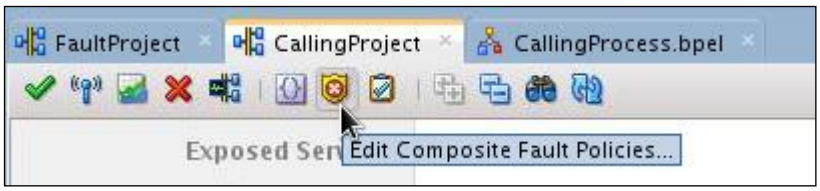


The policy is now created.

d. Save your work and close the Policy wizard.

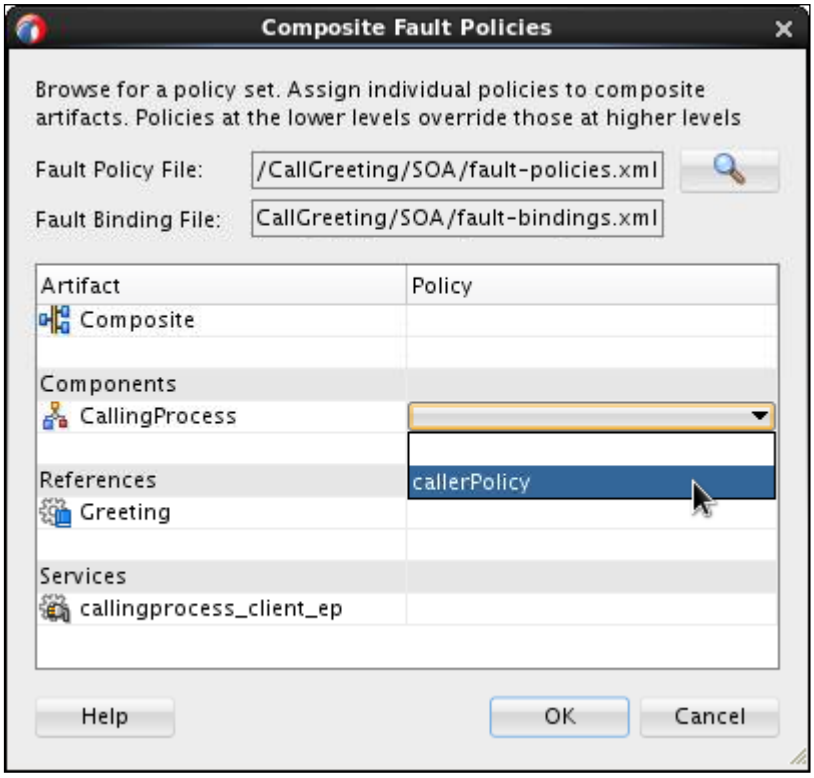
Notice that the `fault-policies.xml` file is created in the project under SOA.

9. Associate the fault policy with the BPEL process CallingProcess.
   a. On the Composite Editor toolbar, click the Edit Composite Fault Policies icon.



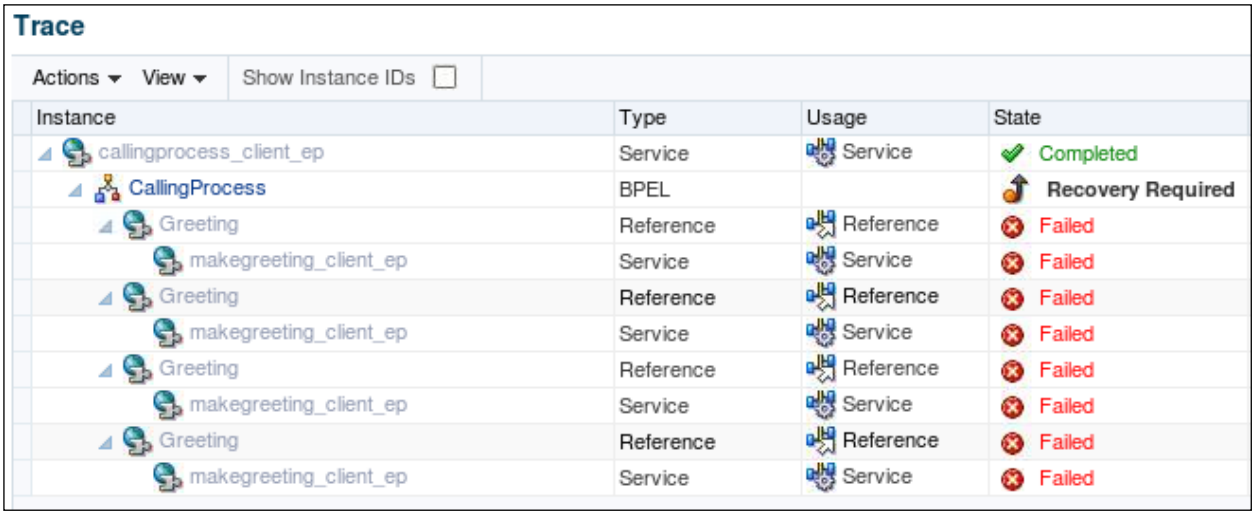
   The Composite Fault Policies screen is displayed. Here you can associate fault policies with the composite or the components therein. In this practice, you associate the policy with CallingProcess.

   b. Select the policy from the drop-down list to the right of the CallingProcess component.



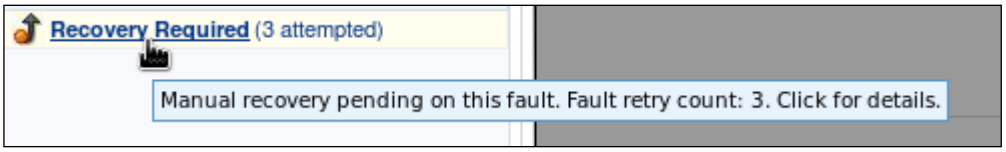
   c. Click OK.

10. Redeploy and test the application.
    a. Redeploy the CallGreeting project to activate the fault policy.
    b. Repeat the preceding step 2 to test the CallGreeting composite.
    c. On the Response tab, click the Launch Flow Trace link.
       The trace indicates that the instance was retried three times after the initial failure.



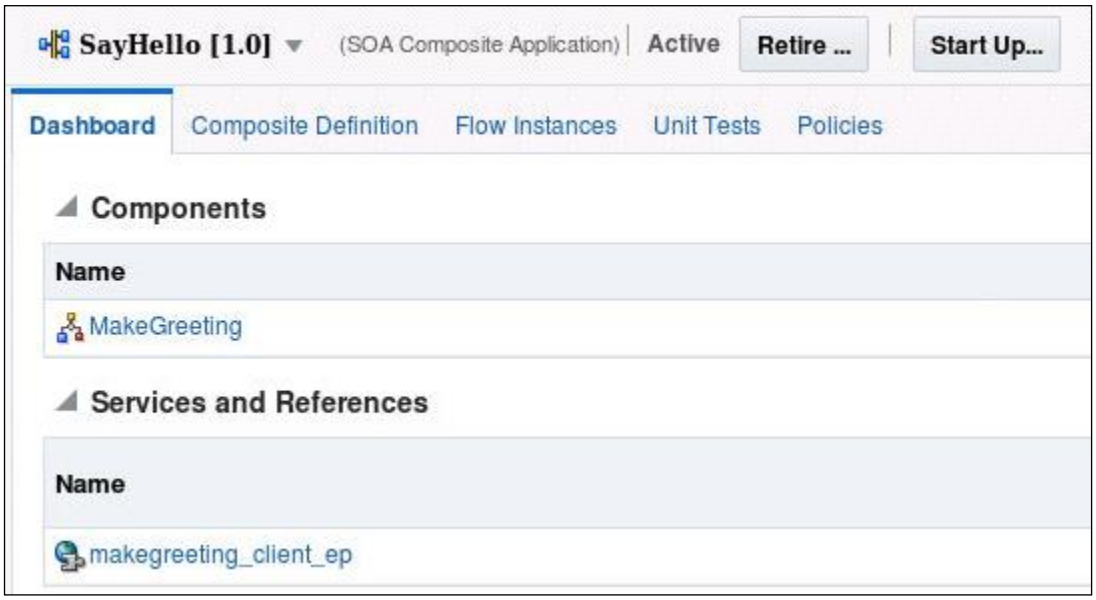
Manually Recovering the Fault

11. 11. Hover your cursor over the text “Recovery Required” on the Faults tab.
    A message is displayed confirming that manual recovery is pending.



12. Activate the SayHello project.
    a. In the Enterprise Manager target navigation pane, expand soa-infra > default.
    b. Select SayHello project.

c. In the SayHello pane, click Start Up.



d. Click Yes to confirm that you want to start the SayHello composite.

13. Manually recover the fault.

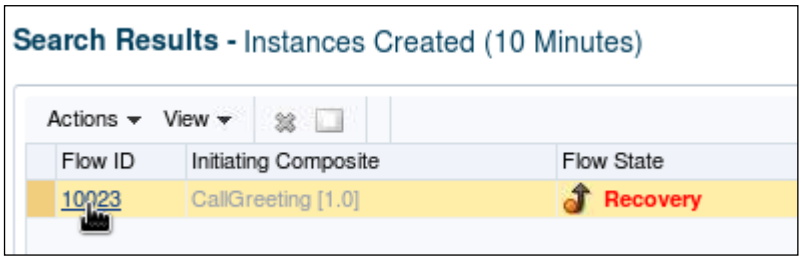
a. In the Enterprise Manager target navigation pane, expand SOA and right-click soa-infra and select Home > Deployed Composites.

b. Select CallGreeting project.

c. In the CallGreeting pane, click Flow Instances.

d. In the Search pane, click Search.

e. In the Search Results pane, click the Flow ID.

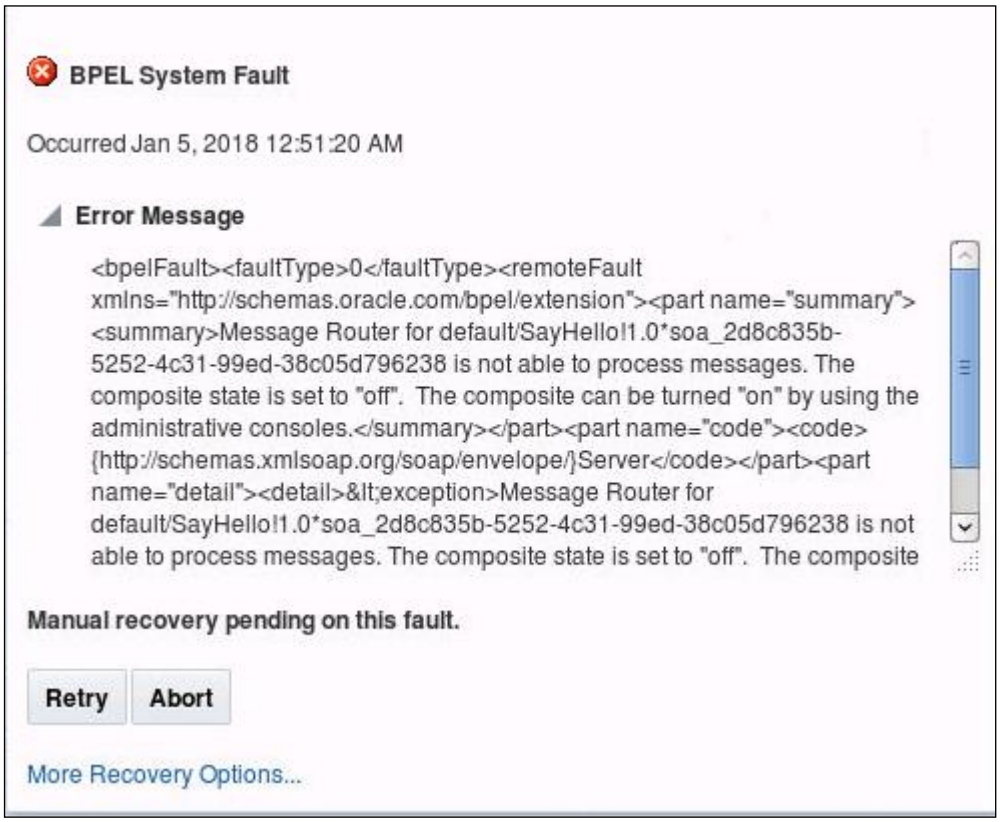


The Flow Trace for the instance is displayed.

f. On the Faults tab, click the Recovery Required link.

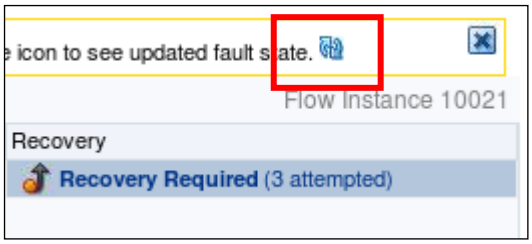
Details of the fault are displayed along with the Retry and Abort options.

g. Click Retry.



The Trace is updated, indicating the completion of the instance.

h. On the Faults tab, click the Refresh icon.



The Recovery status is updated.

i. Close the open Flow Trace window.

Practices for Lesson 9: Implementing Human Workflow

Practices for Lesson 9: Overview

Practices Overview

In this practice, you implement a human task component and email notification functionality in a composite application.

The human task component allows the user to review order details, and then approve or reject the order. Email notification is implemented to notify the customer about the status of the order.

Note: Use *exact* names (both spelling and case are important) as specified for your project and components. A later practice titled “Sharing Functionality” will expect to find these names in use.

Practice 9-1: Seeding the Demo User Community

Overview

In this section, you deploy a web application that contains the demo users for the internal LDAP database of the WebLogic Server. It also contains a servlet that installs these users for you.

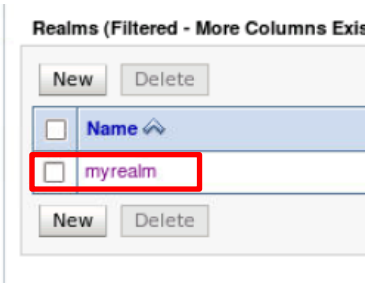
Assumptions

This practice assumes that the Integrated WebLogic Server is running.

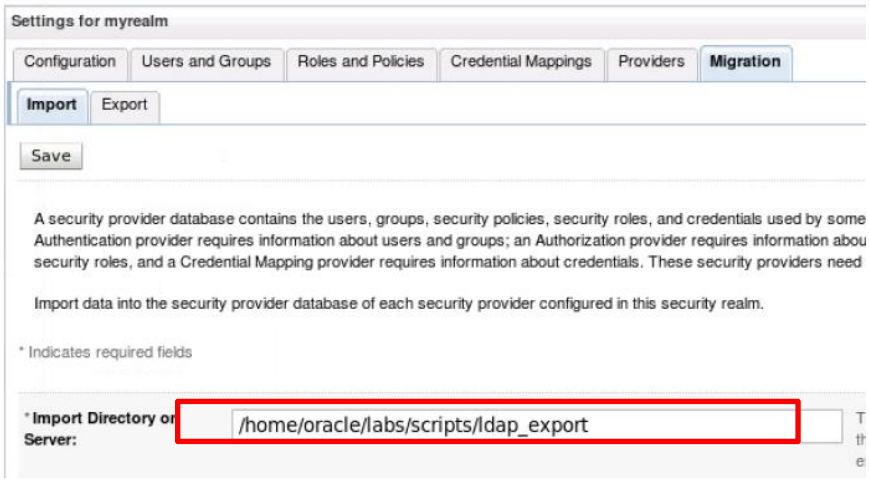
Tasks

1. Import users and groups into WebLogic embedded LDAP server.
2. Open a browser and log into the WebLogic Administration Console: http://localhost:7101/console
3. On the left side of the window click Security Realms.

4. Then select myrealm.



5. Then select the Migration tab. In the Import Directory on Server: input field, enter:
   `/home/oracle/labs/scripts/ldap_export`



6. And click Save.
7. Close the browser.

Practice 9-2: Configuring the Email Driver

Overview

In this practice, you enable Email notifications to work in the SOA Server for Human Tasks and BPEL Email notification.

Assumptions

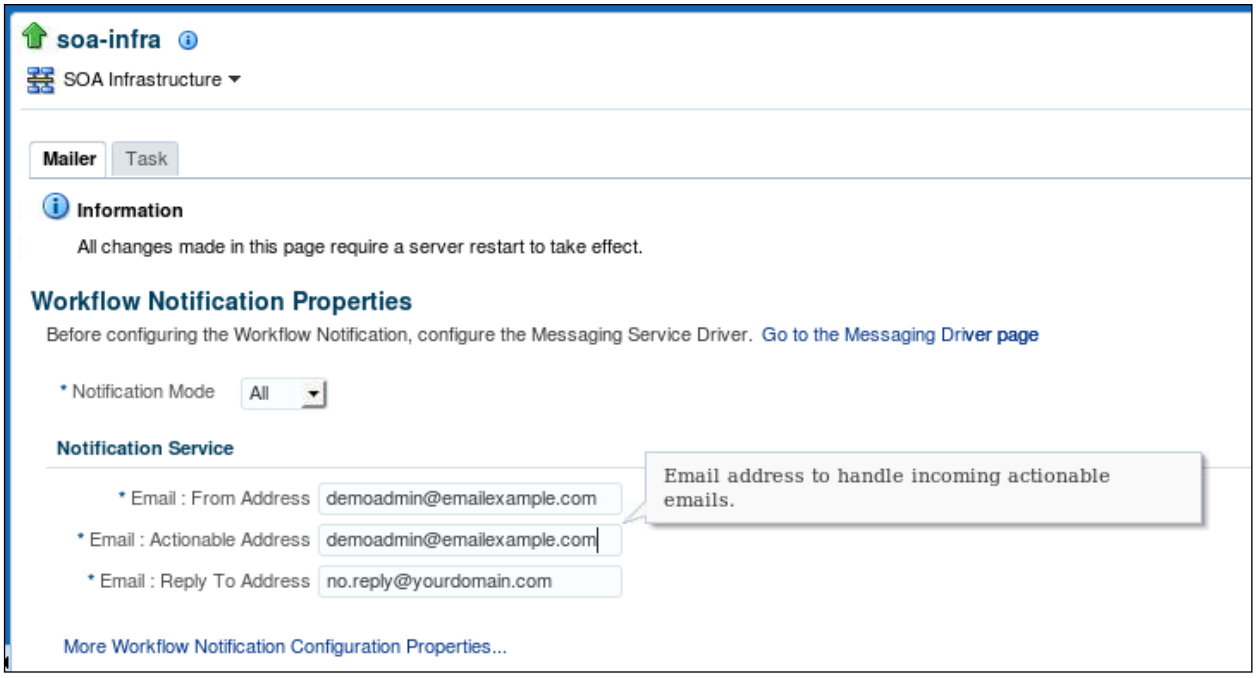
This practice makes no assumptions.

Tasks

1. Open the Oracle Enterprise Manager page (http://localhost:7101/em).
   a. In the Target Navigation pane, expand the SOA folder. Right-click soa-infra and select SOA Administration > Workflow Properties.
   b. On the Workflow Notification Properties page, set and enter the following values:

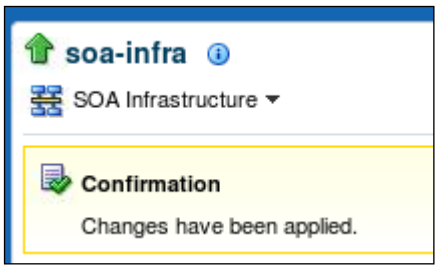
Notification Mode	ALL
Email: From Address	demoadmin@emailexample.com
Email: Actionable Address	demoadmin@emailexample.com
Email: Reply To Address	no.reply@yourdomain.com (default)

   c. Verify your work and click Apply.



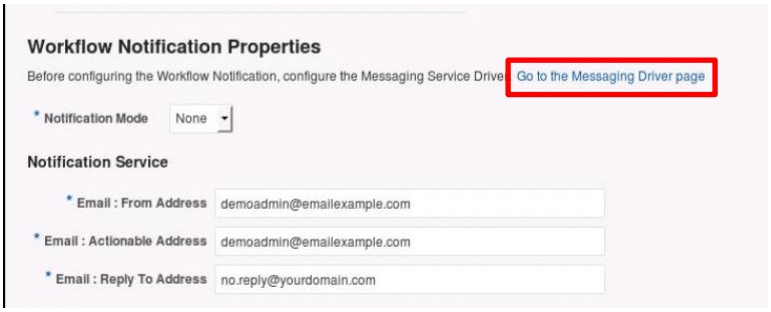
   d. In the Confirmation window, click Yes.

e. On the SOA Infrastructure > Workflow Notification Properties page, verify that you get a confirmation that the changes have been applied.

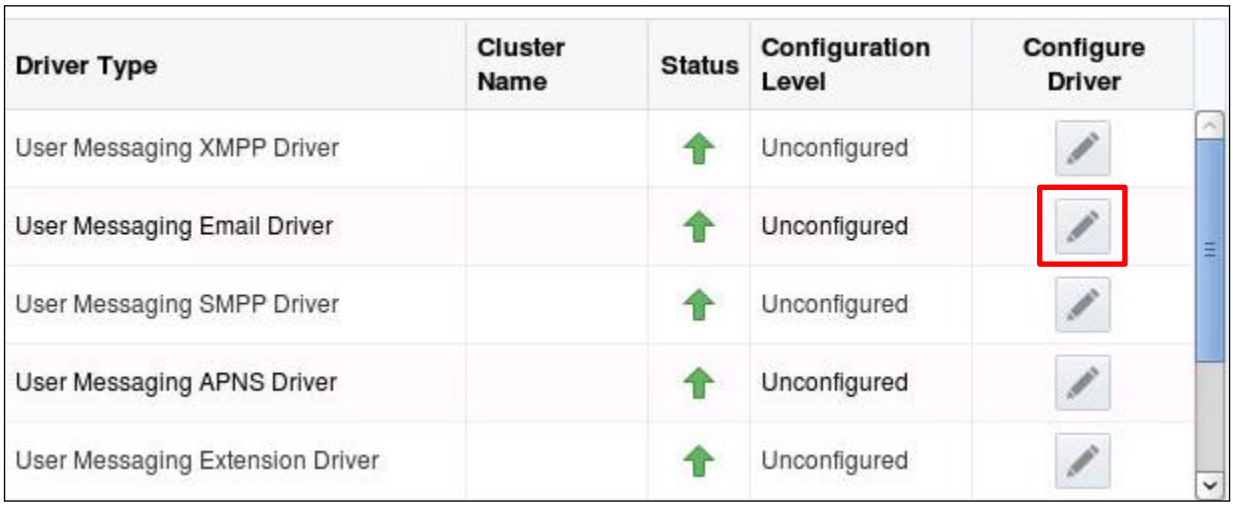


2. Configure the email driver.

a. Click the Go to the Messaging Driver page link.



b. On the **usermessagingserver** page, under Associated Drivers in the Local tabbed page, click the **Configure Driver** (bi-directional arrows) icon in the row for the User Messaging Email Driver (Driver Type).



c. Click Create.

d. Name the configuration `bcaEmail`.

e. Scroll down in the Driver-Specific Configuration section as needed to set the following Driver-Specific Configuration field values:

E-Mail Receiving Protocol	`POP3`
OutgoingMailServer	`emailexample.com`
OutgoingMailServerPort	`25` [This is the default]
Default From Address	`demoadmin@emailexample.com`
IncomingMailServer	`emailexample.com`
IncomingMailServerPort	`110`

f. For incoming Mail IDs, click the pencil

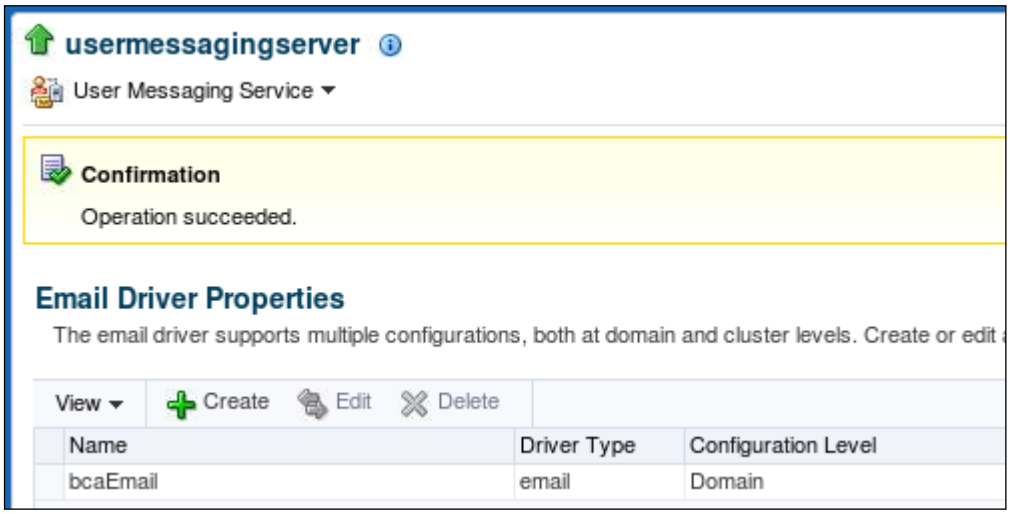
g. For IncomingUserPasswords: set the following:

- Type of Password: **Indirect Password, Create New User** [default]
- Indirect Username/Key: **`demoadmin`**
- Click + Add to add a new Incoming Mail ID
- Incoming Mail Id: `demoadmin@emailexample.com`
- Incoming User ID: `demoadmin`
- Incoming Password: `welcome1`

Click OK and click OK again.

Click OK at the top right of the page to confirm your changes.

h. Verify that you receive a confirmation that changes have been applied.



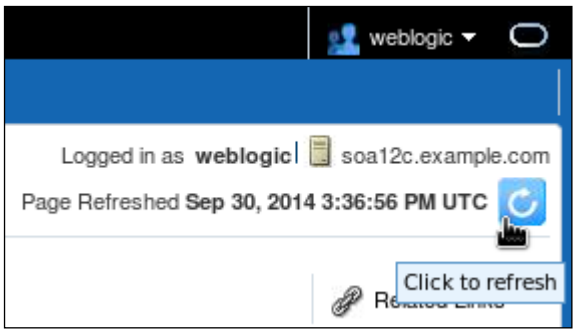
Testing Your Changes

3. Stop and restart the Integrated WebLogic Server to cause your changes to take effect.
4. Test the email driver.
   a. In the Enterprise Manager Target Navigation pane, Expand SOA and right-click soa-infra and select Service Engines > Human Workflow.

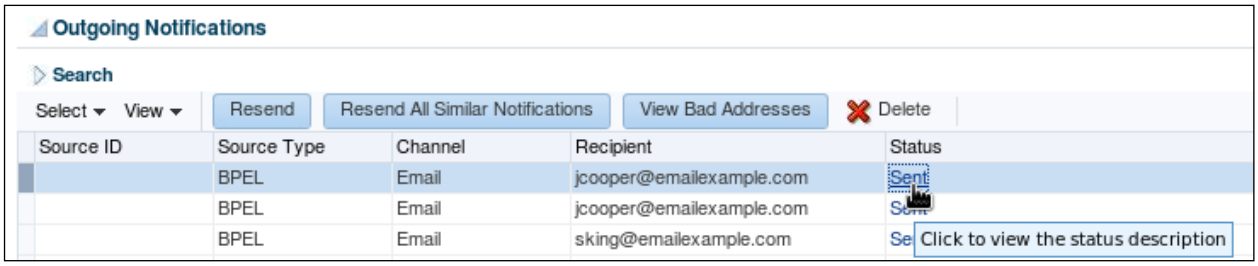
b. On the Human Workflow Engine page, click the Notification Management tab.
c. Click Send Test Notification.
d. In the Send Test Notification dialog box, supply the following values:
   - Send To: `jcooper@emailexample.com`
   - Subject: `Test Message`
   - Content: `Can you see me?`
   - Channel: Email
e. Verify your settings and click Send.

f. The response SENT is displayed in the lower left section of the Send Test Notification pane. Close the pane.

5. Review the sent message status.
   a. Refresh the Notification Management page.

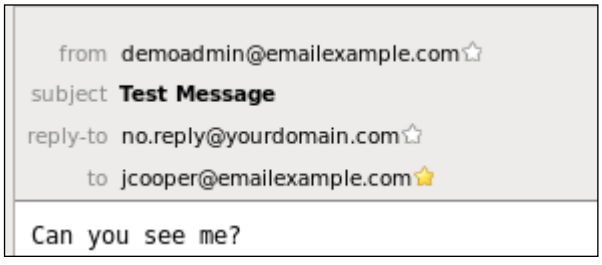


   b. In the Outgoing Notifications section, click the Sent link of the topmost message.



   The Status Description box is displayed.

6. View the email message that was sent to `jcooper`.
   a. Right-click the Thunderbird desktop icon on your desktop and select Open.
   b. In the Thunderbird window, in the All Folders pane, click the `jcooper@emailexample.com` account name.
   c. Click Get Mail on the Thunderbird toolbar.
      An email appears in the inbox.
   d. In the Inbox, click the email to see the message body.



   e. Close or minimize Thunderbird.

Practice 9-3: Creating a Composite Application

Overview

In this practice, you create a new project and create a human task component.

Assumptions

This practice makes no assumptions.

Note: Use *exact* names (both spelling and case are important) as specified for your project and components. A later practice will expect to find these names in use.

Tasks

Creating a Composite Application

In this section, you create a new composite application. You define an exposed service, a human task, and a BPEL service component.

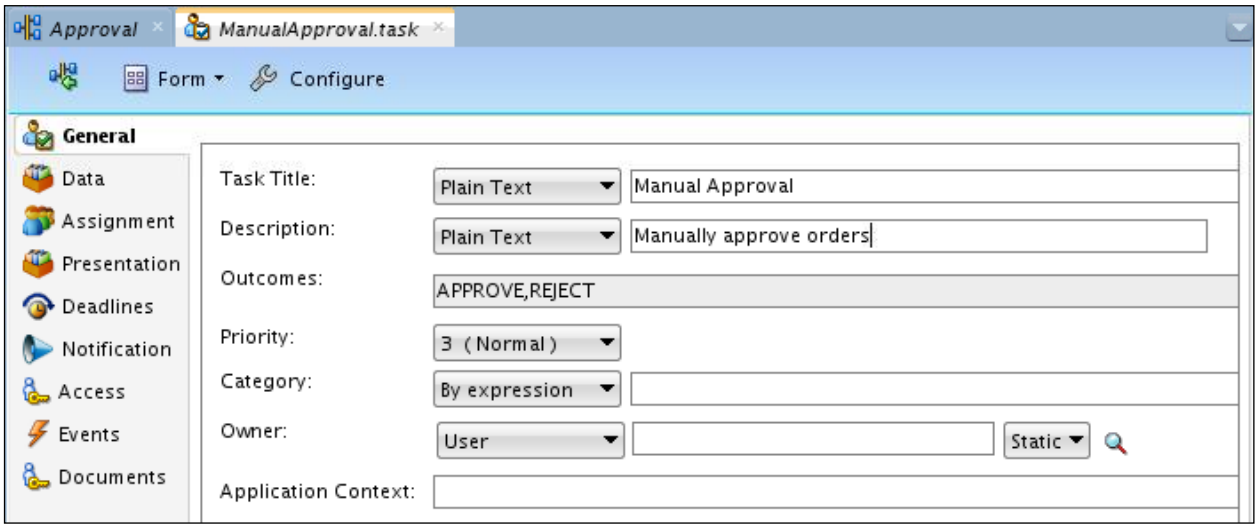
1. In the BPELProjects application, create a new empty SOA project named `Approval`.

   The `composite.xml` file is created and opened for editing.
2. Create a human task component in the assembly model.
   a. In the composite overview window, add a human task component to the Components swimlane.
   b. In the Create Human Task window, enter `ManualApproval` as the Name. Accept the default values for other settings. Click OK.

      Note: The `ManualApproval.componentType` and `ManualApproval.task` files are added to the project. The human task implementation details are stored in the `ManualApproval.task` file, which contains the human task configuration settings.
3. Configure the new human task component.
   a. Right-click the `ManualApproval` human task component icon and select Edit.

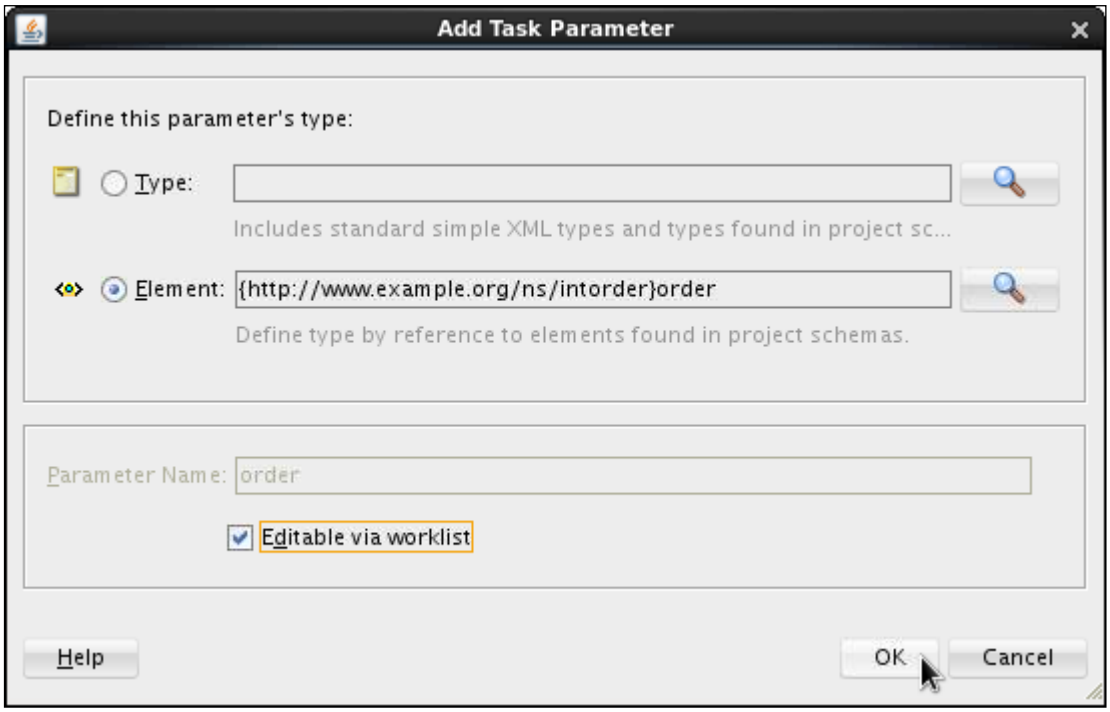
      The Human Task Editor opens the `ManualApproval.task` file.
   b. Configure the Title.
      1) Click the General tab.
      2) Select `Plain Text` from the drop-down menu.
      3) Enter `Manual Approval`.
   c. In the Description field, enter: `Manually approve orders`.

d. Verify your work:



4. Configure the data parameters.
   a. Click the Data tab.
   b. Click the Create icon and select Add other parameter from the drop-down menu.
      The Add Task Parameter window opens.
   c. Select the Element radio button.
   d. Click the “Browse for Complex Types” icon to open the Type Chooser.
      Note: If necessary, point the cursor to an icon to verify its name for selection.
   e. Click the import icon to navigate to `/home/oracle/labs/files/xsd/internalorder.xsd`.
   f. Select the `order` element and click OK.
      You are returned to the Add Task Parameter window.
   g. Select the “Editable via worklist” check box.

h. Use the following screenshot to verify your work, and then save your work.

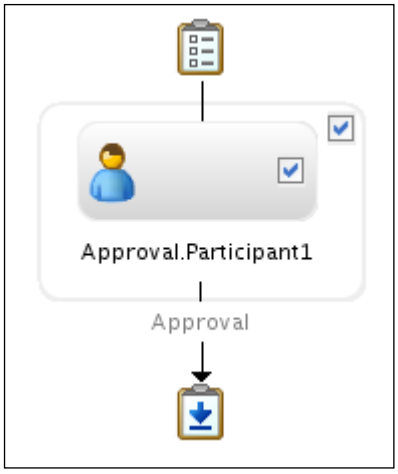


Configuring the Task Assignment and Routing Policy

The following steps associate the task with a particular participant.

5. Define a stage and associate it with a participant.
   a. Click the Assignment tab.
   b. Double-click the heading text `Stage1` in the shaded box.
   c. Name the stage `Approval` and click OK.
   d. Drag a Single Participant from the Workflow Editor section of the Component Palette onto the participant box.

      Approval.Participant1 is added to the box.



e. Double-click Approval.Participant1.

   The Edit Participant Type window opens.

f. In the Participant Names table, click the + Add Property pulldown and select Add User.

   A user row is added to the table.

g. Click the Ellipsis icon in the Value column.

   The Identity Lookup window opens.

   Note: In the next step, you connect to an LDAP Server that provides a list of employees defined in the system. You then assign the task to one of these employees.

6. Look up and select the user `jcooper` as the participant.

   a. Select the Application Server IntegratedWebLogicServer (Resource Palette Connection).

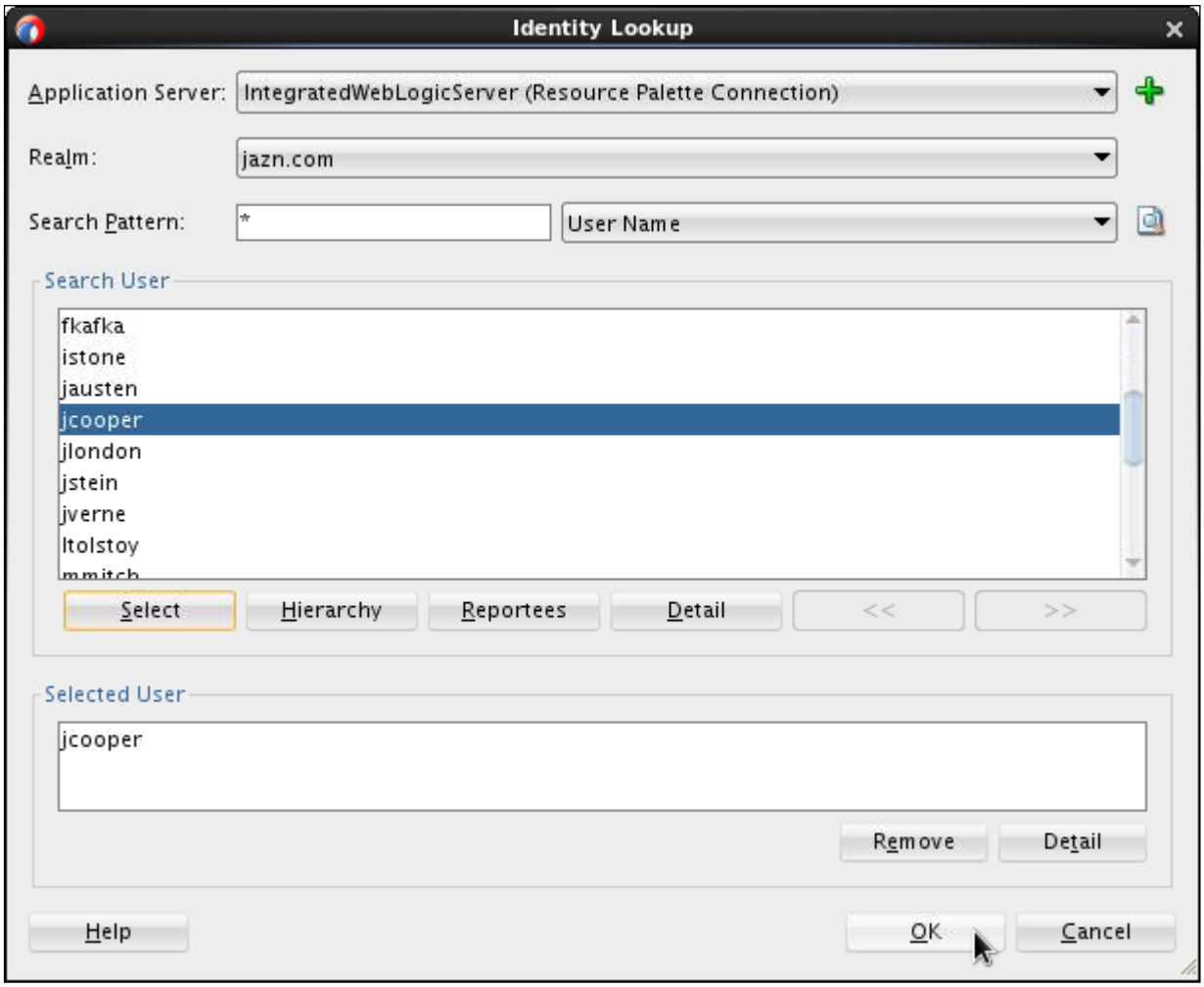
   b. Click the Lookup icon to the right of the Username drop-down menu.

