G Using Service Data Objects (SDO)

G.1 Introduction

Note: The solution for this chapter can be found in c:\po\solutions\apG-SDO

To run this solution, you must have completed labs in chapter 9. Alternatively you can run the setup in Chapter 1 and use the solution from chapter 9 located at c:\po\solutions\ch9

This lab exercise will give you a brief introduction to using Service Data Objects in a SOA Composite for accessing and manipulating data.

To illustrate the usage of SDOs you will create a simple SDO using ADF BC and then use this service to retrieve data from the database using the primary key of the table and then subsequently update the retrieved row.

G.2 Preparing for the lab

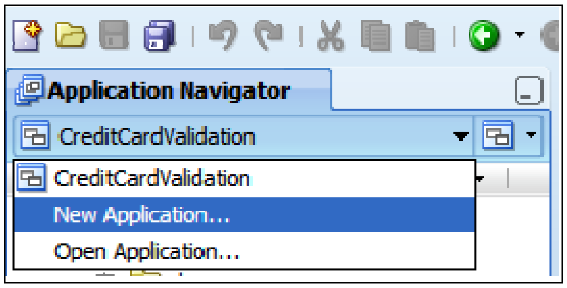
In the *SOADEMO* schema, create a new table called CUSTOMERS, which contains customer information.

1. Start SQL*Plus from the command line
2. Using the *soademo* user, run the script **create_customers_table.sql** found in c:\po\sql directory. This script creates the CUSTOMERS table and populates it.

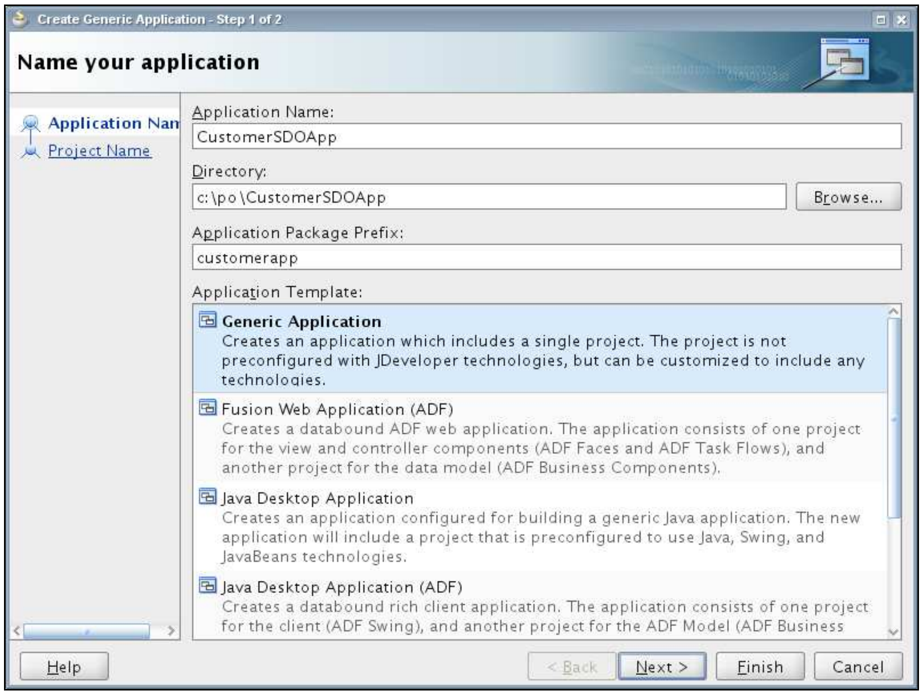
   ```
   sqlplus soademo/soademo @create_customers_table.sql
   ```

G.3 Creating the ADFBC Service

1. Create a new application by using the **File/New** menu, the **New Application** command from the application menu, or by selecting **New Application** from the **Application Navigator** drop-down list.



2. Name your application **CustomerSDOApp** and make its location in c:\po.



Application Name: CustomerSDOApp

Directory: c:\po\CustomerSDOApp

Application Package Prefix: customerapp

Application Template:

Generic Application
Creates an application which includes a single project. The project is not preconfigured with JDeveloper technologies, but can be customized to include any technologies.

Fusion Web Application (ADF)
Creates a databound ADF web application. The application consists of one project for the view and controller components (ADF Faces and ADF Task Flows), and another project for the data model (ADF Business Components).

Java Desktop Application
Creates an application configured for building a generic Java application. The new application will include a project that is preconfigured to use Java, Swing, and JavaBeans technologies.

Java Desktop Application (ADF)
Creates a databound rich client application. The application consists of one project for the client (ADF Swing), and another project for the ADF Model (ADF Business

3. Click **Next**.

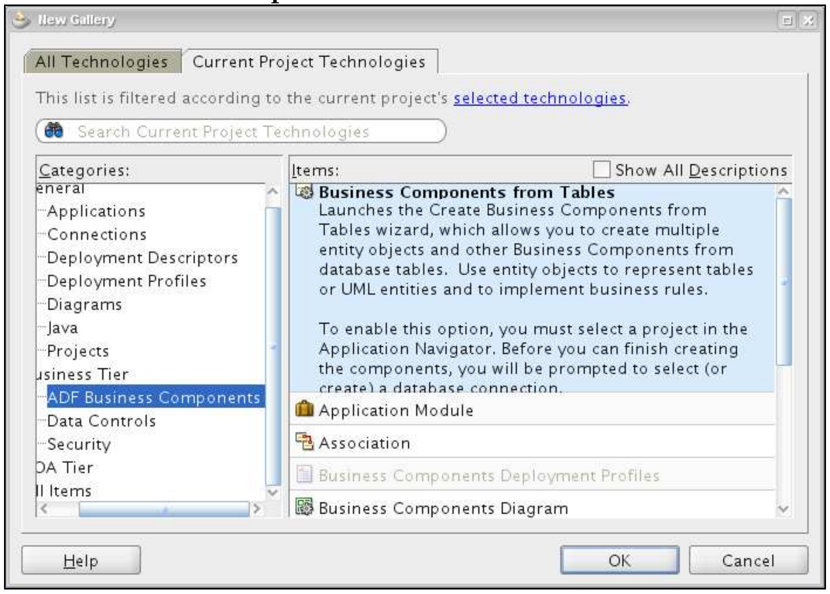
4. Use the name **CustomerSDO** for the project and select **ADF Business Components** for the **Project Technologies**.

