Oracle® SOA Suite
Tutorial
Release 3 (10.1.3.1.0)
B28937-01

September 2006
# Contents

Preface ............................................................................................................................................................................. xi  
Audience ........................................................................................................................................................................... xi  
Documentation Accessibility ............................................................................................................................................... xi  
Related Documents .......................................................................................................................................................... xii  
Conventions .................................................................................................................................................................... xii

1 Introduction to the SOA Order Booking Application and the Tutorial  
1.1 What Does the SOA Order Booking Application Do? ......................................................................................... 1-1  
1.2 Flow of the Application ........................................................................................................................................... 1-2  
1.3 Software Required for Creating and Running the Application ............................................................................. 1-3  
1.4 Contents of the SOADEMO Schema ................................................................................................................ 1-3  
1.5 How This Tutorial Is Organized .......................................................................................................................... 1-5  
1.6 Technologies and Techniques Used in Each Project .......................................................................................... 1-5

2 Setting Up Your Environment  
2.1 Download the Files for the Tutorial .................................................................................................................... 2-1  
2.2 Install Oracle Application Server ....................................................................................................................... 2-1  
2.3 Install Oracle JDeveloper ...................................................................................................................................... 2-2  
2.4 Unzip the Files for the SOA Order Booking Application .................................................................................. 2-2  
2.5 Install the SOADEMO Schema ........................................................................................................................ 2-2  
2.6 Define Data Source and Connection Pool in Oracle Application Server ....................................................... 2-3  
2.6.1 Start Up Application Server Control ................................................................................................................ 2-3  
2.6.2 Create a Connection Pool .................................................................................................................................. 2-3  
2.6.3 Create a Data Source ......................................................................................................................................... 2-7  
2.6.4 Create a Database Adapter Connection Factory ............................................................................................. 2-8  
2.7 Set Up Connections in Oracle JDeveloper .......................................................................................................... 2-9  
2.7.1 Create a Database Connection to the SOADEMO Schema ........................................................................ 2-9  
2.7.2 Create a Connection to Oracle Application Server ........................................................................................ 2-11  
2.7.3 Create a Connection to the Integration Server .................................................................................................. 2-12  
2.8 Create the SOADEMO Application in JDeveloper .......................................................................................... 2-13

3 Creating the CustomerService Project  
3.1 About the CustomerService Project ..................................................................................................................... 3-1  
3.2 Create a New Project for CustomerService ....................................................................................................... 3-2  
3.3 Create Entity Beans from Tables in the Database ......................................................................................... 3-3
3.4 Edit persistence.xml ................................................................. 3-7
3.5 Create a Session Bean in the business Directory ............................. 3-9
3.6 Define Additional Queries in Customer.java ................................... 3-13
3.7 Use a Database Sequence to Generate Address IDs in Address.java ... 3-16
3.8 Edit Session Facade for the Session Bean (CustomerServiceBean.java) ........................................................................ 3-16
3.9 Add and Modify Methods in the Session Bean (CustomerServiceBean.java) .............................................................. 3-17
3.9.1 Modify queryCustomerFindCustomerById .................................. 3-17
3.9.2 Modify queryCustomerFindCustomerByEmail ............................. 3-19
3.9.3 Add getCustomerStatus and addNewCustomer Methods ............. 3-21
3.10 Add JSR-181 Annotations to the Web Service Endpoint Interface (CustomerService.java) ........................................ 3-22
3.11 Create EJB JAR Deployment Profile for the CustomerService Project .. 3-23
3.12 Deploy CustomerService ........................................................ 3-25
3.13 View the WSDL for CustomerService ........................................... 3-26

4 Creating the FulfillmentESB Project

4.1 About the FulfillmentESB Project .................................................. 4-1
4.2 Create a New Project for FulfillmentESB ........................................ 4-2
4.3 Create a System Called "Fulfillment" ............................................. 4-4
4.4 Create the "OrderFulfillment" Routing Service .................................. 4-5
4.5 Create the "Shipment" Routing Service ............................................ 4-8
4.6 Create the "USPSShipment" Adapter (File Adapter) ............................ 4-10
4.7 Create the "FedexShipment" Adapter (Database Adapter) ...................... 4-14
4.8 Create the "FulfillmentBatch" Adapter (JMS Adapter) .......................... 4-19
4.9 Create Routing Rules ................................................................. 4-25
4.9.1 Between OrderFulfillment and Shipment .................................... 4-25
4.9.2 Between OrderFulfillment and JMS Adapter ................................. 4-27
4.9.3 Between Shipment and USPSShipment ........................................ 4-29
4.9.4 Between Shipment and FedexShipment ....................................... 4-35
4.10 Save All Files in the FulfillmentESB Project .................................... 4-39
4.11 Register the FulfillmentESB Project .............................................. 4-39

5 Creating the CreditService Project

5.1 About the CreditService Project .................................................... 5-1
5.2 Create a New Project for CreditService .......................................... 5-2
5.3 Copy the WSDL File ..................................................................... 5-2
5.4 Generate Java from the WSDL ...................................................... 5-4
5.5 Display the List of Files in the Structure Window ............................... 5-7
5.6 Build CreditService ..................................................................... 5-7
5.7 Write the Code to Perform Credit Card Validation ............................... 5-7
5.8 Verify Hostname and Port in CreditService.wsdl ............................... 5-8
5.9 Update the Context-Root .............................................................. 5-9
5.10 Rebuild CreditService ................................................................. 5-9
5.11 Deploy Credit Service to Oracle Application Server ............................ 5-10
6 Creating the RapidService Project