      The Search User area is populated with a list of usernames registered in the LDAP Server.

   c. Highlight the `jcooper` user and click Select.

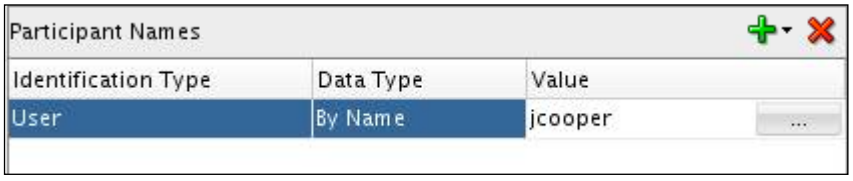
      The `jcooper` user is displayed in the Selected User area.

d. Verify your work and click OK.



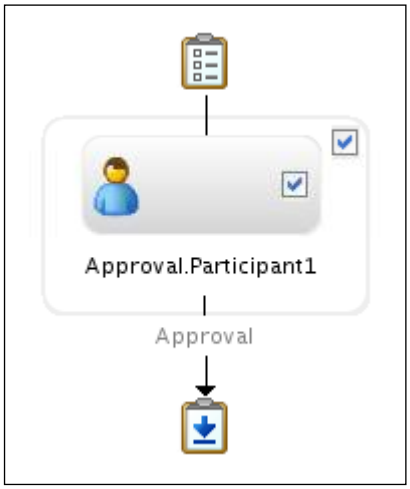
You are returned to the Edit Participant Type window.

e. Verify that `jcooper` appears in the Participant Names table row and click OK.



You are returned to the `ManualApproval.task` window.

f. Verify that the Assignment and Routing Policy: diagram contains the `Approval.Participant1` value.



7. Ensure that the notification settings are enabled to send an email to Assignees when the task is assigned to the user.
   a. Click the Notification tab.
   b. In the Assign row, click the Edit icon (pencil) in the Notification Header column to examine the text used for the email subject.
8. Save your work and close the `ManualApproval.task` window.

Practice 9-4: Implementing Manual Approval in the BPEL Process

Overview

In this practice, you create and configure a BPEL process that receives incoming order data. It invokes the Manual Approval human task that you just completed to seek approval. The process then returns the result of that manual approval to the calling client. In this practice, you also generate an ADF Task Form project for the human task to display the task parameter information in the worklist application.

Assumptions

This practice assumes that you have completed Practice 9-1 successfully.

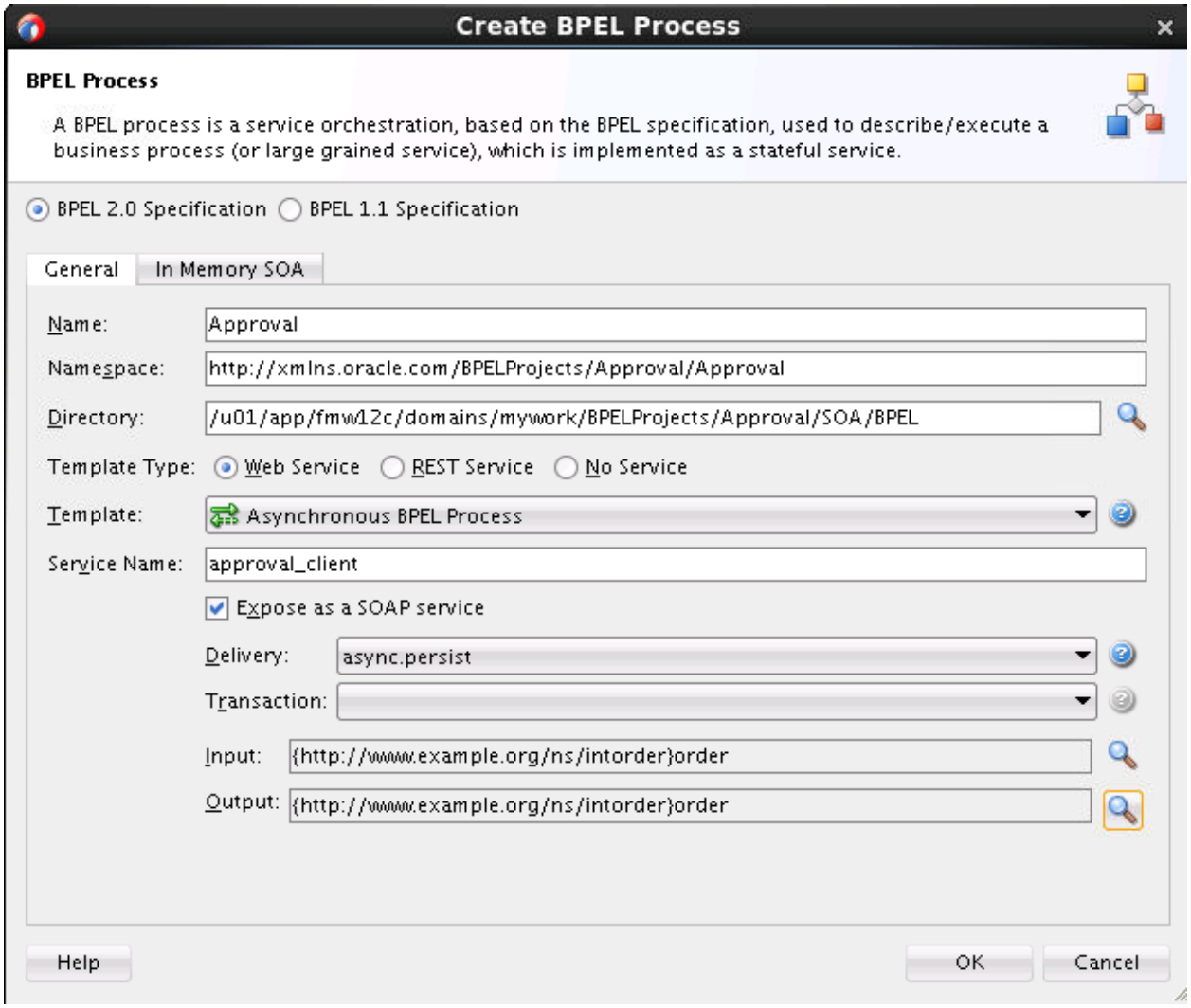
Tasks

Creating and Configuring a BPEL Process

In this section, you create a BPEL process and configure its activities.

1. Create and configure a BPEL service component.
   a. Add a BPEL process to the Components swimlane in the `composite.xml` editor.
   b. Name the process `Approval`.
   c. Verify that the Asynchronous Template is chosen (default).
   d. Leave Expose as SOAP serve selected.
   e. To define the input, use the Browse button to navigate to the `order` element of `internalorder.xsd`.
   f. To define the output, browse to the `order` element of `internalorder.xsd`.

g. Verify your work and click OK.



h. Save your work.

2. Create and configure an Assign activity to copy `inputVariable` to `outputVariable`.
   a. Right-click the BPEL Process Approval and select Edit.
   b. Add an Assign activity to the process between the Receive and Callback activities.
   c. Name the activity `Assign_Output`.
   d. Right-click the new activity and select Edit.
   e. Map the `inputVariable` element `payload` to its corresponding element in `outputVariable`.
   f. Verify and save your work.
3. Create and configure a Human Task activity in the business process.

   Note: This task is associated with the human task component that you created earlier.
   a. Add a Human Task (from the SOA Component section of the Component Palette) to the process immediately following `Assign_Output`.
   b. Right-click the activity and select Edit.

The Edit Human Task window opens.

c. Select the Task Definition `ManualApproval` from the drop-down menu.

Additional configuration fields are exposed.

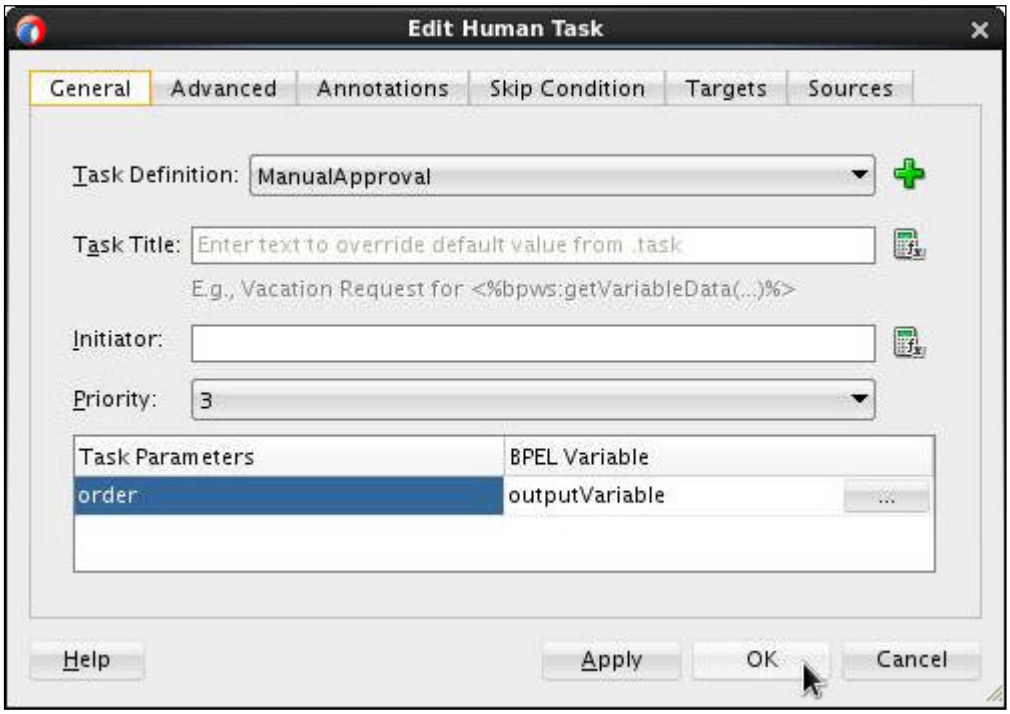
d. Click the Ellipsis (…) icon in the BPEL Variable field for the Task Parameter called `order`.

The Task Parameter window opens.

e. Select the Variables > **output**Variable > payload > `order` element.

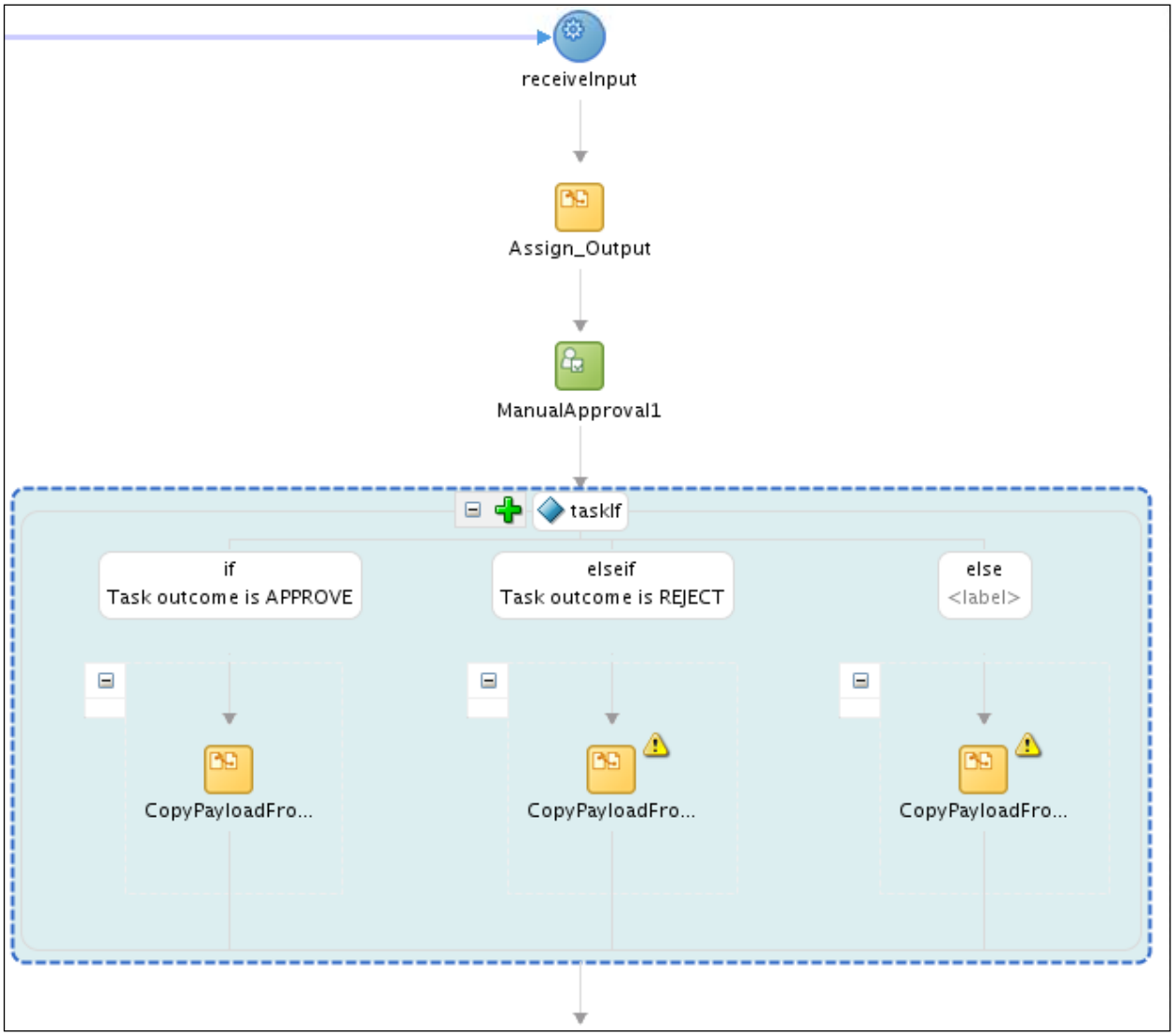
f. Click OK to close the Task Parameter window.

g. Verify your work and click OK.

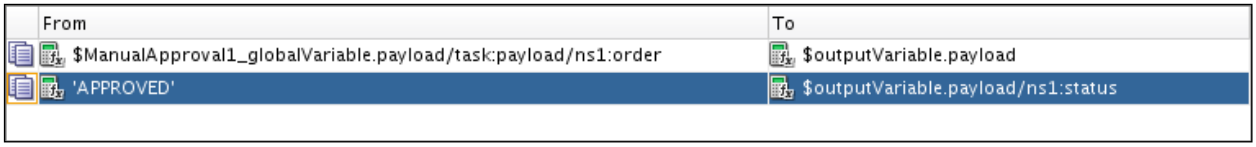


An If activity is added to the process to process the possible outcomes of the manual approval.

Note: These outcomes are part of the configuration of the human task component.



4. Configure the `taskIf` branches to process the various possible task outcomes.
   a. Expand the `taskIf` that appears after the `ManualApproval1` Human Task activity.
   b. Edit the CopyPayloadFromTask Assign activity in the APPROVE branch of the If.
   c. Use the Expression Builder icon to insert the word `'APPROVED'` into the `status` node of `outputVariable`.

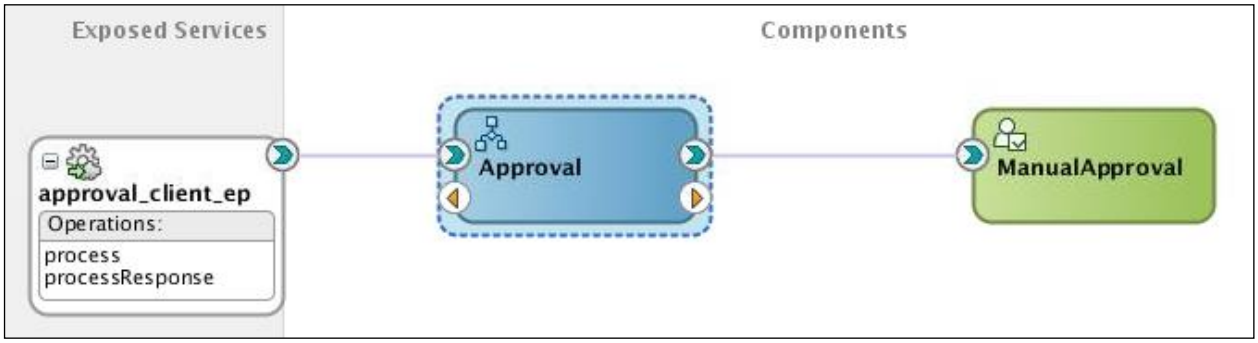


d. Repeat steps b and c to insert the word `'REJECTED'` into the `status` node of the `elseif` REJECT branch of the If.

e. Repeat steps b and c to insert the word `'FAILED'` into the `status` node of the `else` branch of the If.

f. Save your work.

g. Click the Approval tab and verify that a wire is created between the BPEL process `Approval` and the human task `ManualApproval`.



Practice 9-5: Generating the ADF Task Form for the Human Task

Overview

In this section, you use the human task to generate a default ADF Task Form Project, which contains JSF code to display the task parameter information in the worklist application. The worklist application displays the generated Task Form that is viewed by the task assignee to approve or reject the orders that are sent for manual approval.

Note: Because generation of the ADF Task Form Project is CPU-intensive and memory-intensive, you begin this section by shutting down Enterprise Manager and any other unnecessary applications.

Assumptions

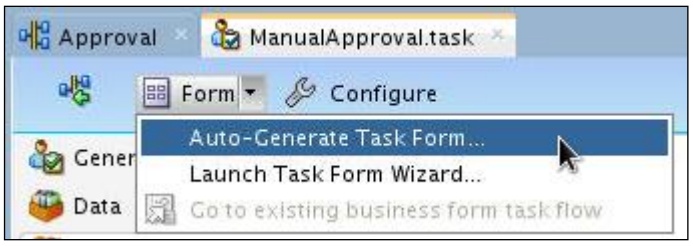
This practice assumes that you have completed all work through Practice 9-4 successfully.

Tasks

1. Patch the SOAINFRA database. A new column was added in 12.2 to the Workflow table and this change was not included in the Integrate Weblogic Server database. You need to patch the database so the Approval application will deploy.
2. In JDeveloper open the file: `/home/oracle/labs/scripts/SQL/SOAInfra.sql`.
3. Click the Run Script Button. Select the SOA Connection. Do not select SOA (Basics). You will see the message that the SOAINFRA.WSTASKMETADATA table is altered.
4. Shut down all unused applications, including your web browser, the Integrated WebLogic Server, and any other applications currently open, except JDeveloper. The next process: form generation, consumes a lot of CPU processing power.
5.
6. Generate the Human Task Form Project.
   a. In the composite overview window, right-click the icon for the human task: `ManualApproval` and select Edit.

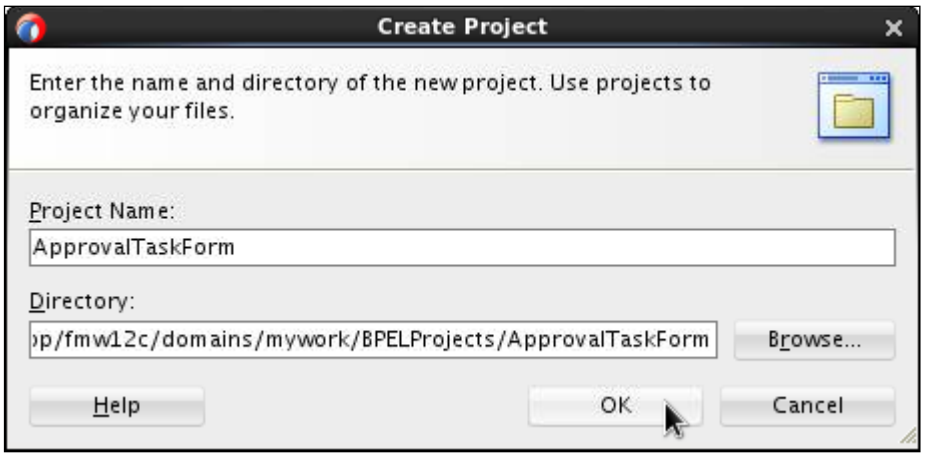
      The `ManualApproval.task` window opens.

   b. From the Form drop-down menu, select Auto-Generate Task Form.



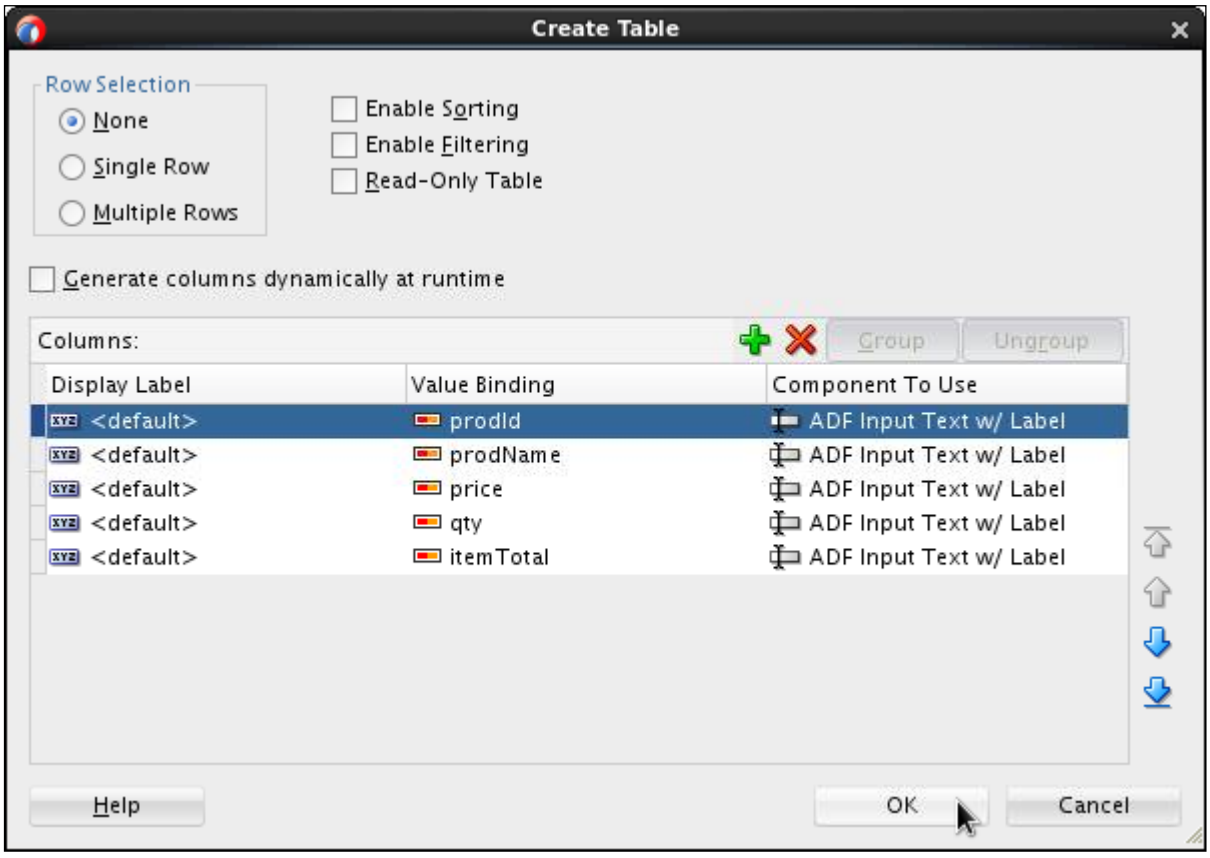
      The Create Project window opens.

c. Name the project `ApprovalTaskForm` and click OK.



Note: JDeveloper starts to consume 100% of the CPU at this time to begin the generation process.

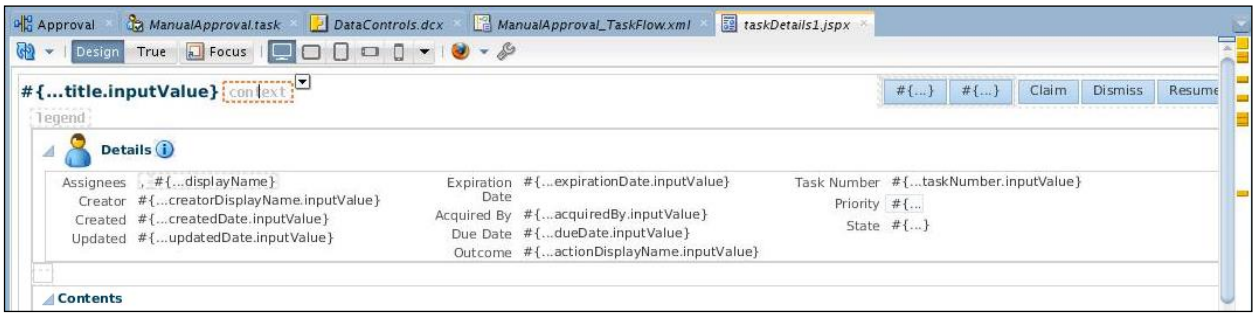
The Create Table window is displayed.



d. To accept the default settings, click OK.

Note: JDeveloper again consumes 100% of the CPU. This can take quite a few minutes, and has some moments when it seems to be “hanging”, but it ***is*** working, so please be patient. When the form is complete, the `ManualApproval_TaskDetails.jspx` window displays the generated task form in the designer.

Note: On rare occasions, the generation process hangs and does not complete. In this case, ask your instructor for guidance. You will want to stop JDeveloper and delete the ApprovalTaskForm folder under the BPELProjects folder, restart JDeveloper and retry step 2.



e. Save your work.

f. Close the `taskDetails1.jspx`, `ManualApproval_TaskFlow.xml` and `DataControls.dcx` windows.

7. Restart the Integrated WebLogic Server from the JDeveloper *Run* menu.

Practice 9-6: Deploying and Testing the Application

Overview

In this practice, you deploy the `Approval` and `ApprovalTaskForm` applications. You then test the applications, including the manual approval of an order and the sending and receipt of email to the approver.

Assumptions

This practice assumes that you have completed all work up to this point successfully.

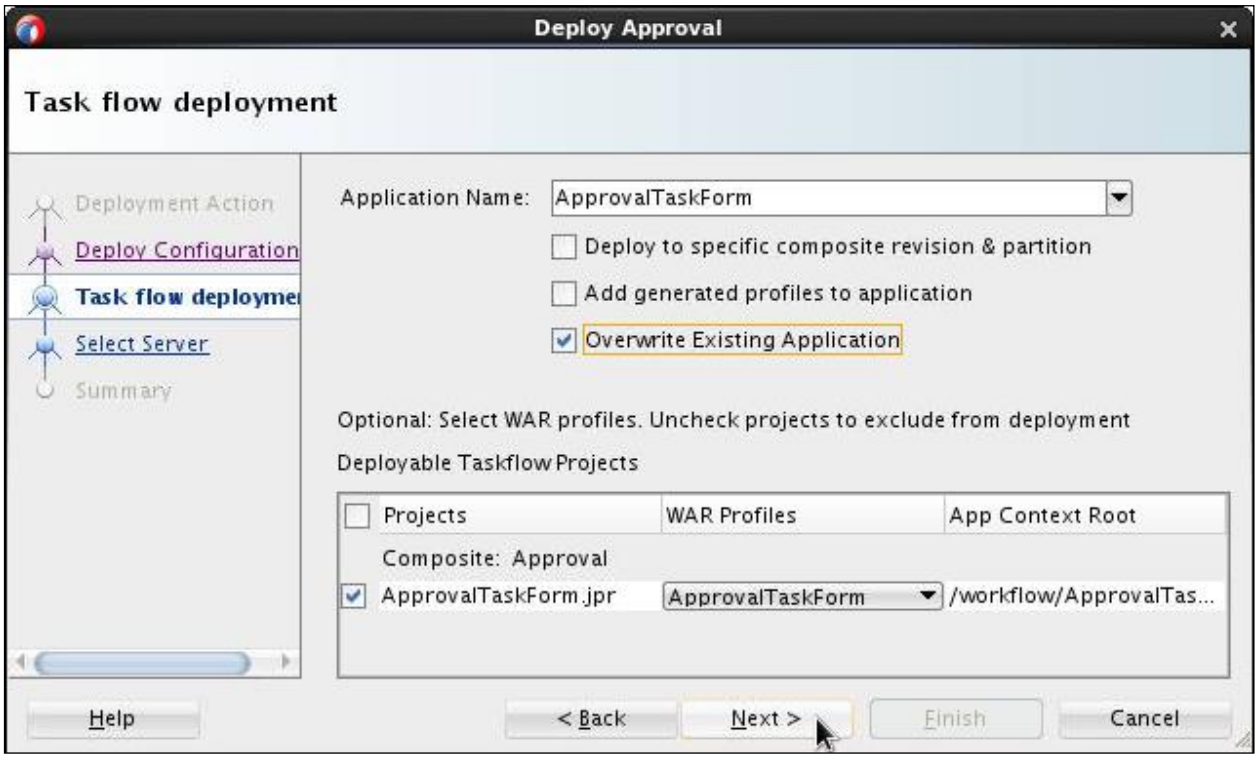
Tasks

Deploying the `Approval` Composite Application

In this section, you deploy the `Approval` and `ApprovalTaskForm` applications. Note that this deployment sequence differs from what you have seen so far in this course.

1. Deploy the `Approval` application to IntegratedWebLogicServer.
   a. In JDeveloper, right-click the application `Approval` and select Deploy > Approval.
   b. Select “Deploy to Application Server.”
   c. Select the “Overwrite any existing composites with the same revision ID” check box. The Task Flow Deployment screen appears.
   d. Select the following options:
      - Application Name: `ApprovalTaskForm`
      - Overwrite Existing Application check box
      - Deployable Taskflow Projects: ApprovalTaskForm.jpr

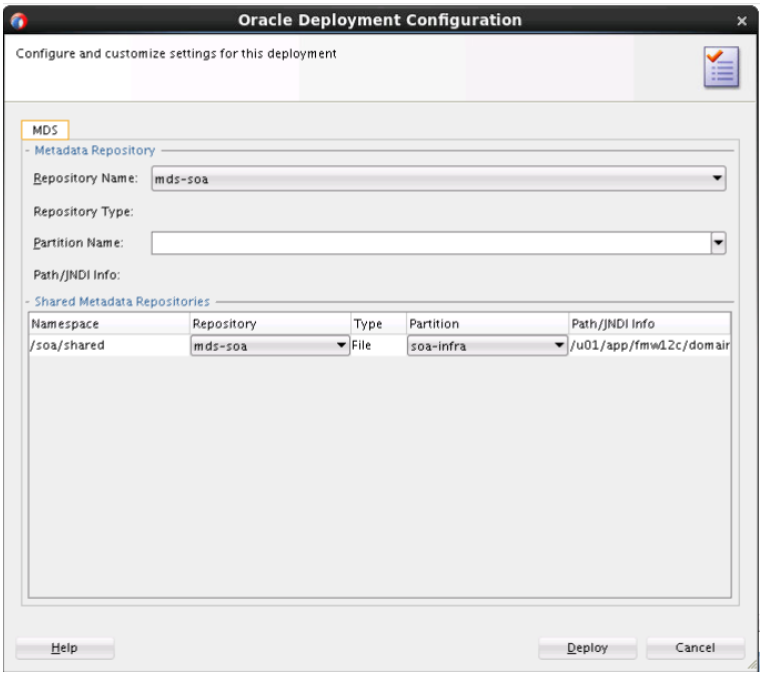
e. Verify your settings and click Next.



f. Select the application server IntegratedWebLogicServer.

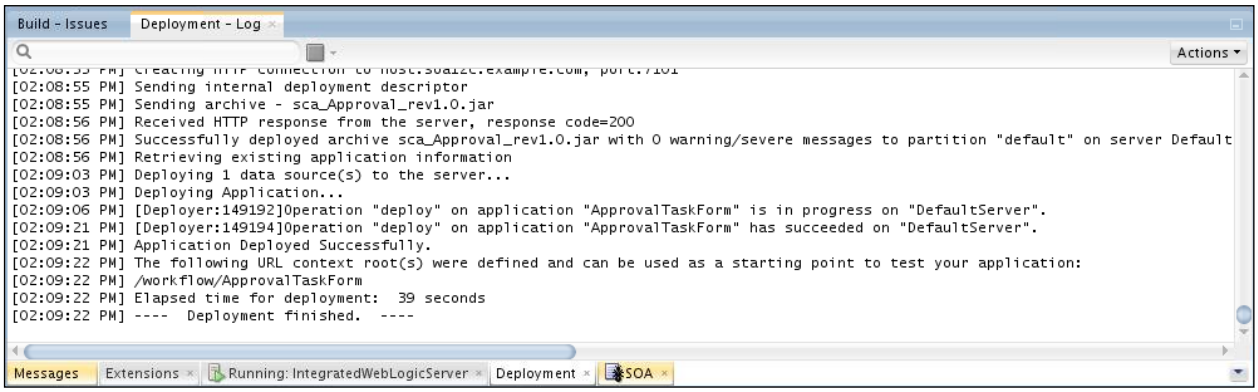
g. Click Finish.

The Oracle Deployment Configuration window appears.

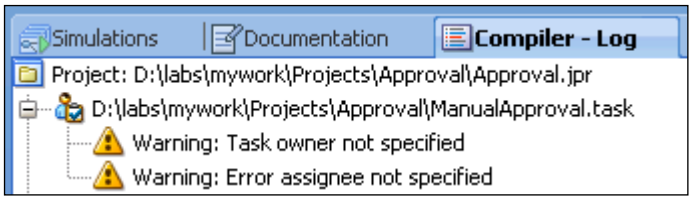


h. Click Deploy.

i. In the Deployment – Log window, verify that deployment is successful.



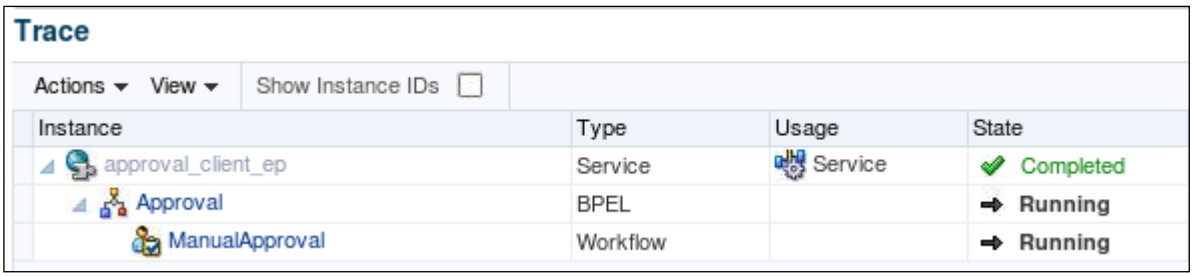
Note: You may ignore the following compiler warnings:



Testing the Composite Application

2. Initiate the test.
   a. Open Oracle Enterprise Manager (http://localhost:7101/em).
   b. In the Target Navigation pane, expand the SOA right-click soa-infra and select Home > Deployed Composites.
   c. Click the “Approval [1.0]” link.
   d. On the “Approval [1.0]” home page, click Test.
   e. Use the Browse button to replace the sample XML data with the contents of the file `/home/oracle/labs/files/xml_in/approval_input.xml`.
   f. Click Test Web Service.

      The Response tab is displayed.
   g. On the Response tab, click the Launch Flow Trace button.
3. On the Flow Trace page, examine the Trace tree that is shown in the following image and answer the questions that follow the image:



Note: If the process Flow Trace tree does not resemble the preceding image, wait a few seconds and refresh the page before you continue. You may need to refresh the page a few times.

Using the Flow Trace tree, answer the following questions:

a. Has the entire process completed? Explain your answer.

b. In this case, what does the last row in the Trace tree that contains the `ManualApproval` Human Workflow component indicate?

Q	Answers to Step 3.
a.	No, the process is still running. If you look in the Trace tree, you can see that the rows containing Approval and ManualApproval indicate their state as Running.
b.	The ManualApproval row indicates that the process is executing the Human Workflow component, which is not yet complete because it is still in the Running state.

4. To examine the `ManualApproval` component state, perform the following steps:

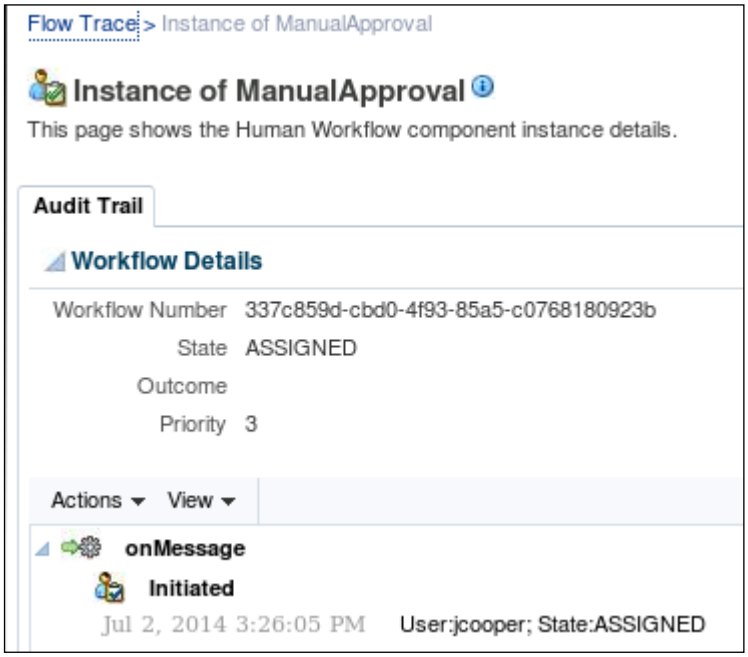
   a. In the web browser window with the Trace Flow page, click the ManualApproval Human Task component link.

   b. On the "Instance of ManualApproval" page, view the information and answer the following questions:

      1) What is the state of the human workflow task?

      2) Who is the task assignee?

Q	Answers to Step 4b).
a.	The task is in the ASSIGNED state. The following image shows the Instance of ManualApproval.
b.	The assignee is `jcooper` (refer to the following image).

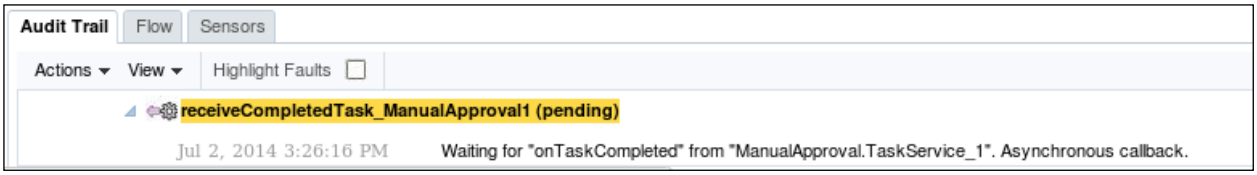


c. On the “Instance of ManualApproval” page, click the Flow Trace locator link at the top of the page.