5. Select **Next** and then select **Finish** to create the empty project.

G.3.1 Creating the Business Components

1. Right-click on the *CustomerSDO* project in the **Application Navigator** and select **New**.

2. In the **New Gallery** window select **ADF Business Components** in the Categories list and **Business Components From Tables** in the Items list

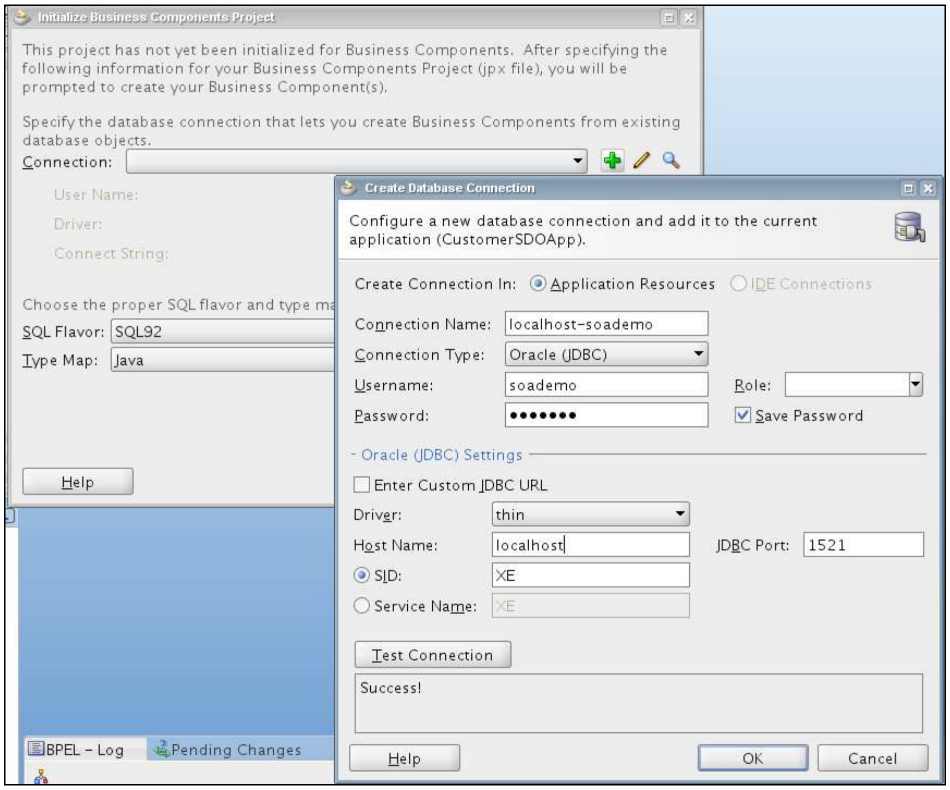


This list is filtered according to the current project's selected technologies.

Business Components from Tables
Launches the Create Business Components from Tables wizard, which allows you to create multiple entity objects and other Business Components from database tables. Use entity objects to represent tables or UML entities and to implement business rules.

To enable this option, you must select a project in the Application Navigator. Before you can finish creating the components, you will be prompted to select (or create) a database connection.

3. Click **OK**
4. In the **Initialize Business Components Project**, create a new connection to the database with the soademo schema.
5. Test the connection to ensure that the user-id and password are correct



Initialize Business Components Project

This project has not yet been initialized for Business Components. After specifying the following information for your Business Components Project (jpx file), you will be prompted to create your Business Component(s).

Specify the database connection that lets you create Business Components from existing database objects.

Connection:

User Name:

Driver:

Connect String:

Choose the proper SQL flavor and type ma

SQL Flavor: SQL92

Type Map: Java

Create Database Connection

Configure a new database connection and add it to the current application (CustomerSDOApp).

Create Connection In: [✓ Application Resources] IDE Connections

Connection Name: localhost-soademo

Connection Type: Oracle (JDBC)

Username: soademo

Role:

Password: •••••••

[✓ Save Password]

Oracle (JDBC) Settings

Enter Custom JDBC URL

Driver: thin

Host Name: localhost

JDBC Port: 1521

[✓ SID:] XE

Service Name: XE

Success!

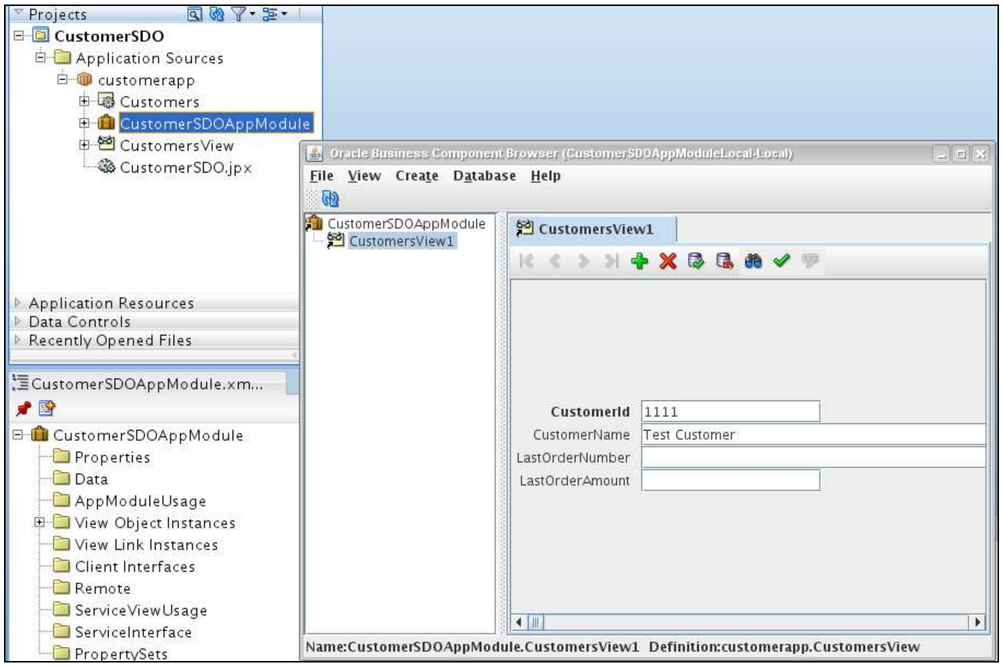
6. Accept the default **SQL** Flavor and **Type Map** settings and click on **OK** to continue.
7. In the **Entity Objects** window click on **Query** to retrieve tables. Select **CUSTOMERS** from the **Available** list and move it to the **Selected** list.
8. Click **Next**
9. In the **Updatable View Objects** window, select **Customers (SOADEMO.CUSTOMERS)** from the **Available** list and move it to the **Selected** list and click **Next**
10. Accept defaults in the **Read-Only View Objects** window and click **Next**
11. In the **Application Module** window enter the name of the module as **CustomerSDOAppModule** and click Next
12. Click **Finish** in the **Diagram** window.
13. Click **Save All** to save the project

G.3.2 Testing the Application Module

You can test the application module created in the previous step.

14. To test it, right-click on CustomerSDOAppModule (not the application name) and select Run. This will start a Java application.

15. Double-click on CustomersView1 in the left-hand pane. This will open up a form. You should see one row that was populated when you created the CUSTOMERS table.



CustomerId	1111
CustomerName	Test Customer
LastOrderNumber	
LastOrderAmount	

Name:CustomerSDOAppModule.CustomersView1 Definition:customerapp.CustomersView

16. Close the Java application by selecting **File>Exit** in its toolbar.

G.3.3 Creating the service interface

Now create a service interface for your **CustomersView1** view object.

1. Double click on the **CustomerSDOAppModule** in the Application Navigator. This opens the application module configuration panel.