6.1 About the RapidService Project ................................................................. 6-1
6.2 Create a New Project for RapidService .................................................... 6-1
6.3 Add JSR-181 Library to the RapidService Project ................................. 6-2
6.4 Create Item.java ....................................................................................... 6-4
6.5 Create Quote.java ..................................................................................... 6-6
6.6 Create RequestQuote.java ......................................................................... 6-7
6.7 Check Files in the Application Navigator .................................................. 6-9
6.8 Compile the Files .................................................................................... 6-10
6.9 Publish the Project as a Web Service ....................................................... 6-10
6.10 Verify the Hostname and Port in the Generated WSDL File ................... 6-15
6.11 Check Files in the Application Navigator ................................................ 6-16
6.12 Set the Context Root to RapidService ................................................... 6-16
6.13 Edit the Deployment Descriptor .............................................................. 6-17
6.14 Deploy the RapidService Project ............................................................ 6-18
6.15 View the WSDL for RapidService .......................................................... 6-19

7 Creating the SelectManufacturer Project

7.1 About the SelectManufacturer Project ...................................................... 7-1
7.2 Create a New BPEL Project for SelectManufacturer .................................. 7-1
7.3 Create "SelectService" Partner Link .......................................................... 7-3
7.4 Define Variables for the SelectManufacturer Project ............................... 7-4
7.5 Receive Order Data from the Client through a Receive Activity ............... 7-7
7.6 Assign Values to be Returned ................................................................. 7-9
7.7 Return Values to the Client Using an Invoke Activity ............................... 7-12
7.8 Deploy the BPEL Process ........................................................................ 7-14
7.8.1 Deploying Using Ant from the Developer Prompt ............................... 7-15
7.8.2 Viewing SelectManufacturer in the Oracle BPEL Control .................. 7-16

8 Creating the SOAOrderBooking Project

8.1 About the SOAOrderBooking Project ..................................................... 8-1
8.1.1 Blocks in the SOAOrderBooking Project ............................................ 8-2
8.1.2 Blocks Shown in Minimized View ...................................................... 8-2
8.2 Create a New BPEL Project for SOAOrderBooking ................................ 8-4
8.3 Copy Files ............................................................................................... 8-5
8.4 Define Variables for the SOAOrderBooking Project ............................... 8-5
8.5 Create "client" Partner Link ...................................................................... 8-8
8.6 Receive Input from the Client (Receive Activity) ..................................... 8-10
8.6.1 Create the Receive Activity ................................................................. 8-10
8.6.2 Create a Sensor for the Receive Activity ............................................ 8-12
8.7 Insert Order Information in the Database ("InsertOrderIntoDB" Scope) .... 8-16
8.7.1 Create a Database Adapter for Writing to the ORDERS Table ............. 8-17
8.7.2 Create a Database Adapter for Retrieving the Order ID from the Database 8-21
8.7.3 Create the "InsertOrderIntoDB" Scope ................................................ 8-24
8.7.4 Retrieve the Order ID from the Database Sequence ("GetOrderId" Invoke Activity) 8-28
8.7.5 Prepare the Order ID and Order Status Information ("AssignOrderStatus" Assign Activity) 8-31