5. To determine why the application is still running, click the Approval BPEL Component link in the web browser window with the Trace Flow page.
6. On the “Instance of Approval” page, view the process details and answer the following questions:
   a. What is the name of the last activity executed?
   b. Explain what the last activity execute state represents at this time.

   Note: *Do not* close the “Instance of Approval” page yet.

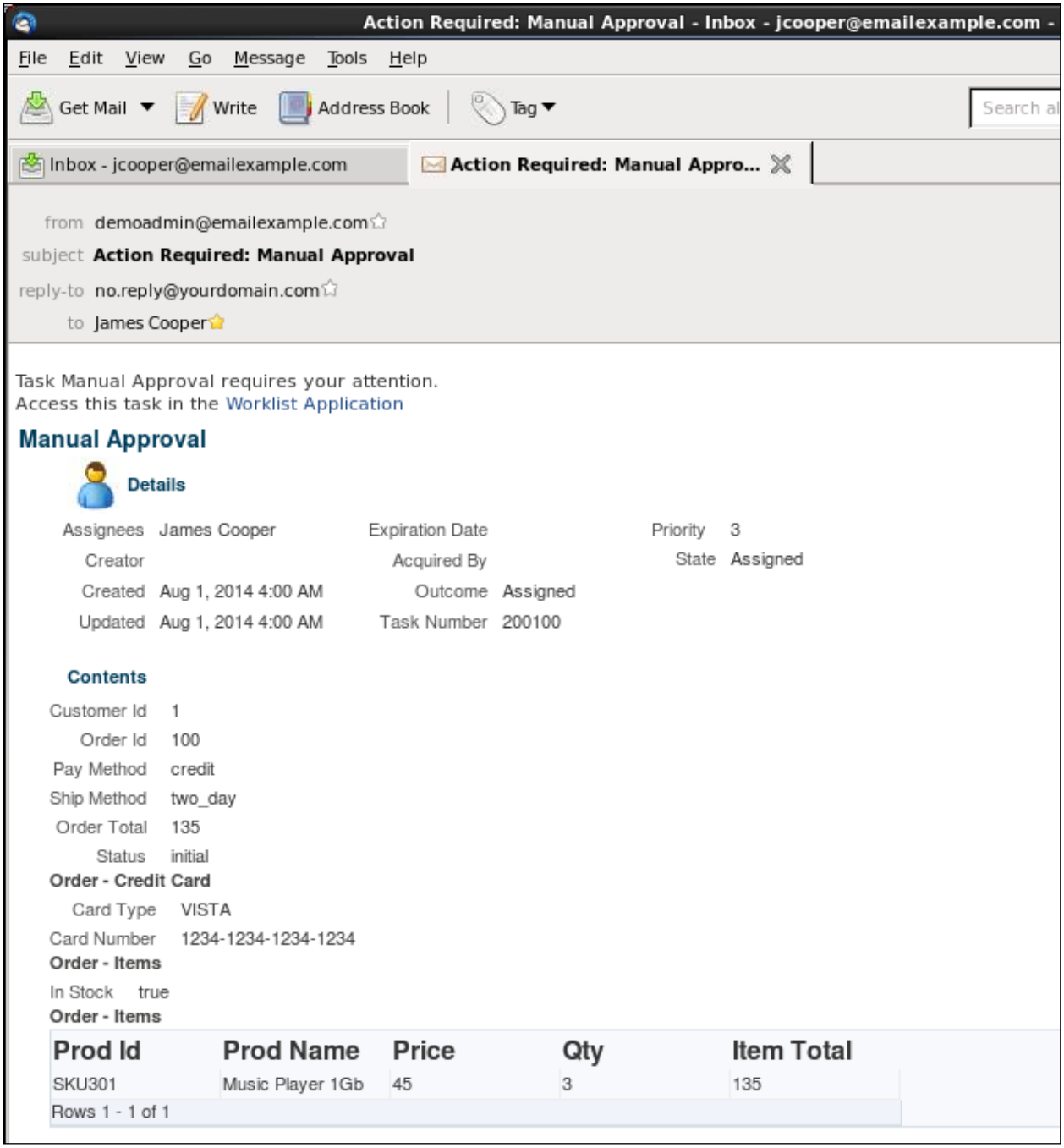
Q	Answers for Step 8.
a.	The name of the last activity executed is `receiveCompletedTask_ManualApproval1`.
b.	The `receiveCompletedTask_ManualApproval1` is in a *pending* state. The activity is waiting for an asynchronous callback message from the human workflow task that indicates that it has been completed. The implications are that the human workflow process is not complete.



Viewing the Email Messages and the Worklist Application

7. View the email message that is sent to the task assignee `jcooper`.
   a. Right-click the Thunderbird desktop icon on your desktop and select Open.
   b. In the Thunderbird window, in the All Folders pane, click the `jcooper@emailexample.com` account name.
   c. Click Get Mail on the Thunderbird toolbar.
      An email appears in the Inbox.

d. In the Inbox, verify the presence of an email message and click the entry to see the message body.



8. To open the worklist application and log in as `jcooper` to make changes to the shipping method and approve the order, perform the following steps:

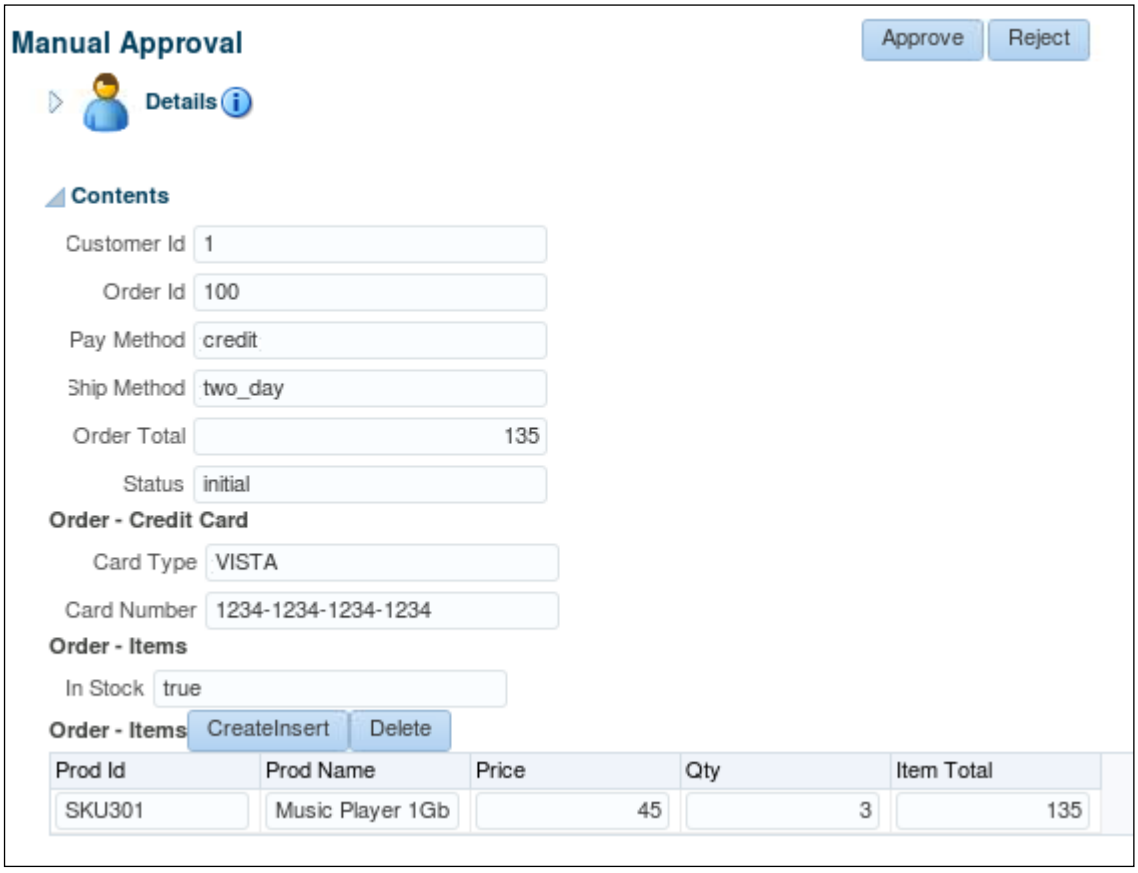
   a. To access the worklist application in a Firefox web browser window, enter the URL http://soa12c.example.com:7101/integration/worklistapp.

   Tips:

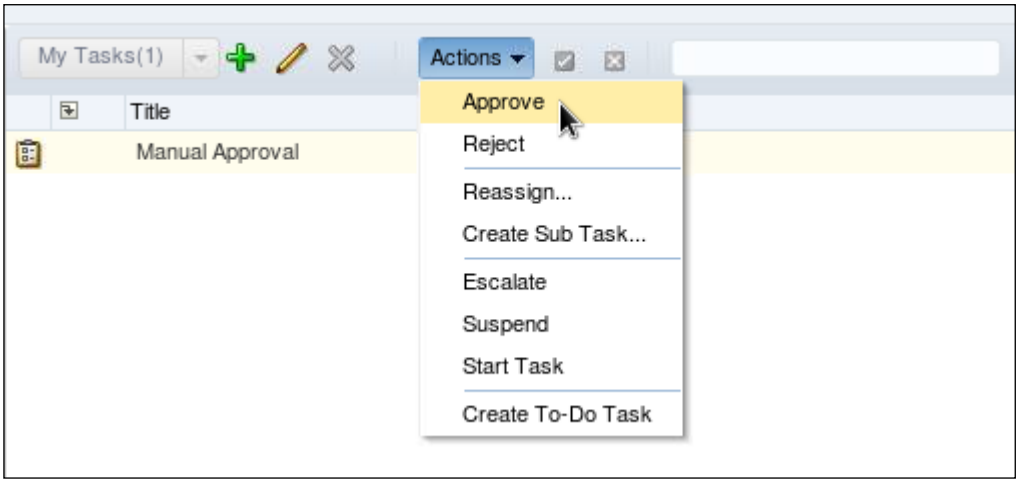
   - **Do *not* substitute "localhost" for the fully qualified host name in the URL.**
   - **Use only the Firefox browser.**

b. On the Oracle BPM Worklist application page, enter `jcooper` as the Username and `welcome1` as the Password, and click Login.

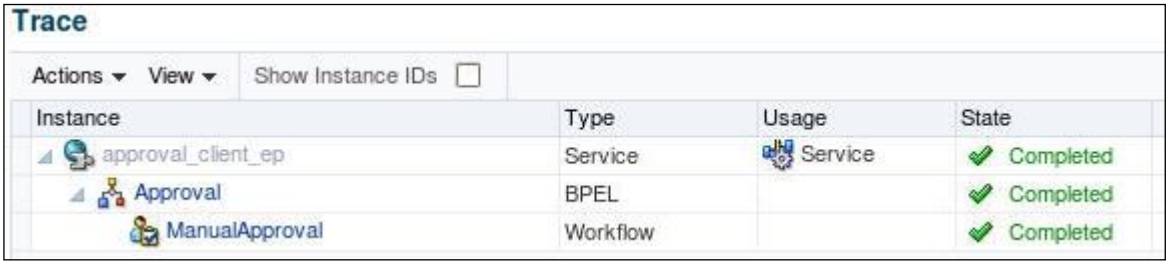
c. On the Worklist home page, the Inbox is selected and the task with the title “Manual Approval” should be present in the top-right pane on the page. Click the “Manual Approval” task entry to display the order details.



9. Click Approve (on either the task form as shown in the preceding screenshot or the toolbar shown as follows) to approve the order and allow the process to complete.



10. Return to the Flow Trace and click Approval. Examine the steps of the business process and verify that they completed successfully.



11. If you want, and if time permits, run another test and, in the worklist application, reject the order.

Practice 9-7: Sending Order Status Email Notification to a Customer [Optional]

Overview

In this practice, you first add a Database adapter to the composite application. You then modify the `Approval` BPEL Process. You create a Scope, and then add activities to the Scope to look up the customer email address from a database and send an email notification to the customer.

Assumptions

This practice assumes that you have completed Practice 9-5 successfully.

Tasks

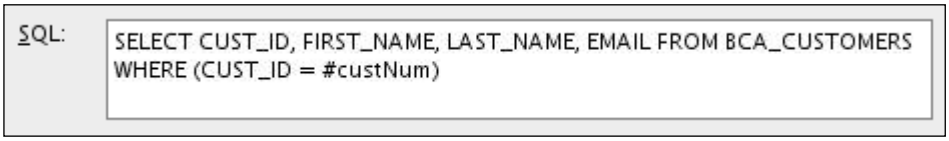
Creating and Configuring a Database Adapter

1. In the assembly model (`composite.xml`) for the project `Approval`, add a Database adapter to the External References swimlane.

   The Adapter Configuration wizard opens.
2. To configure the Database adapter, complete the following steps:

Step	Screen	Choices or Values
a.	Database Adapter Reference	Name: `CustomerLookup`
b.	Service Connection	Click Browse. Select SOA connection. Click Copy Connection. Verify the following settings: • Connection: **`SOA`** • User Name: **`soainfra`** • Driver: **`org.apache.derby.jdbc.clientDriver`** • Connect String: **`jdbc:derby//localhost:1527/soainfra`** • JNDI Name: **`eis/DB/SOA`**
c.	Operation Type	Select the “Perform an operation on a Table” option. Ensure that the Select check box is selected. Deselect other operations.
d.	Select Table	Click Import Tables.
e.	Import Tables	Set the BCA_% filter.

Step	Screen	Choices or Values
		Click Query.
f.	Import Tables	Copy the `BCA_CUSTOMERS` table from the Available list to the Selected list. Click OK.
g.	Select Table	Verify that the `BCA_CUSTOMERS` table is listed.
h.	Relationships	Click Next.
i.	Attribute Filtering	Select the firstName, lastName, and email attributes. Deselect the remaining options. **Note:** The custID (key attribute) cannot be deselected. Click Next.
j.	Define Selection Criteria	Next to the Parameters section, click Add.
k.	Parameter Name	Enter `custNum`. Click OK.
l.	Define Selection Criteria	Next to the SQL field, click Edit.
m.	Expression Builder	Click Add.
n.	Expression Builder	Form the condition `custId` EQUAL `custNum`, where: • First Argument Query Key: `custID` • Operator: `EQUAL` • Second Argument: Select the Parameter option, and select `custNum`. Click OK.
o.	Define Selection Criteria	Verify that your SQL statement is correct by comparing it with the following screenshot.
p.	Advanced Options	Click Next.
q.	JCA Endpoint Properties	Click Next.
r.	Finish	Click Finish.



3. Save your work.

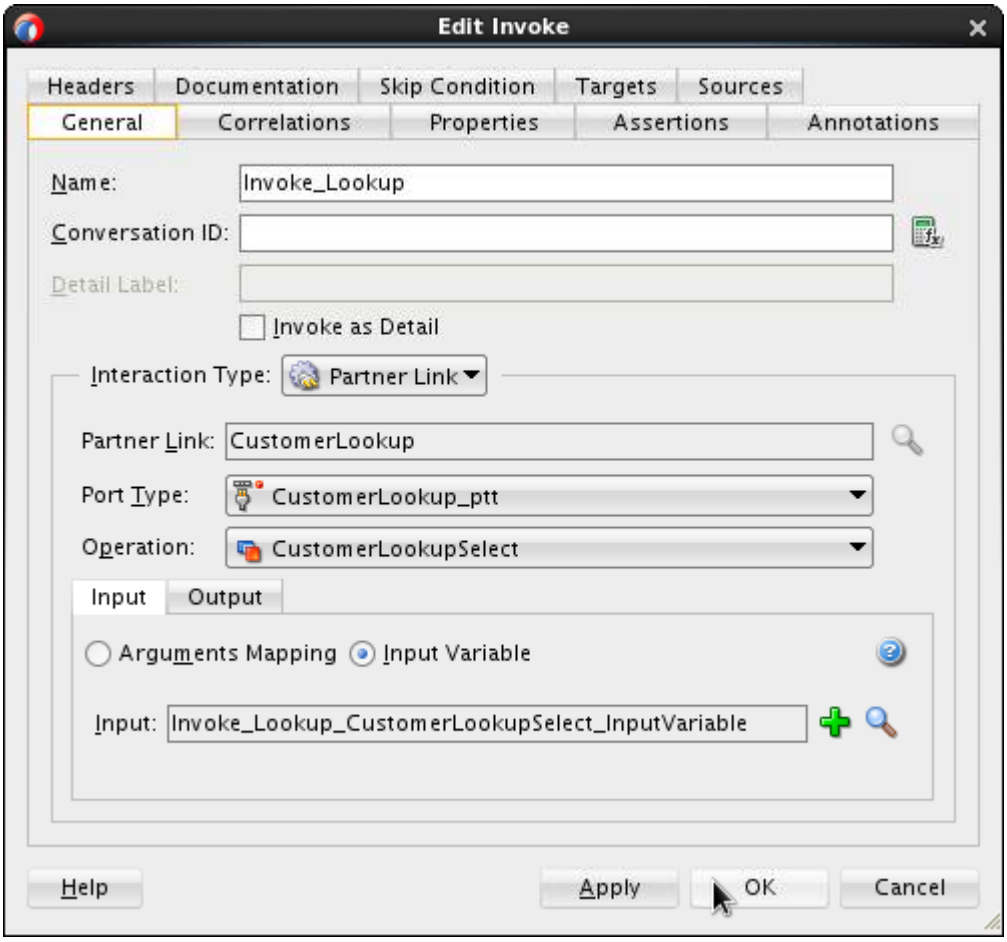
Modifying the Approval BPEL Process

4. In the composite overview window, wire the BPEL component Approval to CustomerLookup.

   This causes CustomerLookup to appear as a partner link in the BPEL editor.
5. Create and configure a Scope activity in Approval.
   a. If it is not already open for edit, right-click the Approval component and select Edit.
   b. For visual clarity, collapse the `taskIf` activity.
   c. Add a new Scope just above the `callbackClient` activity.
   d. Name the new Scope `Scope_Email`.
6. Create and configure an Invoke activity to look up the customer email address.
   a. Expand `Scope_Email`.
   b. Add an Invoke activity to `Scope_Email`.
   c. To specify the partner link, drag and drop Invoke to the partner link CustomerLookup.

      The Edit Invoke dialog box appears.
   d. Name the Invoke `Invoke_Lookup`.
   e. Create a new Input variable.
   f. Create a new Output variable.

g. Verify and save your work.

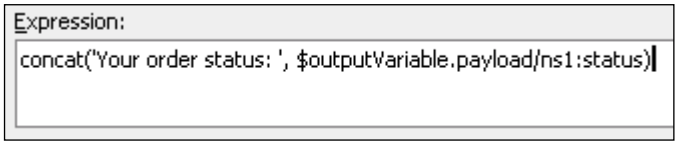


7. Create and configure an Assign activity.
   a. Add an Assign activity within `Scope_Email` just above `Invoke_Lookup`.
   b. Name the activity `Assign_Lookup`.
   c. Right-click the new activity and select Edit.
   d. Map the `inputVariable` element `customerId` to the `custNum` element of the `Invoke_Lookup… inputVariable`.
   e. Verify and save your work.

8. Create and configure an `Email` activity.
   a. Add an `Email` activity (Oracle Extensions) within `Scope_Email` just below `Invoke_Lookup`.
   b. Name the activity `Email_Customer`.
   c. Right-click the `Email` activity and select Edit.

d. To specify the To field, use Expression Builder to reference the `Invoke_Lookup` Output variable `email` field.

e. To specify the Subject field, use Expression Builder to construct the value `concat('Your order status: ',$outputVariable.payload/ns1:status).`

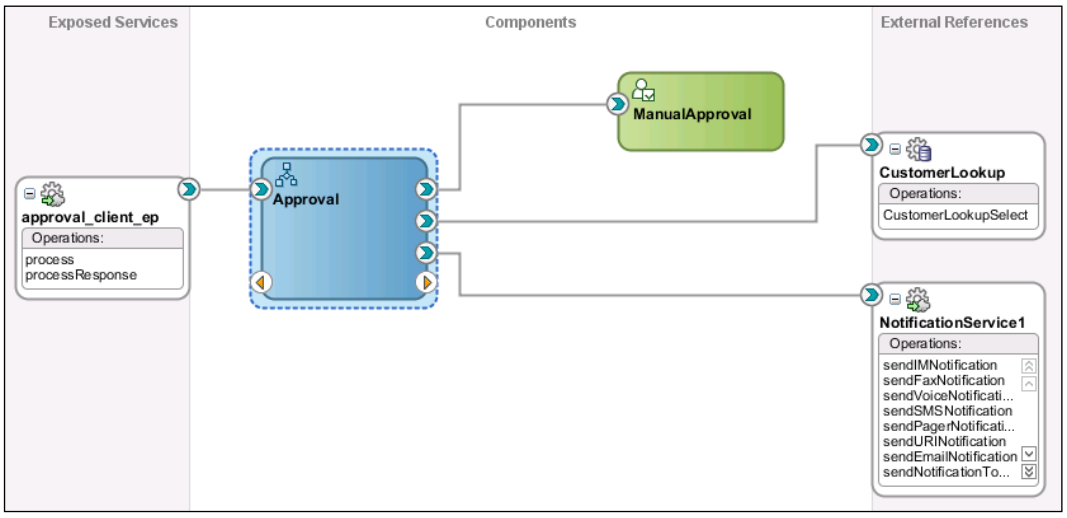


f. To specify the Body field, you can use a mix of text and XPath expressions. XPath expressions must be enclosed with **<%** and **%>** begin and end tags. The body of the email can be clear text or HTML. For this practice, it is recommended that you copy and paste the following example from the file `/home/oracle/labs/files/xml/email-body.xml`.

```
concat('Your order status is
',$outputVariable.payload/ns1:status,'. Thank you for shopping
with us. For more information contact customer support on 1-800-
MUSICFANS.')
```

g. Save your work and close the BPEL editor.

9. Verify your assembly model.



Practice 9-8: Redeploying and Testing the Application

Overview

In this practice, you redeploy the `Approval` and `ApprovalTaskForm` applications. You repeat the test of the applications, including the manual approval of an order and the sending and receipt of email to **both the approver and the end customer**.

Assumptions

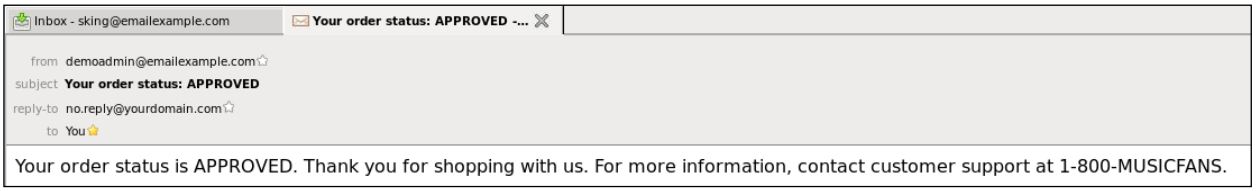
This practice assumes that you have completed all work up to this point successfully.

Tasks

Undeploying the Application

1. Complete all steps in Practice 9-6 to redeploy and test the application.
2. Return to the email program. Select the user `sking` and click the toolbar button “Get Mail.”

   Note: It may take a few moments for the email to arrive in the Inbox.
3. Verify that the customer email was sent correctly.



4. When you have finished, you can close the various editors that are open within JDeveloper.

Practices for Lesson 10: Sharing Functionality in Oracle SOA Suite

Overview

Practices for Lesson 10: Overview

Practices Overview

In this practice, you first import a collection of templates into the SOA Design Time Repository. You create a project, and use a BPEL component template to accelerate development of the project. You then modify the project. These modifications include the creation and calling of a BPEL in-line subprocess. You then test the finished project.

Practice 10-1 Using a Component Template

Overview

In this section, you import the contents of a previously exported Design Time Repository. These contents are in a `.jar` file and they include several project and component templates. In this and subsequent practices, you use these templates to accelerate the building of larger projects.

Assumptions

This practice assumes the following:

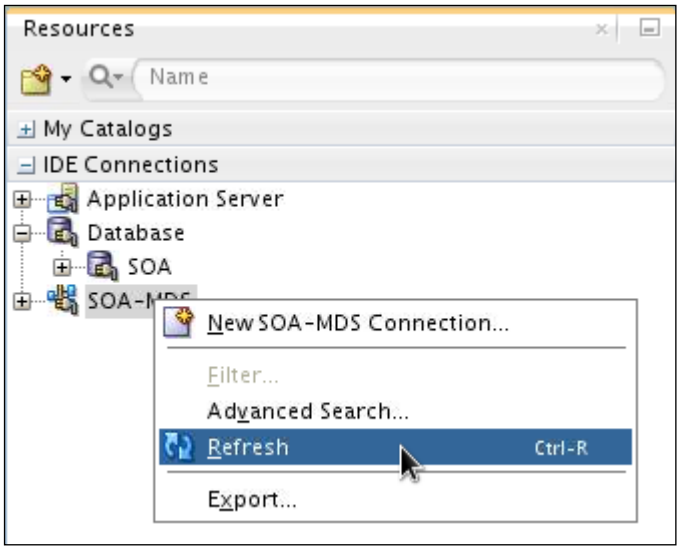
- You have completed Practice 5 for the lesson titled “Using JMS and JDBC Adapters” by using *exact* project and component names as specified in that practice, and that the `ValidateCreditCard` application is still deployed to the application server.
- You have completed Practice 9 for the lesson titled “Implementing Human Workflow and Notifications” by using *exact* project and component names as specified in that practice, and that the `Approval` application is still deployed to the application server.

If your environment does not meet these assumptions, deploy the working solutions for the missing practices as described in `/home/oracle/labs/soln/restoringSolutions.pdf`.

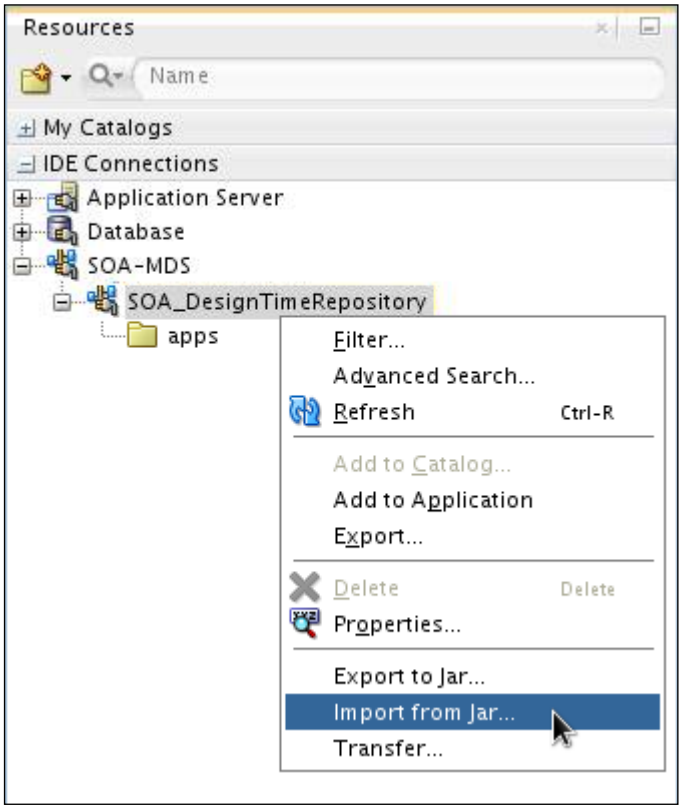
Tasks

Importing Templates to the SOA Design Time Repository

1. Import templates from the `.jar` file to the Design Time Repository.
   a. In JDeveloper, in the IDE Connections section of the Resources tab, right-click SOA-MDS and select Refresh.

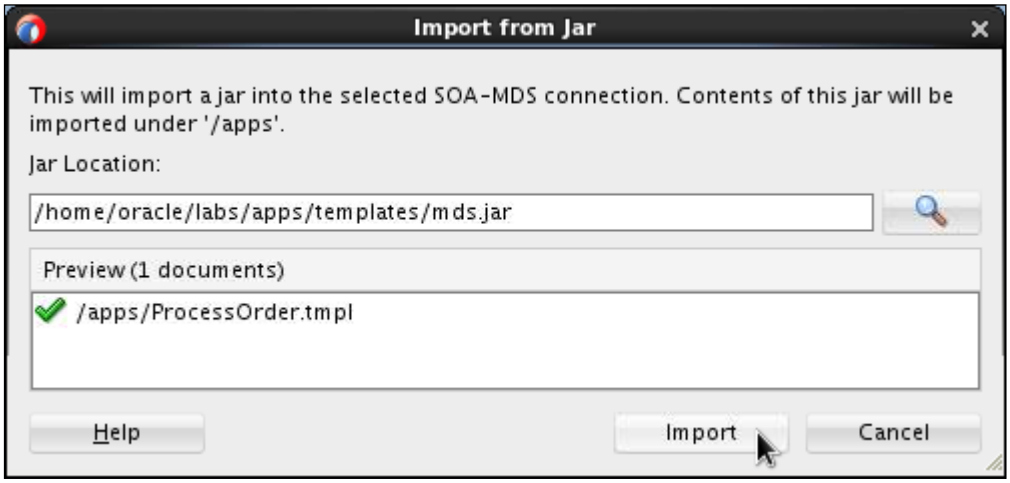


   b. Expand the SOA-MDS folder and right-click SOA_DesignTimeRepository.
   c. In the context menu, select Import From Jar.

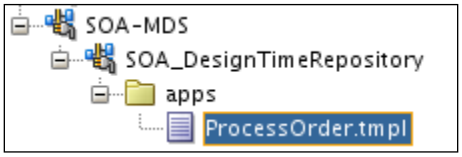


d. Navigate to the Jar location `/home/oracle/labs/apps/templates/mds.jar` and click Open.

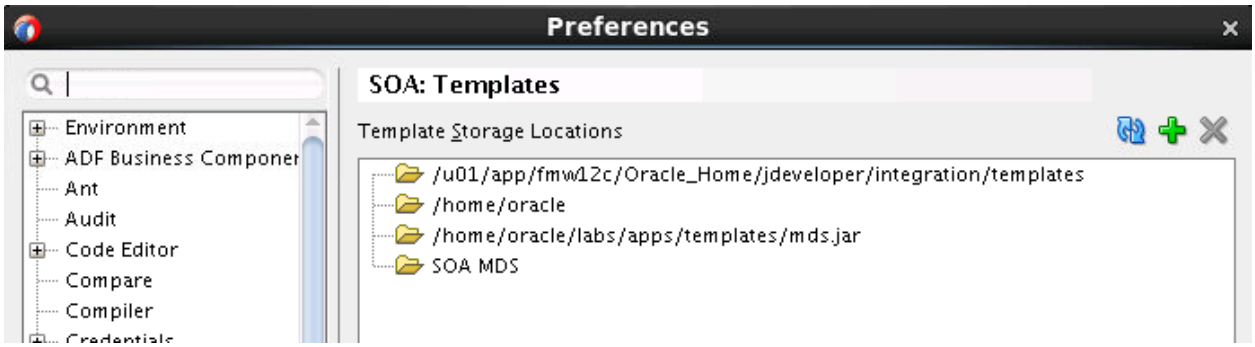
e. In the Import from Jar window, click Import.



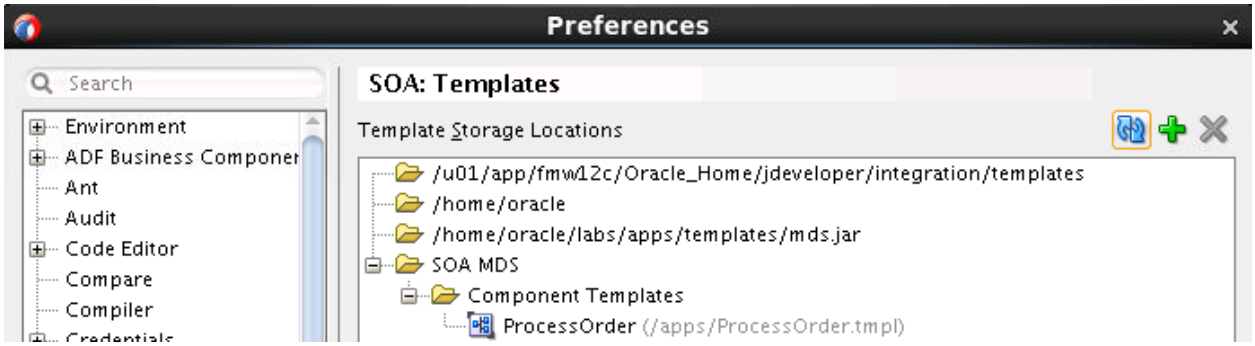
f. In the IDE Connections pane, expand the /apps folder and verify successful import of the specified template.



g. Select from the menu: Tools > Preferences > SOA > Templates. Notice the template does not show up. Yo need to make Jdeveloper see the template you imported.



h. Click the Refresh Templates Button  and the newly imported template appears. Click OK to close the window.



Creating a Project

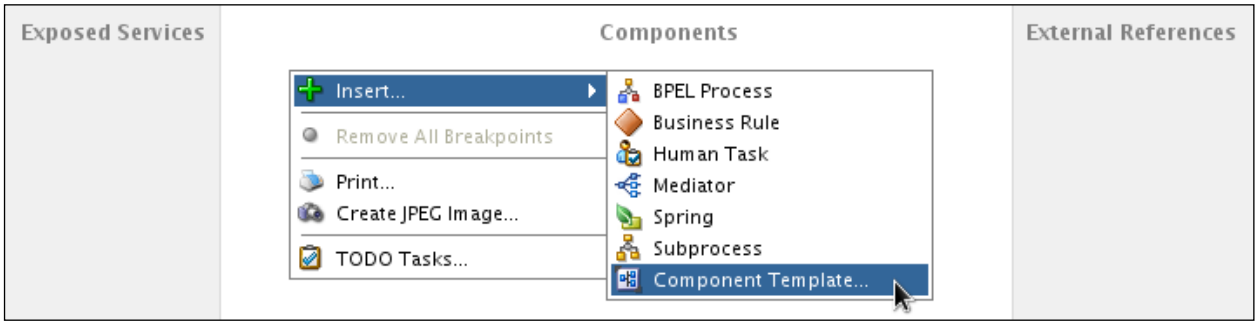
2. In the BPELProjects application, create a new empty SOA project named `OrderFlow`.

   The `composite.xml` file is created and the overview window is opened for editing.

Adding a BPEL Component from a Template

3. Add a BPEL Component from a template.

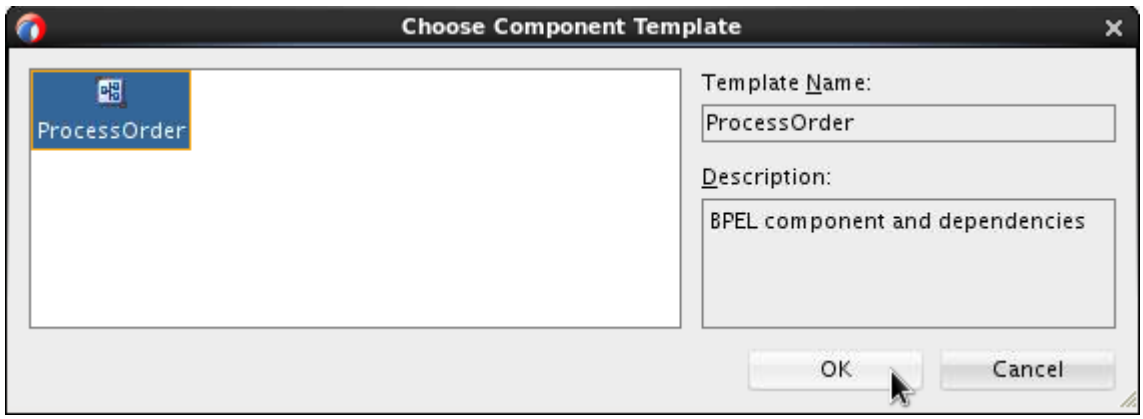
   a. In the component overview window, right-click in the Components swimlane and select Insert > Component Template.



   The Choose Component Template dialog box appears.

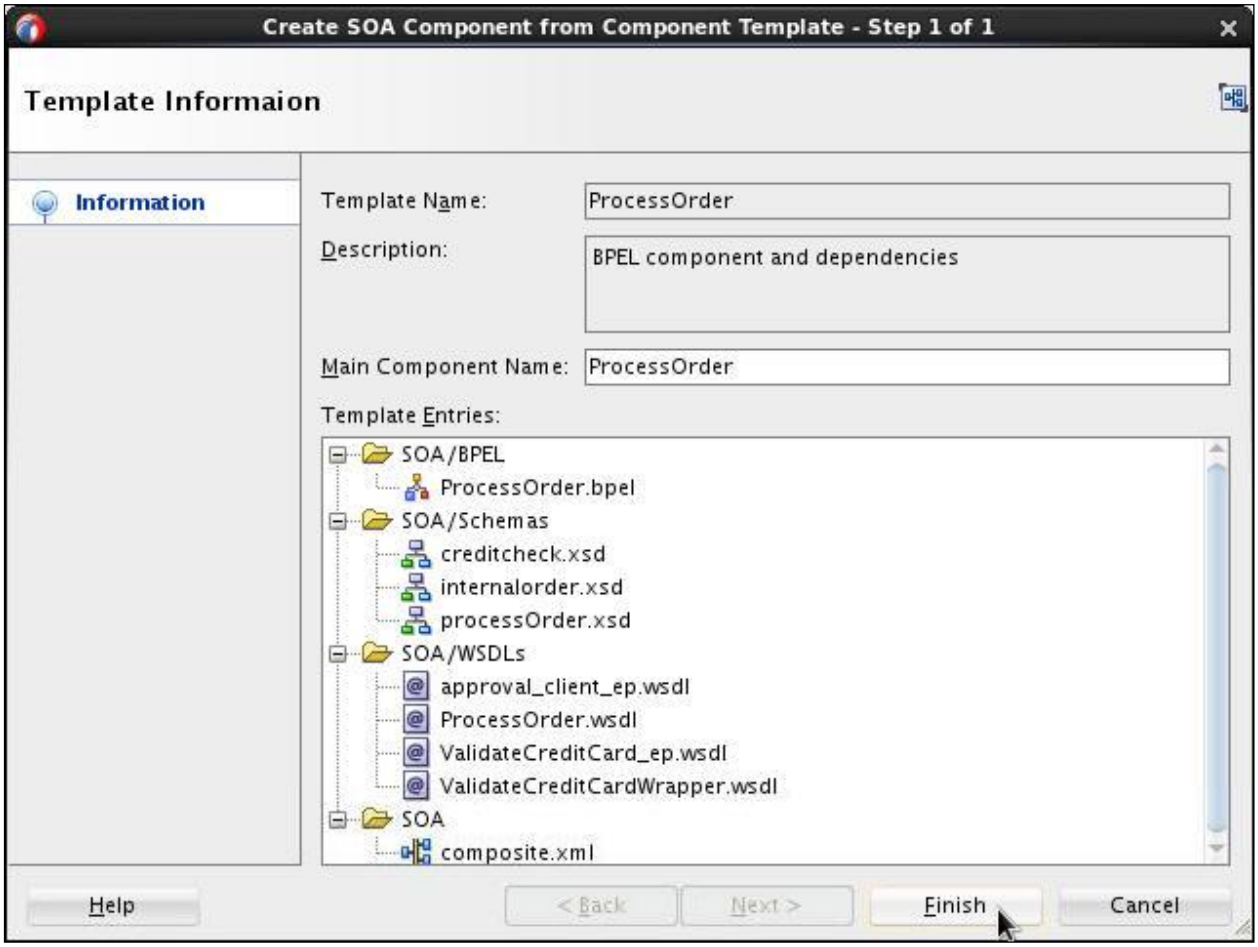
Note: If the Component Template option does not appear, stop the Integrated WebLogic Server. Close and reopen JDeveloper. Restart the Integrated WebLogic Server. You can resume your work while it is restarting.

b. Select ProcessOrder and click OK.