2. Click on **Service Interface**

3. Click on the + sign in the top-right corner to add a new interface. This starts a wizard for defining the new service

4. In the **Service Interface** window, change the name of the Web Service to **CustomerSDOService**. Use the default target name space and click Next

5. Click Next on the **Service Custom Methods** page

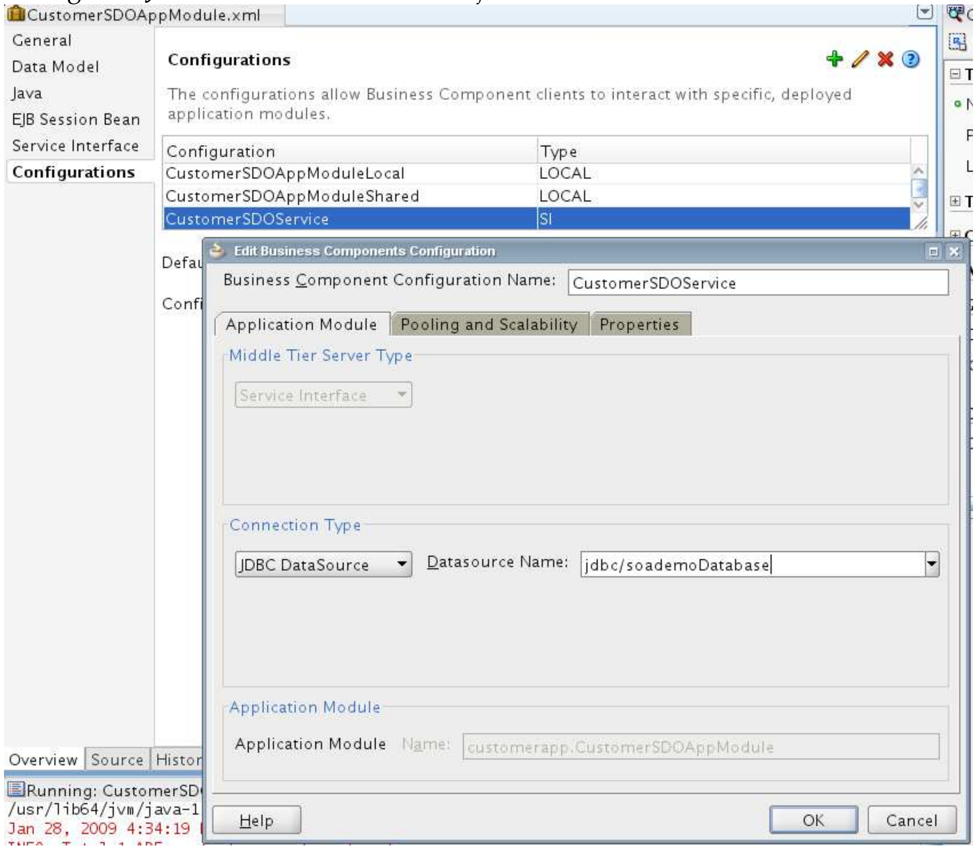
6. In the **Service View Instances**, select *CustomersView1* and add it to the **Selected** list.

7. Select the **CustomersView1** in the **Selected** list. This will populate the **Basic Operations** tab. Select all the operations listed (don't forget to scroll) and click on Next.
8. Click **Finish**

G.3.4 Deploying the service

At this point you now have a new service interface created for the CustomersView1 view object. Before you can deploy this service you need to ensure that service can participate in a distributed transaction. To do that, you need to configure it to use the JDBC data source, **jdbc/soademoDatabase** that was created for the soademo schema in the earlier labs.

1. Click on **Configurations** in the *CustomerSDOAppModule*
2. Edit the *CustomerSDOService* configuration
3. Change the **JDBC DataSource Name** to *jdbc/soademoDatabase*



Configuration	Type
CustomerSDOAppModuleLocal	LOCAL
CustomerSDOAppModuleShared	LOCAL
CustomerSDOService	SI

Business Component Configuration Name: CustomerSDOService

JDBC DataSource Datasource Name: jdbc/soademoDatabase

Application Module Name: customerapp.CustomerSDOAppModule

4. Click **OK**.
5. Back on the **Configurations** page, select **CustomerSDOService** as the default configuration
6. Click **Save All** to save the project
7. You now need to create a deployment profile. Right-click on the CustomerSDO project in the application navigator and select **Project Properties**

8. Select **Java EE Application** in the left-hand panel
9. Change the **Java EE Web Application Name** to *CustomerSDO-webapp*
10. Change **the Java EE Web Context Root** to *customer-app*
11. Click on **Deployment** in the left-hand panel
12. Click **New** to create a new deployment profile
13. Select **Business Components Service Interface** as the Archive Type
14. Change the Name to **customerSDOProfile**
15. Click **OK**
16. Expand the **customerSDOProfile** and select Middle Tier and click on **Edit**...
17. Change the **Enterprise Application Name** to *CustomerSDO*
18. Change the name of the ear file to *customer-app.ear*
19. Click **OK**
20. Click **OK** to close the **Project Properties** window
21. **Save All** to save the project
22. In the **Application Menu** in the toolbar, select **Deploy-> CustomerSDOApp_customerSDOProfile -> MyServerConnection**
23. Select the *soa_server1* managed server as the target.

G.3.5 Testing the Customer SDO service

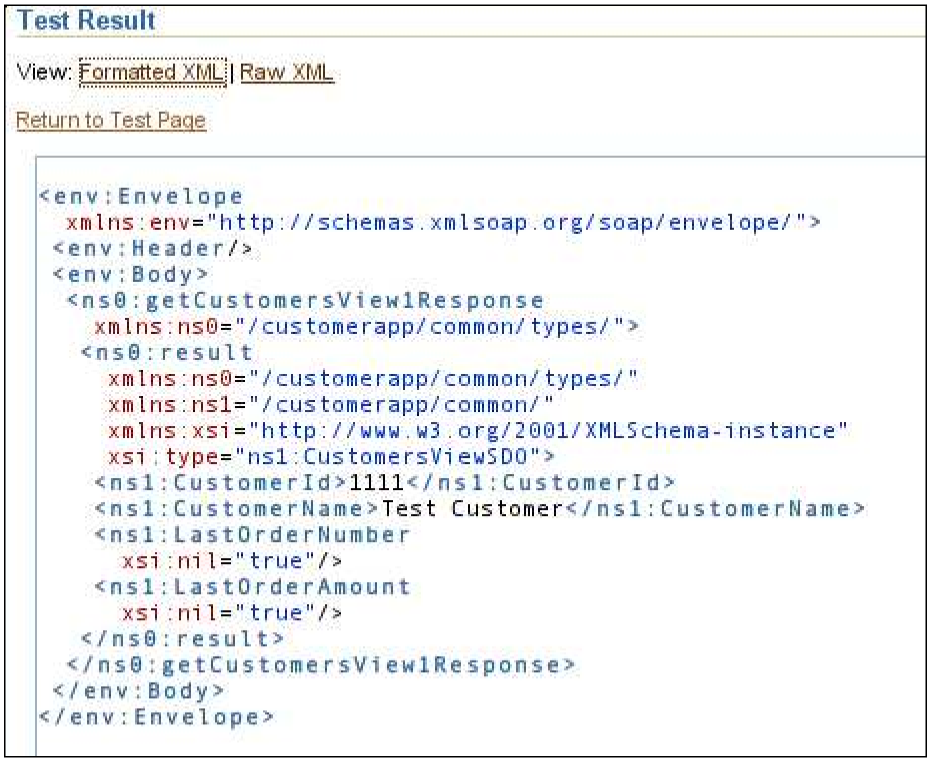
To test the service, start your browser and navigate to

http://localhost:8001/customer-app/CustomerSDOService

You can also use EM to test the service by selecting the **Test** icon in the **Web Services** pane of the *CustomerSDOApp_customerSDOProfile* application page.

You should see a Web Service testing page. In the **Operations** drop-down list select **getCustomersView1**. The page will re-render to show only customerid text input. Enter **1111** and click on **Invoke**.

In the test result window, click on **Formatted XML**, you should see the appropriate customer information.