8.7.6 Create the Mapping File ("TransformOrder" BPEL Service) 8-34

8.7.7 Insert the Order Information into the Database ("InsertOrder" Invoke Activity) 8-36

8.7.8 Minimize the "InsertOrderIntoDB" Scope 8-39

8.7.9 Retrieve Information About the Customer ("CustomerService" Scope) 8-39

8.8.1 Create the "CustomerService" Partner Link 8-40

8.8.2 Create the "CustomerService" Scope 8-42

8.8.3 Assign Customer ID to the findCustomerById Operation ("AssignRequest" Assign Activity) 8-43

8.8.4 Create a Variable to Contain the Results of findCustomerById ("customerServiceResponse" Process Variable) 8-45

8.8.5 Invoke findCustomerById ("GetCustInfo" Invoke Activity) 8-47

8.8.6 Create the "AssignInitialCustomerResponse" Assign Activity 8-49

8.8.7 Copy the Customer's First and Last Names to the inputVariable ("AssignCustomerResponse" Assign Activity) 8-52

8.8.8 Minimize the "CustomerService" Scope 8-54

8.9 Verify the Customer's Credit Card ("CreditService" Scope) 8-54

8.9.1 Create "CreditValidatingService" Partner Link 8-55

8.9.2 Create the "CreditService" Scope 8-57

8.9.3 Assign the Credit Card Number and Credit Card Type Information ("InitializeRequest" Assign Activity) 8-60

8.9.4 Verify the Customer's Credit Card ("InvokeCreditService" Invoke Activity) 8-62

8.9.5 Create the "OrderBookingFault" Process Variable 8-65

8.9.6 Check the Results of the Credit Card Validation (Switch Activity) 8-67

8.9.6.1 Specify the Condition for <case> 8-67

8.9.6.2 Set the Value of the OrderBookingFault Variable ("AssignFault" Assign Activity) 8-68

8.9.6.3 Create the "ThrowCreditFault" Throw Activity 8-70

8.9.7 Minimize the "CreditService" Scope 8-70

8.10 Set up Oracle Business Rules 8-70

8.10.1 Set up the Repository File 8-71

8.10.2 Create a Dictionary in the Repository 8-72

8.10.3 Copy OrderBookingRules.xsd to the Oracle Application Server Machine 8-73

8.10.4 Generate JAXB Classes for the Elements in the XML Schema 8-74

8.10.5 Import the JAXB Classes into the Oracle Business Rules Data Model 8-76

8.10.6 Define a Variable in the Data Model 8-77

8.10.7 Create a Ruleset 8-78

8.10.8 Create Rules 8-79

8.10.8.1 Create the "belowLimit" Rule 8-79

8.10.8.2 Create the "overLimit" Rule 8-82

8.10.8.3 Create the "platinumMember" Rule 8-86

8.10.9 Log out of Rule Author 8-89

8.10.10 Copy the Files to the JDeveloper Machine 8-89

8.11 Determine If an Order Requires Manual Approval ("RequiresManualApproval" Decide Activity) 8-89

8.11.1 Create the Activities in the "RequiresManualApproval" Decide Activity 8-90

8.11.2 Copy Order Total and Customer Status Information ("BPEL_Var_To_Rule_Facts" Assign Activity) 8-94
8.11.3 Copy the ConversationId to the dsIn Variable ("BPEL_Header" Assign Activity) 8-98
8.11.4 Create the "requiresApproval" Process Variable ................................................. 8-99
8.11.5 Copy the Result of the Decision Service to the requiresApproval Variable ("Facts_To_BPEL_Var" Assign Activity) 8-100
8.11.6 Minimize the "RequiresManualApproval" Decide Activity .............................. 8-102
8.12 Set Up a Form to Process Orders That Require Manual Approval ("requiresApproval" Switch) 8-102
8.12.1 Create the Switch ............................................................................................... 8-104
8.12.2 Set the Condition for the <case> ....................................................................... 8-104
8.12.3 Create a Sequence in the <case> Branch .......................................................... 8-106
8.12.4 Create a Human Task ....................................................................................... 8-106
8.12.5 Create a Form for the Worklist Application ..................................................... 8-111
8.12.6 Accept the Default Settings for the Remaining Human Task Activities .......... 8-111
8.13 Handle the Manager’s Response ("requiresApproval" Switch) ......................... 8-112
8.13.1 Handle the Reject Case ...................................................................................... 8-112
8.13.1.1 Set the Status of the Order (Assign Activity) ................................................ 8-112
8.13.1.2 Create a Throw Activity ................................................................................ 8-113
8.13.2 Handle the Approve Case .................................................................................. 8-114
8.13.3 Handle the Expired Case ................................................................................... 8-114
8.13.4 Accept the Default Settings for Each Case ....................................................... 8-114
8.13.5 Minimize the "requiresApproval" Switch .......................................................... 8-114
8.14 Choose a Supplier ("SelectSupplier" Scope) ......................................................... 8-115
8.14.1 Create the "SelectManufacturer" Partner Link ................................................. 8-116
8.14.2 Create the "RapidService" Partner Link ............................................................ 8-118
8.14.3 Create the "SelectSupplier" Scope ...................................................................... 8-119
8.14.4 Create a Flow Activity ....................................................................................... 8-122
8.14.5 Set the Activities for Select Manufacturer ...................................................... 8-122
8.14.5.1 Create a Scope for SelectManufacturer .......................................................... 8-122
8.14.5.2 Create a Transform Activity ("TransformSelectRequest" Transform Activity) .... 8-125
8.14.5.3 Create an Invoke Activity ............................................................................... 8-126
8.14.5.4 Create a Receive Activity .............................................................................. 8-128
8.14.6 Set the Activities for Rapid Manufacturer ....................................................... 8-130
8.14.6.1 Create a Scope for Rapid Manufacturer .......................................................... 8-131
8.14.6.2 Create a Transform Activity ("TransformRapidRequest" Transform Activity) ...... 8-132
8.14.6.3 Create an Invoke Activity ............................................................................... 8-134
8.14.7 Create a Switch to Pick the Lower-Priced Quote ............................................. 8-137
8.14.7.1 Create the Switch ......................................................................................... 8-137
8.14.7.2 Set the Condition for the Switch ..................................................................... 8-137
8.14.7.3 Set the Activities for Select Manufacturer ...................................................... 8-140
8.14.7.4 Set the Activities for Rapid Distributor ........................................................... 8-142
8.14.8 Minimize the "SelectSupplier" Scope ................................................................. 8-144
8.15 Determine the Shipping Method ("PostFulfillmentReq" Scope) ......................... 8-145
8.15.1 Create the "OrderFulfillment" Partner Link ....................................................... 8-145
8.15.2 Create the "PostFulfillmentReq" Scope ............................................................. 8-147
8.15.3 Copy Order Information to Scope Variable ("initializeRequest" Assign Activity) 8-149
8.15.4 Invoke OrderFulfillmentESB ("PostFulfillmentReq" Invoke Activity) ............. 8-150
9  Creating the OrderBookingESB Project

9.1  About the OrderBookingESB Project ................................................................. 9-1
9.2  Create a New Project for OrderBookingESB....................................................... 9-2
9.3  Create a System Called "OrderBooking" ............................................................... 9-3
9.4  Create the "OrderBookingService" Routing Service .......................................... 9-4
9.5  Invoke the SOAOrderBooking Process ("OrderBookingProcess" External Service) ... 9-5
9.6  Set up Routing Rules for OrderBookingService .................................................. 9-7
9.7  Save All Files in the OrderBookingESB Project ................................................... 9-11
9.8  Register the OrderBookingESB Project ............................................................... 9-11

10  Interfacing the Client Application with the SOA Order Booking Application

10.1 About the SOADemo-Client Application............................................................ 10-1
10.2 Invoking Services from CustomerService .......................................................... 10-2
10.2.1 Generate a Web Service Proxy for CustomerService ....................................... 10-2
10.2.2 Build the Files .................................................................................................. 10-6
10.2.3 Write Code to Invoke the Web Service Proxy for CustomerService ................ 10-7
10.2.3.1 Verifying the Login ..................................................................................... 10-7
10.2.3.2 Registering New Customers ........................................................................ 10-8
10.3 Invoking the OrderBookingESB Project ............................................................ 10-9
10.3.1 Retrieve the Concrete WSDL URL ................................................................. 10-9
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.3.2</td>
<td>Create the Project in the Client and Create the Web Service Proxy</td>
<td>10-10</td>
</tr>
<tr>
<td>10.3.3</td>
<td>Build the Files</td>
<td>10-15</td>
</tr>
<tr>
<td>10.3.4</td>
<td>Write Code to Invoke the Web Service Proxy for OrderBookingESB</td>
<td>10-15</td>
</tr>
<tr>
<td>10.4</td>
<td>Deploying the Client Application</td>
<td>10-17</td>
</tr>
</tbody>
</table>
This document describes how to build the SOA Order Booking demo application.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

**Audience**

This document is intended for developers who are interested in developing applications based on service-oriented architecture (SOA).