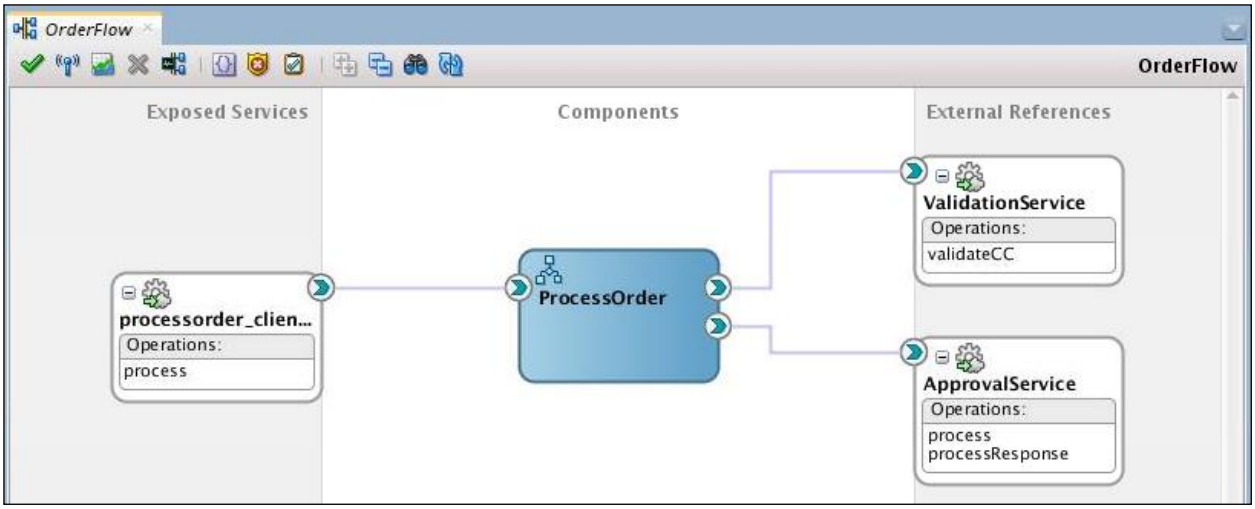
The Create SOA Component from Component Template dialog box appears.

c. Examine the template entries and click Finish.



The composite overview window is updated.

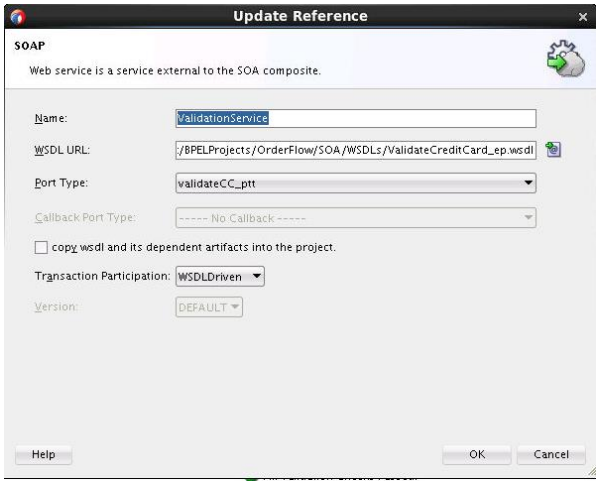
d. Verify your project and save your work.



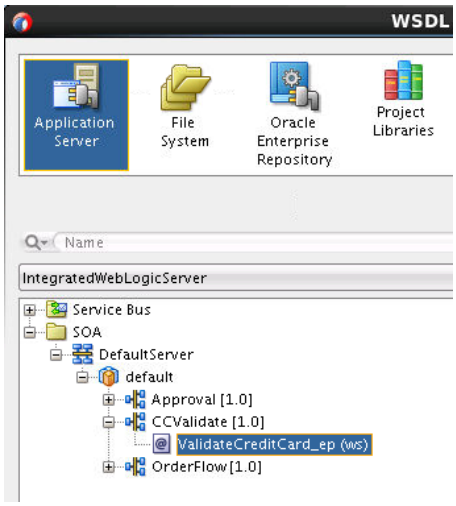
Note: Expect many error messages as you save. Recall that JDeveloper performs a validation check each time you save. These errors indicate that the project is not fully configured, which is expected at this point.

e. The ValidationService is from a different server and the WSDL must be changed for it to access the service deployed on your server. This is not unusual with templates. You will update the WSDL for the validation Service first, which will remove the old WSDL and the wiring to the BPEL process. You will rewire the BPEL process and update the Invoke activity so it sees the new, correct WSDL.

f. In the `composite.xml` diagram, right-click the ValidationService icon and select Edit.



g. Click the Find existing WSDLs button and locate the CCValidate service you deployed and select the endpoint: `ValidateCreditCard_ep(ws)`.

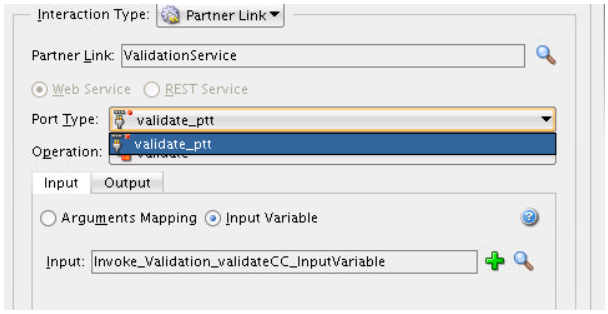


h. This will remove the wiring to the BPEL process. Click Yes twice when prompted to delete the unused WSDL files. Then, re-wire the BPEL process to the ValidationService.

i. Open the BPEL process ProcessOrder for editing.

j. Right-click the Invoke_Validation activity to edit it.

k. Select validate_ptt for the port type. Select both the Input and Output tabs to make sure the variables are set. Click OK and close the Edit Invoke dialog. Verify and save your work.



Creating and Configuring a Database Adapter

4. In the assembly model (`composite.xml`) for the `OrderFlow` project, add a Database adapter to the External References swimlane.

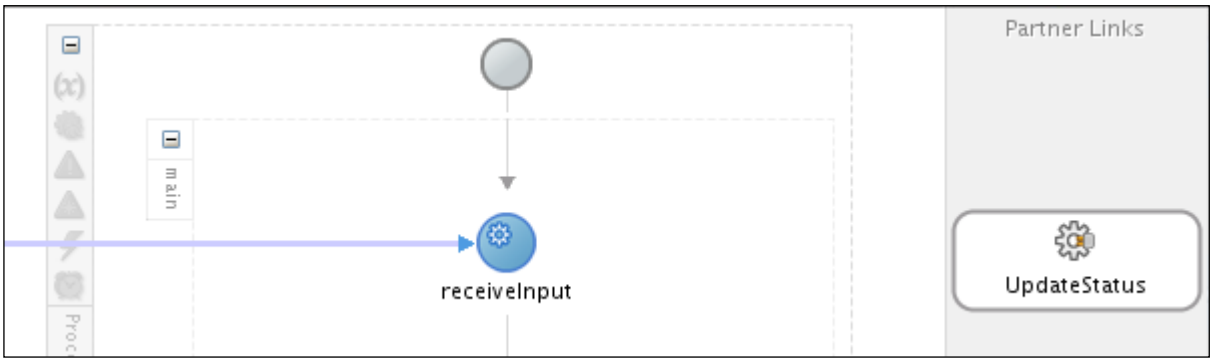
   The Adapter Configuration wizard opens.

5. To configure the Database adapter, complete the following steps:

Step	Screen	Choices or Values
a.	Database Adapter Reference	Name: `UpdateStatus`
b.	Service Connection	Verify the following settings: • Connection: **SOA** • User Name: **soainfra** • Driver: **org.apache.derby.jdbc.clientDriver** • Connect String: **jdbc:derby//localhost:1527/soainfra** • JNDI Name: **eis/DB/SOA**
c.	Operation Type	Select the “Perform an operation on a Table” option. Ensure that the Update Only check box is selected. Deselect the other operations.
d.	Select Table	Click Import Tables.
e.	Import Tables	Set the BCA_% filter. Click Query.
f.	Import Tables	Move the `BCA_ORDERS` table from the Available list to the Selected list. Click OK.
g.	Select Table	Verify that the `BCA_ORDERS` table is listed.
h.	Relationships	Click Next.
i.	Attribute Filtering	Deselect all attributes except `status`.
j.	Advanced Options	Click Next.
k.	JCA Endpoint Properties	Click Next.
l.	Finish	Click Finish.

6. In the Composite Editor, wire the ProcessOrder BPEL process to the UpdateStatus external reference.
7. Open the BPEL process for editing.

8. Verify that UpdateStatus is shown in the Partner Links column and save your work.

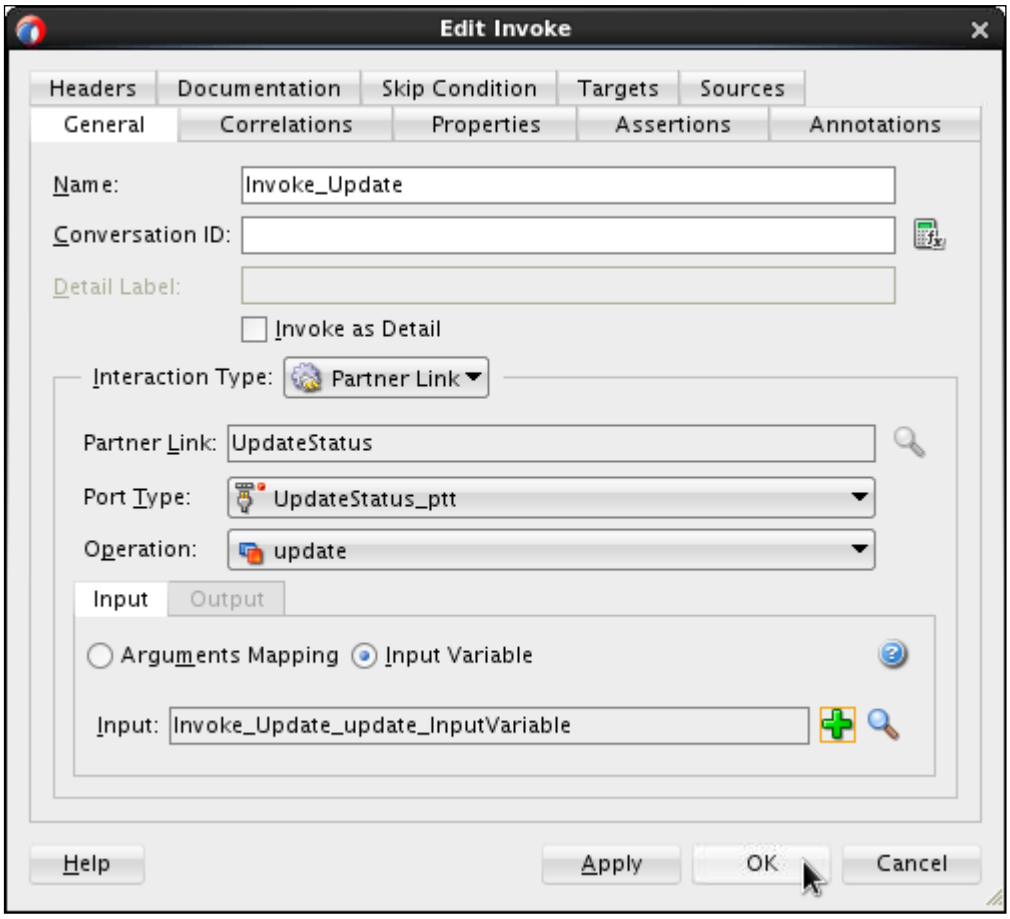


Modifying the BPEL Process

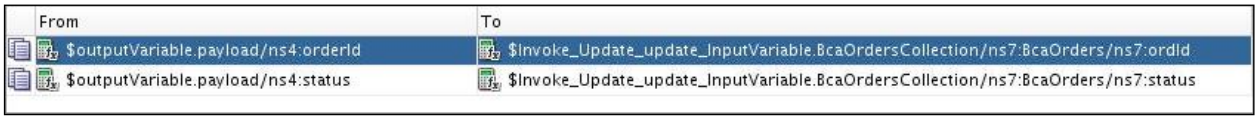
9. Add a new Scope activity to the BPEL process, just below the `GetValidation` activity.
10. Create and configure an Invoke activity.
    a. Add an Invoke activity to the new Scope.
    b. To specify the partner link, drag and drop the Invoke activity to the partner link UpdateStatus.

       The Edit Invoke dialog box appears.
    c. Name the new activity `Invoke_Update`.
    d. Create a new Input variable.

e. Verify and save your work.



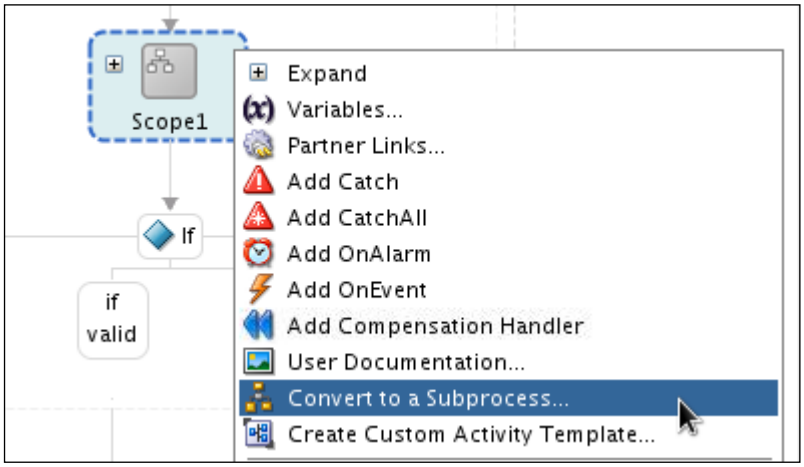
11. Create and configure an Assign activity.
    a. Add an Assign activity within the Scope activity, just above `Invoke_Update`.
    b. Name the activity `SetUpdate`.
    c. Right-click the new activity and select Edit.
    d. Map the `outputVariable` element `orderId` to the `ordId` element of `Invoke_Update_update_inputVariable`.
    e. Map the `outputVariable` element `status` to the `status` element of `Invoke_Update_update_inputVariable`.
    f. Verify and save your work.



Converting the Scope to a Subprocess

Updating the order status is something that will need to be done multiple times in this business process. For this reason, it makes sense to convert the update function (as defined in the scope activity) to a subprocess.

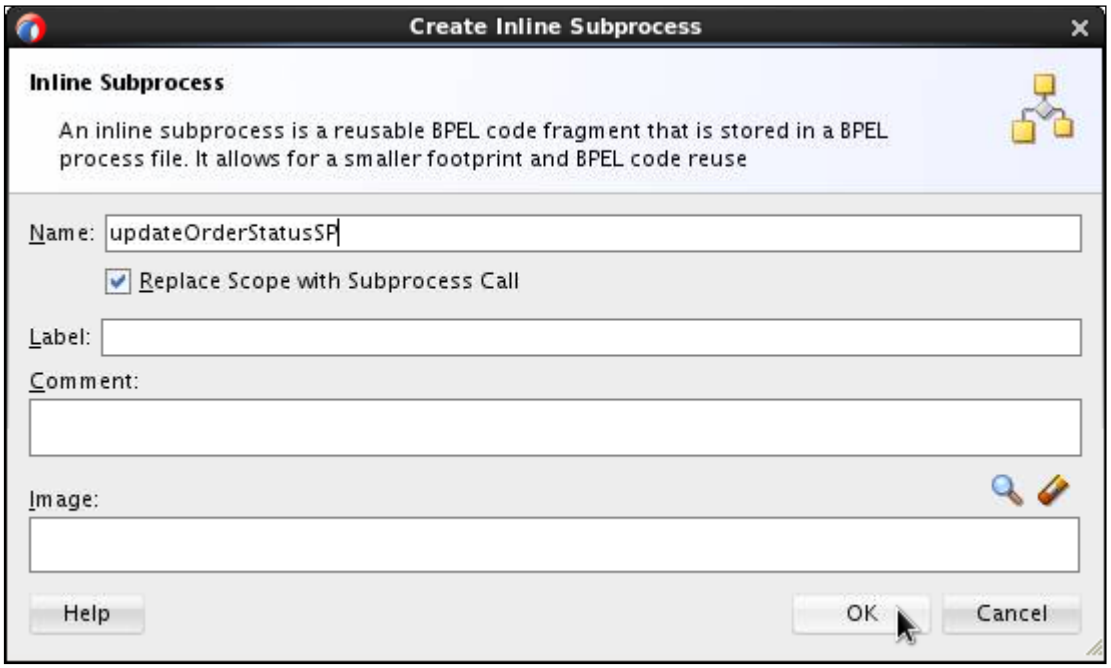
12. Convert the `Scope1` scope to a subprocess.
    a. Collapse the scope and right-click the scope icon.
    b. Select Convert to a Subprocess.



The Create Inline Subprocess dialog box appears.

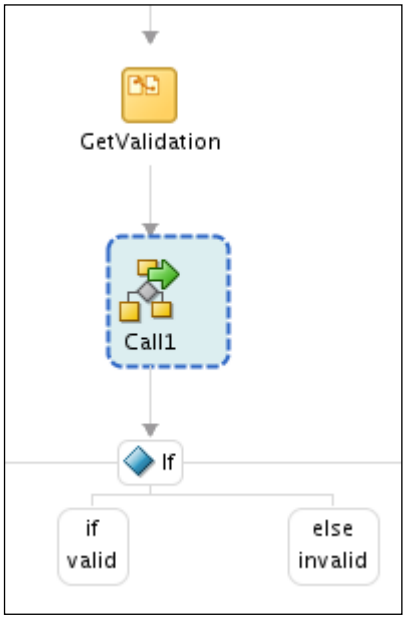
    c. Name the subprocess updateOrderStatusSP.

Leave the Replace Scope with Subprocess Call check box selected. This automatically replaces the current scope with a Call activity.



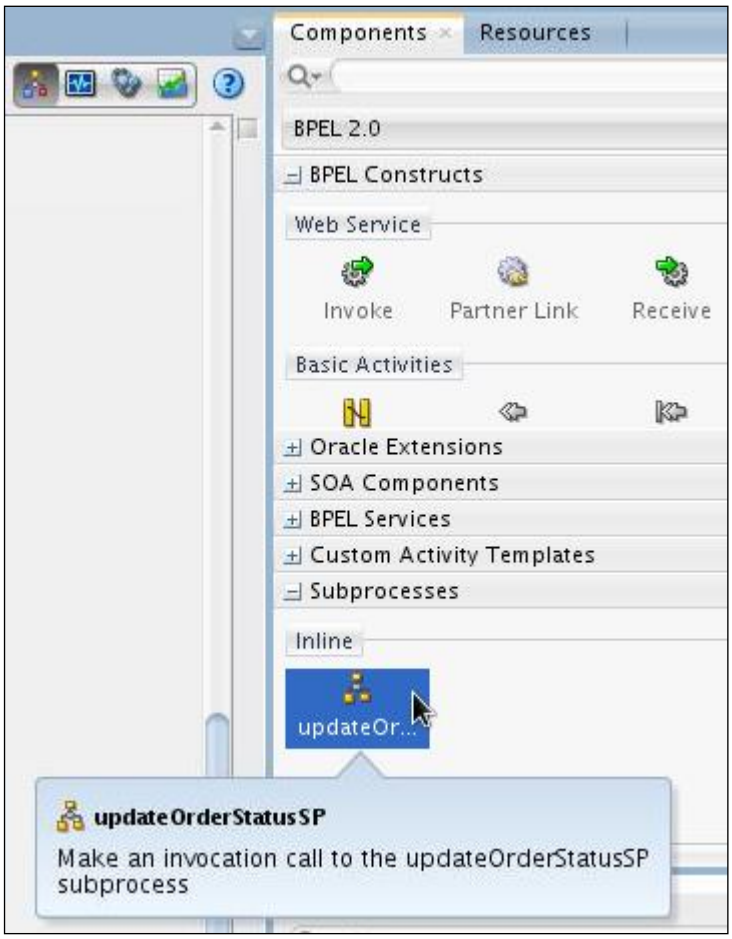
    d. Click OK.

A call activity is displayed in the process.

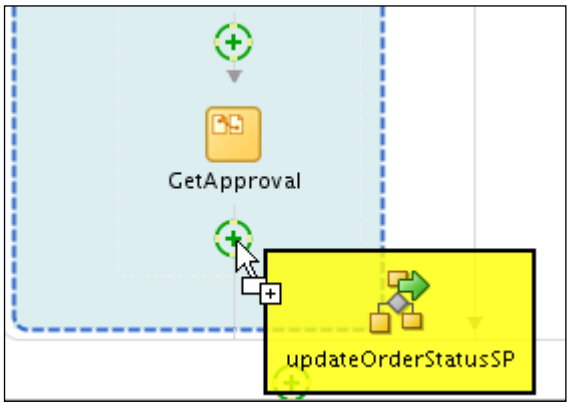


Note: The subprocess is displayed in the Component Window under Subprocesses. You can see and edit the definition of the subprocess when switching from the Main Process to the subprocess at the top of the BPEL editor. All partner links that are available in the Main Process are displayed, in case you want to add another Invoke. Changes to the subprocess are reflected in every call.

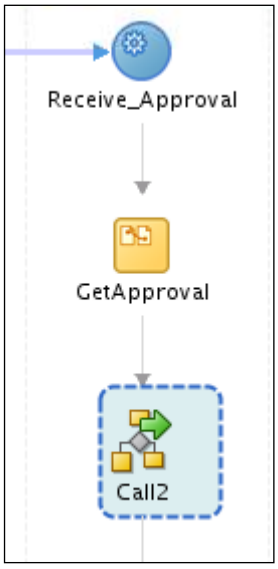
13. Add a call activity after ValidationService.
    a. Expand the Subprocesses section of the Component Palette.
       The updateOrderStatusSP subprocess is displayed.



    b. Drag the updateOrderStatusSP subprocess from the Component Palette to the BPEL process, within the If branch of the If activity, just below the GetApproval activity.



A new call activity is added to the process. That is all that needs to happen to invoke the subprocess and have the order status updated to whatever value is returned by the call to the approval process (APPROVED or REJECTED).



14. Save your work.

Practice 10-2: Deploying and Testing the `OrderFlow` Application

Overview

In this practice, you deploy and test the `OrderFlow` composite application.

Assumptions

This practice assumes that you have completed all work in Practice 10-1 successfully.

Tasks

Deploying the Composite Application

1. Deploy the `OrderFlow` application to IntegratedWebLogicServer.

   Remember to select the “Overwrite any existing composites with the same revision ID” check box.

Testing the Composite Application

In this section, you submit an order that will first have the credit card validated, and if the card is valid, will have the order manually approved. As employee `jcooper`, you log in to the Worklist application to approve the order.

Before you do that, you open a SQL script, the Worklist application, and the Enterprise Manager test window.

2. In JDeveloper, open the `/home/oracle/labs/scripts/SQL/JDBlist_orders.sql` file.
3. In a web browser, open the Worklist application:
   a. Access the Oracle BPM Worklist application (http://soa12c.example.com:7101/integration/worklistapp).
   b. Log in to the application as `jcooper` with the password `welcome1`.
4. In a second web browser tab, open the Oracle Enterprise Manager test page.
   a. Access Enterprise Manager at http://localhost:7101/em.
   b. In the Target Navigation pane, expand the SOA > soa-infra > Home > Deployed Composites in the tree. Click the “OrderFlow [1.0]” link.
   c. On the “OrderFlow [1.0]” home page, click Test.
   d. On the Test Web Service page, scroll down to the Input Arguments section to the Request tab.
   e. Use the Browse button to replace the sample XML data with the contents of the `/home/oracle/labs/files/xml_in/processOrder.xml` file.
5. Launch the test.

6. After launching the If you receive an error
   a. In JDeveloper, with the JDBlist_orders.sql tab selected, click Run Script.

	ORD_ID	CUST_ID	PAY_METHOD	SHIP_METHOD	CARRIER	CARD_TYPE	CARD_NUMBER	TOTAL_PRICE	DATE_ORDERED	DATE_SHIPPED	STATUS
1	1001	1	credit	two_day	(null)	ACARD	5678-5678-5678-5678	6500.00	2014-08-30	(null)	waiting
2	1002	2	credit	next_day	(null)	VCARD	1234-1234-1234-1234	4190.00	2014-08-30	(null)	waiting
3	1003	1	credit	five_day	(null)	ACARD	5678-5678-5678-5678	12000.00	2014-08-30	(null)	waiting

   b. In Enterprise Manager, click Test Web Service.
   c. Switch to the Worklist application tab.
   d. Click the Refresh icon.
   e. Click the Manual Approval task entry.

      The order details are displayed in the lower-right pane.
   f. Click Approve.
   g. In JDeveloper, with the JDBlist_orders.sql tab selected, click Run Script.

	ORD_ID	CUST_ID	PAY_METHOD	SHIP_METHOD	CARRIER	CARD_TYPE	CARD_NUMBER	TOTAL_PRICE	DATE_ORDERED	DATE_SHIPPED	STATUS
1	1001	1	credit	two_day	(null)	ACARD	5678-5678-5678-5678	6500.00	2014-08-30	(null)	APPROVED
2	1002	2	credit	next_day	(null)	VCARD	1234-1234-1234-1234	4190.00	2014-08-30	(null)	waiting
3	1003	1	credit	five_day	(null)	ACARD	5678-5678-5678-5678	12000.00	2014-08-30	(null)	waiting

   h. Return to the Enterprise Manager tab.
   i. On the Response tab, click the Launch Flow Trace link.
   j. Examine the audit trail for ProcessOrder.
   k. View each of the payload options and observe the changes in the Status field.

   Note: If it took more than 30 seconds to complete steps b–f, your results in step g will vary. The status field will read ‘VALID.’ Enterprise Manager will report a test failure because the test will have timed out. For the purpose of this course, you can ignore this and rerun the test.

Practices for Lesson 11:
Using the REST Adapter

Practices for Lesson 11: Overview

Practice Overview

In this practice, you add a REST interface to the CCValidate project and test it. Then you call the new REST interface from another SOA composite. You'll use WSDL operations to handle the REST calls and XML and JSON transformations.

Lastly you create a service to call an external REST service and configure the REST Interface using the Native Format Builder, without using pre-defined XSDs.

Practice 11-1: Exposing a REST Service

Overview

In this practice, you create a service template from the existing CCValidate project and create a new composite based on that template and modify it by adding a REST interface.

Assumptions

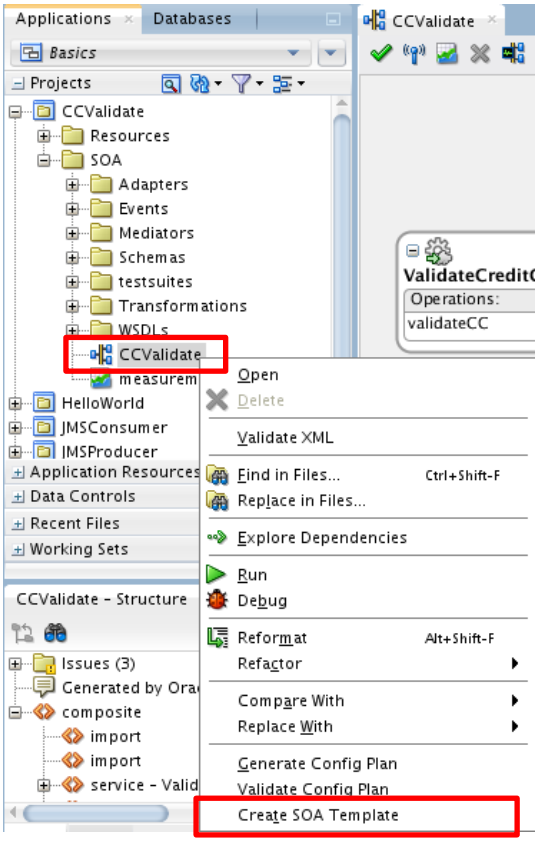
This practice assumes the following:

- You have successfully built and deployed the CCValidate project that was described in Practice 5 for the lesson titled “Using JMS and JDBC Adapters.”

Tasks

Defining a REST Interface

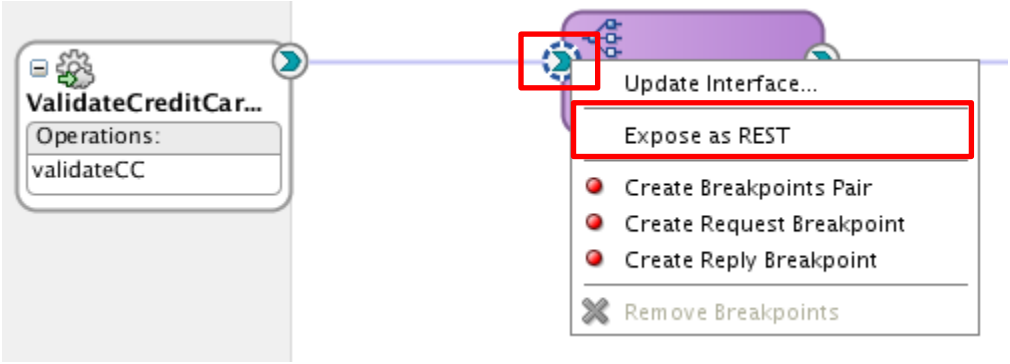
1. In the Basics application, open the `composite.xml` file for the CCValidate project.
2. Right click the CCValidate composite.xml file and select Create SOA Template.



3. Name the template CCValidateTemplate and click Next and Finish. Click OK to dismiss the success message.

4. Create a new SOA project named CCValidateWithREST. On the Configure SOA setting screen, select SOA Template and select the CCValidateTemplate you just created. Click Finish to create the composite based on the template.

5. Add a REST interface to CCValidateWithREST project.
   a. Right-click the Mediator process interface and select Expose as REST.



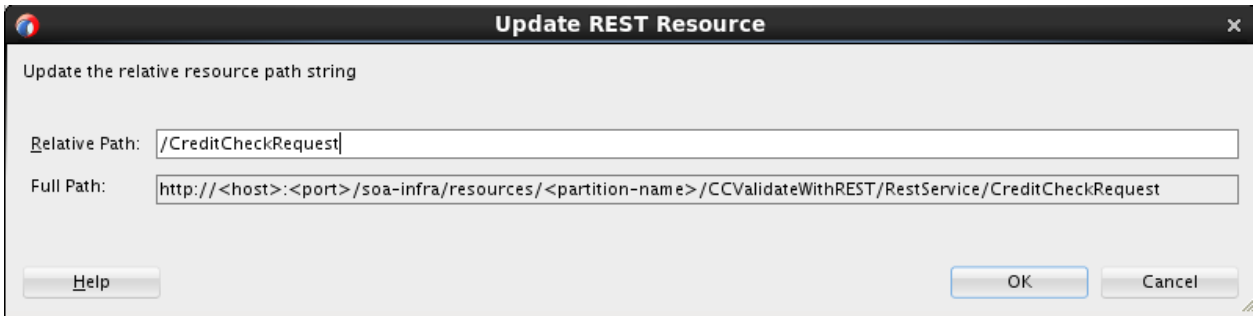
   The Create Rest Binding window is displayed.

   b. Set the following fields:
      - Name: `CCValidate_RS`. Notice the service will invoke components using WSDL interfaces option is selected. Read the rest of the information to see how the REST resources and verbs will be mapped to WSDL operations and incoming payloads translated into XML.
      - Click Next.
      - Set the Description to `Rest Interface`.

Define a resource.

6. Double click the Resource Path "/".

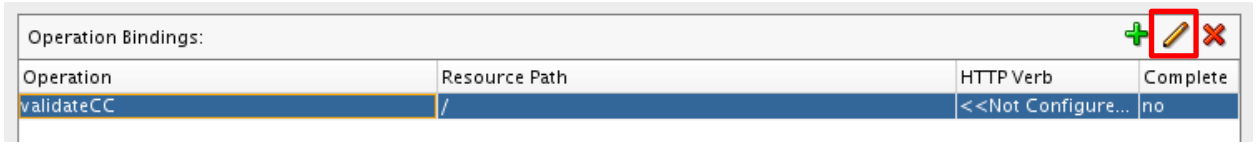
7. Enter `/CreditCheckRequest` as a relative path to the resource.



8. Click OK.
9. Modify the operation binding.

   In the Operation Bindings: section, notice the operation validateCC has no configured HTTP Verb: `<< Not Configured>>`.

   a. Select validateCC and then click the pencil to edit the operation binding.



   The REST Operation Binding dialog box is displayed.

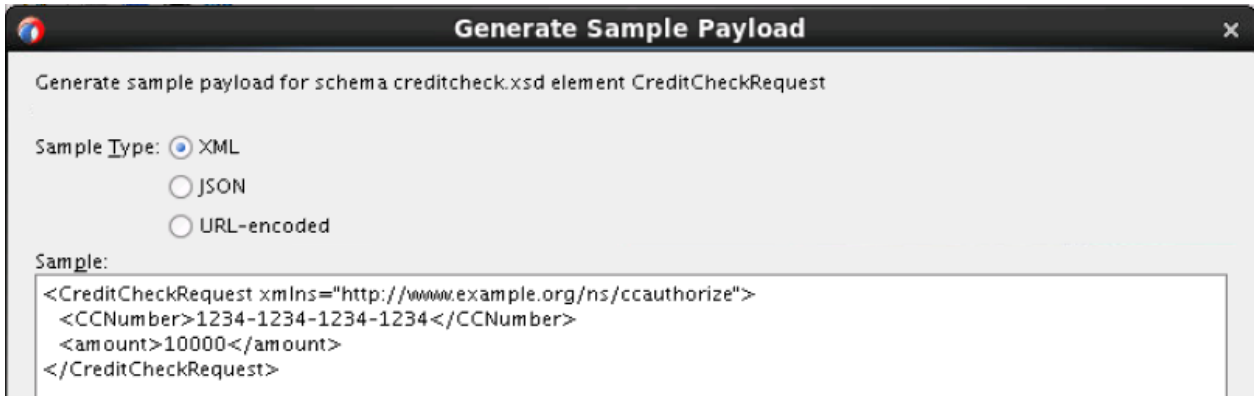
   b. Notice the verb is set to GET. Since this composite REST interface will be called by other SOA composite services, you need to change the operation to POST so that its inputs are populated correctly.

   c. Request

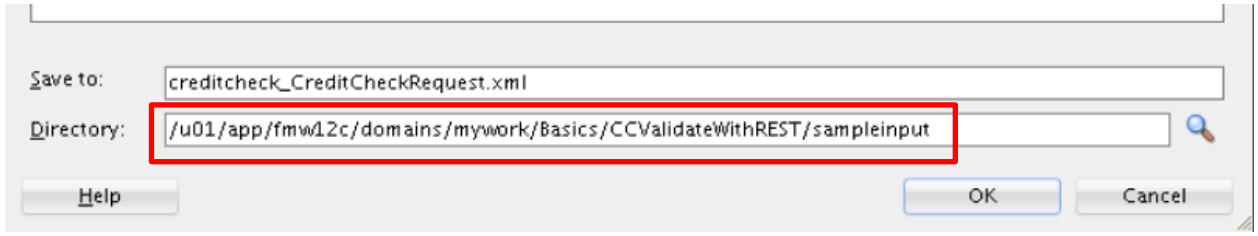
      1) Notice the Schema URL has automatically been set to `creditcheck.xsd`. This is because the REST interface will use the WSDL operations and XSD to handle the input parameters.
      2) Notice the URI Parameters have been removed when you set the HTTP verb to POST.
      3) Check JSON and XML checkboxes on, so you can use either format for the request. Click the “Generate Sample Payload” button to see what will be in the body of the POST.

   When you test this as a POST operation, you will need to supply XML and JSON payloads for testing. You will create and save these test payloads now.

4) Click the Generate Sample Payload button, and select XML. Modify the XML values as shown:

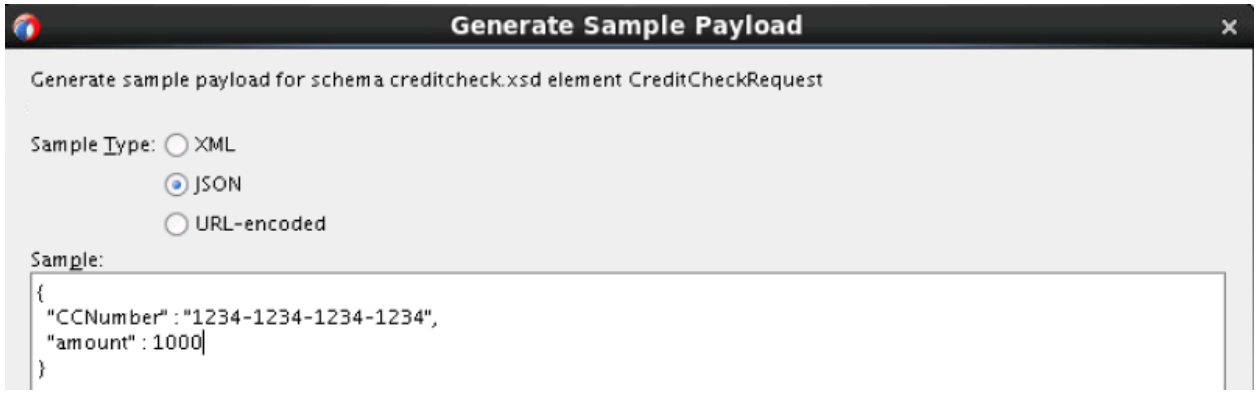


5) Save the input to a local folder within the project named sampleinput, as shown:

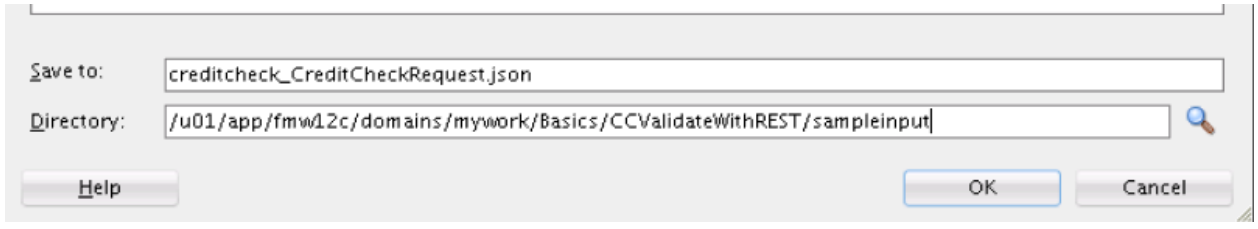


6) Click OK to save the payload sample.