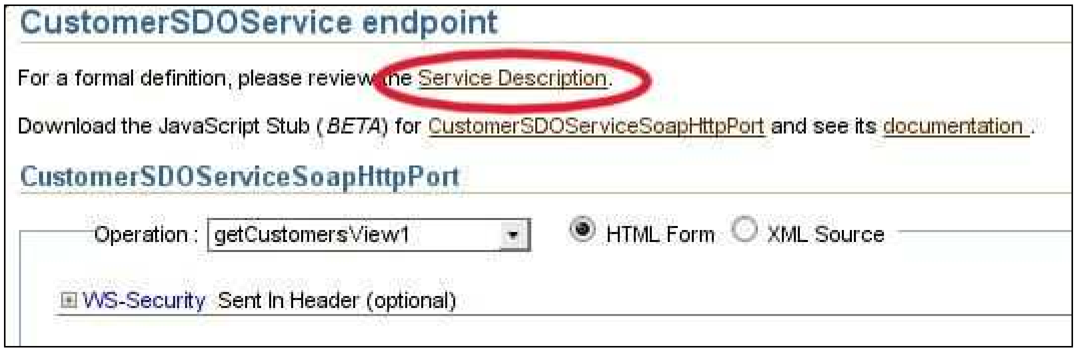
```
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Header/>
 <env:Body>
  <ns0:getCustomersView1Response
    xmlns:ns0="/customerapp/common/types/">
   <ns0:result
     xmlns:ns0="/customerapp/common/types/"
     xmlns:ns1="/customerapp/common/"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:type="ns1:CustomersViewSDO">
    <ns1:CustomerId>1111</ns1:CustomerId>
    <ns1:CustomerName>Test Customer</ns1:CustomerName>
    <ns1:LastOrderNumber
      xsi:nil="true"/>
    <ns1:LastOrderAmount
      xsi:nil="true"/>
   </ns0:result>
  </ns0:getCustomersView1Response>
 </env:Body>
</env:Envelope>
```

G.4 Using the SDO in POProcessing Composite

In this step you will modify the POProcessing composite to use the customer SDO Service. The approveLargeOrder BPEL process will be modified to retrieve the customer information using the customer id in the input order and replace the ID in the input with the name. You will also update the customer information in the database with the information of the last approved order ID and the order value.

G.4.1 Adding the CustomerSDOService as a component

1. Open the POProcessing application in JDeveloper or use the completed solution from Chapter 9 from c:\po\solutions\ch9.
2. Open the composite.xml and add a new Web Service component to the External References swim lane on the composite diagram.
3. In the **Create Web Service** window, enter the name of the service as **CustomerSDOService**.
4. Browse to service test page as mentioned in step 3.4.5 and copy the WSDL URL referenced by the **Service Description** hyperlink



CustomerSDOService endpoint

For a formal definition, please review the Service Description.

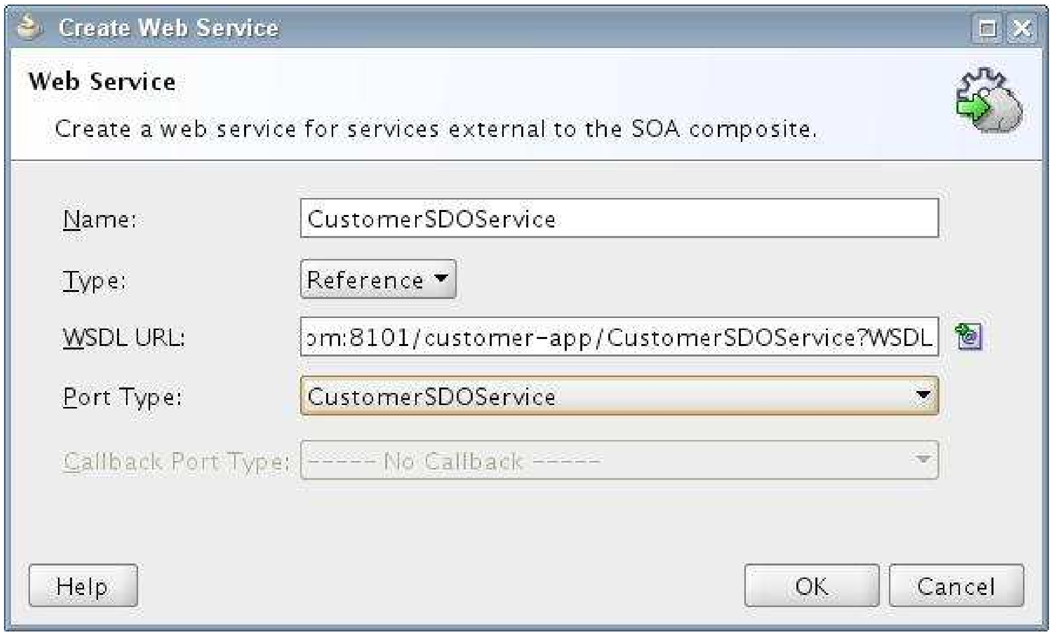
Download the JavaScript Stub (*BETA*) for CustomerSDOServiceSoapHttpPort and see its documentation .

CustomerSDOServiceSoapHttpPort

Operation : getCustomersView1 [✓ HTML Form] XML Source

WS-Security Sent In Header (optional)

5. Paste the URL into the **WSDL URL** field in the **Create Web Service** window and press the Tab key to process the URL. The wizard will load the WSDL and populate the Port Type appropriately. Press **OK** to complete the definition.



Create Web Service

Web Service

Create a web service for services external to the SOA composite.