**Documentation Accessibility**

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

**Accessibility of Code Examples in Documentation**

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

**Accessibility of Links to External Web Sites in Documentation**

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.
TTY Access to Oracle Support Services
Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents
For more information related to the demo application, see the following documents in the Oracle Application Server Release 3 (10.1.3.1.0) documentation set:

- *Oracle SOA Suite Quick Start Guide*
- *Oracle SOA Suite Developer’s Guide*

Conventions
The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td><code>monospace</code></td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
Introduction to the SOA Order Booking Application and the Tutorial

This chapter describes the components of the SOA Order Booking application, and the requirements for creating and running it.

This chapter contains the following sections:

- Section 1.1, "What Does the SOA Order Booking Application Do?"
- Section 1.2, "Flow of the Application"
- Section 1.3, "Software Required for Creating and Running the Application"
- Section 1.4, "Contents of the SOADEMO Schema"
- Section 1.5, "How This Tutorial Is Organized"
- Section 1.6, "Technologies and Techniques Used in Each Project"

1.1 What Does the SOA Order Booking Application Do?

Run by a fictitious company called "Global Company", the SOA Order Booking application processes orders placed by customers. The application routes each order to two suppliers, Select Manufacturer and Rapid Service, to get quotes. The application chooses the supplier that provided the lower quote to fulfill the order.

To invoke the SOA Order Booking application, Global Company provides customers with a web front-end application called the SOADemo-Client application. The SOADemo-Client application enables customers to browse products and place their orders. This web front-end application is built using Application Development Framework (ADF) technology.

When the customer places an order using the SOADemo-Client application, the client application invokes the SOA Order Booking application. The SOA Order Booking application consists of different projects, each of which performs a specific function.

The entry point to the SOA Order Booking application is the OrderBookingESB project. This project invokes the SOAOrderBooking project, which is a Business Process Execution Language (BPEL) project that orchestrates all the services needed by the SOA Order Booking application.

For more information on projects in the application, see Section 1.6, "Technologies and Techniques Used in Each Project".

The SOA Order Booking application contains two types of projects:

- Projects that provide services. These projects include CustomerService, CreditService, SelectManufacturer, and RapidService. The services offered by these
projects are implemented as web services so that they can be invoked by different types of clients. For example, the CustomerService project is invoked by the SOADemo-Client application and also by the SOAOrderBooking BPEL project.

- Projects that define the flow of action in the application. These projects use the ESB (Enterprise Service Bus) or the BPEL technologies. These projects include OrderBookingESB, FulfillmentESB, and SOAOrderBooking, and they invoke the projects that provide services. They can also invoke other BPEL and ESB project. For example, the OrderBookingESB project invokes the SOAOrderBooking project (which is a BPEL project), and the SOAOrderBooking project invokes the FulfillmentESB project.

Table 1–1 provides a short description of the projects in the SOA Order Booking application.

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderBookingESB</td>
<td>Provides the starting point for the SOA Order Booking application. The SOADemo-Client application invokes OrderBookingESB, which invokes SOAOrderBooking, which invokes everything else.</td>
</tr>
<tr>
<td>SOAOrderBooking</td>
<td>Defines the main flow of the application. It invokes all the web services and performs the appropriate actions based on the results.</td>
</tr>
<tr>
<td>CustomerService</td>
<td>Provides web services for looking up existing customers in the database, and for adding new customers.</td>
</tr>
<tr>
<td>CreditService</td>
<td>Provides web services for checking whether or not a customer’s credit card is valid.</td>
</tr>
<tr>
<td>SelectManufacturer</td>
<td>Provides price quote for orders. This is one of the suppliers.</td>
</tr>
<tr>
<td>RapidService</td>
<td>Provides price quote for orders. This is another supplier.</td>
</tr>
<tr>
<td>FulfillmentESB</td>
<td>Determines how an order is to be shipped based on the dollar amount of the order.</td>
</tr>
</tbody>
</table>

In addition to the SOA Order Booking application and its projects, this tutorial also describes the SOADemo-Client application, which is the web front-end to the SOA Order Booking application.

1.2 Flow of the Application

This section describes the order in which the projects are invoked.

1. When a customer places an order using the SOADemo-Client application, this action invokes the OrderBookingESB project.

2. The OrderBookingESB project invokes the SOAOrderBooking project, which defines the main flow of the SOA Order Booking application.

3. The SOAOrderBooking project inserts the order information in the database.

4. The SOAOrderBooking project retrieves customer information from the database. It does this by invoking the CustomerService project.

5. The SOAOrderBooking project checks the customer’s credit. It does this by invoking the CreditService project.
6. The SOAOrderBooking project then determines if the order requires manual approval. It does this by using the rules defined in a rules repository. The application uses these rules:
   - If the customer’s status is platinum, then the order is approved automatically, regardless of the dollar amount of the order.
   - For non-platinum customers, if the dollar amount of the order is greater than or equal to $1000, then the order requires manual approval. If the order is under $1000, then the order is approved automatically.

7. For orders that require manual approval, a manager needs to log into the Worklist application and approve (or reject) the order.

8. For approved orders, the SOAOrderBooking project requests quotes from the suppliers: SelectManufacturer and RapidService.

9. After getting responses from the suppliers, the SOAOrderBooking project selects the supplier that responded with the lower quote.

10. The SOAOrderBooking project selects a shipping method (USPS or Fedex) for the order. It does this by invoking the FulfillmentESB project.