7) Now do the same for the JSON. Click the Generate Sample Payload Button again and choose JSON and modify the payload as shown:



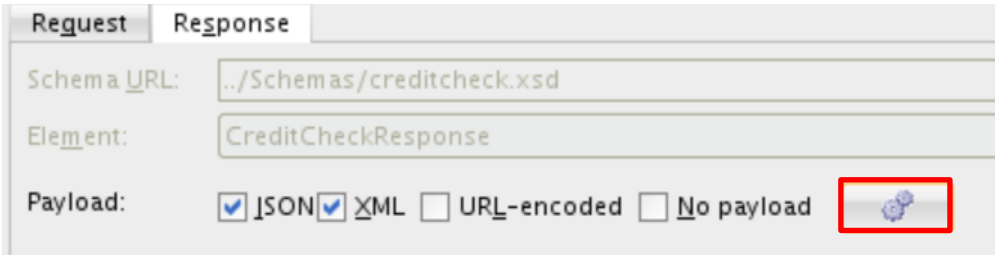
8) Again, save the input to a local folder within the project named sampleinput, as shown:



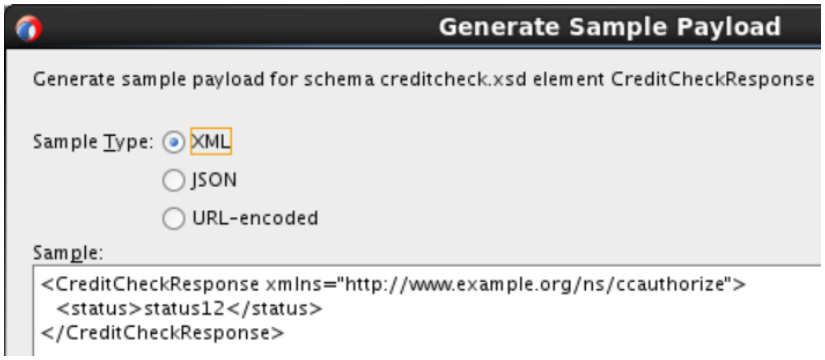
9) Click OK to save the input sample. Click the Response tab.

d. Response

1) Notice the Schema URL has automatically been set to `creditcheck.xsd`. This is because the REST interface will use the WSDL operations and XSD to handle the response generated.
2) Select the JSON and XML check boxes.
3) Click the Generate Sample Payload button.



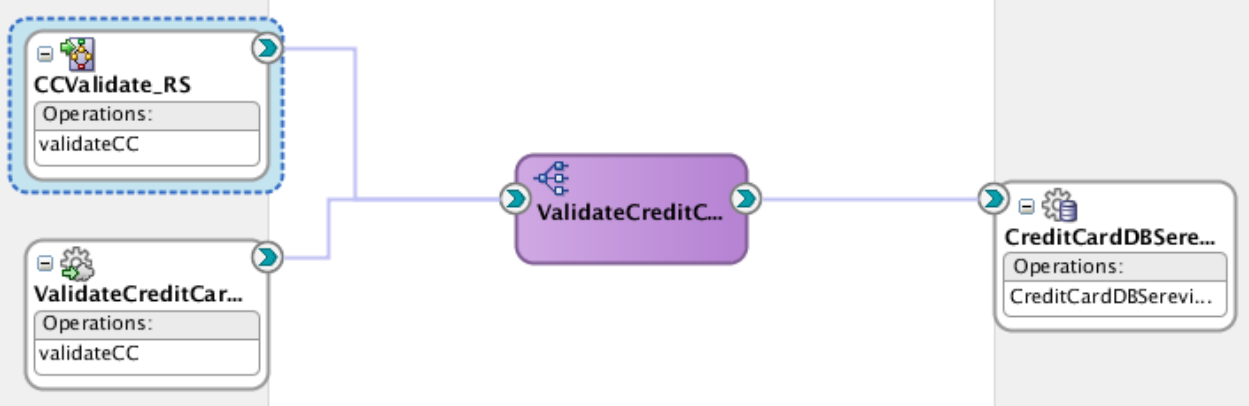
4) In the Generate Sample Payload window, select the JSON option.
5) Notice you can see the sample XML and JSON style outputs by selecting the appropriate radio button.
6) Click OK when you are done viewing the two types.



7) Click OK to exit the REST Operation Binding screen. You are returned to the REST Binding Configuration Wizard. If required, you can use this screen to define new resources, operations and parameters. In this case you are done, so click Finish.
8) Note: In the 12.1 version of SOA Suite you would have to create and use a mediator to transform incoming REST JSON into XML for BPEL, as 12.1 BPEL did not work with JSON natively. In the current, 12.2 release, BPEL can now work with JSON natively, so a mediator is not needed.
9) In these next exercises you will use WSDL operations to handle REST requests and responses and SOA Suite will handle the transformation for you, enabling you to manipulate BPEL variables using XPath.
10) While BPEL is capable of working with JSON natively, without the need of an XSD and WSDL, this requires more complicated variable definition and frequently requires writing JavaScript to manipulate the JSON variables and their objects.

11) Consequently, most developers choose to use WSDL to implement REST inbound and outbound operations, enabling them to use XPath within BPEL directly. This makes development, maintenance and debugging simpler, yet still enables a SOA composite to support both XML for SOAP interfaces and XML and JSON data formats for REST Interfaces and external services.

10. Verify that the composite application assembly model resembles the following image:



11. Deploy the CCValidateWithREST project to the IntegratedWebLogicServer.

Practice 11-2: Test a REST Interface

Overview

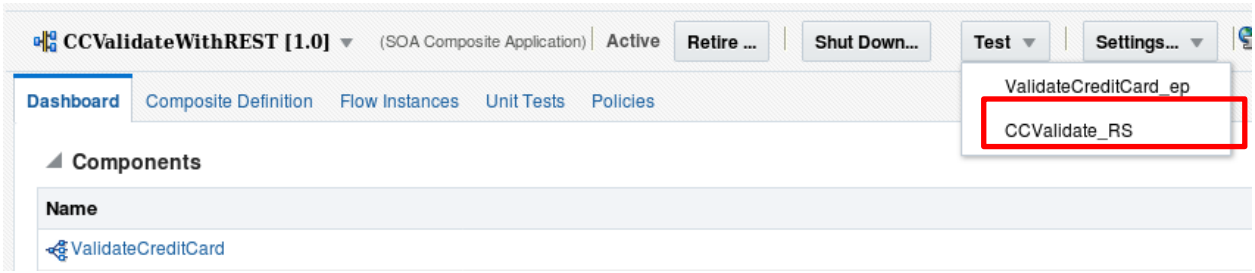
In this practice, you test the SOA project that consumes the exposed REST service that you just developed and tested.

Assumptions

This practice assumes that you have completed Practice 11-1 successfully.

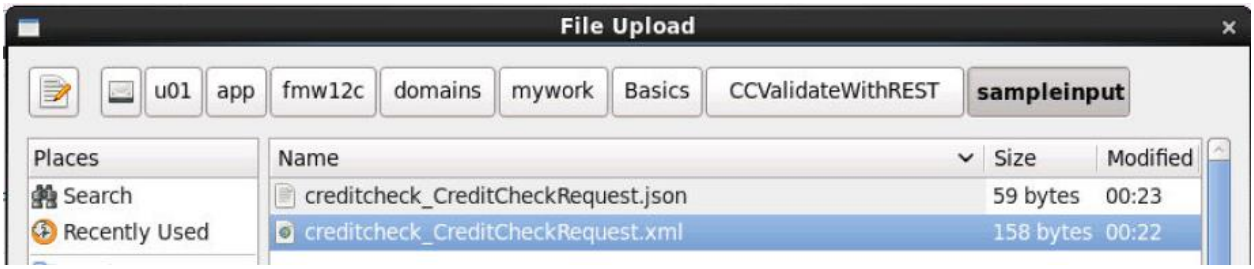
Tasks

2. In Enterprise Manager, navigate to the test page for the CCValidateWithREST project you just deployed.
   a. Open Enterprise Manager.
   b. Navigate to the CCValidateWithREST project.
   c. On the CCValidateWithREST home page, click Test. Notice there are now two test points:



   The CCValidate_RS interface invokes the new REST interface you just added. Select this and the test page opens.
   d. On the Test Web Service page, scroll down and notice that the Representation Media Type for Request and Response have two choices: application/xml and application/json. This shows that the service can be requested or respond with either format.
   e. Select application/xml/ for both.

f. In the Input Arguments section of the Request tab expand the Triangle to the left of “Content and click the Browse button to find and load the XML sample payload you previously created. Browse to `/u01/app/fmw12c/domains/mywork/Basics/CCValidateWithREST/sampleinput` and select the `creditcheck_CreditCheckRequest.xml` file.



Click Open and the payload is entered into the content area.

g. Click Test Web Service. Notice the response comes back formatted as XML.

h. Now select application/json for the Response type and test the service again. This time the response comes back formatted as JSON.

i. You have successfully exposed an existing SOAP-based service with a REST interface. If you want to test the JSON payload, follow the same steps as e-g above, but select application/json for the Request type and load the JSON sample payload.

j. Next you will call this service’s REST interface from another composite application.

Creating Practice 11-3: Using an Existing REST Interface

Overview

In this practice, you create a SOA project that consumes the exposed REST service that you just developed and tested.

Assumptions

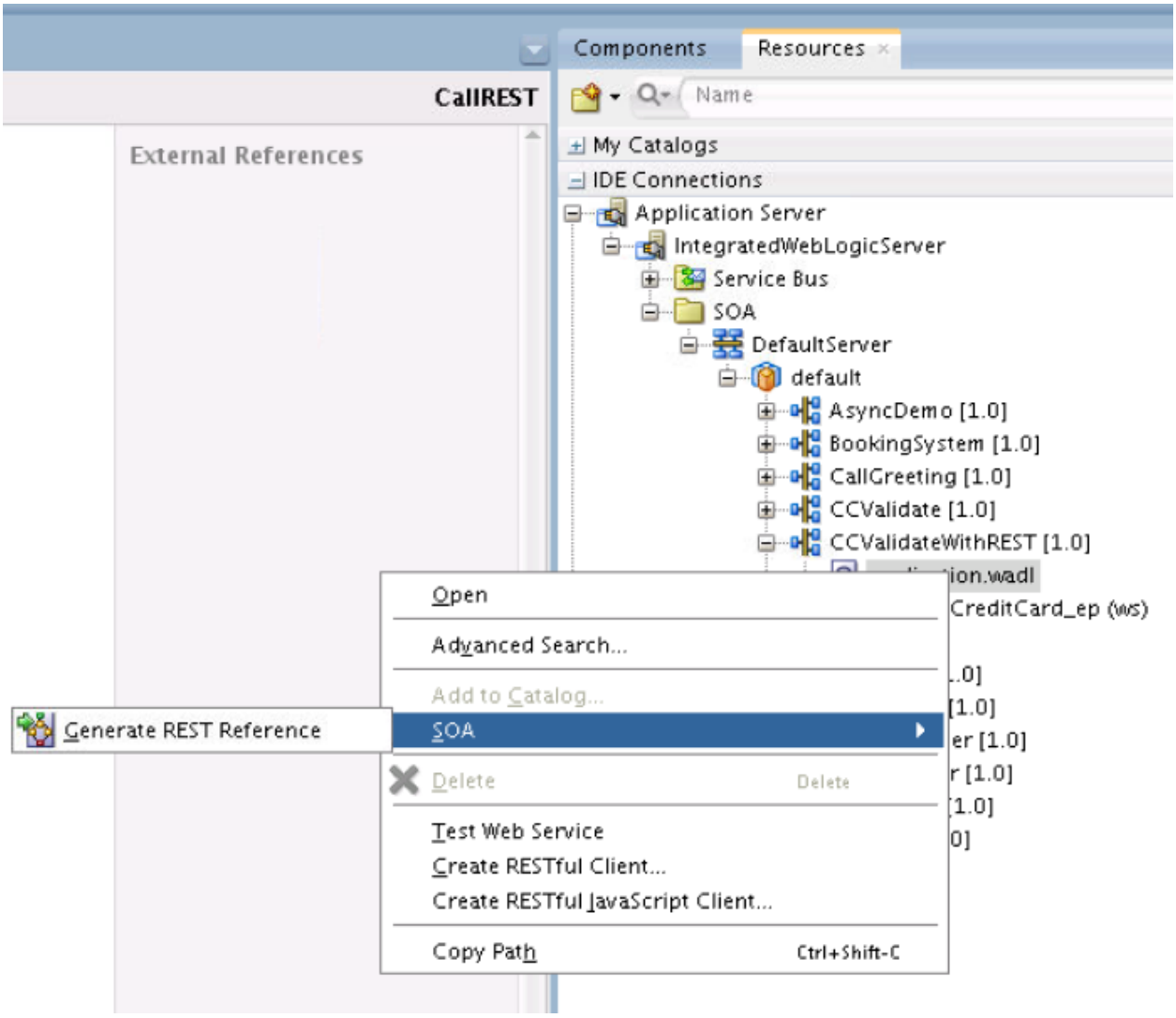
This practice assumes that you have completed Practice 11-2 successfully.

Tasks

1. In the BPELProjects application, create a new empty SOA project named CallREST.
   The `composite.xml` file is created and the overview window is opened for editing.

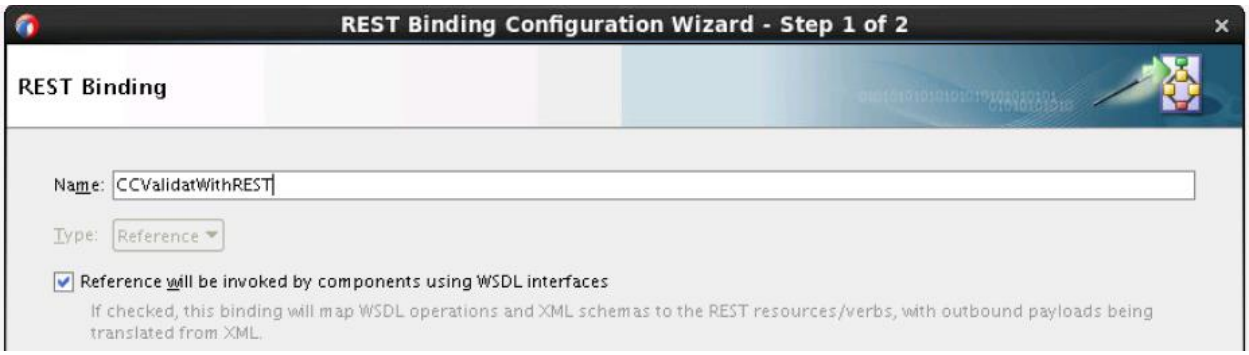
Creating and Configuring a REST Interface

2. Add a REST binding to the External References swimlane. Use the Respurces window to expand the Application Server , IntegratedWebLogic Server Connection, then SOA, then DefaultServer, then default, then CCValidateWithREST and right click the `application.wadl` and select SOA > Generate REST Reference.



The “Create Rest binding” dialog box is displayed.

3. Enter **`CCValidateWithREST`** for the name and select the Reference will be invoked by components using WSDL interfaces check box. Click Next.



4. You'll be using a composite already deployed to SOA Suite, so it has a WADL already defined. You will use the existing WADL to define the bindings for the interface.
5. Click Finish and OK when prompted to localize the files.

   The definition of the REST binding is completed. The REST External Reference has been added.

Creating a BPEL Process

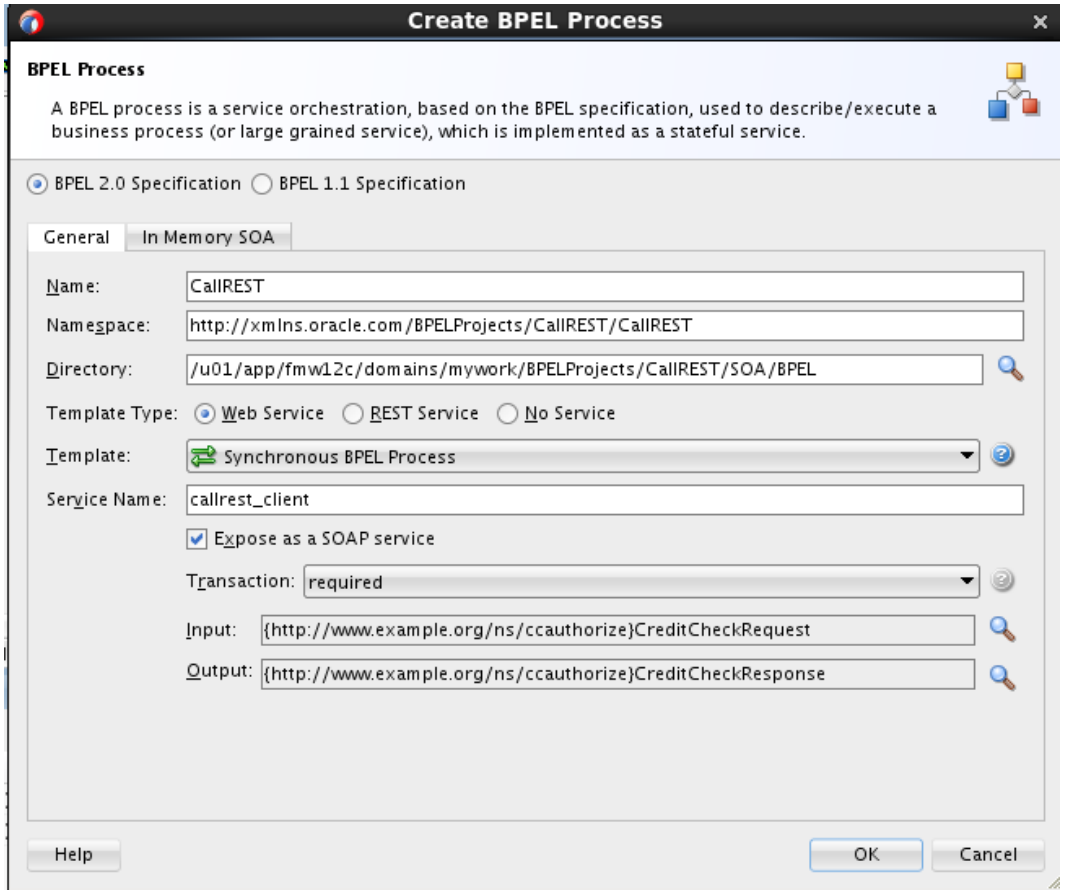
In this section, you create a BPEL process that invokes the REST interface that you created.

6. Use the Components window to add a BPEL process to the application.

   The Create BPEL Process dialog box is displayed.

   a. Name: `CallREST`
   b. Template: Synchronous
   c. Template Type: Web Service
   d. Expose as SOAP service is selected
   e. Create an input variable based on the CreditCheckRequest element of the `creditcheck.xsd` file (already in the project).
   f. Create an output variable based on the CreditCheckResponse element of the `creditcheck.xsd` file.

g. Verify your work and click **OK**.



Note: The Template Type REST Service is used if the BPEL will implement the exposed interface of an existing SOA composite.

In this example, the BPEL will be exposed as a SOAP service. But, you could easily expose it as a REST service following the steps you used in the previous section to expose the mediator with a REST interface. You could do the same thing with a BPEL component.

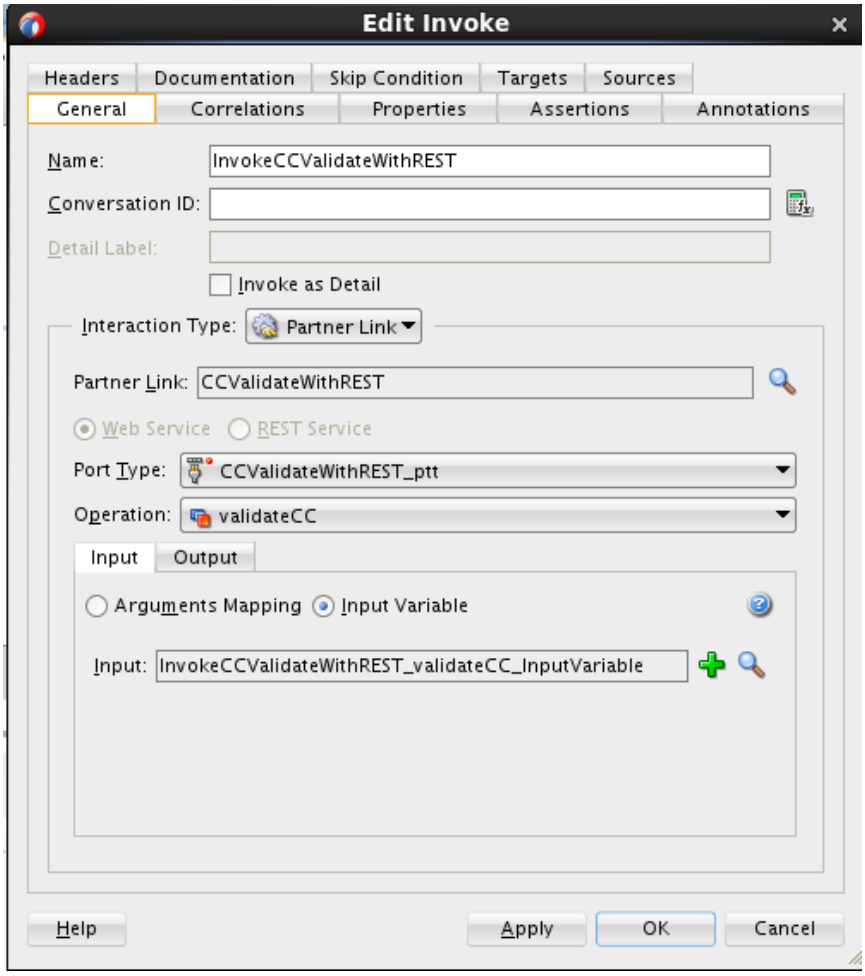
7. Wire the BPEL process to the CCValidateWithREST External Reference binding.

Defining Activities in the BPEL Process

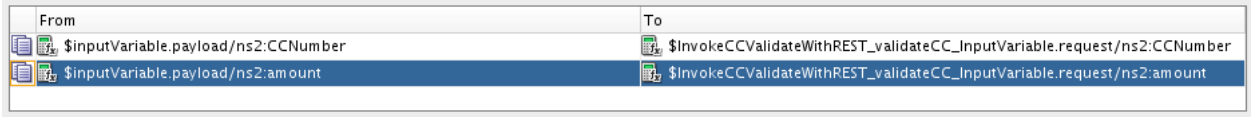
8. Open the BPEL Process for editing.
9. Create and configure an Invoke activity.
   a. Add an **Invoke** activity to the BPEL process.
   b. Map the Invoke activity to the CCValidateWithREST partner link.

      The Edit Invoke window is displayed.
   c. Name the new activity **`Invoke_CCValidateWithREST`**.
   d. Automatically create input and output messages.

e. Verify your work and click OK.

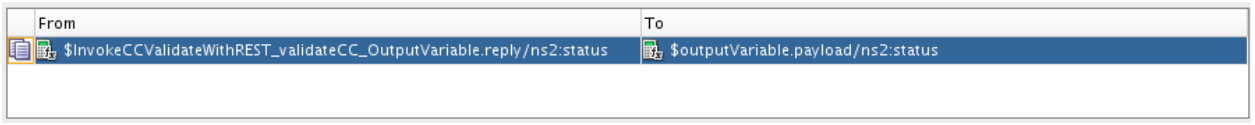


10. Create and configure an Assign activity to set the input message for the REST service.
    a. Add an **Assign** activity before the Invoke activity.
    b. Name the new activity **`Assign_CreditCard`**.
    c. In the Assign Editor, map the CCNumber element in the process input variable to the CCNumber element in the input variable for the REST service.
    d. Map the amount element in the process input variable to the amount element in the input variable for the REST service.

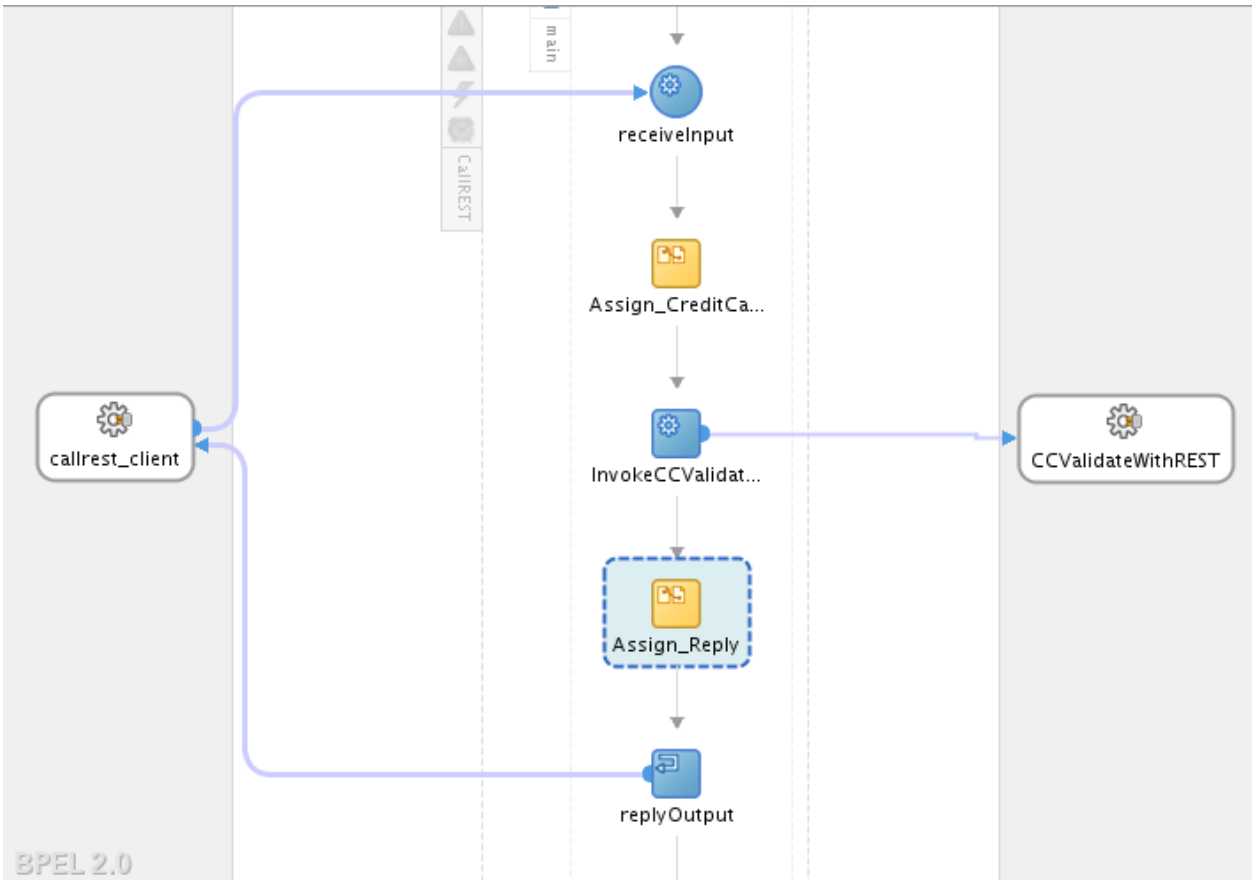


    e. Click OK.
11. Create and configure an Assign activity to set the output message for the BPEL process.
    a. Add an **Assign** activity after the Invoke activity.
    b. Name the new activity **`Assign_Reply`**.
    c. In the Assign Editor, map the status element in the REST service output variable to the status element of the output variable for the BPEL process.

d. Verify your work and click OK.



12. Verify and save your work.



Practice 11-4: Deploying and Testing the Project

Overview

In this practice, you deploy and test the project.

Assumptions

This practice assumes that you have completed all work through Practice 11-3 successfully.

Tasks

1. Deploy the CallREST application to the IntegratedWebLogicServer.

   Remember to select the “Overwrite any existing composites with the same revision ID” check box.
2. In Enterprise Manager, navigate to the test page for the CallREST project.
   a. Open Enterprise Manager.
   b. Navigate to the CallREST project.
   c. On the CallREST home page, click Test.

      The test page opens.
   d. On the Test Web Service page, scroll down to the Input Arguments section to the Request tab. Expand **payload** and enter the following values:

      CCNumber: `1234-1234-1234-1234`

      amount: `1000`
   e. Click Test Web Service.

      The Response tab is displayed.
   f. Verify that the status value returned is VALID, as shown in the following screenshot:

   g. On the Response tab, click the Launch Message Flow Trace link.

h. In the Flow Trace, click the View Payload link for Assign_Reply in CallREST.

The called project CCValidateWithREST has successfully processed the message. The REST adapter has converted the JSON to XML, and returned the status of VALID.

If you have time, add a REST interface to the CallREST BPEL process in the CallREST project. Set the HTTP verb to GET so you can enter the parameters manually. Redeploy and re-test CallREST. This time test the REST interface you just added.

Thought question: What would you need to do if you wanted to be able to call the CCValidateWithREST service with a GET operation in addition to the POST operation? If you set this up, how would you configure the Invoke action when you wired it to the Partner Link in the BPEL process?

Practice 11-5: Use the Native Format Builder to Generate XSDs for REST Interfaces

Overview

In this practice, you use the Native Format Builder to generate XSDs to support the use of WSDL for RESTful services.

In the previous exercises, you were supplied with the XSDs for the REST request and response structures. However, you may need to access REST services but don't have the necessary XSDs. Using XSDs and XPath make working with JSON in SOA Suite much easier.

While you can use pure JSON, or write the XSDs by hand or use utilities that can convert JSON to XSD, SOA Suite includes the Native Format Builder utility that can introspect RESTful URLs and their JSON outputs and generate the needed XSDs. In this part of the practice, you'll use it to access a REST service.

Tasks

1. Test the provided external web service. It is an API for getting sunlight hours for a given longitude and latitude. It is provided by https://sunrise-sunset.org as an example **only**, and **should not** be used in production.
2. Enter this URL in your browser: http://api.sunrise-sunset.org/json?lat=36.7201600&lng=-4.4203400

3. You should see some JSON returned (yours may differ):

```
{"results":{"sunrise":"7:27:53 AM","sunset":"5:28:36
PM","solar_noon":"12:28:14
PM","day_length":"10:00:43","civil_twilight_begin":"6:59:59
AM","civil_twilight_end":"5:56:30
PM","nautical_twilight_begin":"6:28:21
AM","nautical_twilight_end":"6:28:08
PM","astronomical_twilight_begin":"5:57:25
AM","astronomical_twilight_end":"6:59:04 PM"},"status":"OK"}
```

3. In the BPELProjects application, create a new empty SOA project named **GetSunlightHours**.

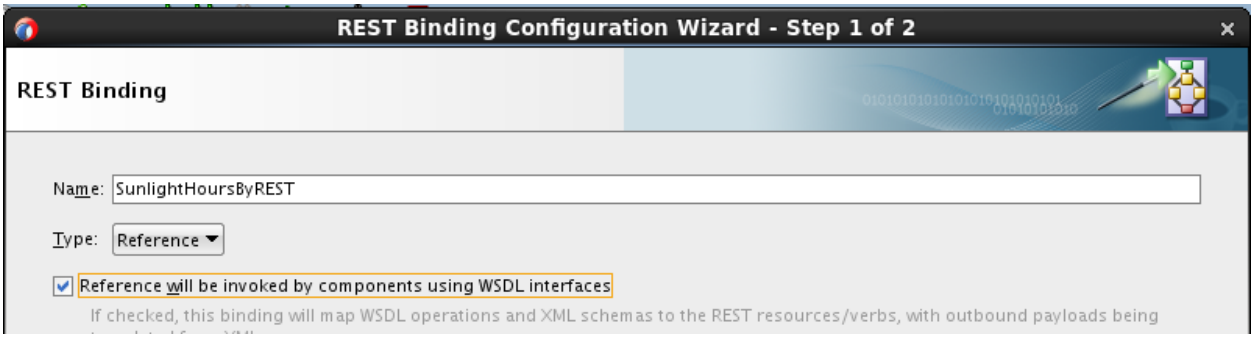
The `composite.xml` file is created and the overview window is opened for editing.

Creating and Configuring a REST Interface

4. Add a REST binding to the External References swimlane.

The "Create Rest binding" dialog box is displayed.

Enter **`SunlightHoursByREST`** for the name and select the Reference will be invoked by components using WSDL interfaces check box. Click Next.

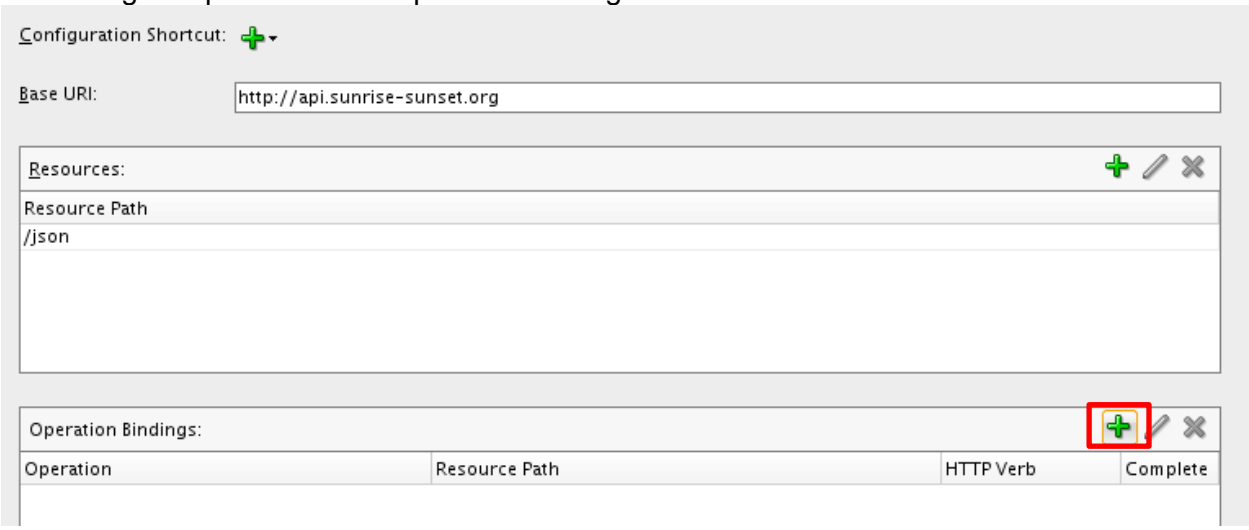


5. You do not have the WADL for this external REST service. Therefore, you will need to use the Native Format Builder to generate the XSDs needed for the WSDL interface.

   Enter the Base URI: http://api.sunrise-sunset.org

   Set the Resource Path: /json

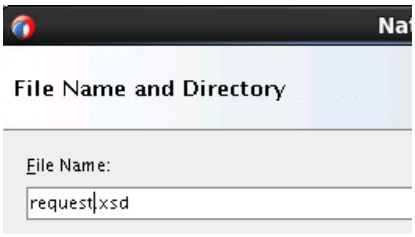
6. Click the green plus to add an operation binding.



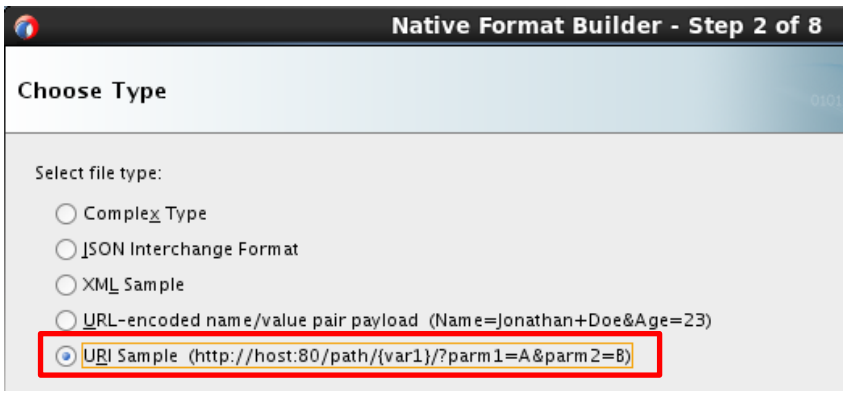
7. The REST Operation Binding page opens. Set the HTTP Verb to GET.
8. You need to define the Schema URL, which is currently greyed out. Click the Define Schema for Native Format button to the right of the Schema URL field.

9. The Native Format Builder Opens. Click Next.

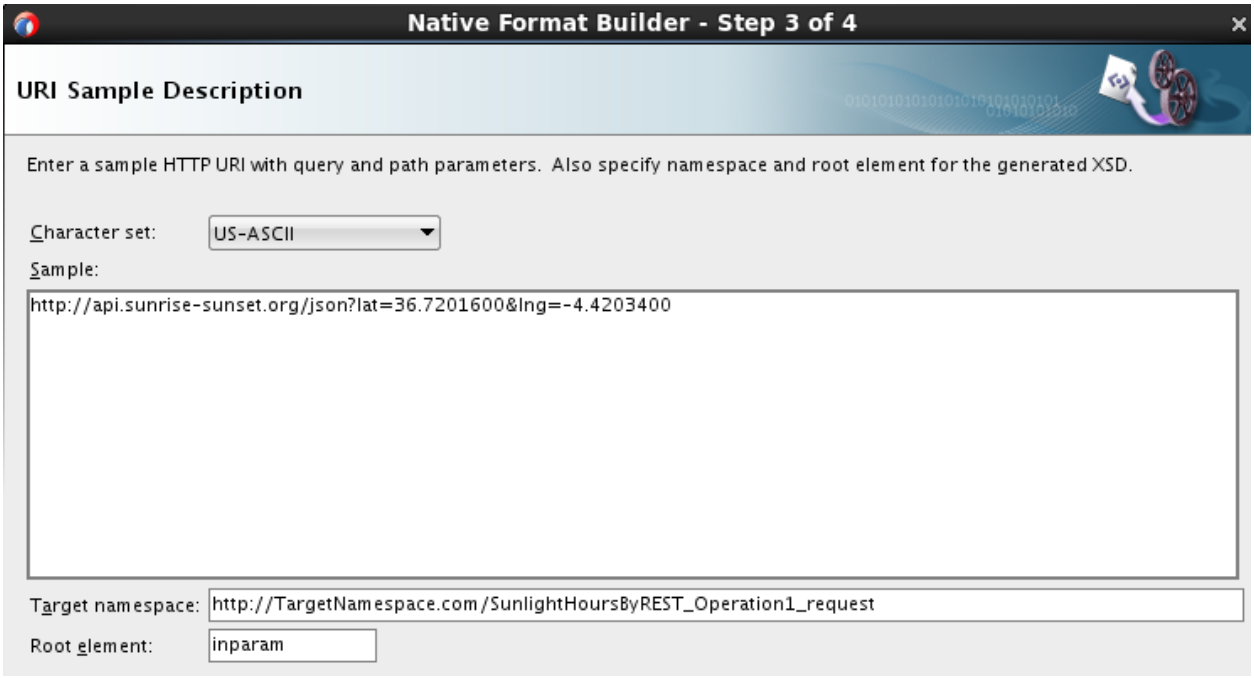
10. Change the file name for the XSD to `request.xsd`. Click Next.



11. There are different ways to generate the XSD from native sources. For the request parameter XSD, you will use the URI Sample. Select this then click Next.



12. Go back to the browser and copy the URL from the URL you previously and paste it into the Sample box. Set the Root element to **inparam**. Click Next.

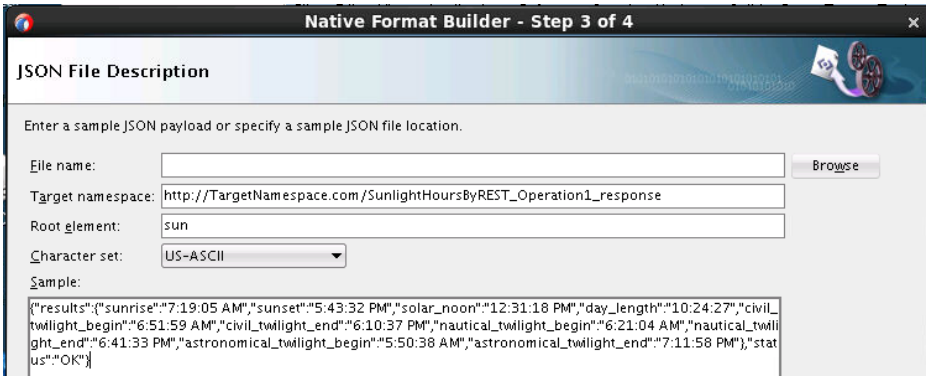


13. Notice the URL output has been parsed and an XSD is created. If you needed to, you could edit the generated XSD here. In this case, you do not need to make any changes. Click Next and Finish.
14. Return to the REST Operation Binding dialog box. Notice that the URI parameters are created, the style and types and expression are all configured. You now need to create the response XSD. Click the Response tab.