Name: CustomerSDOService

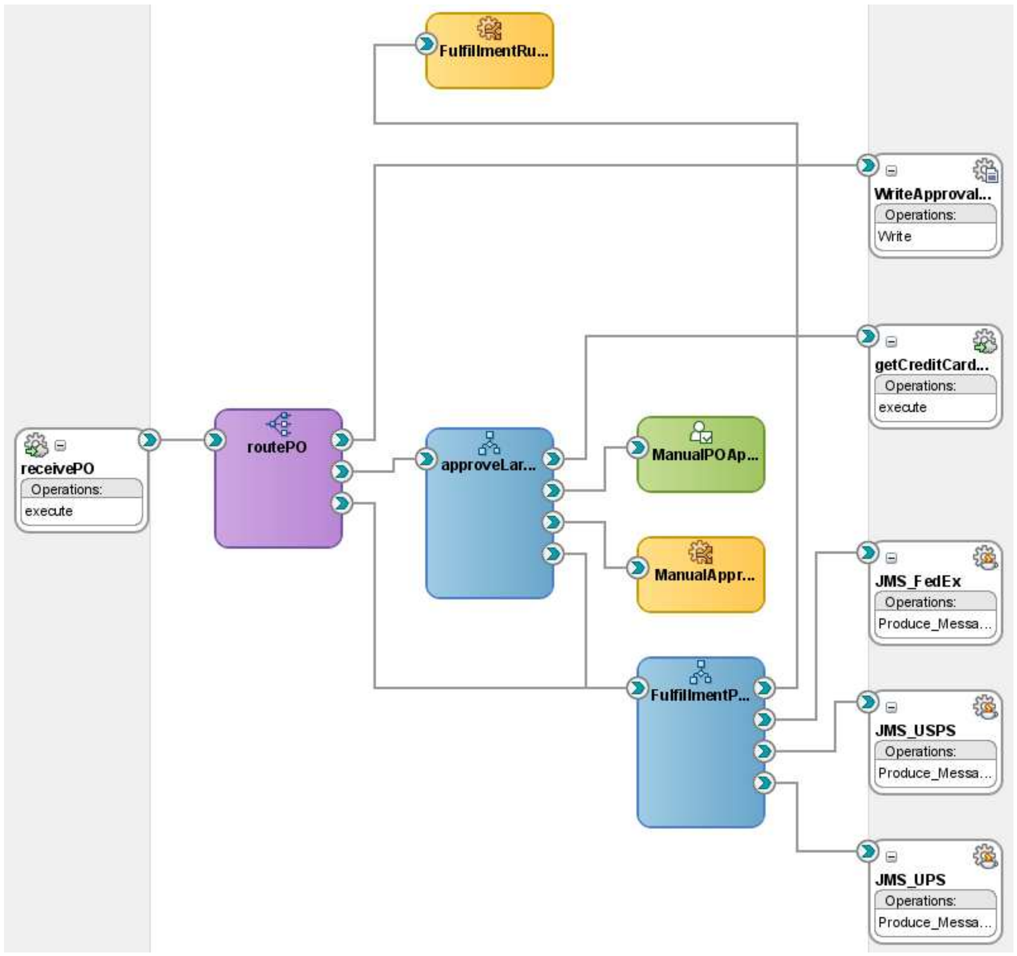
Type: Reference

WSDL URL: om:8101/customer-app/CustomerSDOService?WSDL

Port Type: CustomerSDOService

Callback Port Type: ------ No Callback ------

6. Connect the **approveLargeOrder** BPEL process to the **CustomerSDOService** web service



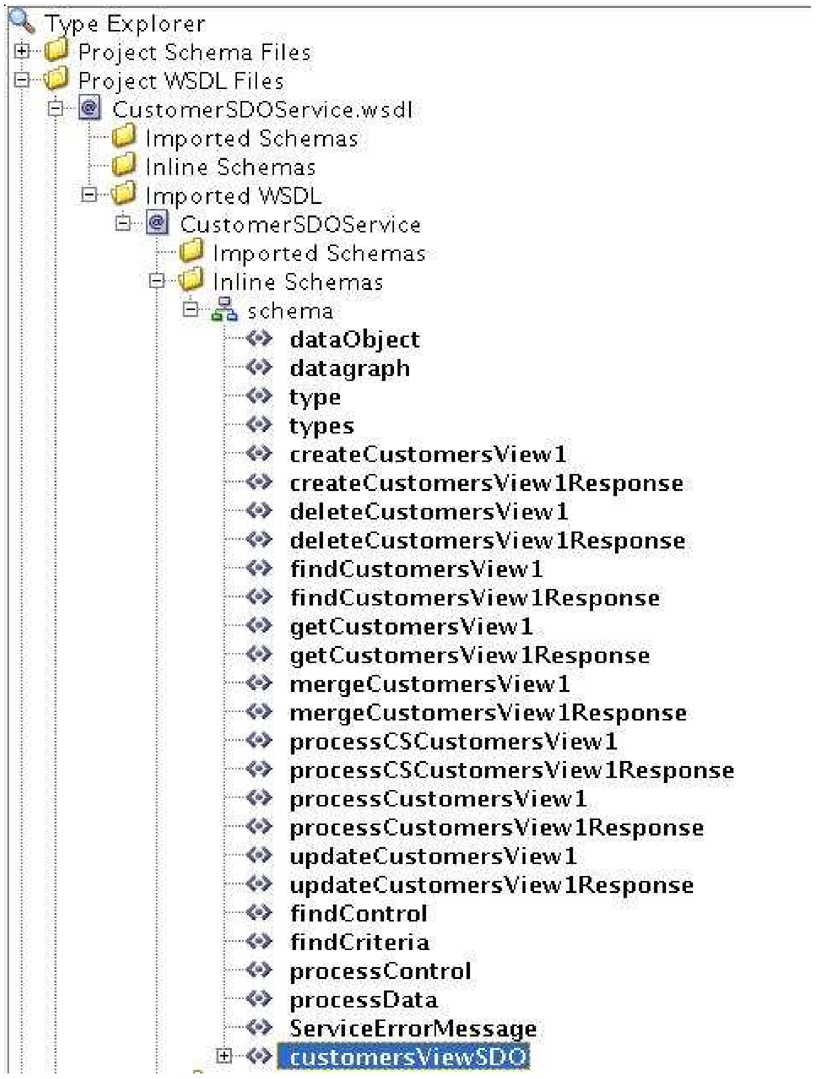
7. Save all

G.4.2 Using the SDO in BPEL

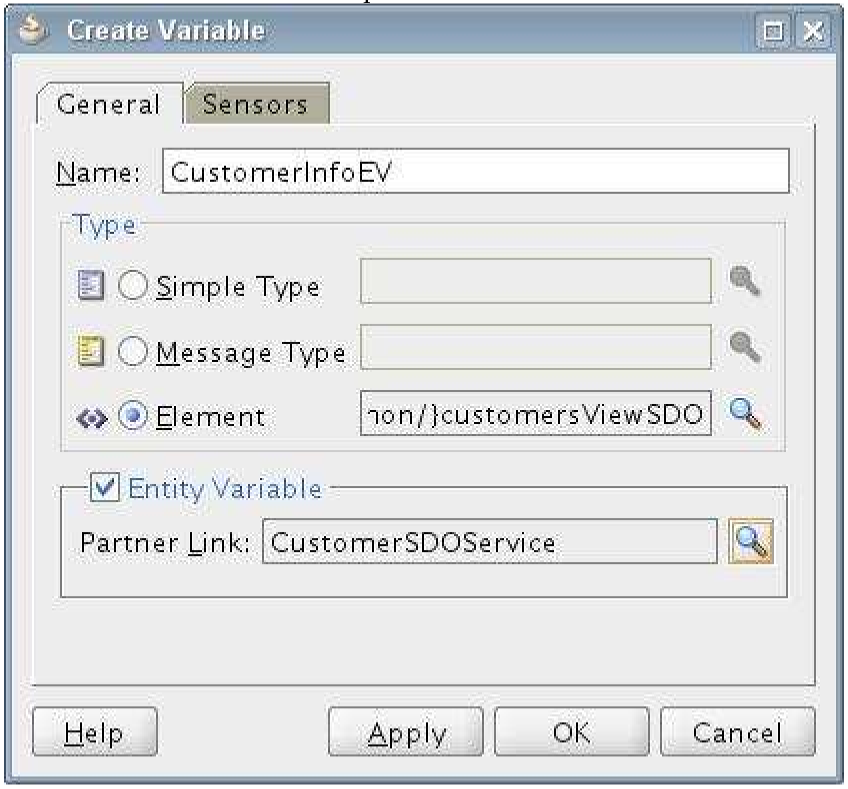
In this step you will use Entity Variables to operate on the underlying Service Data Object. Entity Variable is a new feature introduced in 11g.

G.4.2.1 Creating the Entity Variable

1. Open the **approveLargeOrder** BPEL process
2. Create a new **Entity Variable** by clicking on the **Variables** icon **(x)** in the main scope for approveLargeOrder.
3. In the **Variables** window, click on **+** to add a new variable
4. Name the variable **CustomerInfoEV**
5. Select **Element** and click on the browse icon to select an element. In the **Type Chooser**, traverse down the **Project WSDL Files** tree as shown below and select **customersViewSDO** element as shown in the following image.