11. The FulfillmentESB project checks the dollar amount of the order. If the amount is under $500, then it selects the USPS as the shipper. Otherwise, Fedex is the shipper.

12. The SOAOrderBooking project sets the order status in the database.

13. The SOAOrderBooking project sends an email to the customer. It does this through the Email service, which is available in BPEL.

### 1.3 Software Required for Creating and Running the Application

To create the SOA Order Booking and SOADemo-Client applications, and to run them, you need the following software:

**Table 1–2  Required Software**

<table>
<thead>
<tr>
<th>Software</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle JDeveloper</td>
<td>Oracle JDeveloper is the development tool that you use to develop the applications. You need Oracle JDeveloper version 10.1.3.1.0.</td>
</tr>
<tr>
<td>Oracle Application Server</td>
<td>Oracle Application Server is the runtime platform on which you deploy and run the applications. You need Oracle Application Server Release 3 (10.1.3.1.0).</td>
</tr>
<tr>
<td>Oracle Database</td>
<td>Oracle Database is used to store data such as customer, order, and product information. To run the SOA Order Booking application, you create a schema called SOADEMO in the database and install tables with sample data in the schema. Projects in the SOA Order Booking application read from and write to tables in the SOADEMO schema. You need release 10g (10.2.x) of the Oracle Database. You cannot use the Oracle Lite database that is shipped with the Windows version of Oracle Application Server.</td>
</tr>
</tbody>
</table>

### 1.4 Contents of the SOADEMO Schema

The SOADEMO schema consists of the following tables:
The database sequences in the SOADEMO schema are:

<table>
<thead>
<tr>
<th>Table 1–4 Sequences in the SOADEMO Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence</td>
</tr>
<tr>
<td>ADDRESS_EJB_SEQ_ID_GEN</td>
</tr>
<tr>
<td>EJB_SEQ_ID_GEN</td>
</tr>
<tr>
<td>ORDER_SEQ_ID_GEN</td>
</tr>
</tbody>
</table>
Running in a Multi-Lingual or Multibyte Environment
If you are running the Oracle BPEL Process Manager in a multi-lingual environment or need multibyte support, it is recommended that your database character set encoding be Unicode. This means that the database character set encoding should be AL32UTF8. If the character set encoding is not Unicode, there may be possible loss or misinterpretation of data.

1.5 How This Tutorial Is Organized
The steps of the tutorial begin in Chapter 2, “Setting Up Your Environment”, where you begin by downloading and installing the required software and configuring the required connections in JDeveloper and Oracle Application Server.

Chapter 3 to Chapter 9 describe the projects in the SOA Order Booking application. Each chapter covers one project. You start by creating the projects that provide web services. After you have these services in place, you can create the SOAOrderBooking project, which, because it defines the main flow in the application, invokes almost all the web services.

Chapter 10, "Interfacing the Client Application with the SOA Order Booking Application", covers how the SOADemo-Client application invokes services in the SOA Order Booking application.

1.6 Technologies and Techniques Used in Each Project
The SOA Order Booking application integrates SOA Suite technologies such as BPEL and ESB, and uses them to invoke web services in a defined flow sequence. The web services are independent of each other and are generated in different ways.

Table 1–5 lists the technologies and techniques used in the projects in the SOA Order Booking application.

<table>
<thead>
<tr>
<th>Project</th>
<th>Technology and Techniques Used</th>
</tr>
</thead>
</table>
| CustomerService| ■ Uses EJB 3.0 entity objects that are generated from database tables. JDeveloper is used to generate the entity objects.  
■ Uses JSR-181 Web Services Metadata annotations.  
■ Uses a stateless session bean as the session facade. The session bean is generated by JDeveloper. |
| CreditService  | ■ Shows "top-down" implementation of web services: starting with a WSDL file, you use JDeveloper to generate Java classes from the WSDL file. |
| RapidService   | ■ Shows "bottom-up" implementation of web services: starting with Java classes, you use JDeveloper to generate a WSDL file.  
■ Uses JSR-181 Web Services Metadata annotations in the Java files. |
<p>| SelectManufacturer | ■ Shows a simple asynchronous BPEL process with Receive and Invoke activities. |</p>
<table>
<thead>
<tr>
<th>Project</th>
<th>Technology and Techniques Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>FulfillmentESB</td>
<td>Shows routing services that use filters to check input data. The filters then direct the requests to the appropriate targets. Uses transformation rules to transform data appropriately for writing to databases and files. Database adapters and file adapters perform the writes. Shows a routing service that routes to a JMS adapter.</td>
</tr>
<tr>
<td>SOAOrderBooking</td>
<td>Shows how to use BPEL to orchestrate a flow sequence. Invokes the services provided by all the projects (except for OrderBookingESB). Invokes other BPEL flows (the SelectManufacturer BPEL project). Invokes ESB project (the FulfillmentESB project). Shows how to integrate Oracle Business Rules with BPEL. Shows Decision Service. Sends email through the Email service. Uses the flow activity to send requests to RapidService and SelectManufacturer. Uses the human task to set up a step that requires manual approval.</td>
</tr>
<tr>
<td>OrderBookingESB</td>
<td>Invokes a BPEL project (the SOAOrderBooking project) by using a SOAP service.</td>
</tr>
<tr>
<td>SOADemo-Client</td>
<td>Shows how to invoke an ESB project from an ADF application (the &quot;place order&quot; button invokes the OrderBookingESB project). Shows how to call the CustomerService project from the &quot;login&quot; button.</td>
</tr>
</tbody>
</table>
This chapter describes how to set up the required software for creating and running the SOA Order Booking application.