15. As you did in the previous step, click the gear icon for the Schema URL to invoke the Native Format Builder. Click Next.
16. Set the filename to `response.xsd`. Click Next.

17. For the response itself, you will use the JSON Interchange Format to generate the XSD. On the Choose Type screen make sure the JSON Interchange Format option is selected (it's the default) and click Next.
18. If you had a sample JSON file you could use that as the sample payload, but since you have the JSON in the browser from your test of the REST service, simply copy the entire JSON content of the browser. Select the JSON, and right-click and select Copy.

19. Paste this into the Sample box in the JSON File Description Window using Ctrl-V. Set the Root element to `sun`.



20. Click Next and view the generated XSD based on the JSON returned from the service. Click Next and then Finish. Return to the REST Operation Binding screen. The Schema URL now displays the `response.xsd` that you just generated.

Make sure **both** the JSON and XML boxes are checked for the Payload and click OK. Click Finish to exit the REST Binding Configuration Wizard. The definition of the REST binding is completed and a new REST External Reference binding has been added to your project.

Creating a BPEL Process

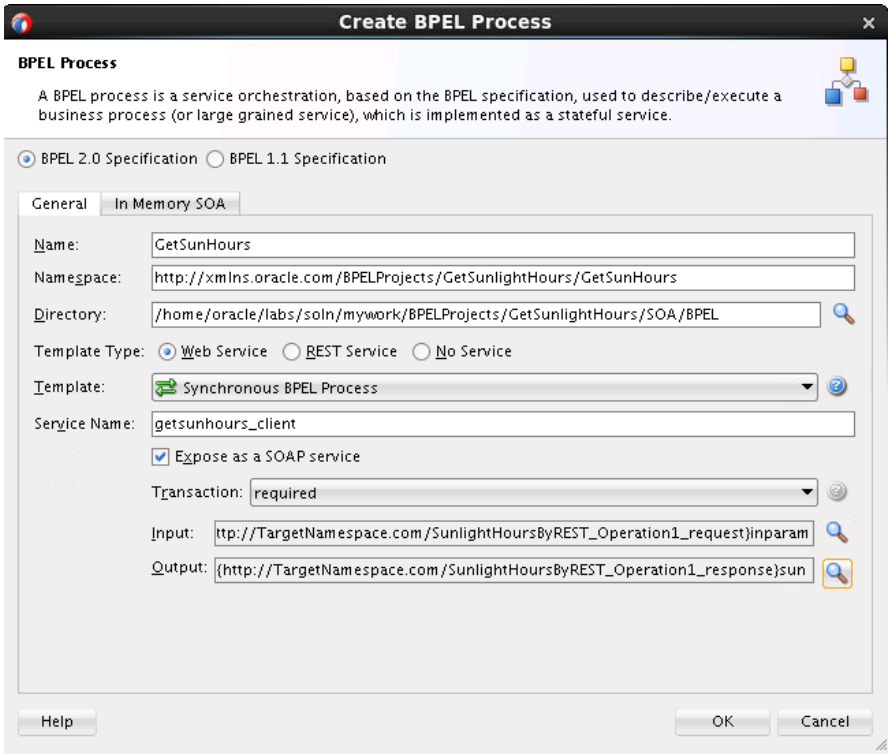
In this section, you create a BPEL process that invokes the REST interface that you created.

21. Add a BPEL process to the application.

The Create BPEL Process dialog box is displayed.

a. Name: `GetSunHours`

b. Template: Synchronous
c. Template Type: Web Service
d. Expose as SOAP service is selected
e. Create an input variable based on the **inparam** element of the `request.xsd` file (already in the project schema files).
f. Create an output variable based on the **sun** element of the `response.xsd` file (already in the project schema files).
g. Verify your work and click **OK**.

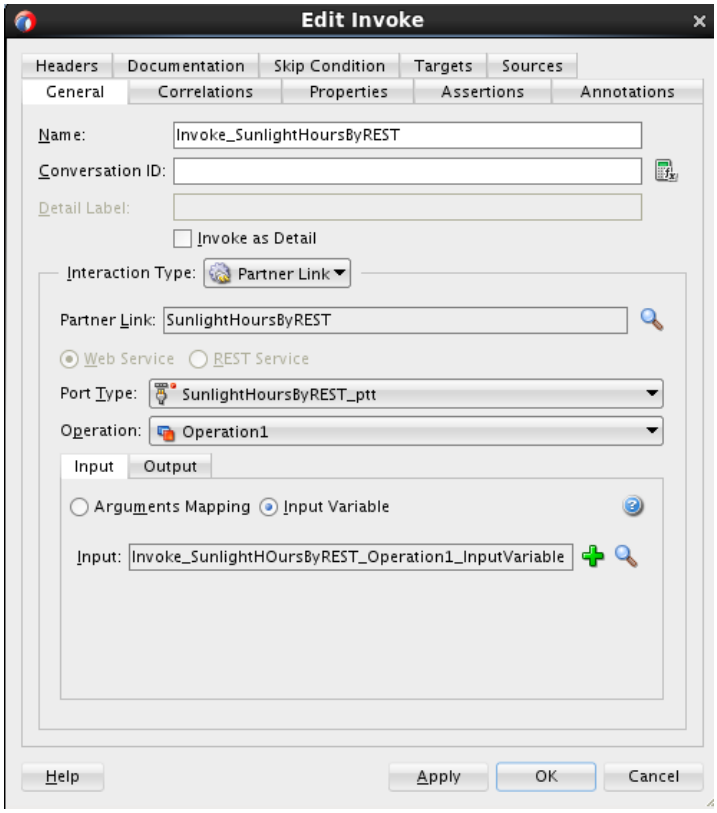


22. Wire the BPEL process to the GetSunlightHoursByREST External Reference binding.

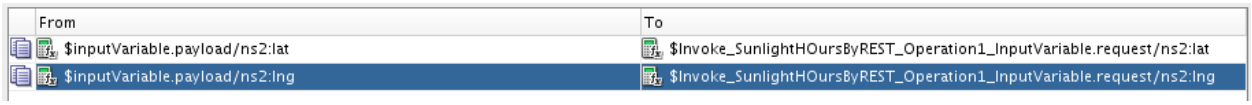
Defining Activities in the BPEL Process

23. Open the BPEL Process for editing.
24. Create and configure an Invoke activity.
a. Add an **Invoke** activity to the BPEL process.
b. Map the Invoke activity to the SunlightHoursByREST partner link.
The Edit Invoke window is displayed.
c. Name the new activity **`Invoke_SunlightHoursByREST`**.
d. Automatically create input and output variables.

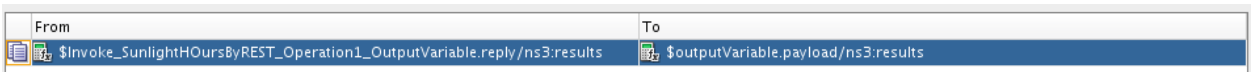
e. Verify your work and click OK.



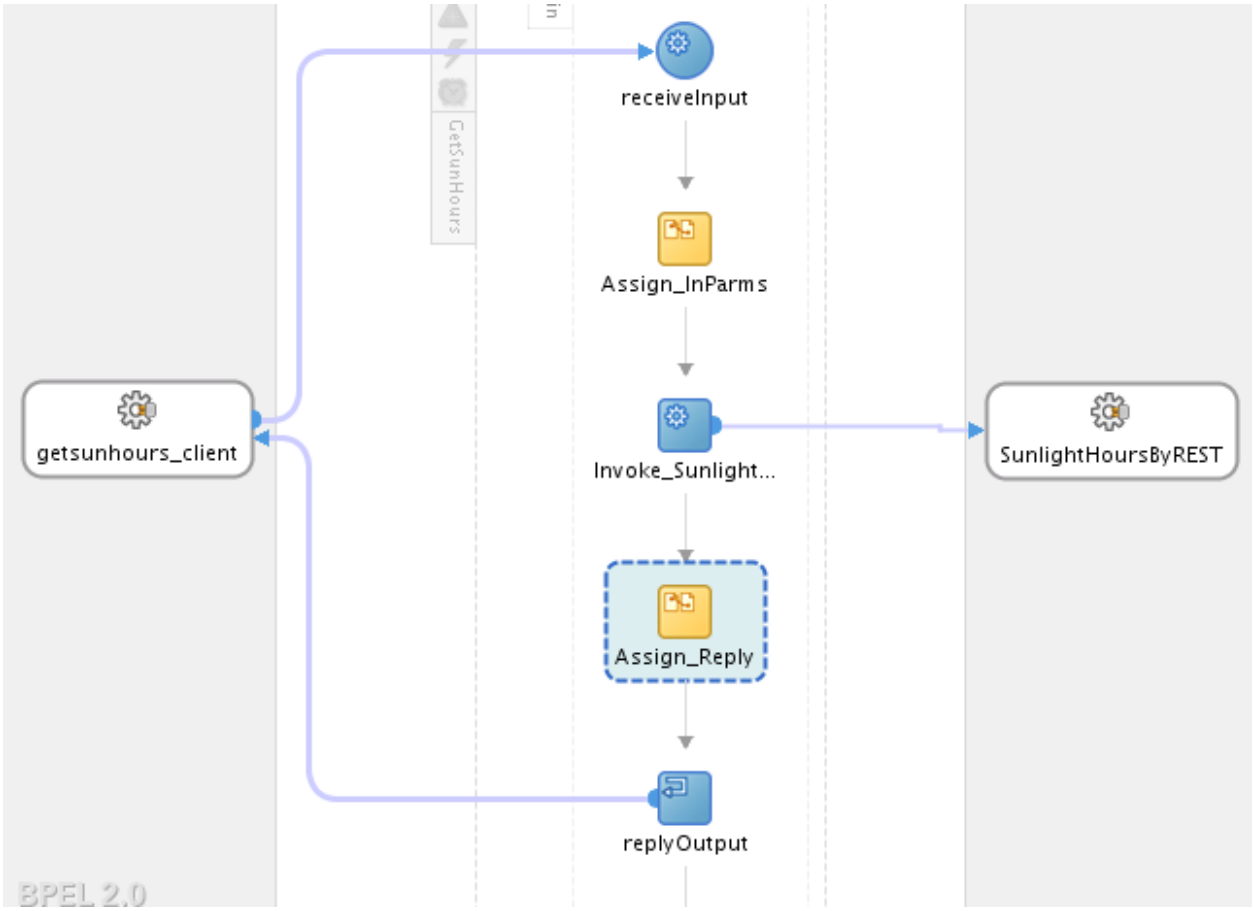
4. Create and configure an Assign activity to set the input message for the REST service.
   a. Add an **Assign** activity before the Invoke activity.
   b. Name the new activity **`Assign_Inparms`**.
   c. In the Assign Editor, map the lat element in the process input variable to the lat element in the input variable for the REST service.
   d. Map the lng element in the process input variable to the lng element in the input variable for the REST service.



   e. Click OK.
5. Create and configure an Assign activity to set the output message for the BPEL process.
   a. Add an **Assign** activity after the Invoke activity.
   b. Name the new activity **`Assign_Reply`**.
   c. In the Assign Editor, map the results element in the REST service output variable to the results element of the process output variable for the BPEL process. Since both payloads are identical, the mapping of elements with the same names will occur automatically at run time.
   d. Verify your work and click OK.



6. Verify and save your work.



Practice 11-6: Deploying and Testing the Project

Overview

In this practice, you deploy and test the project.

Assumptions

This practice assumes that you have completed all work through Practice 11-5 successfully.

Tasks

1. Deploy the GetSunlightHours application to IntegratedWebLogicServer.

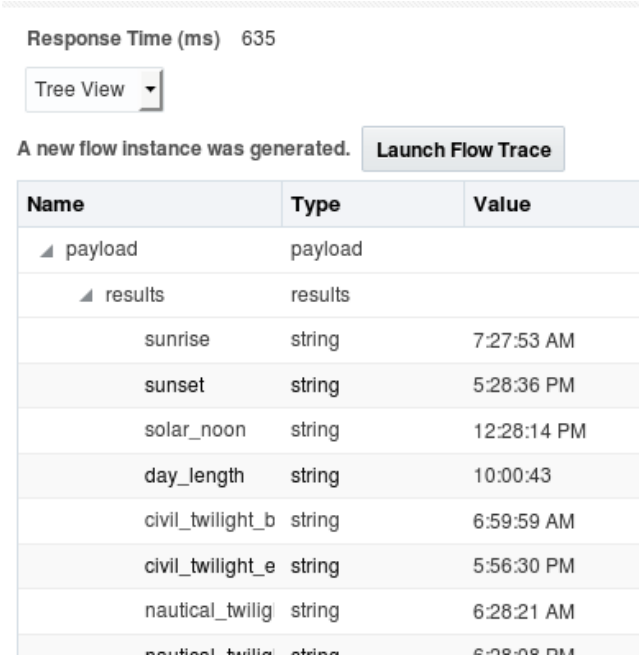
   Remember to select the “Overwrite any existing composites with the same revision ID” check box.
2. In Enterprise Manager, navigate to the test page for the GetSunlightHours project.
   a. Open Enterprise Manager. Click the Target Navigation icon, expand SOA and right-click soa-infra and select Home> Deployed Composites.
   b. Click the GetSunlightHours composite.
   c. On the GetSunlightHours home page, click Test.

      The test page opens.
   d. On the Test Web Service page, scroll down to the Input Arguments section to the Request tab. Expand **payload** and enter the following values:

      lat: `36.7201600`

      lng: `-4.4203400`
   e. Click Test Web Service.

   NOTE: If you test the service and it fails with `an unknown host exception: api.sunrise-sunset.org`, this is expected in the OU lab environment due to security constraints, so you are done with the practice. If the service call worked, you would see the remote web service results returned and displayed.



f.

Practices for Lesson 12: Developing Business Rules

Practices for Lesson 12: Overview

Practices Overview

In this lesson, you learn basic vocabulary and concepts for Oracle Business Rules, as well as how to configure business rules and Decision Tables. You also learn how to invoke the configured business rule service component from a BPEL process.

Practice 12-1: Creating and Configuring a Composite Application

Overview

In this practice, you create a composite application that processes order data. The assembly model includes a Business Rules component that determines whether or not the order requires manual approval. The application includes an external reference that exposes the `Approval` application (from Practice 9) as a web service. The assembly model also includes a BPEL process, which invokes the service component and external reference in the course of its execution.

Assumptions

This practice assumes that you have completed Practice 9 for the lesson titled “Implementing Human Workflow and Notifications” successfully, and have deployed the `Approvals` and `ApprovalTaskForm` applications.

Tasks

1. In the BPELProjects application, create a new empty SOA project named `OrderProcessing`.

   The `composite.xml` file is created and the overview window is opened for editing.

Creating and Configuring an External Reference to the `Approval` Project

In this section, you add an External Reference to the `Approval` project that you created and deployed in the previous practice. The application is invoked as a web service from a BPEL process within this composite.

2. Create and configure a web service in the assembly model (`composite.xml`) as an External Reference.
   a. Add a SOAP web service to the External Reference swimlane.

      The Create Web Service window opens.

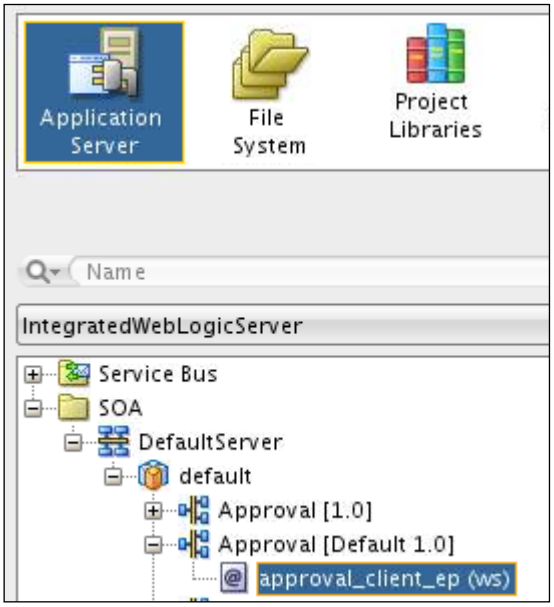
   b. Name the service `ApprovalService`.

   c. To define the WSDL URL, perform the following steps:

      1) Click the Find Existing WSDL button.

         The WSDL Chooser opens.

2) With Application Server selected, navigate to SOA > Default Server > default > Approval [Default 1.0] > approval_client_ep (ws).



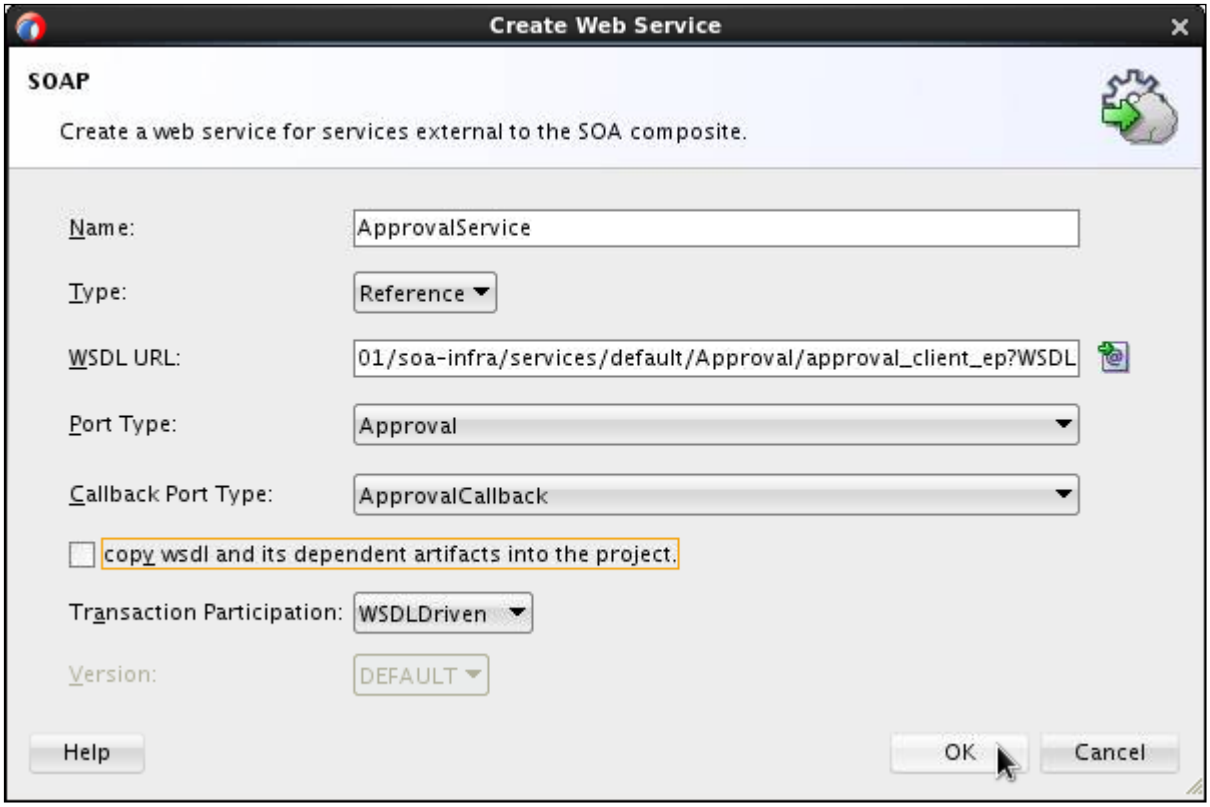
3) Click OK to close the WSDL Chooser.

d. Set the Port Type to Approval.

e. Set the Callback Port Type to ApprovalCallback.

f. Deselect the “copy wsdl and its dependent artifacts into the project” check box.

g. Verify your work and click OK.

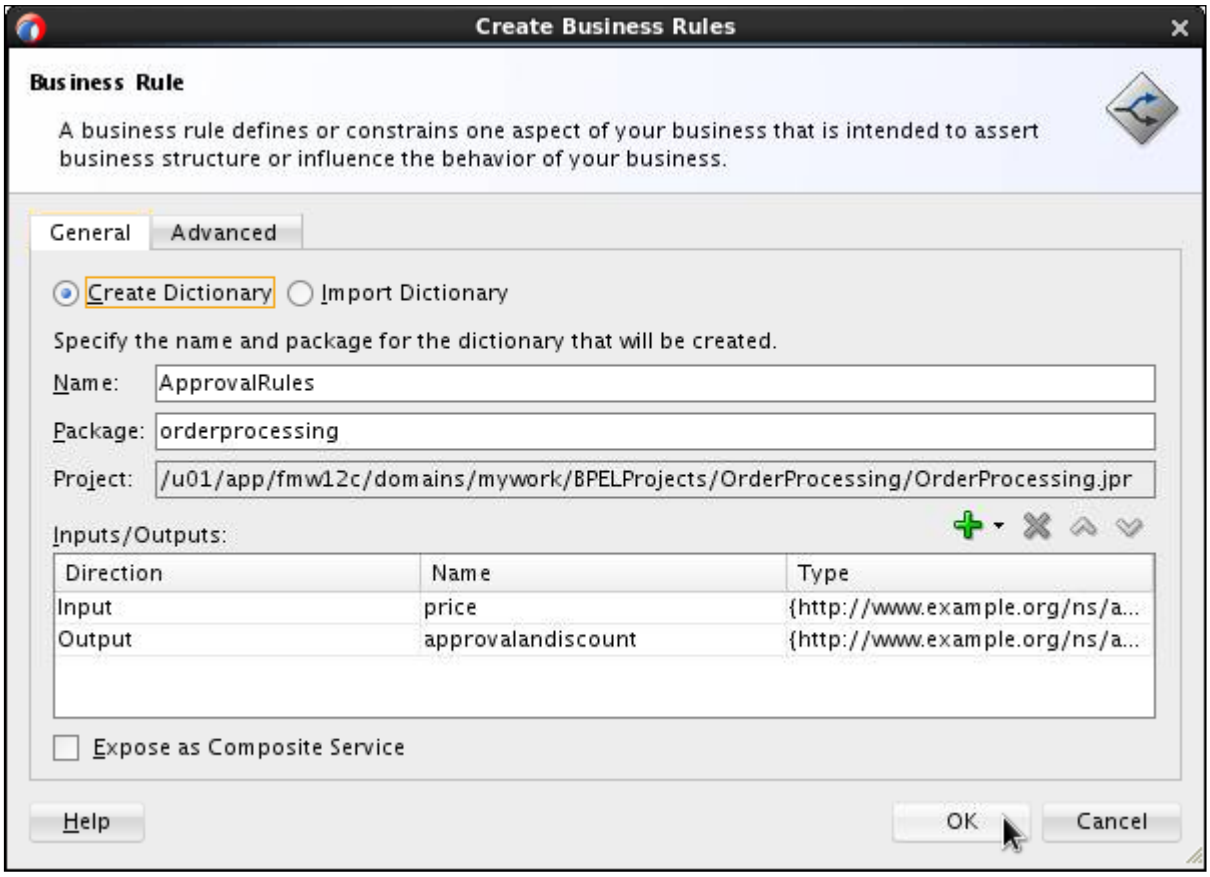


Creating the `ApprovalRules` Business Rules Component

In this section, you create a Business Rules component that implements the if/then logic to determine whether or not the current order needs human authorization. The decision is based on the total value of the order (for example, “if total is greater than 4000, require approval”). The Business Rules component is invoked from a BPEL process within this application.

3. Create and configure a Business Rules component in the assembly model.
    a. Add a Business Rules component to the Components swimlane.
    b. Name the component `ApprovalRules`.
    c. Add input.
        1) Select Create > Input.
        2) Import `/home/oracle/labs/files/xsd/orderapproval.xsd`.
        3) Select the `price` element. Click OK.
    d. Add output.
        1) Select Create > Output.
        2) Select the `approvalandiscount` element of `orderapproval.xsd`. Click OK.
    e. Define a name for the decision service.
        1) Click the Advanced tab.
        2) Name the service `ApprovalRulesService` (case-sensitive).
    f. Toggle back to the General tab.

g. Verify your work and click OK.



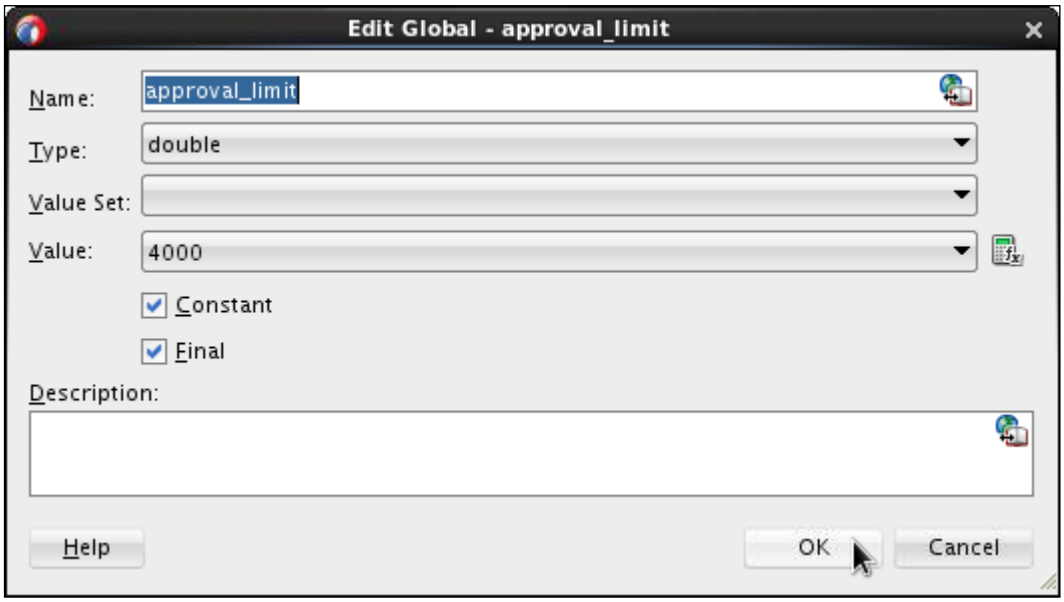
h. Save your work.

Creating a Global to Hold the Approval Limit

In this section, you create a global value to hold the approval limit value, which is used in the rule conditions that are created in the subsequent steps of this practice.

4. Create and initialize a global that is defined for the approval limit as a value of `4000`.
   a. Right-click the ApprovalRules component icon and select Edit.
   b. Click the Globals tab on the left side of the editor.
   c. Click the Create icon.
   d. Name the global `approval_limit`.
   e. Use the drop-down menu to set the type to `double` (*with a lowercase "d"*).
   f. Use Expression Builder to define a value of `4000`.
   g. Select the Constant check box and ensure that the Final check box is selected.

h. Verify and save your work.



Renaming the Default Ruleset

In this section, you rename the default initial ruleset that is already created in the dictionary, before you create the rules in the ruleset.

5. Rename the `Ruleset1` ruleset.
   a. Click the `Ruleset1` tab on the left side of the editor.
   b. In the Name field, enter the text: `approvalruleset`, and press Enter.

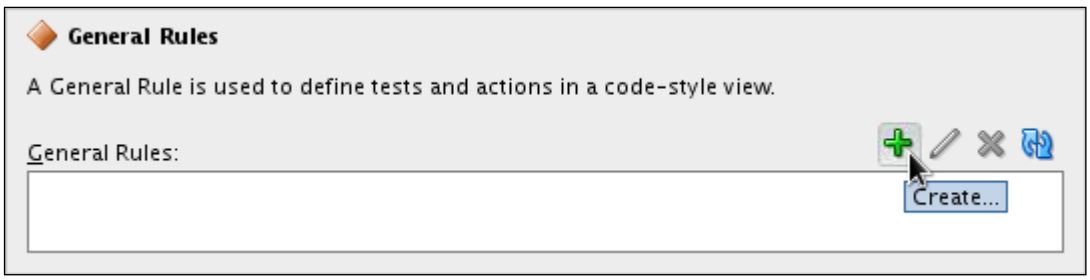
      Note: Renaming the ruleset is not strictly required, but applying a meaningful name makes future maintenance easier.

Creating the `abovelimit` Rule in the `approvalruleset` Ruleset

In this section, you create a rule in `approvalruleset`. The rule checks to see if the total price is greater than the approval limit and if so, it sets the required approval Boolean result to `true`.

Note: The structure that contains the result is called `approvalandiscount`, which contains a `discount` element. The `discount` element is not used in this course application.

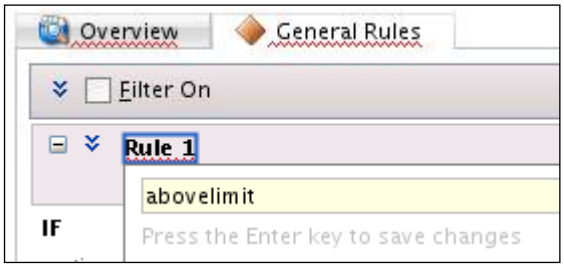
6. Create a new rule.
   a. In the General Rules pane, click Create.



   A new rule named `Rule1` is created.

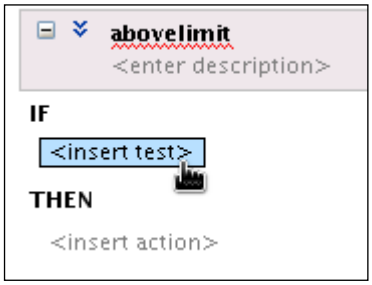
b. Click the new rule and rename it `abovelimit`.

Note: Remember to press Enter after entering the new name.



7. Create the test (or condition) in the IF section of the `abovelimit` rule.

a. Click the `<insert test>` entry.



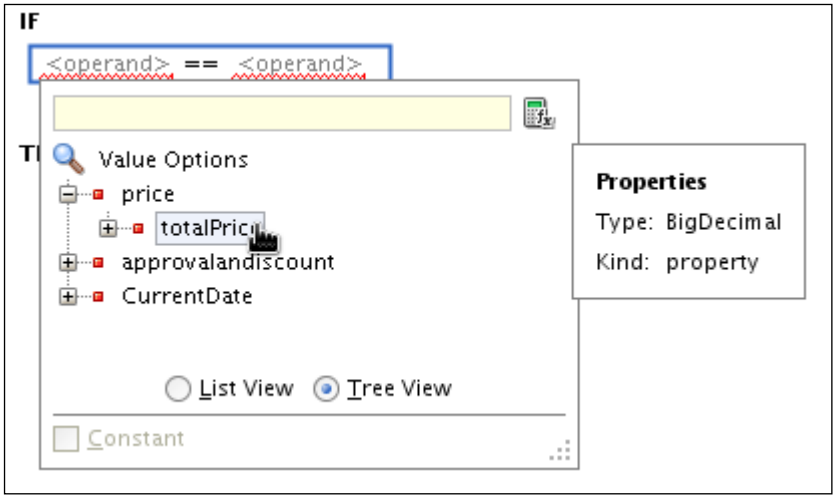
b. Select simple test.

Note: This creates a conditional structure with the format `<operand>` == `<operand>`. It also creates another `<insert test>` below it. Each part needs to be clicked to specify its value.

c. Set the left-side `<operand>`.

1) Click the `<operand>` text.

2) Select `price.totalPrice` from the drop-down menu.



d. Set the operator.

1) Click the "==" text.

2) Select the greater-than (>) sign from the drop-down menu.

e. Set the right-side `<operand>`.

1) Click the `<operand>` text.

2) Select `approval_limit` from the drop-down menu.

Note: `approval_limit` is the global that you created and set to the value `4000`.

8. Create the action in the THEN section.

a. Click the `<insert action>` text.

b. Select `assert new` from the drop-down menu.

Note: The `assert new` action creates a new fact object that needs to be defined and initialized to hold the desired values. In addition, another `<insert action>` is added below the one being created. This allows you to create multiple actions for a rule. In this case, you create one action.

c. Set the object created by the `assert new` action.

1) Click the `<target>` text.

2) Select the `approvalandiscount` option from the drop-down menu.

Note: The `approvalandiscount` structure (as defined in the XSD that is imported to define the output result structure for the Business Rule decision service) has child elements called *properties*. You need to set their values for the present condition.

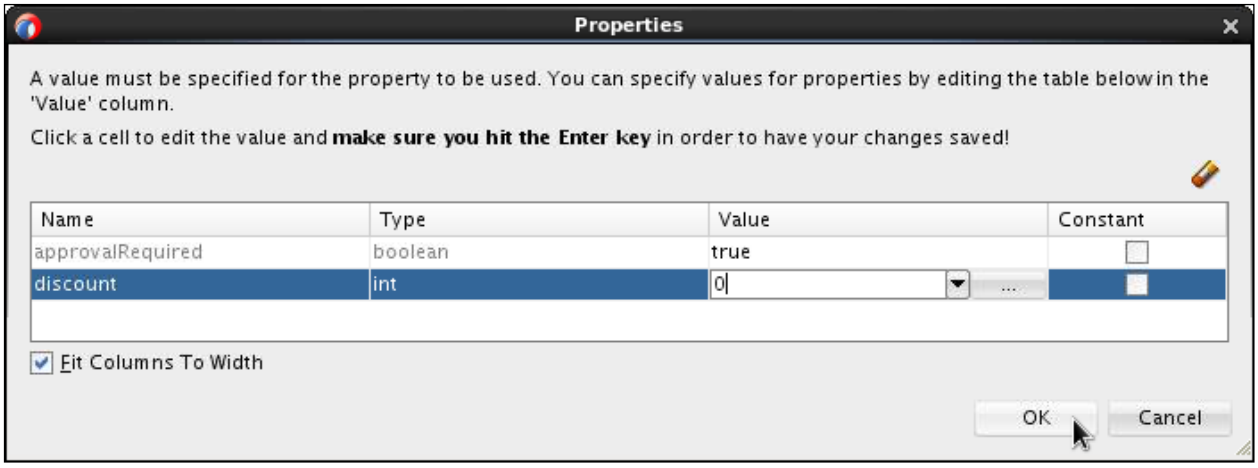
d. Set the `approvalandiscount` property values.

1) Click the `<edit properties>` text.

e. Set the `approvalRequired` property value to `true` (which can be selected from the drop-down menu in the Value column).

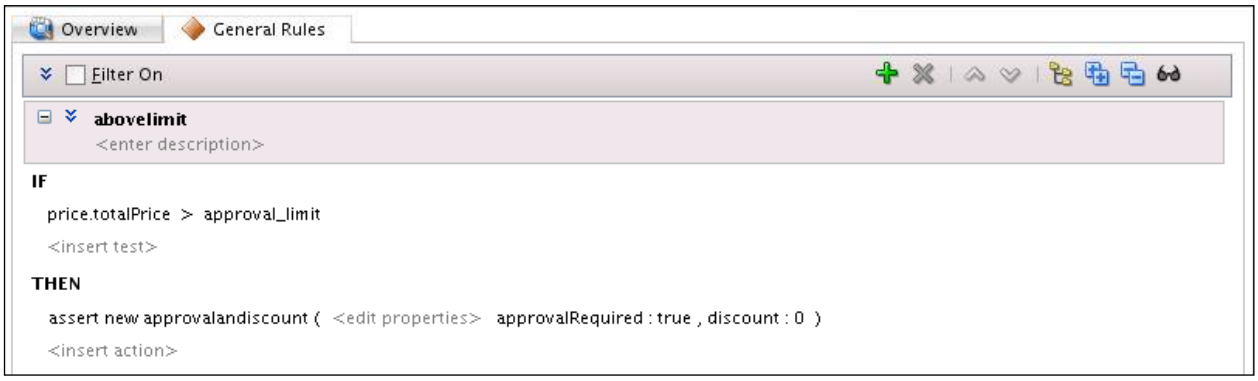
1) Enter the value `0` (zero) in the discount Value column.

2) Verify your work and click OK.



The Properties pane is closed.

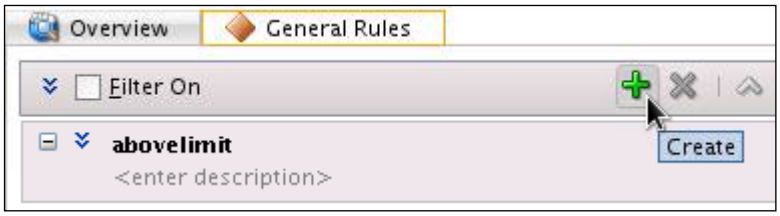
f. Verify and save your work.



Creating the `withinlimit` Rule in the `approvalruleset` Ruleset

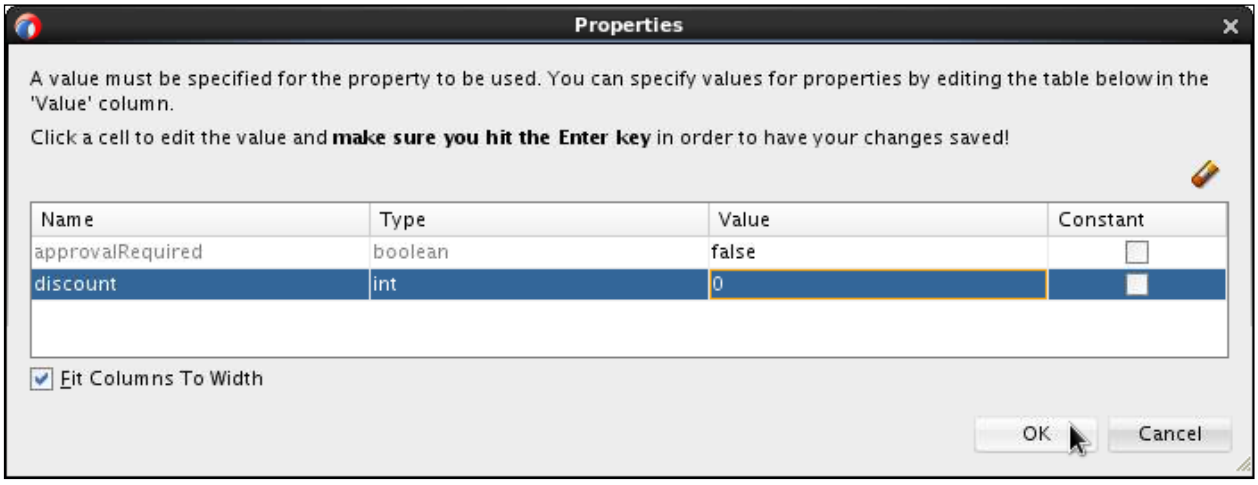
In this section, you create a `withinlimit` rule that tests the condition where the total price is less than or equal to the approval limit, in which case the `approvalRequired` property should be set to a value of `false`.