Type Explorer
Project Schema Files
Project WSDL Files
CustomerSDOService.wsdl
Imported Schemas
Inline Schemas
Imported WSDL
CustomerSDOService
Imported Schemas
Inline Schemas
schema
dataObject
datagraph
type
types
createCustomersView1
createCustomersView1Response
deleteCustomersView1
deleteCustomersView1Response
findCustomersView1
findCustomersView1Response
getCustomersView1
getCustomersView1Response
mergeCustomersView1
mergeCustomersView1Response
processCSCustomersView1
processCSCustomersView1Response
processCustomersView1
processCustomersView1Response
updateCustomersView1
updateCustomersView1Response
findControl
findCriteria
processControl
processData
ServiceErrorMessage
customersViewSDO

6. Back in the **Create Variable** window, select **Entity Variable** and choose the **CustomerSDOService** for the partner link.



Name: CustomerInfoEV

Type

Simple Type

Message Type

Element: ıon/}customersViewSDO

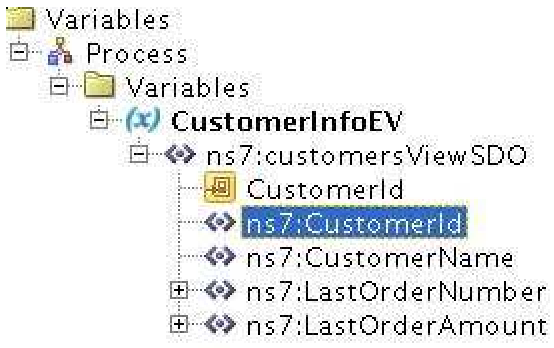
[✓] Entity Variable

Partner Link: CustomerSDOService

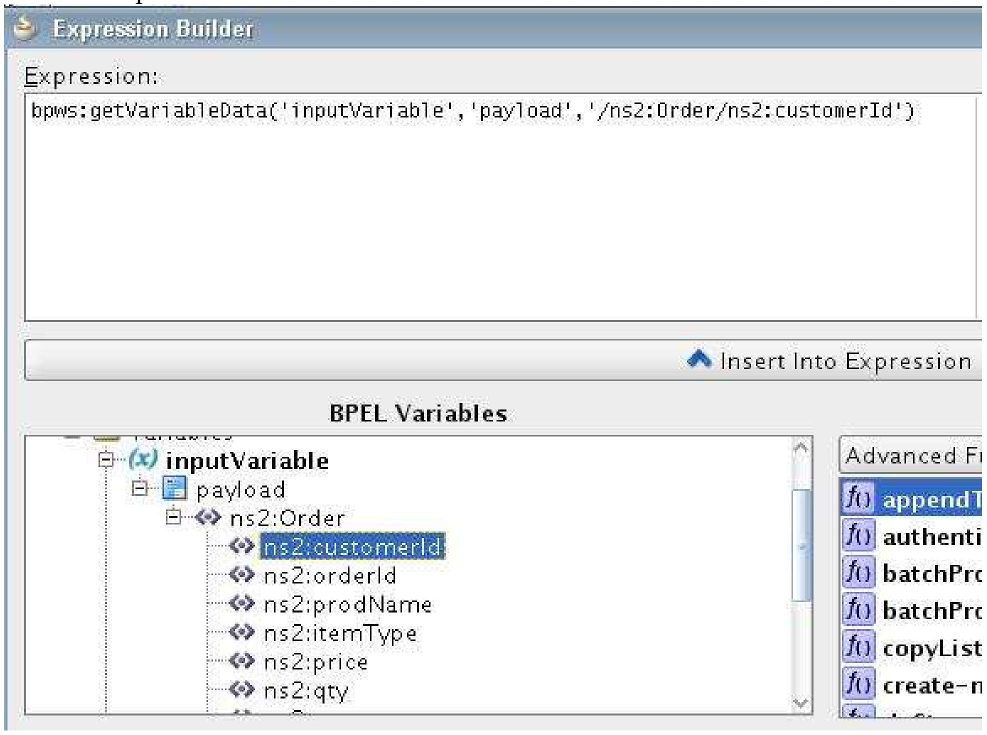
G.4.2.2 Binding the Entity Variable

You have now associated an entity variable with the CustomerSDOService. To retrieve a row from the database, you will need to assign a value for the primary key. The action of assigning the value, initates the retrieval of the corresponding row. This row becomes the current row represented by the entity variable. Any changes done to the contents of this variable is the same as changing the column values of the row and these changes are automatically applied to the database in a consistent manner.

1. Drag a **Bind Entity** activity and drop it just after the **receiveInput** activity.
2. Rename the activity to **BindCustomerId**.
3. Double-click the activity to open its properties. Select **CustmerInfoEV** as the entity variable and add a unique key by clicking on the **+** icon. For the **Key Local Part,** click on the browse variables icon ***(x)*** and select **ns7:CustomerId** from the CustomerInfoEV variable. Note that you may have a different namespace prefix other than ns7.



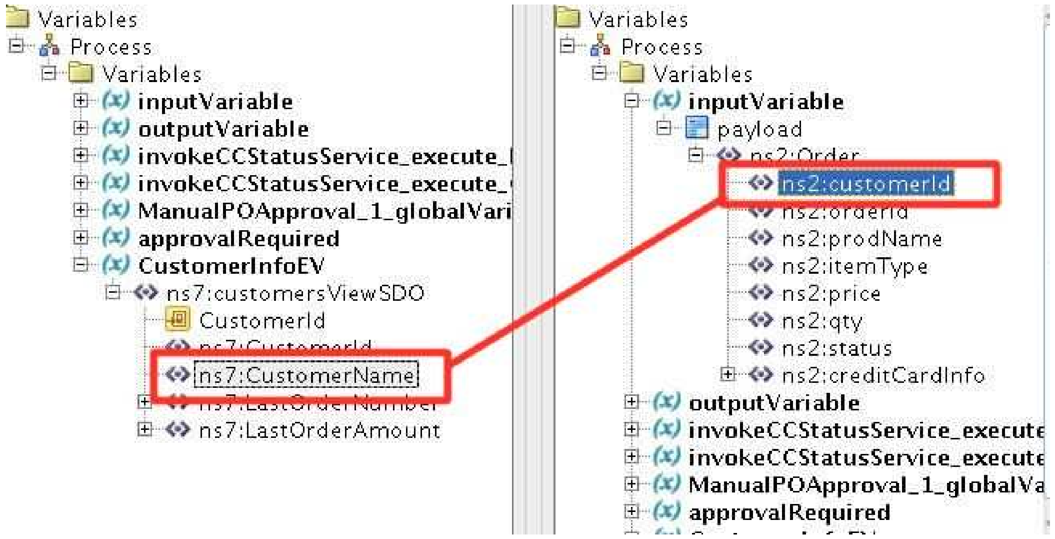
4. For the **Key Value** click on the express builder icon and select the customer Id from the input variable.



```
bpws:getVariableData('inputVariable','payload','/ns2:Order/ns2:customerId')
```