This chapter contains the following sections:

- Section 2.1, "Download the Files for the Tutorial"
- Section 2.2, "Install Oracle Application Server"
- Section 2.3, "Install Oracle JDeveloper"
- Section 2.4, "Unzip the Files for the SOA Order Booking Application"
- Section 2.5, "Install the SOA DEMO Schema"
- Section 2.6, "Define Data Source and Connection Pool in Oracle Application Server"
- Section 2.7, "Set Up Connections in Oracle JDeveloper"
- Section 2.8, "Create the SOA DEMO Application in JDeveloper"

### 2.1 Download the Files for the Tutorial

To use this tutorial, you need to download files for:

- Oracle Application Server. This is your runtime environment.
- JDeveloper. This is your development environment.
- SOA Order Booking demo application. This is the application that this tutorial describes. The zip file that contains the files for the application is `soademoe_101310_prod.zip`.

You can download the files from the Service-Oriented Architecture page on the Oracle Technology Network site ([http://www.oracle.com/technology/soa](http://www.oracle.com/technology/soa)).

### 2.2 Install Oracle Application Server

Install Oracle Application Server Release 3 (10.1.3.1.0) from the zip file you downloaded. You can perform either the Basic installation or the Advanced installation. If you choose the Advanced installation, ensure that you install the SOA Suite, which is installed automatically in the Basic installation.

The tutorial assumes that your Oracle Application Server installation uses the default port of 8888. The installer configures Oracle HTTP Server or OC4J to listen on this port if the port is not in use. If the port is in use, the installer will try the next number on the list. See the *Oracle Application Server Installation Guide* for your platform for details.
If your installation uses a different port number, you need to change the port number in some files. These files will be pointed out as you develop the application.

Remember the password that you set for the oc4jadmin user. You will need this password for managing Oracle Application Server.

For details on installing Oracle Application Server, see the Oracle Application Server Installation Guide for your platform.

### 2.3 Install Oracle JDeveloper

Install JDeveloper Version 10.1.3.1.0 from the zip file you downloaded. You can install JDeveloper and Oracle Application Server on the same machine or on different machines.

The tutorial assumes that JDeveloper and Oracle Application Server are installed on the same machine. If you installed them on different machines, you will need to update the hostname in certain files. These files will be pointed out as you develop the application.

### 2.4 Unzip the Files for the SOA Order Booking Application

Unzip the soademo_101310_prod.zip file that you downloaded for the SOA Order Booking application. You can unzip the files into a temporary directory. When you create the application, you will create the files for the application in a different directory. You will copy some files from the temporary directory to the application directory when you develop the SOA Order Booking application.

For example, you can unzip the files into C:\temp, and you can build your application in C:\soademo.

### 2.5 Install the SOADEMO Schema

You need to have DBA privileges for the database where you want to install the SOADEMO schema.

---

**Note:** You must install the SOADEMO schema on Oracle Database 9i, 10g, or XE. If you want to install the schema on an Oracle Lite database, you need to modify the SQL scripts that install the SOADEMO schema and its objects.

---

To install the SOADEMO schema:

1. Change directory to where you unzipped the soademo_101310_prod.zip file for the SOA Order Booking application.

2. Check that the directory contains the following files:
   - build.sql
   - createSchema.sql
   - createSchemaObjects.sql
   - populateSchemaTables.sql

3. Check that the ORACLE_HOME and ORACLE_SID environment variables are set.

4. Run build.sql as DBA using SQL*Plus:
> sqlplus "sys/password as sysdba"
SQL> @build.sql

The script creates a schema called SOA DEMO with "ORACLE" as its password, and populates it with the tables described in Section 1.4, "Contents of the SOA DEMO Schema".

2.6 Define Data Source and Connection Pool in Oracle Application Server

You need to define data sources and connection pools in Oracle Application Server so that the SOA Order Booking application will be able to access the database during runtime.

2.6.1 Start Up Application Server Control

Application Server Control is the web-based tool that you use to manage Oracle Application Server. To access Application Server Control, enter the following URL in a browser:

http://hostname:port/em

hostname specifies the machine running Oracle Application Server.

port specifies the HTTP listening port. The installer configures Oracle HTTP Server or OC4J to listen on port 8888 if the port is not in use. If the port is in use, the installer will try the next number on the list. See the Oracle Application Server Installation Guide for your platform for details.

When Application Server Control prompts you to log in, enter oc4jadmin as the username. The password for oc4jadmin is the password that you set during installation.

2.6.2 Create a Connection Pool

Connection pools maintain connections to your database. For the SOA Order Booking application, you need to create a connection pool called soademo_pool.

1. Log into Application Server Control.
2. Click the home link to display the OC4J:home page.
3. On the OC4J:home page, click the Administration tab.
4. Click the icon in the Go To Task column for Services > JDBC Resources. See Figure 2–1 below. This displays the JDBC Resources page.
5. In the JDBC Resources page, click the Create button in the Connection Pools section (not in the Data Sources section). This displays the Create Connection Pool - Application page (Figure 2–2).
6. Click Continue to accept the defaults on the Create Connection Pool - Application page. This displays the Create Connection Pool page.

7. On the Create Connection Pool page, set the following values:
   - **Name**: enter `soademo_pool`.
   - **URL section**: enter the information to connect to the database where you installed the SOADEMO schema.
   - **Credentials section**: enter the username and password to connect to the database.
8. Click **Test Connection**. This displays the Test Connection page. Click **Test** on that page to verify that the connection information is valid. If the test failed, verify the connection values and try again.

9. Click **Finish**.

Continue with the next section to create a data source that uses the connection pool that you just created.
2.6.3 Create a Data Source

Create a data source that uses the connection pool that you created in the previous section.

1. On the JDBC Resources pages, click the Create button in the Data Sources section. This displays the Create Data Source - Application & Type page.

![Figure 2–4 Create Data Source - Application & Type Page]

2. Click Continue on the Create Data Source - Application & Type page to accept the default values. This displays the Create Data Source - Managed Data Source page.