9. Create a new rule.
   a. In the General Rules pane of the `approvalruleset` tab, click Create.

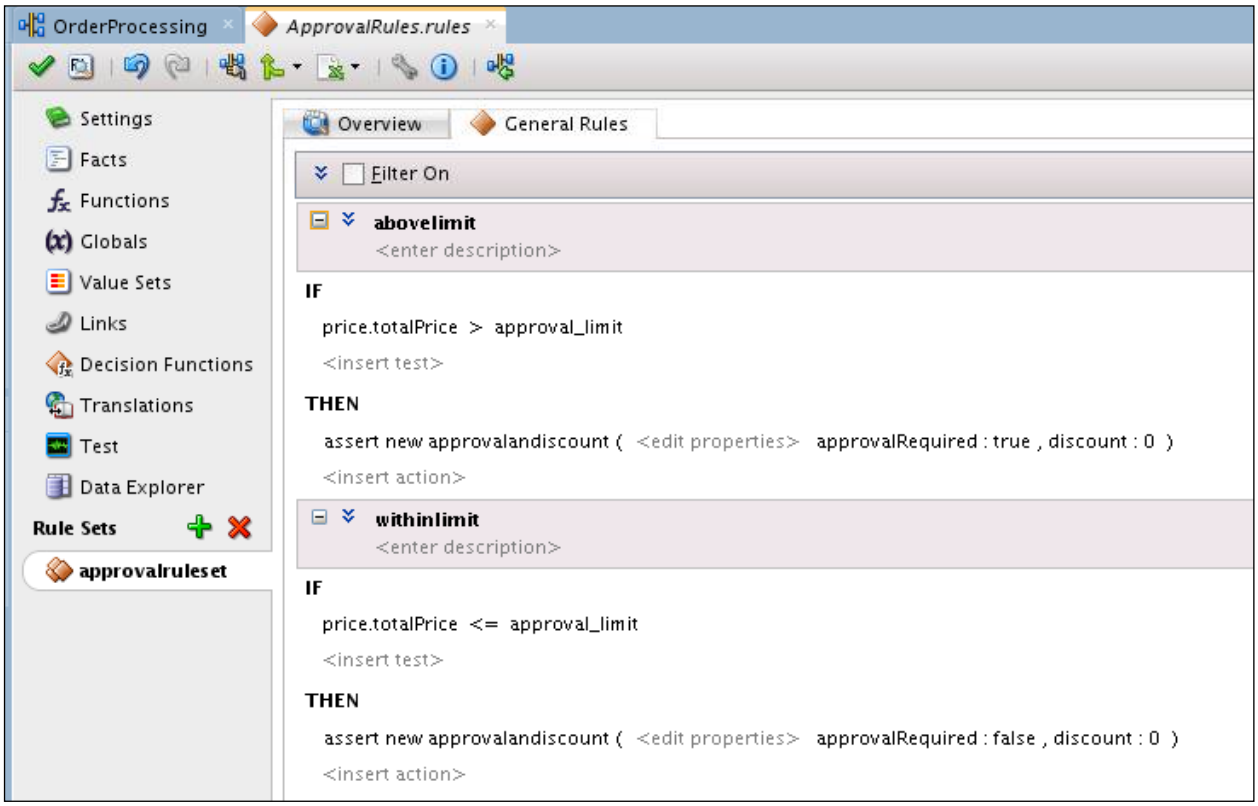


   b. Click the new rule name and rename it `withinlimit`.
   c. Configure the test (condition) where the conditional expression is expressed as price.totalPrice <= approval_limit.

d. In the THEN section, click `<insert action>`, select the `assert new approvalandiscount` object (as you did for the `abovelimit` rule), and set the properties so that `approvalRequired` is `false` and discount is `0` (zero).

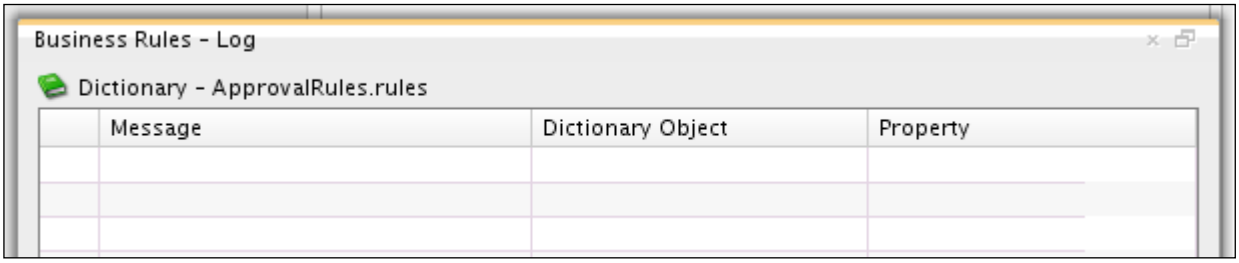


e. Verify your work.



10. In the `ApprovalRules.rules` window, click the Validate icon on the toolbar to ensure that the rule dictionary contains valid definitions.

The Business Rule Validation – Log opens near the bottom of the pane. A successful validation results in a blank log.

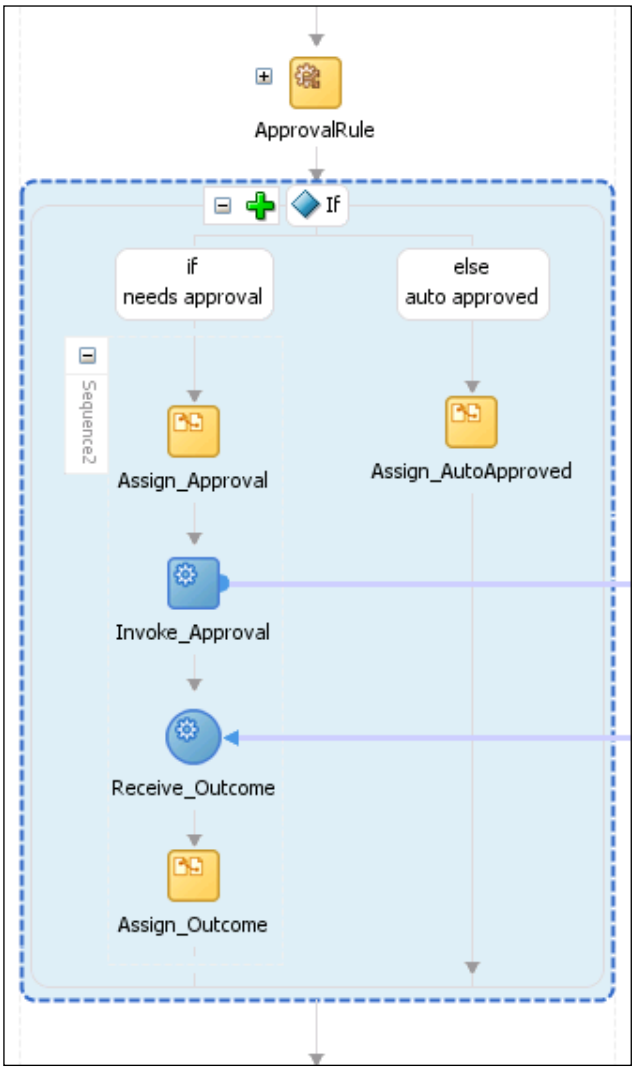


11. Save your work and close `ApprovalRules.rules`.

Practice 12-2: Implementing Business Rules in a BPEL Process

Overview

In this section, you create the BPEL process and configure a series of activities. You first add a business rule to determine whether or not the order data requires manual approval. You then add and configure activities so that those orders that require approval are passed to the Approval application that you completed in the previous practice. The outcome of that approval is then stored in a variable and passed to additional activities (not shown here) for additional processing.



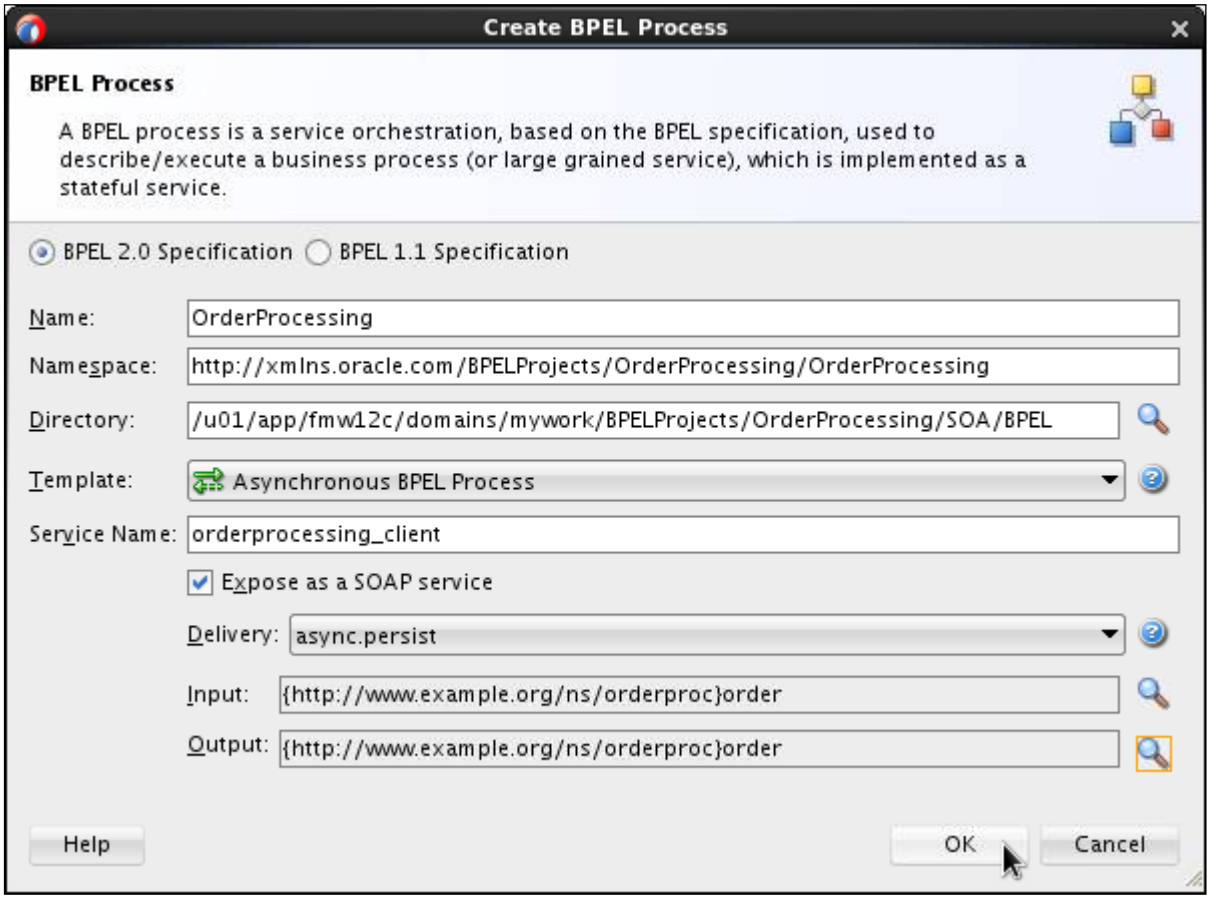
Assumptions

This practice assumes that you have completed Practice 12-1 successfully.

Tasks

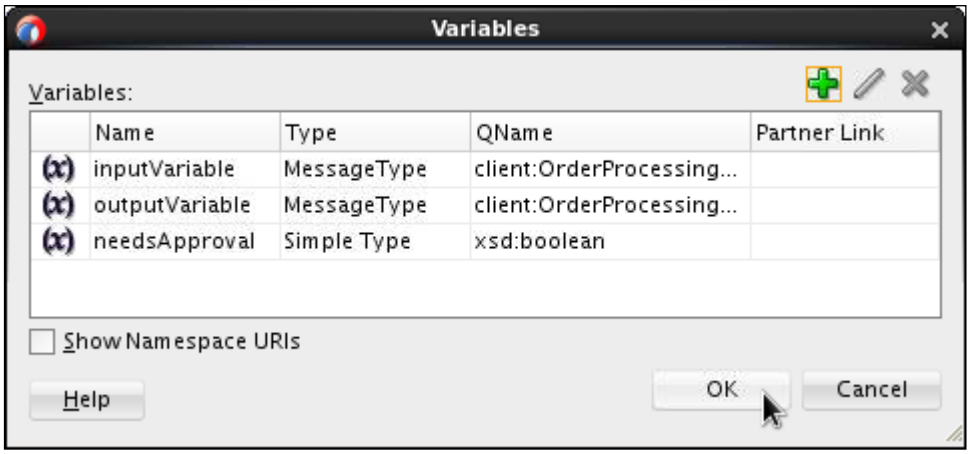
Creating and Configuring a BPEL Process

1. Create and configure a BPEL service component.
   a. Add a BPEL process to the Components swimlane in the `composite.xml` editor.
   b. Name the process `OrderProcessing`.
   c. Accept the default Asynchronous Template.
   d. To define the input, perform the following steps:
      1) Click the Browse icon to access the Type Chooser window. Then, click the Import Schema file icon to import the file `/home/oracle/labs/files/xsd/orderProcessing.xsd` into the project.
      2) Deselect the 'Maintain original directory structure for imported files' check box.
      3) Select the `order` element.
      4) OK.
   e. To define the output, browse to the `order` element of `orderProcessing.xsd`.
   f. Verify your work and click OK.

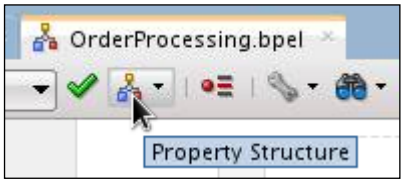


   g. Save your work.

2. In the composite overview window, wire the BPEL process to ApprovalService.
   ApprovalService is associated with the BPEL process as a partner link.
3. Wire the BPEL process to the ApprovalRules business rule component.
4. Right-click the BPEL component and select Edit.
5. Create a process-level Boolean variable named `needsApproval`. Hint: Click the Create button, Set the variable name to needsApproval, select Type, click Browse icon, expand XML Schema Simple Types, select Boolean and click OK.



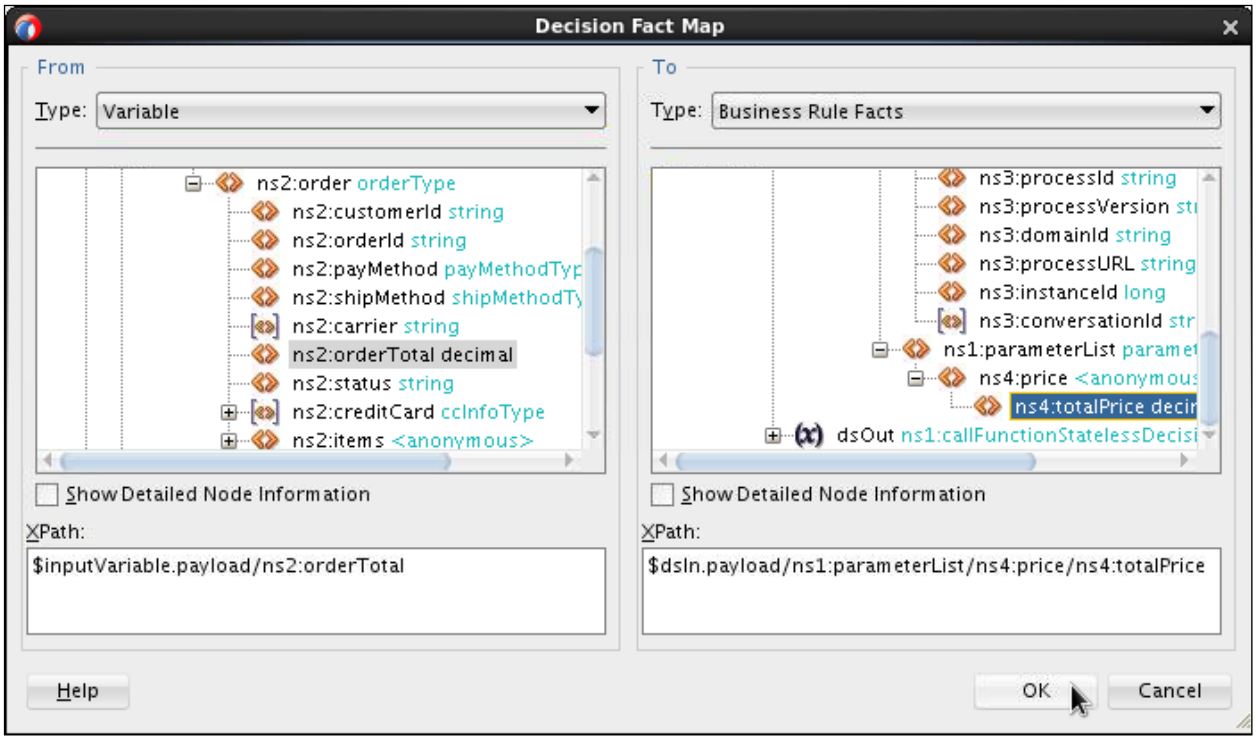
Note: The preceding screenshot was produced by clicking the Variables icon on the Process Scope toolbar. If you select the Property Structure > Variables option from the BPEL editor toolbar menu (see the following), the screen differs, but you still are able to create the desired variable.



6. Create and configure a Business Rules component.
   a. Add a Business Rule SOA Component to the process between the Receive and the callback activities.
   b. Rename the rule `ApprovalRule`.
   c. Right-click ApprovalRule and select Edit.
      The Business Rule settings window opens.
   d. Specify the dictionary by selecting ApprovalRules from the drop-down menu.
      Additional configuration options appear. The Service is set to ApprovalRulesService, and the Operation is set to “Execute function and reset the session.”
   e. Assign Input Facts.
      1) Click the Create icon.
      2) The Decision Fact Map window opens.
      3) In the left pane, navigate to the `orderTotal` inputVariable.

4) In the right pane, navigate to the `totalPrice` element of the `dSIn – payload – callFunction… parameterList – price` variable.

5) Verify your work and click OK.



f. Assign Output Facts.

1) Click the Assign Output Facts tab.

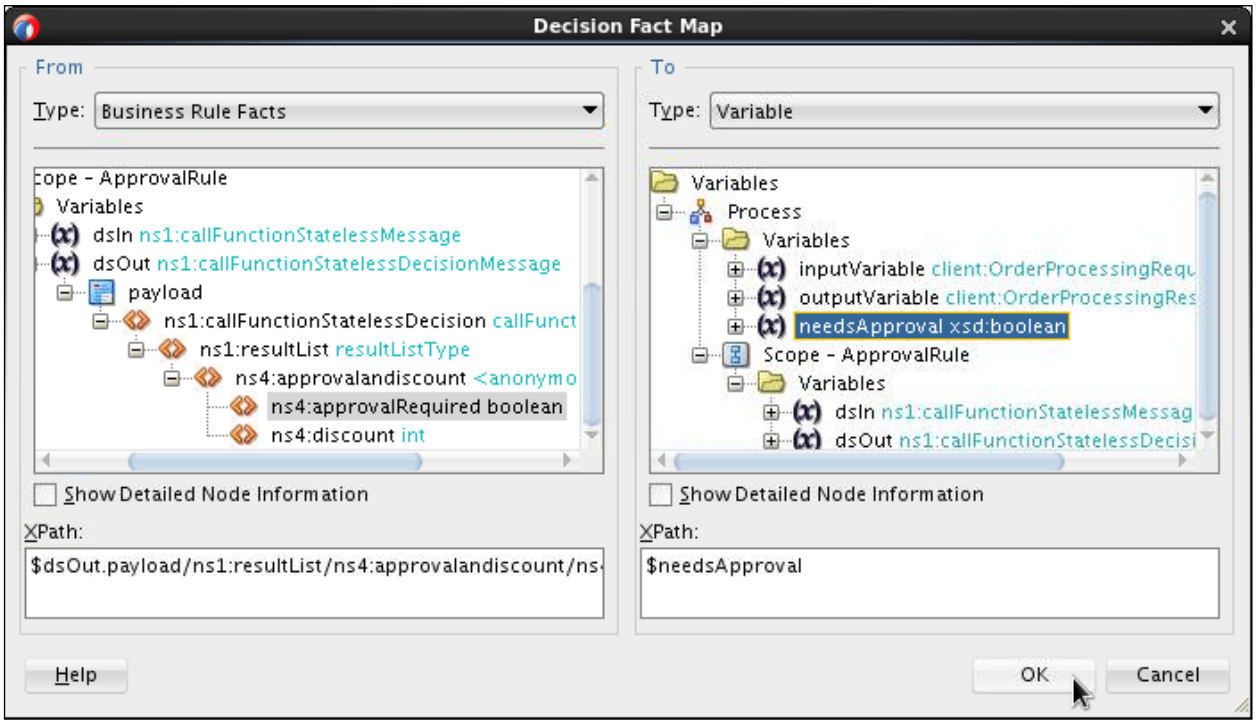
2) Click the Create icon.

3) The Decision Fact Map opens.

4) In the left pane, navigate to the approvalRequired element of the dsOut – payload – callFunction… - resultList - approvalanddiscount variable.

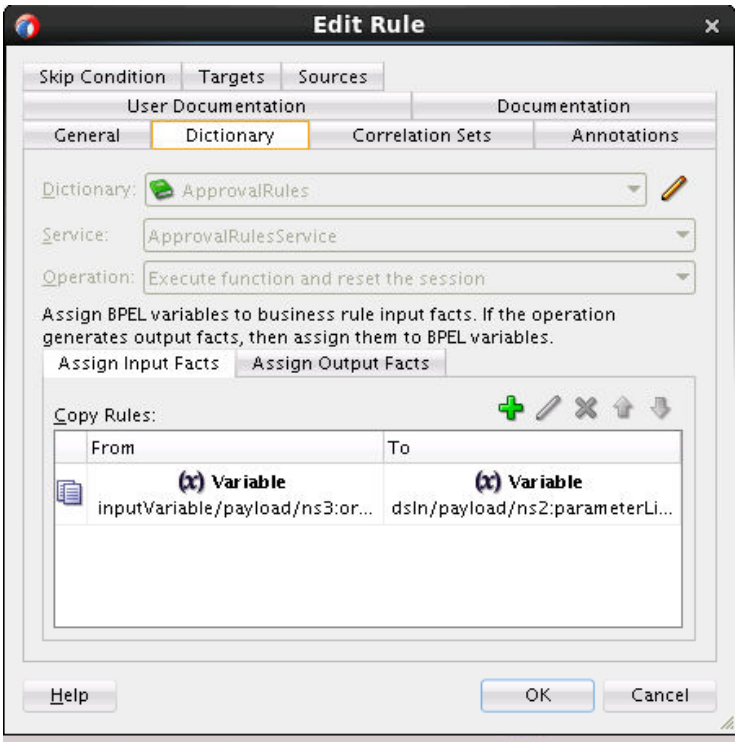
5) In the right pane, navigate to the process variable `needsApproval`.

6) Verify your work and click OK.

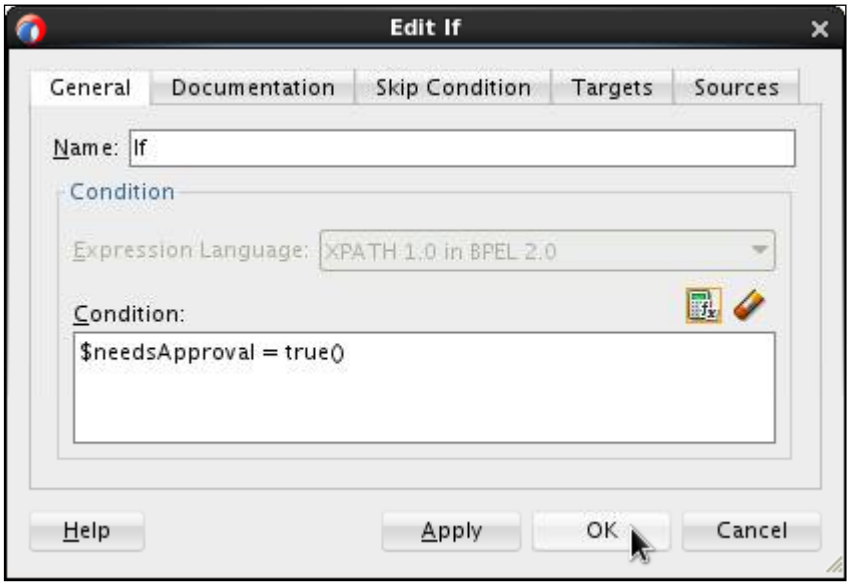


You are returned to the Business Rule dialog box.

g. Verify your work and click OK.



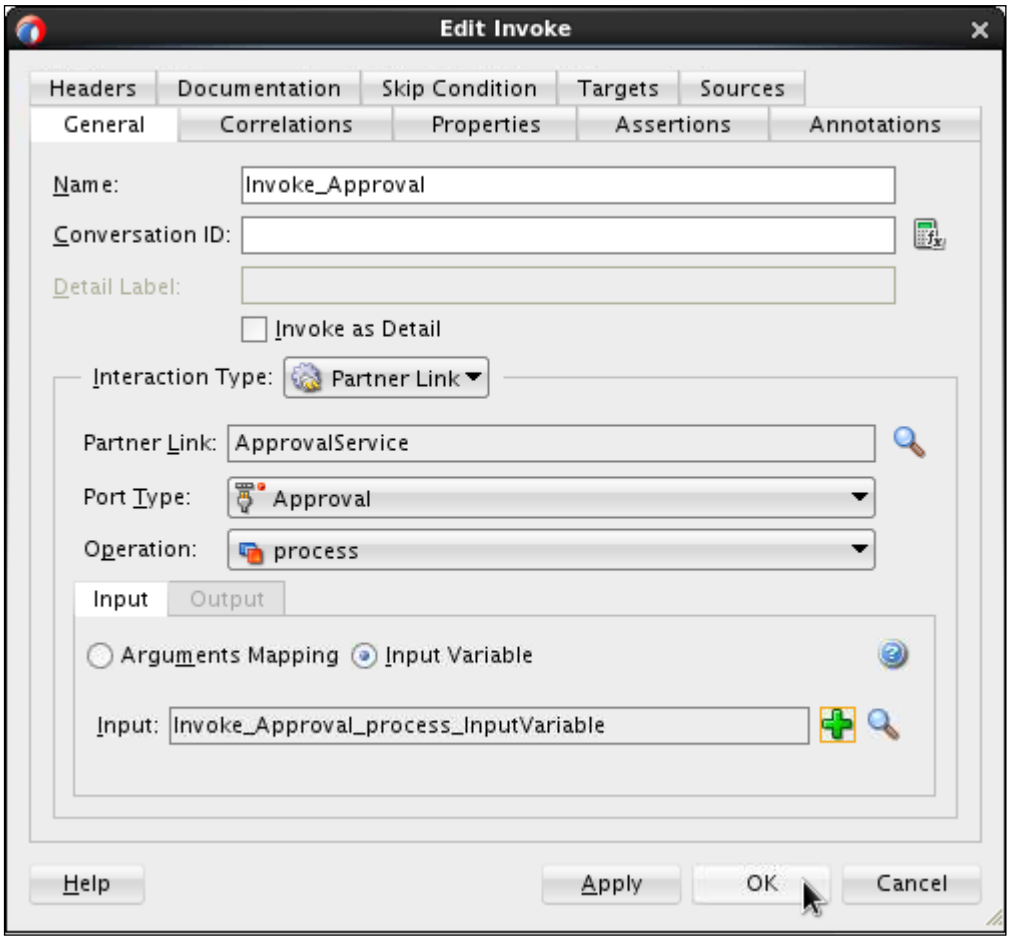
7. Create and configure an If activity.
   a. Add an If activity following the Business Rule activity.
   b. Set the conditional expression of the If to test whether `$needsApproval=true()`.
   c. Label the branches `'if needs approval'` and `'else auto approve'`.



8. Add an Assign activity to the <`else auto approve`> branch.
   a. Add an Assign activity to the <`else auto approve`> branch.
   b. Name the activity `Assign_AutoApproved`.
   c. Right-click `Assign_AutoApproved` and select Edit.
   d. Use Expression Builder to assign the value `'APPROVED'` to the status element of `inputVariable`.
   e. Verify and save your work.

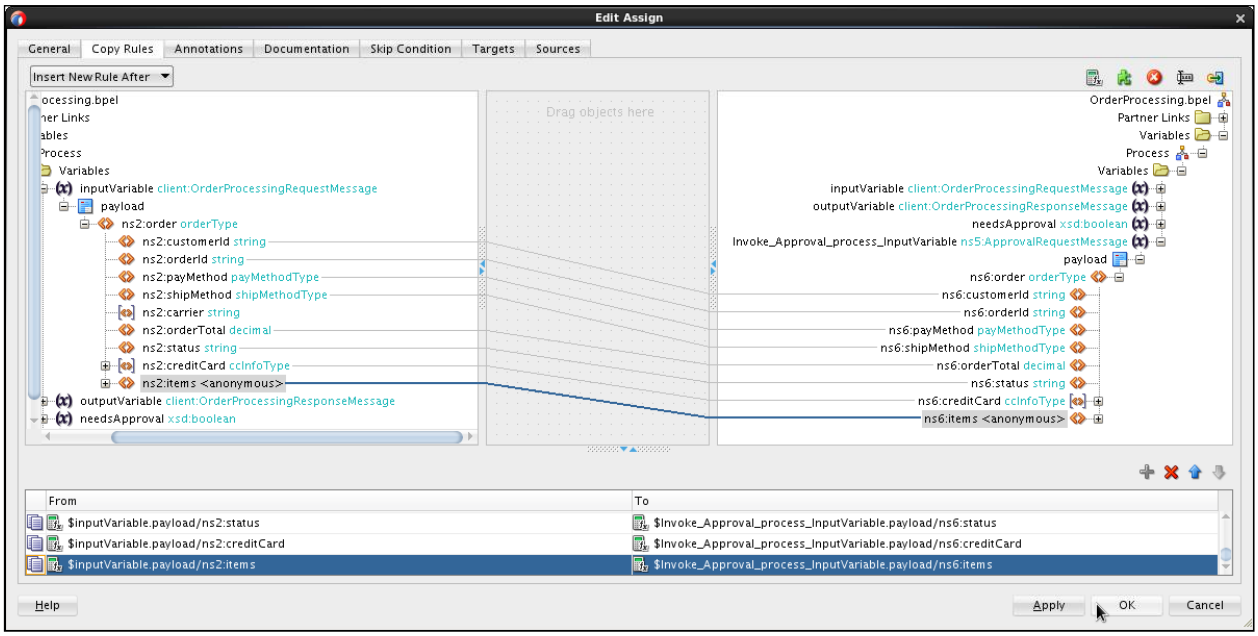
9. Create and configure an Invoke activity in the `<if>` branch to call the Approval application.
   a. Add an Invoke activity to the `<if>` branch.
   b. Wire the Invoke activity to the partner link `ApprovalService`.
      The Edit Invoke window is displayed.
   c. Name the Invoke `Invoke_Approval`.
   d. Create a new Input variable.

e. Verify and save your work.



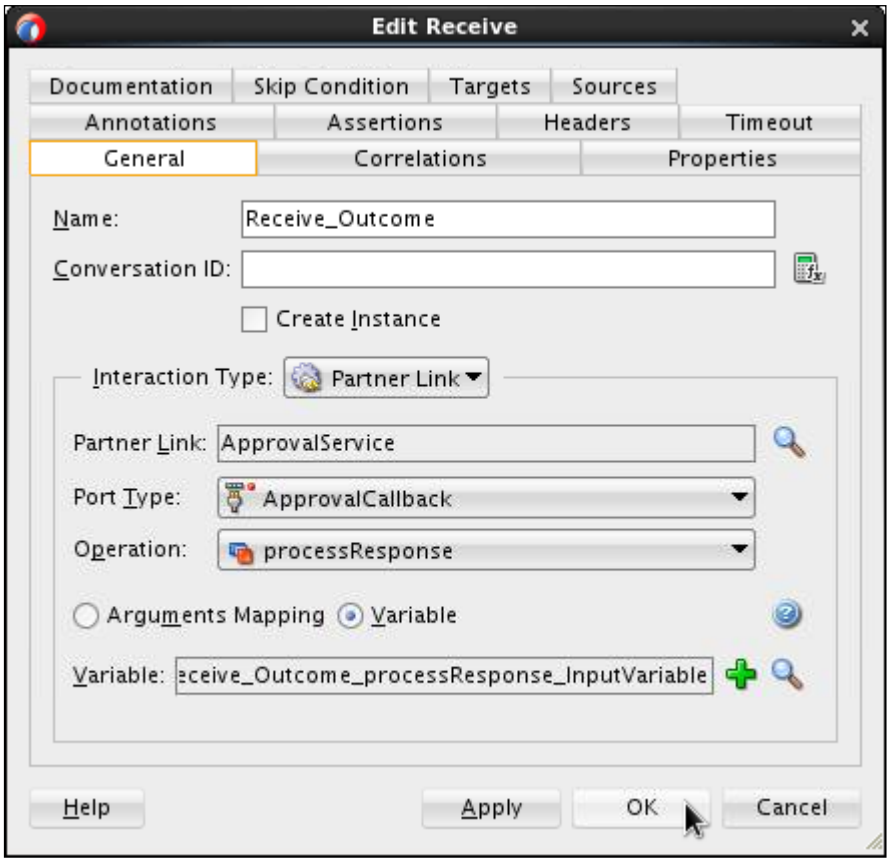
10. Create and configure an Assign activity.
    a. Add an Assign activity just above `Invoke_Approval`.
    b. Name the activity `Assign_Approval`.
    c. Right-click the new activity and select Edit.
    d. Map each individual element of `inputVariable` (except `carrier`) to its corresponding element in `Invoke_Approval_process_InputVariable`.

e. Verify and save your work.



11. Create and configure a Receive activity.
    a. Add a Receive activity just below `Invoke_Approval`.
    b. Right-click the new activity and select Edit.
    c. Name the activity `Receive_Outcome`.
    d. Use the Browse button to choose the partner link `ApprovalService`.
    e. Auto-create the variable `Receive_Outcome_processResponse_InputVariable`.

f. Verify and save your work.



12. Create and configure an Assign activity to save the approval outcome to `inputVariable`.
    a. Add an Assign activity just below `Receive_Outcome`.
    b. Name the activity `Assign_Outcome`.
    c. Right-click the new activity and select Edit.
    d. Map the `Receive_Outcome` variable element `status` to the corresponding element in `inputVariable`.
    e. Verify and save your work.

13. Create and configure an Assign activity to populate the business process output variable, which is returned to the calling client.
    a. Add an Assign activity just above `callback_Client`.
    b. Name the activity `Assign_Output`.
    c. Right-click the new activity and select Edit.
    d. Map `inputVariable.payload` to `OutputVariable.payload`.
    e. Verify and save your work.

Practice 12-3: Deploying and Testing the Application

Overview

In this practice, you deploy and test the `OrderProcessing` composite application.

Assumptions

This practice assumes the following:

- You have completed Practice 9 for the lesson titled “Implementing Human Workflow and Notifications” successfully, and that the `Approval` application is still deployed to the application server.
- You have completed all work in Practice 12-2 successfully.

Tasks

Deploying the OrderProcessing Composite Application

1. Deploy the `OrderProcessing` application to IntegratedWebLogicServer.

   Remember to select the “Overwrite any existing composites with the same revision ID” check box.

Testing the Composite Application

In this section, you submit an order whose total is greater than 4000 so that the `ApprovalRules` business rule returns a result that requires manual approval of the order. As in the previous practice, as employee `jcooper`, you check your email to read the notification about the assigned task, and then log in to the Worklist application to approve the order.

2. Open the Oracle Enterprise Manager page (http://localhost:7101/em).
   a. In the Target Navigation pane, expand the SOA > soa-infra > node in the tree. Right-click soa-infra and select Home > Deployed Composites.
   b. Click the OrderProcessing [1.0] link.
   c. On the “OrderProcessing [1.0]” home page, click Test.
   d. On the Test Web Service page, scroll down to the Input Arguments section to the Request tab and select XML view.
   e. Replace the XML data with the contents of the file `/home/oracle/labs/files/xml_in/businessRules_input.xml`.
   f. Click Test Web Service.
   g. On the Response tab, click the Launch Flow Trace link.

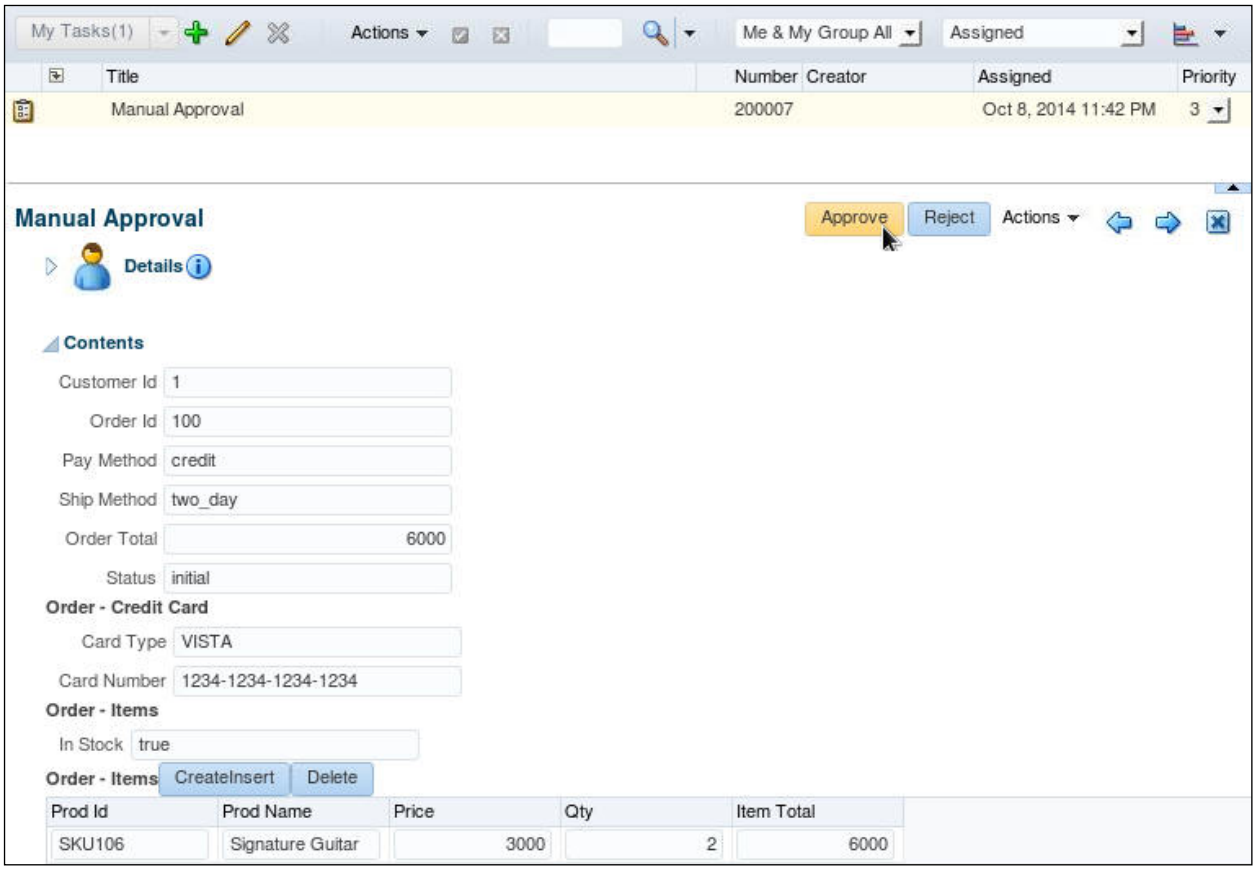
h. Which entries have completed? Which are still running? Why?

Instance	Type	Usage	State
orderprocessing_client_ep	Service	Service	Completed
OrderProcessing	BPEL		Running
ApprovalRules	Decision		Completed
ApprovalService	Reference	Reference	Completed
approval_client_ep	Service	Service	Completed
Approval	BPEL		Running
ManualApproval	Workflow		Running

3. Manually approve the order in the Worklist application:
   a. Access the Oracle BPM Worklist application at http://soa12c.example.com:7101/integration/worklistapp.
   b. Log in to the application as user `jcooper` with the password `welcome1`.