5. Click **OK** in the **Bind Entity** window to complete the activity definition.

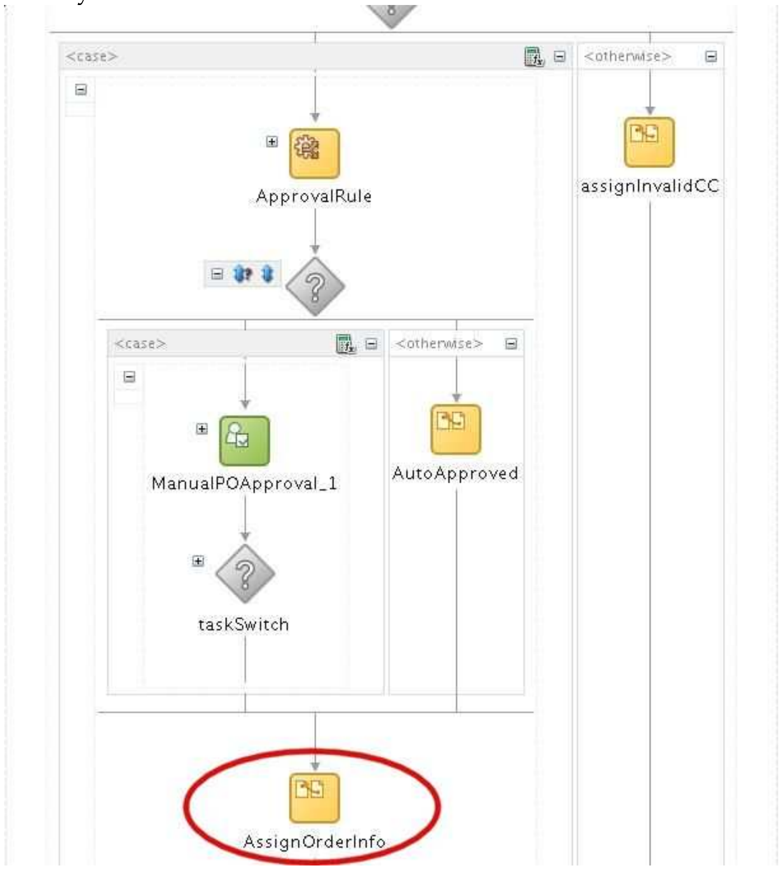
6. Add as **Assign** activity to directly below the *BindCustomerID* activity and assign the customer name from the CustomerInfoEV variable to the customerId in the input variable. Ignore the warning “Variable is not initialized” warning on this assign activity.



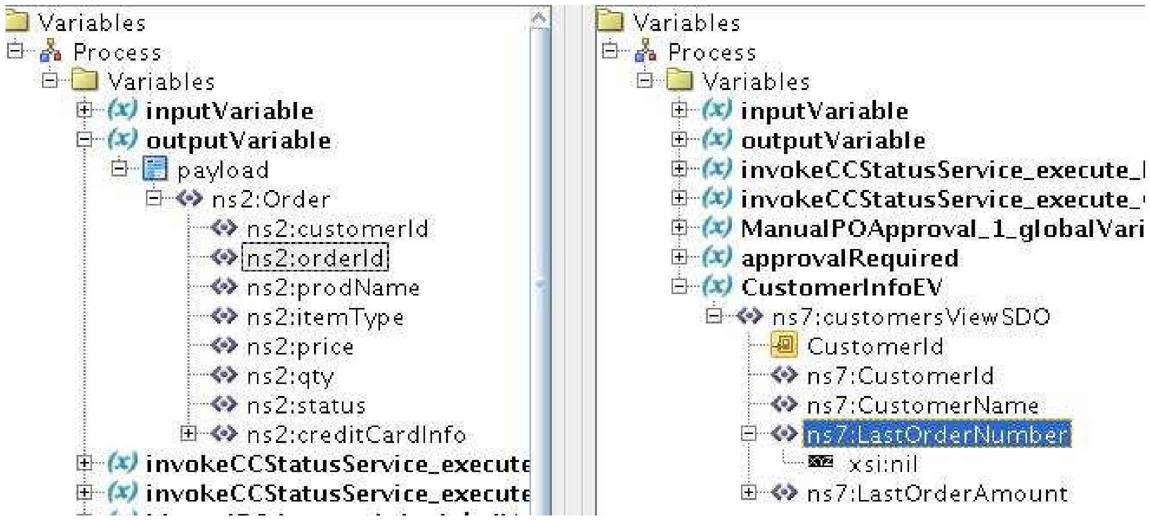
G.4.2.3 Updating the Entity Variable

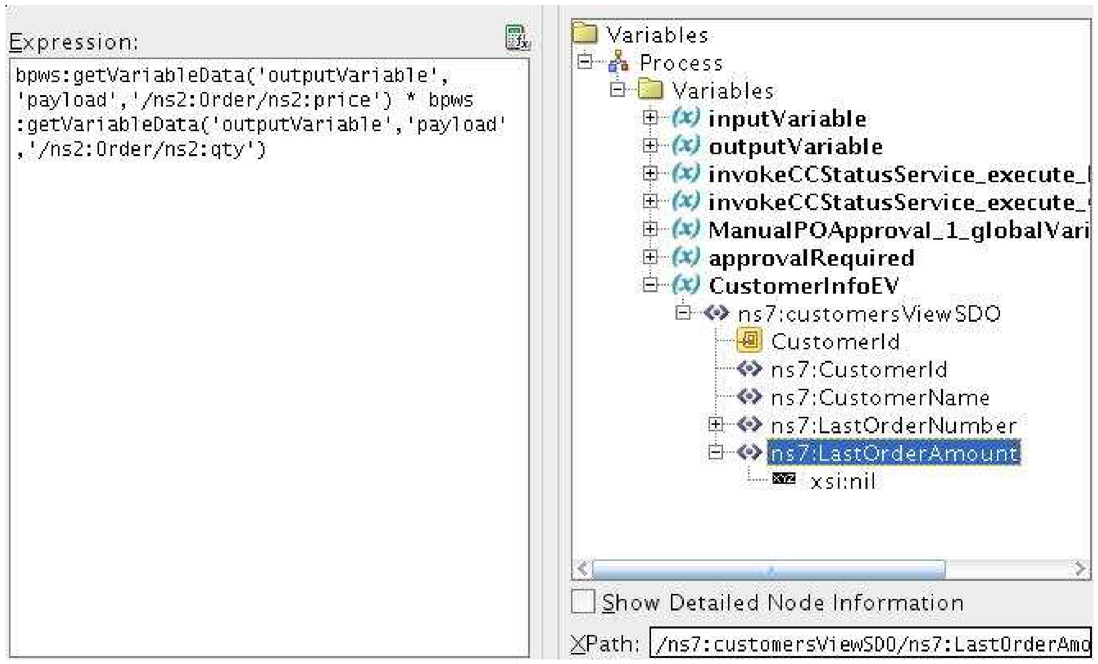
In this step you will update the currently retrieved customer data with order information.

1. Create an assign activity in the case branch which has the **ApprovalRule** rules activity as shown below.



2. In the assign activity, create two copy operations. One for copying the **orderId** from the **outputVariable** to **LastOrderNumber** in **CustomerInfoEV**. The second copy operation should copy the value (**price** * **qty**) from the **outputVariable** to **Last OrderAmount** in the entity variable.