3. On the Create Data Source - Managed Data Source page, enter the following values:
   - Name: enter soademoDS.
   - JNDI Location: enter jdbc/soademoDS.
   - Connection Pool: select soademo_pool.

   You can leave the other fields at their default values.

![Figure 2–5 Create Data Source - Managed Data Source Page]
4. Click **Finish**. This takes you back to the JDBC Resources page.

5. On the JDBC Resources page, click the icon in the Test Connection column for the **soademoDS** data source. This displays the Test Connection page. Click **Test** on the Test Connection page. You should get a confirmation message that the connection was made successfully.

### 2.6.4 Create a Database Adapter Connection Factory

Perform the following steps to create a database adapter connection factory:

1. Click the **OC4J:home** breadcrumb link at the top of the page.
2. Click the **Applications** link.
3. Click the **default** link in the applications table.
4. Click **DbAdapter** in the Modules table.
5. Click the **Connection Factories** link.
6. In the Connection Factories section, click the **Create** button. (Note: do not click the Create button in the Shared Connection Pools section.)
7. On the Create Connection Factory: Select Interface page, click **Continue** to accept the default values.
8. On the Create Connection Factory page, enter the following values:
   - **JNDI Location**: enter `eis/DB/soademo`.
   - **xADataSourceName**: enter `jdbc/soademoDS`.

For the other fields, accept the default values.
2.7 Set Up Connections in Oracle JDeveloper

In JDeveloper, you set up connections to the database, to Oracle Application Server, and to the Integration Server in Oracle Application Server. These connections enable you to view the data in the SOADEMO schema, and to deploy the applications to Oracle Application Server from JDeveloper.

Starting JDeveloper

To start Oracle JDeveloper:

1. Navigate to the directory where you installed Oracle JDeveloper and double-click the jdeveloper.exe executable.

   If this is the first time Oracle JDeveloper is run, a window asking "Do you wish to migrate?" appears.

2. Click No to continue. You will build the entire application from scratch.

2.7.1 Create a Database Connection to the SOADEMO Schema

In JDeveloper, create a database connection to the SOADEMO schema:

---

**Note:** In this tutorial, the database connection is named SOADEMO. You can use a different name if you want, but using the same naming conventions will make it easier to follow the instructions.
1. In Oracle JDeveloper, select View > Connection Navigator.

2. Right-click the Database node and select New Database Connection.

3. Click Next on the Welcome page.

4. In Step 1, Type, enter the following values:
   - Connection Name: enter SOADEMO.
   - Connection Type: select Oracle (JDBC).

   Click Next.

5. In Step 2, Authentication, enter the following values:
   - Username: enter SOADEMO.
   - Password: enter oracle.
   - Role: leave blank.
   - Deploy Password: select the check box.

   Click Next.

6. In Step 3, Connection, enter the following values:
   - Driver: select thin.
   - Host Name: enter the name of the machine running the database where you installed the SOADEMO schema.
   - JDBC Port: enter the port number for the database. The default value is 1521.
   - SID: enter the system identifier for the database. The default value is ORCL.

   If you are unsure about the database connection values, check with your database administrator.

   Click Next.

7. In Step 4, Test, click Test Connection. If the test is not successful, check that the database is available and that the connection values are correct. You can click the Back button to return to the previous page to edit the connection values.

8. Click Finish. The connection appears below the Database node in the Connection Navigator.

9. You can now examine the schema from Oracle JDeveloper. In the Connection Navigator, expand Database > soademo > SOADEMO. Expand the Sequences and Tables nodes and verify that the tables match those listed in Section 1.4, "Contents of the SOADEMO Schema".
2.7.2 Create a Connection to Oracle Application Server

In JDeveloper, create a connection to Oracle Application Server.

1. In Oracle JDeveloper, select View > Connection Navigator.

2. Right-click the Application Server node and select New Application Server Connection.

3. Click Next on the Welcome page.

4. In Step 1, Type, enter the following values:
   - Connection Name: enter SoademoApplicationServer.
   - Connection Type: select Oracle Application Server 10g 10.1.3.

5. In Step 2, Authentication, enter the following values:
   - Username: enter oc4jadmin. This is the name of the administration user.
   - Password: enter the password for oc4jadmin. This is the password that you set when you installed Oracle Application Server.
   - Deploy Password: select the check box.

6. In Step 3, Connection, enter the following values:
   - Connect To: select Single Instance.
   - Host Name: enter the name of the machine where you installed Oracle Application Server.
   - OPMN Port: enter the OPMN port for the Oracle Application Server instance.

   You can determine this port by looking in the ORACLE_
Set Up Connections in Oracle JDeveloper

HOME\opmn\conf\opmn.xml file. You want the port number specified in the request attribute. For example:

```
<notification-server>
    <port local="6100" remote="6202" request="6005"/>
```

- OC4J Instance Name: enter home.

Click Next.

7. In Step 4, Test, click Test Connection. If the test is not successful, check that the Oracle Application Server instance is available and that the connection values are correct. You can click the Back button to return to the previous page to edit the connection values.

8. Click Finish. The connection appears below the Application Server node in the Connection Navigator.

2.7.3 Create a Connection to the Integration Server

In JDeveloper, create a connection to the BPEL and ESB servers running on the Oracle Application Server instance.

1. In Oracle JDeveloper, select View > Connection Navigator.

2. Right-click the Integration Server node and select New Integration Server Connection.

3. Click Next on the Welcome page.

4. In Step 1, Name, enter the following value:
   - Connection Name: enter SoademoIntegConnection.

Click Next.