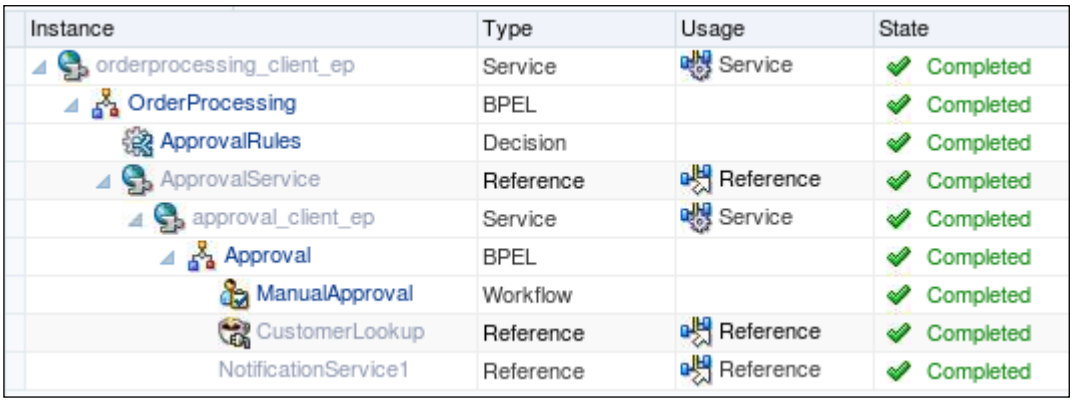
      On the Worklist home page, the Inbox is selected and the Manual Approval task is present in the top-right pane.
   c. Click the Manual Approval task entry.

      The order details are displayed in the lower-right pane.
   d. Click Approve.



   e. Log out of the Worklist application and close the webpage.

4. Relaunch the Flow Trace and verify that the process has now completed.



Instance	Type	Usage	State
orderprocessing_client_ep	Service	Service	Completed
OrderProcessing	BPEL		Completed
ApprovalRules	Decision		Completed
ApprovalService	Reference	Reference	Completed
approval_client_ep	Service	Service	Completed
Approval	BPEL		Completed
ManualApproval	Workflow		Completed
CustomerLookup	Reference	Reference	Completed
NotificationService1	Reference	Reference	Completed

Additional Optional Tests

If you have time, you can submit some additional orders. Consider these scenarios:

5. Initiate a test with an order that requires manual approval and, at the manual approval stage, reject it.

6. In the Flow Trace, examine the callbackClient payload.

Practice 12-4: Adding a Shipping Rules Business Rule [Optional]

Overview

In this practice, you modify, redeploy, and test the `OrderProcessing` composite application. You add a second Business Rules component, which (if the order is approved) determines the shipping company for the order based on the delivery options included in the order data.

Assumptions

This practice assumes that you have completed Practice 12-3 successfully, and that the `OrderProcessing` application is still deployed to the application server.

Tasks

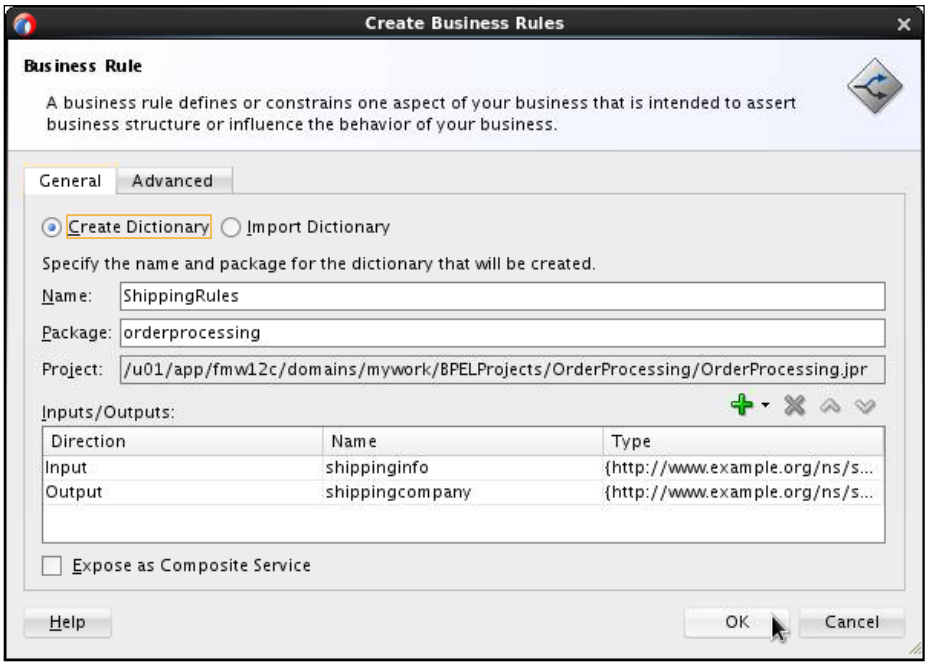
Creating and Configuring the `ShippingRules` Business Rules Component

In this section, you create a second Business Rules component. This component uses a Decision Table to assign a shipping carrier based on the shipping method specified in the order detail. You configure the decision service interface by defining the service name and specifying the input and output fact structures, which are drawn from an XSD file that you import into the project.

1. Create and configure another Business Rules component in the assembly model.
   a. In `composite.xml`, add a Business Rules component to the Components swimlane.
   b. Name the component `ShippingRules`.
   c. Add input.
      1) Select Create > Input.
      2) Import `/home/oracle/labs/files/xsd/ordershipment.xsd`.

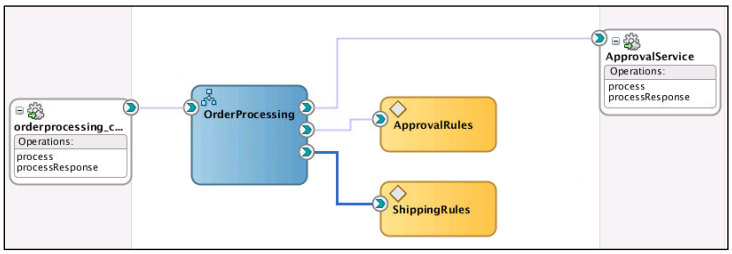
         Remember to deselect the “Maintain original directory structure for imported files” check box.
      3) Select the `shippinginfo` element.
   d. Add output.
      1) Select Create > Output.
      2) Navigate to the `shippingcompany` element of `ordershipment.xsd`.
   e. Define a name for the decision service.
      1) Click the Advanced tab.
      2) Name the service `ShippingRulesService` (*case-sensitive*).

f. Return to the General tab. Verify and save your work.



g. Wire the OrderProcessing BPEL component to the ShippingRules component.

h. Verify and save your work.



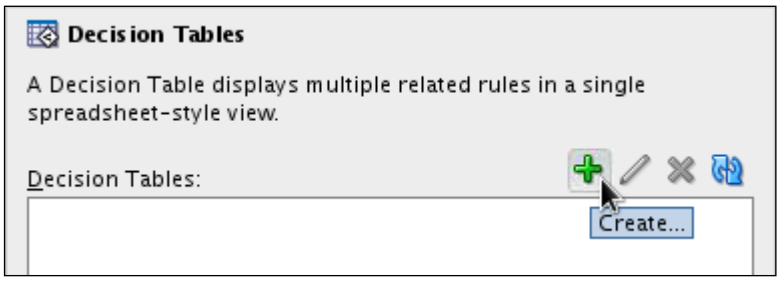
Defining the `ShippingRules` Decision Table

In this section, you edit the `ShippingRules` Business Rules component and create the dictionary definitions, including the Decision Table and its related rules.

2. Right-click the ShippingRules component and select Edit.
   The `ShippingRules.rules` window opens.
3. Rename the default ruleset `shippingruleset`.

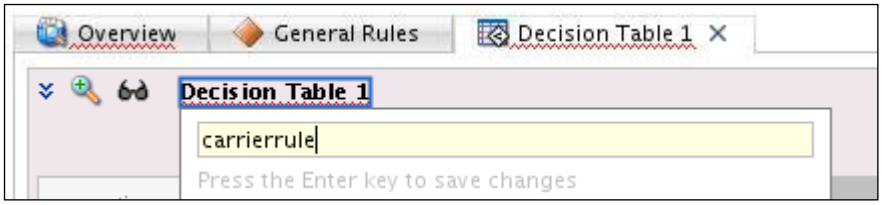
Creating a Decision Table

4. Create and configure a Decision Table.
   a. Click the Create icon in the Decision Tables pane.



   b. Rename the `DecisionTable 1` rule `carrierrule`.

      Tip: Press Enter to save the new name.

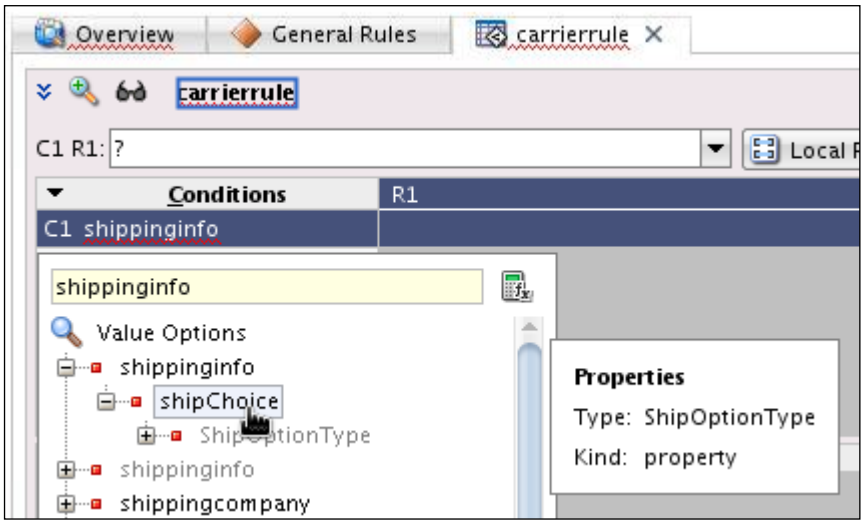


Creating Rules in the Decision Table

In this section, you define three rules in the Decision Table. The rules are created after you define the condition. After a rule comparison value is set for the condition, you can specify the action.

5. Create the Decision Table condition.
   a. In Decision Table Conditions, click the `<insert condition>` text.

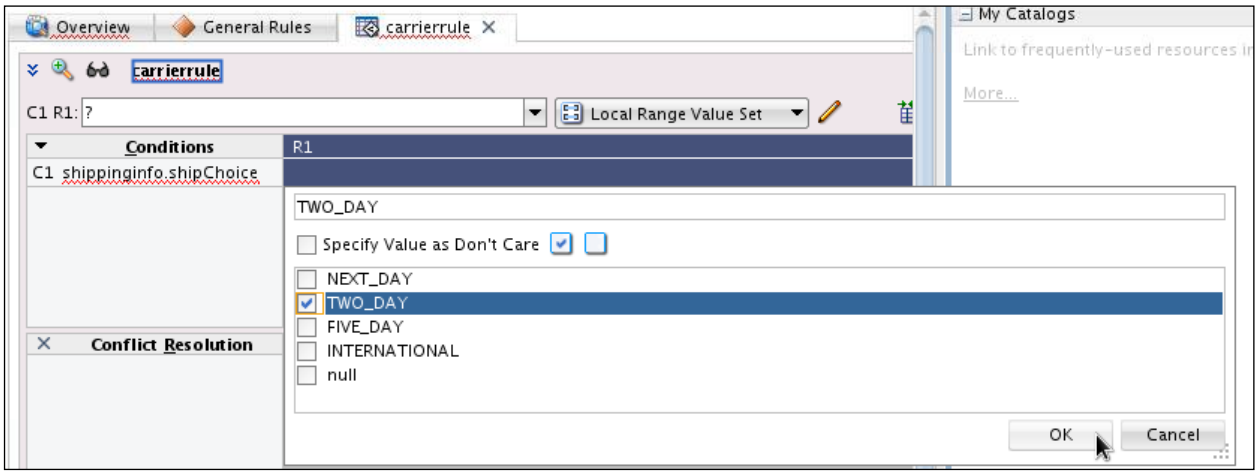
b. From the Conditions drop-down menu, double-click `shippinginfo` and click `shipChoice`.



Note: Adding and setting the condition creates the first rule template in the column titled R1.

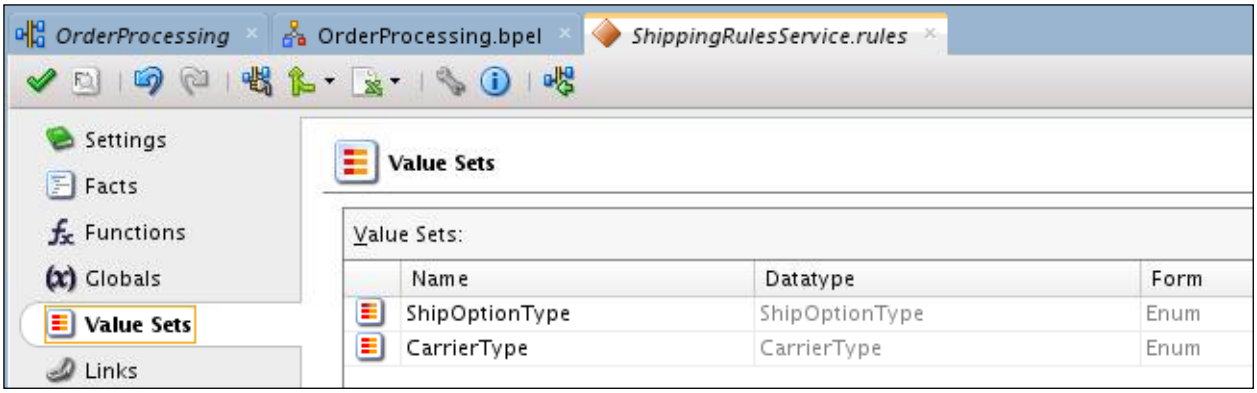
6. To set the comparison value TWO_DAY shipping in the rule for R1, perform the following steps:

a. In the R1 column and the C1 condition row, click the cell and select the `TWO_DAY` check box from the drop-down menu.



Note: The value `TWO_DAY` is added to the selected cell.

Have you considered where the list of values for the shipChoice element comes from? If you are curious, take a look at the Value Sets definitions in the `shippingRules.rules` dictionary, as in the following example:



Ask yourself, how did these value set definitions get created? Do their names offer any insight?

Apart from the fact that these value sets have an XML Fact type definition (refer to the Facts section in the `ShippingRules.rules` definition), these XML Facts were derived from XML elements, in `ordershipment.xsd`, which were used to define the Input and Output variables for the ShippingRules Business Rules component.

If you examine the `ordershipment.xsd` file, you can find the definitions for these types shown with the associated enumerations. For example, the `shipOptionType` definition in the XSD is as follows:

shipOptionType restricts xsd:string	
enumeration	next_day
enumeration	two_day
enumeration	five_day
enumeration	international

If you are curious, consider editing the corresponding ShipOptionType value set to view how its definition resembles the XSD element definition.

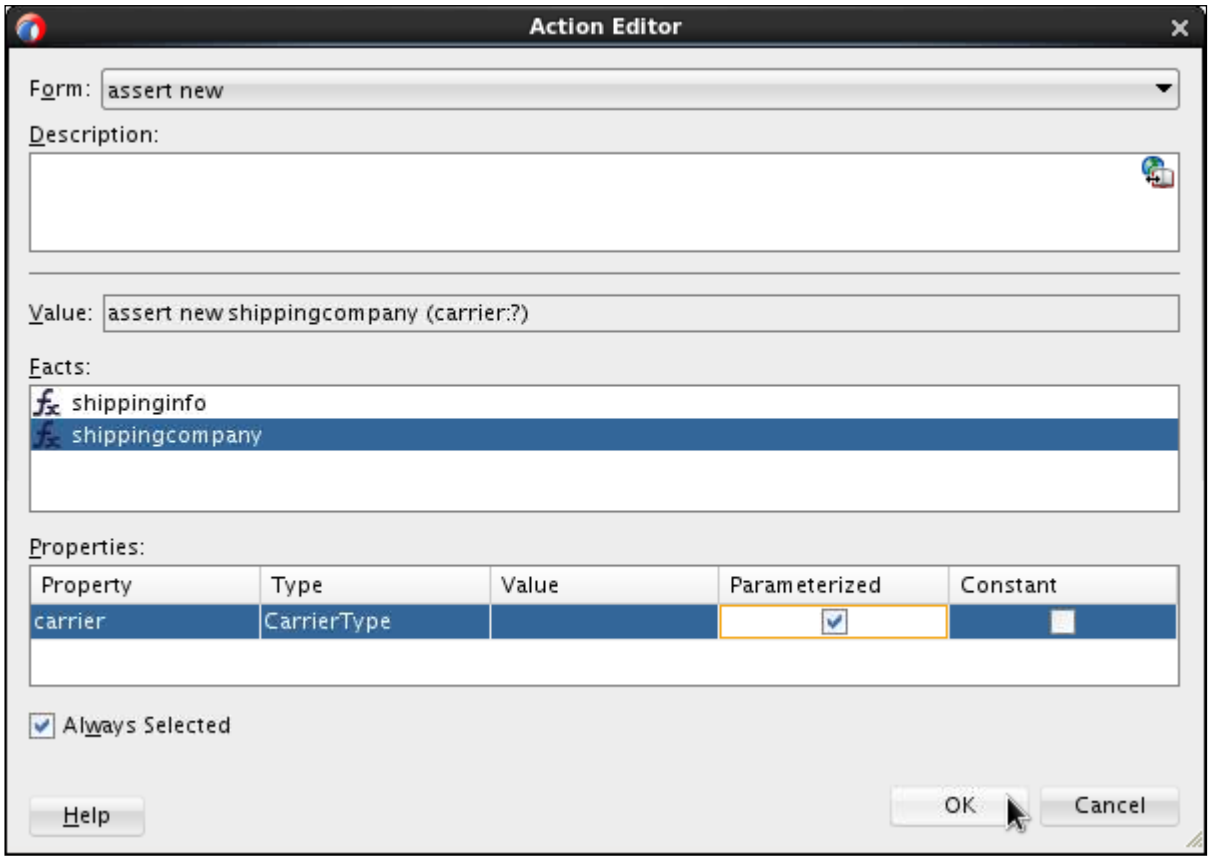
b. Create the action for the rule and set the action values.

1) In the first column under the Actions section, click the `<insert action>` text.
2) Double-click `assert new`.

   The Action Editor window opens.
3) Select the `shippingcompany` entry in the Facts section.
4) In Properties for the `carrier` property, select Parameterized.

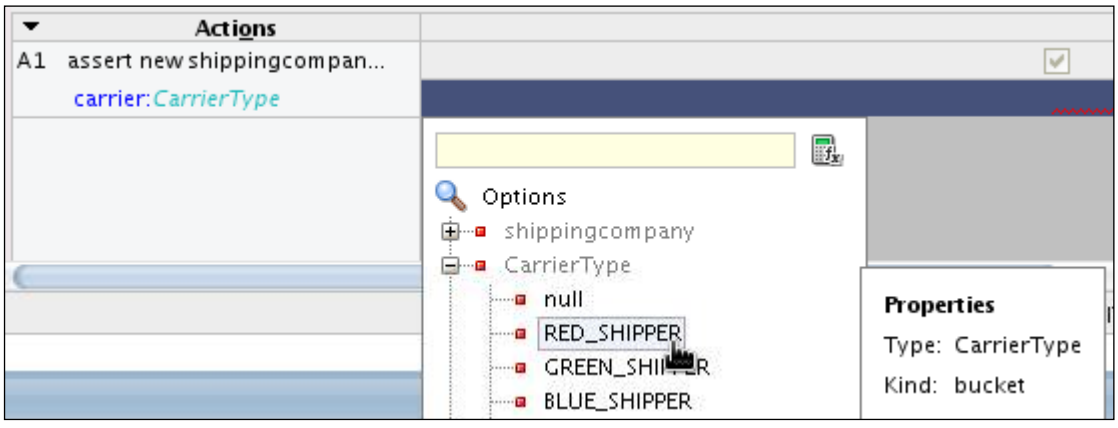
   Note: When you select Parameterized, a cell is added for each rule in the Decision Table, and the property can be set on a per-rule basis.
5) At the bottom of the window, select Always Selected.

6) Verify your work and click OK.



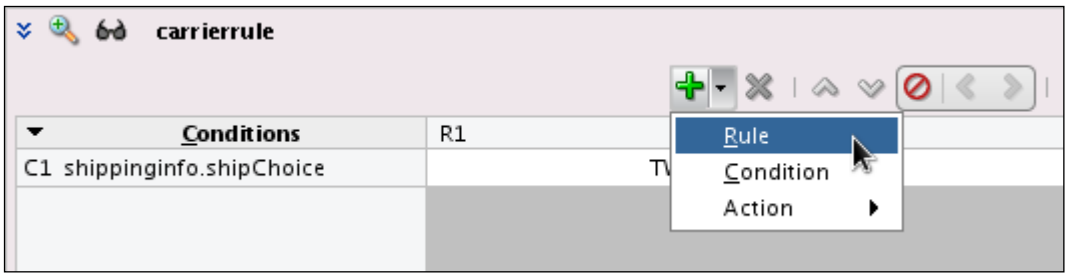
Note: The Value field is constructed from the selections you make. The Always Selected check box ensures that the action is always selected for each rule.

c. In the A1 row that is created for the `assert new shippingcompany` action, click the cell under the R1 column and select the CarrierType `RED_SHIPPER` value.



Note: The configuration that you have performed so far in the Decision Table effectively allows the business rule to apply the logic "If the shipping choice is `TWO_DAY`, then assign carrier `RED_SHIPPER`." The remaining steps configure similar logic for the other shipping choices and carriers defined in the value sets that you examined earlier.

7. Create the second rule in the Decision Table for the Green Shipper.
   a. On the toolbar at the top of the Decision Table, click the Create icon and select Rule.



   A new column is added to the table.
   b. To set the condition in the new column, click the cell in the C1 row and select the `FIVE_DAY` check box.
   c. In the A1 row for the new column, click the intersecting cell and select the CarrierType `GREEN_SHIPPER` from the drop-down menu.
8. To create the third rule in the Decision Table for the Blue Shipper, perform the following steps:
   a. On the toolbar at the top of the Decision Table, click the Create icon and select Rule.
   b. To set the condition comparison value in the new rule column, click the cell in the C1 row and select both the `NEXT_DAY` and `INTERNATIONAL` check boxes.
   c. To set the action values, click the A1 cell in the new column and select the CarrierType `BLUE_SHIPPER` from the drop-down menu.
9. Verify your work.

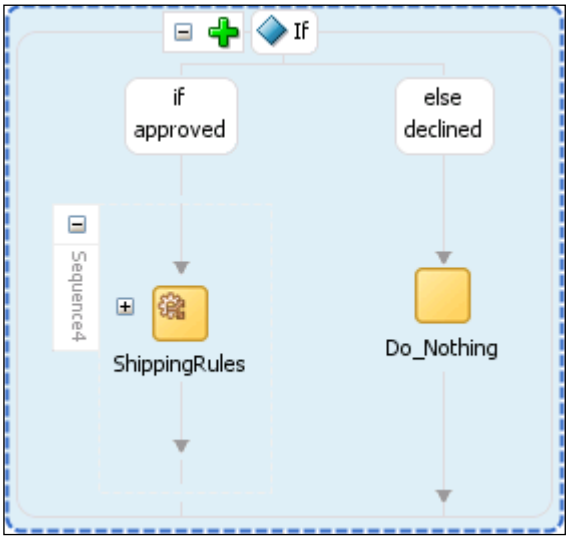
Conditions	R1	R2	R3
C1 shippinginfo.shipChoice	TWO_DAY	FIVE_DAY	NEXT_DAY,INTERNATIONAL
Conflict Resolution			
Actions			
A1 assert new shippingcompany	☑	☑	☑
carrier:CarrierType	CarrierType.RED_SHIPPER	CarrierType.GREEN_SHIPPER	CarrierType.BLUE_SHIPPER

10. Save your work.
11. Close the `ShippingRules.rules` window.

Practice 12-5: Modifying the BPEL Process [Optional]

Overview

In this section, you modify the BPEL process that was created earlier in this practice. You test whether or not the order was approved. If the order is approved, a business rule activity is employed to determine an appropriate carrier for the order. The order data is updated accordingly and returned to the calling client. The following image shows the portion of the business process that you are about to build:



Assumptions

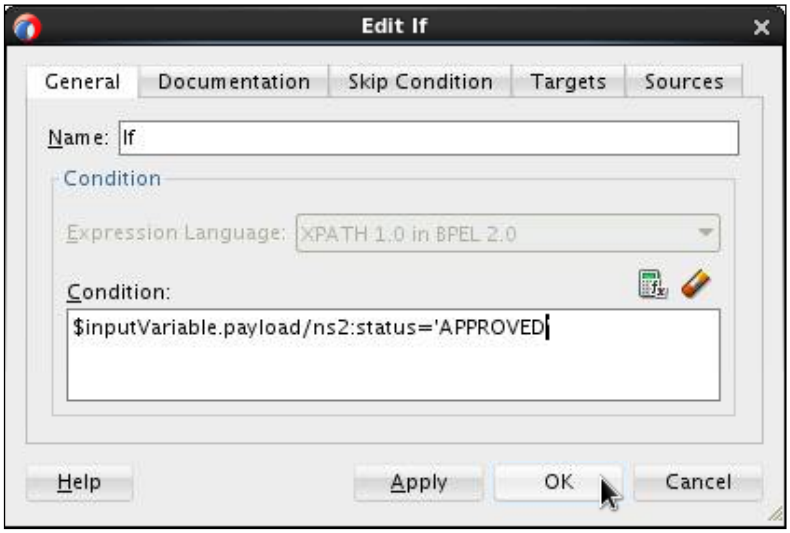
This practice assumes that you have completed Practice 12-4 successfully, and that the `OrderProcessing` application is still deployed to the application server.

Tasks

Finishing Processing of Approved Orders

1. Open the BPEL Process OrderProcessing for editing.
2. Create and configure an If activity to test the approval status of the order.
   a. Collapse the existing If activity for clarity.
   b. Add a second If activity between the first If and the Assign_Output activities.
   c. Label the branches `'if APPROVED'` and `'else REJECTED'`.

d. Set the conditional expression of the If to test whether `status=APPROVED`. **Hint**: Use the expression builder to create the test expression.



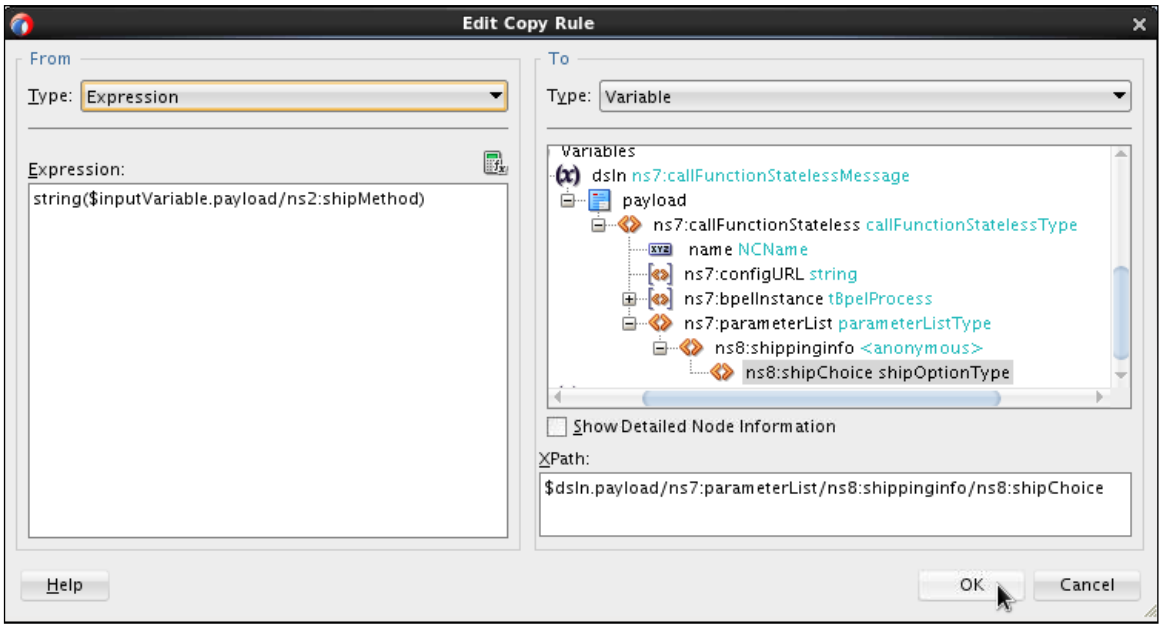
Configuring the <else REJECTED> Branch

3. Add an Empty activity to the <else REJECTED> branch. Name it `Do_Nothing`.

Configuring the <if APPROVED> Branch

4. Create and configure a business rule activity to choose the shipping company.
   a. Add a Business Rule activity to the <if APPROVED> branch.
   b. Right-click the new activity and select Edit.
   c. Name the rule `ShippingRules`.
   d. Specify the dictionary by selecting `ShippingRules` from the drop-down menu.
   e. Assign Input Facts.
      1) Click the Create icon.
         The Decision Fact Map window opens.
      2) In the left pane, select Expression from the Type drop-down menu.
      3) Use Expression Builder to define the expression `string($inputVariable.payload/ns4:shipMethod)`.
      4) In the right pane, select the `shipChoice` element of the `dsIn` variable.

5) Verify your work and click OK.



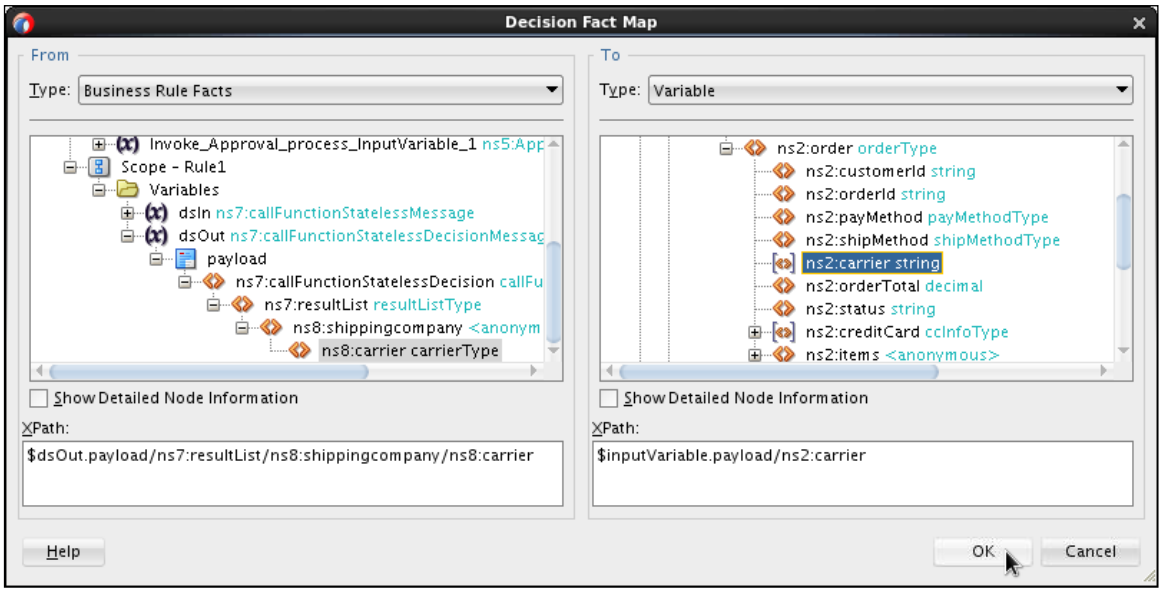
Note: Wrapping the `shipMethod` element in the `string` function as you did in the preceding step 3 is not strictly required. However, if you remove the `string` function, you get a compiler warning about incompatible types for the specified Assign Input Facts settings.

f. Assign Output Facts.

1) Click the Assign Output Facts tab.
2) Click the Create icon.

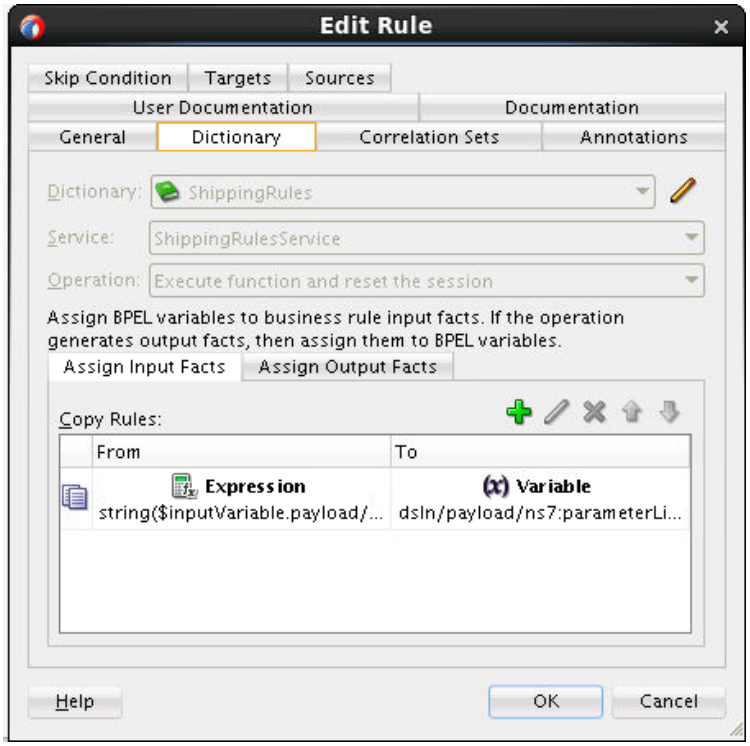
   The Decision Fact Map window opens.
3) In the left pane, select the `carrier` element of the `dsOut` variable.
4) In the right pane, select the `carrier` element of `inputVariable`.

5) Verify your work and click OK.



You are returned to the Edit Rule dialog box.

g. Verify your work and click OK.



5. Close `OrderProcessing.bpel`.

Practice 12-6: Deploying and Testing the `OrderProcessing` Application [Optional]

Overview

In this practice, you redeploy and test the `OrderProcessing` composite application.

Assumptions

This practice assumes the following:

- You have completed Practice 9 for the lesson titled “Implementing Human Workflow and Notifications” successfully, and that the `Approval` application is still deployed to the application server.
- You have completed all work in Practice 12-5 successfully.

Tasks

Redeploying the Enrollment Composite Application

1. Redeploy the `OrderProcessing` application to IntegratedWebLogicServer.

Testing the Composite Application

In this section, you again submit an order whose total is greater than 4000 so that the `ApprovalRules` business rule returns a result that requires manual approval of the order. As in the previous practice, as employee `jcooper`, you check your email to read the notification about the assigned task, and then log in to the Worklist application to approve the order.

2. Open the Oracle Enterprise Manager page (http://localhost:7101/em).
   a. In the Target Navigation pane, expand the SOA > soa-infra > Home > Deployed Composite default nodes in the tree. Click the “OrderProcessing [1.0]” link.
   b. On the “OrderProcessing [1.0]” home page, click Test.
   c. On the Test Web Service page, scroll down to the Input Arguments section to the Request tab and select XML view.
   d. Replace the XML data with the contents of the file `/home/oracle/labs/files/xml_in/businessRules_input.xml`.
   e. Click Test Web Service.
3. Manually approve the order in the Worklist application:
   a. Access the Oracle BPM Worklist application at http://localhost:7101/integration/worklistapp.
   b. Log in to the application as `jcooper` with the password `welcome1`.

      On the Worklist home page, the Inbox is selected and the Manual Approval task is present in the top-right pane.
   c. Click the Manual Approval task entry.

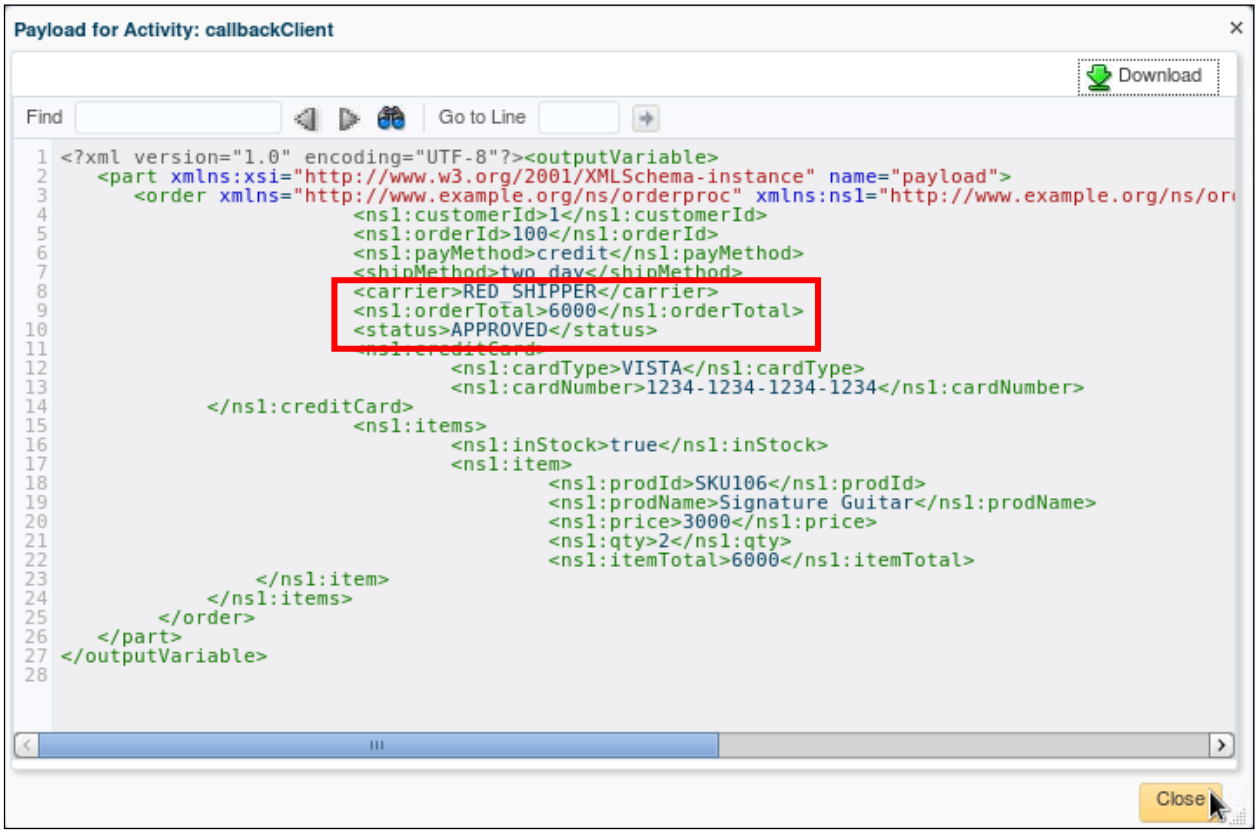
      The order details are displayed in the lower-right pane.

d. Click Approve.

   The Ship Method value is `two_day`. Recall that this affects the shipping company that is used for the shipping request.

e. Log out of the Worklist application and close the webpage.

4. Return to the Flow Trace page and click the Refresh icon in the top-right corner of the page.
5. Scroll down the Trace tree and verify that the carrier selected is `RED_SHIPPER`.
   a. Click the OrderProcessing BPEL process.
   b. Expand the payload for the callbackClient activity.
   c. Verify that the shipMethod is `two_day`, and the resulting carrier value is `RED_SHIPPER`.



Additional Optional Test

If you have time, you can submit some additional orders. Consider these scenarios:

6. Initiate a test with an order that requires manual approval and, at the manual approval stage, reject it.