Expression:

```
bpws:getVariableData('outputVariable',
'payload','/ns2:Order/ns2:price') * bpws
:getVariableData('outputVariable','payload'
,'/ns2:Order/ns2:qty')
```

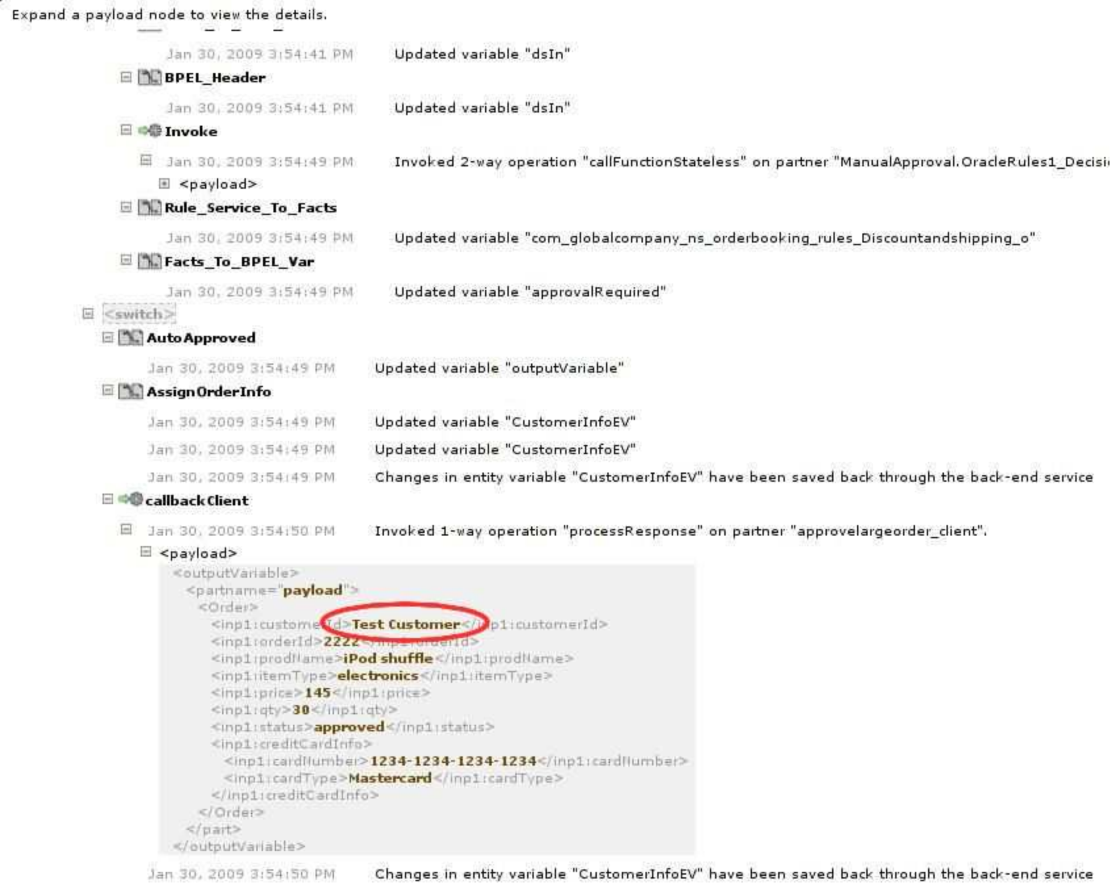
XPath: /ns7:customersViewSDO/ns7:LastOrderAmo

3. Save all to save the project.

G.4.3 Deploying and Testing

1. Deploy the POProcessing composite
2. Run it using an order between $1000 and $5000
3. View the instance using the SOA console.
4. In the Flow Trace window, click on approveLargeOrder to open the BPEL audit trail.

   Expand the **<payload>** node at the end under the **callbackClient** node. You should see the customerId element value is now the name of the customer that was retrieved from the database as a result of the Bind activity you added to the flow.

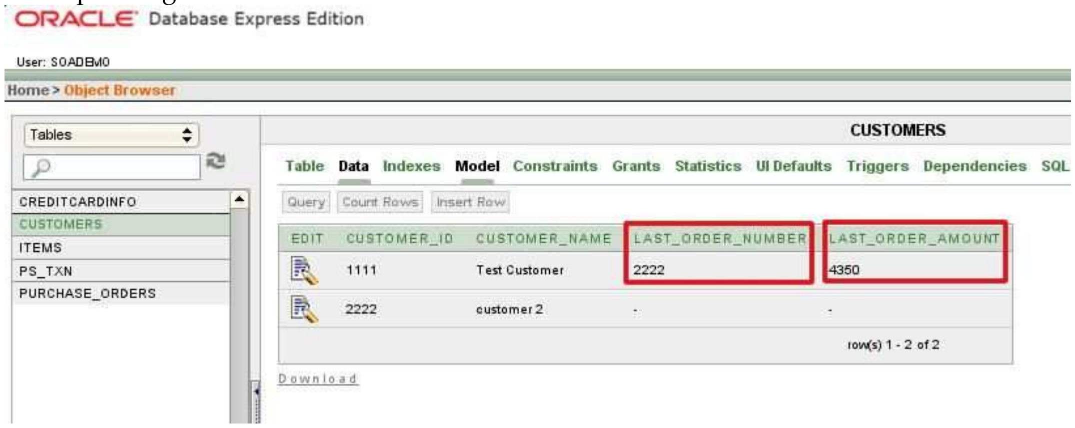


```
Expand a payload node to view the details.

        Jan 30, 2009 3:54:41 PM     Updated variable "dsIn"
  BPEL_Header
        Jan 30, 2009 3:54:41 PM     Updated variable "dsIn"
  Invoke
        Jan 30, 2009 3:54:49 PM     Invoked 2-way operation "callFunctionStateless" on partner "ManualApproval.OracleRules1_Decisi
          <payload>
  Rule_Service_To_Facts
        Jan 30, 2009 3:54:49 PM     Updated variable "com_globalcompany_ns_orderbooking_rules_Discountandshipping_o"
  Facts_To_BPEL_Var
        Jan 30, 2009 3:54:49 PM     Updated variable "approvalRequired"
<switch>
 AutoApproved
       Jan 30, 2009 3:54:49 PM     Updated variable "outputVariable"
 AssignOrderInfo
       Jan 30, 2009 3:54:49 PM     Updated variable "CustomerInfoEV"
       Jan 30, 2009 3:54:49 PM     Updated variable "CustomerInfoEV"
       Jan 30, 2009 3:54:49 PM     Changes in entity variable "CustomerInfoEV" have been saved back through the back-end service
 callbackClient
       Jan 30, 2009 3:54:50 PM     Invoked 1-way operation "processResponse" on partner "approvelargeorder_client".
         <payload>
          <outputVariable>
            <partname="payload">
              <Order>
                <inp1:customerId>Test Customer</inp1:customerId>
                <inp1:orderId>2222</inp1:orderId>
                <inp1:prodName>iPod shuffle</inp1:prodName>
                <inp1:itemType>electronics</inp1:itemType>
                <inp1:price>145</inp1:price>
                <inp1:qty>30</inp1:qty>
                <inp1:status>approved</inp1:status>
                <inp1:creditCardInfo>
                  <inp1:cardNumber>1234-1234-1234-1234</inp1:cardNumber>
                  <inp1:cardType>Mastercard</inp1:cardType>
                </inp1:creditCardInfo>
              </Order>
            </part>
          </outputVariable>
       Jan 30, 2009 3:54:50 PM     Changes in entity variable "CustomerInfoEV" have been saved back through the back-end service
```

5. Query the CUSTOMER table in the database using SQL*Plus or your web browser through Application Express (http://localhost:8080/apex). You can also do this using the **Database Connections** tab in JDeveloper (**View** menu). If using Application Express, login as soademo/soademo and click on **Object Browser**. In the Tables list select **CUSTOMERS** and click on **Data**. You should see the LAST_ORDER_NUMBER and LAST_ORDER_VALUE set to the

corresponding order id and the value.



EDIT	CUSTOMER_ID	CUSTOMER_NAME	LAST_ORDER_NUMBER	LAST_ORDER_AMOUNT
	1111	Test Customer	2222	4350
	2222	customer 2	-	-

row(s) 1 - 2 of 2

6. Try creating a new customer in the database and submit a new PO, but this time change the customer ID to reflect the ID of the new customer created.