5. In Step 2, Connection, enter the following values:
   - Application Server: enter SoademoApplicationServer, which is the Application Server connection that you just created (in Section 2.7.2, "Create a Connection to Oracle Application Server").
   - Hostname: The value is derived from the Application Server connection.
   - Port Number: enter the port number that Oracle Application Server listens at for HTTP requests from clients. This is typically the Oracle HTTP Server port. If there is no Oracle HTTP Server component, then specify the OC4J port. The default port is 8888.
   - Add Hostname to the List of Proxy Exceptions: select this option.

Click Next.

6. In Step 3, Test Connection, click Test Connection. If the test is not successful, check that the Oracle Application Server instance is available and that the connection values are correct. You can click the Back button to return to the previous page to edit the connection values.

7. Click Finish. The connection appears below the Integration Server node in the Connection Navigator.
2.8 Create the SOADEMO Application in JDeveloper

In JDeveloper, perform these steps to create an application called "SOADEMO", which will contain the projects for the SOA Order Booking application:

1. In Oracle JDeveloper, select View > Application Navigator.
2. Right-click the Applications node and select New Application.
3. In the Create Application dialog, enter these values:
   - Application Name: enter SOADEMO.
   - Directory Name: Accept the default directory location or edit it to specify a different directory location. JDeveloper will create this directory, which will contain all the projects in the SOA Order Booking application.
   - Application Package Prefix: enter oracle.soadeMO.
   - Application Template: select No Template [All Technologies].
   Click OK.
4. In the Create Project dialog, click Cancel. You will create the projects later.

This tutorial refers to the directory that you specified as SOADEMO.
This chapter describes how to create the CustomerService project. It contains these sections:

- **Section 3.1, "About the CustomerService Project"
- **Section 3.2, "Create a New Project for CustomerService"
- **Section 3.3, "Create Entity Beans from Tables in the Database"
- **Section 3.4, "Edit persistence.xml"
- **Section 3.5, "Create a Session Bean in the business Directory"
- **Section 3.6, "Define Additional Queries in Customer.java"
- **Section 3.7, "Use a Database Sequence to Generate Address IDs in Address.java"
- **Section 3.8, "Edit Session Facade for the Session Bean (CustomerServiceBean.java)"
- **Section 3.9, "Add and Modify Methods in the Session Bean (CustomerServiceBean.java)"
- **Section 3.10, "Add JSR-181 Annotations to the Web Service Endpoint Interface (CustomerService.java)"
- **Section 3.11, "Create EJB JAR Deployment Profile for the CustomerService Project"
- **Section 3.12, "Deploy CustomerService"
- **Section 3.13, "View the WSDL for CustomerService"

### 3.1 About the CustomerService Project

The CustomerService project provides methods that enable client applications, such as the SOADemo-Client application, to retrieve customer information from the database and add customers to the database.

**Highlights of the CustomerService project:**

- The project is invoked directly by the SOADemo-Client application. When a customer logs in using the client application, the client application invokes the CustomerService project to validate the user.
- The project uses Java Persistence API entity objects to manage the CUSTOMER and ADDRESS tables in the SOADEMO schema in the database.
- The project uses an EJB 3.0 stateless session bean with JSR-181 Web Services annotations. This means that application servers that support JSR-181 are able to publish the bean as a web service during deployment. Methods in CustomerService that you want to make public (that is, invocable by client applications)
applications) are defined in this bean. Client applications can then invoke the methods through Web Services.

- The SOAOrderBooking project, which is a BPEL process, also invokes the CustomerService project to get details on the customer, such as the customer’s status (for example, platinum or gold).

### 3.2 Create a New Project for CustomerService

Start by creating a new project for CustomerService in JDeveloper:

1. Right-click the SOADEMO application, and select **New Project**.
2. In the New Gallery, in the Categories section, expand **General** and select **Projects**. In the Items section, select **Empty Project**.

![New Gallery for the CustomerService Project](image)

Click **OK**.

3. In the Create Project dialog, enter **CustomerService** in the **Project Name** field.

![Create Project Dialog for the CustomerService Project](image)

Click **OK**.

In the Application Navigator, you should see an empty CustomerService project located under the SOADEMO application.
4. Select File > Save to save your work.

3.3 Create Entity Beans from Tables in the Database

You can use JDeveloper to generate entity objects for the Customer and Address tables in the SOADEMO schema. These entity objects use the Java Persistence API available in EJB 3.0.

To generate the entity beans:

1. Right-click the CustomerService project, and select New. In the New Gallery, in the Categories section, expand Business Tier and select EJB. In the Items section, select Entities From Tables (JPA/EJB 3.0). Click OK.

![New Gallery for Creating Entity Objects from Database Tables](image)

This launches the Create Entities From Tables wizard.

2. On the Welcome page of the Create Entities From Tables wizard, click Next.

3. In Step 1, Database Connection Details, select the database connection and click Next.
4. In Step 2, Select Tables, check that:
   - **Name Filter** is blank.
   - **Auto-Query** is not selected.
   - **Schema** is set to SOADEMO.
   - **Tables** is selected.

5. Select **Address** and **Customer** and click the single blue arrow to move them to the Selected box.
Create Entity Beans from Tables in the Database

Creating the CustomerService Project

6. In Step 3, Entities From Tables:
   - Package Name: set it to org.soademo.customerservice.persistence.
   - Place member-level annotations on: Fields.
   - Implement java.io.Serializable: select this option.
   - Collection type for relationship fields: select java.util.List.

7. In Step 4, Specify Entity Details:
   - In Table Name, select SOADEMO.ADDRESS:
     - Entity Name: Address.
     - Bean Class: org.soademo.customerservice.persistence.Address.
Create Entity Beans from Tables in the Database

3-6  Oracle SOA Suite Tutorial

Figure 3–8  Create Entities From Tables Wizard: Step 4, Specify Entity Details, Showing Values for Address Table

In Table Name, select SOADEMO.CUSTOMER:
- Entity Name: Customer.

Figure 3–9  Create Entities From Tables Wizard: Step 4, Specify Entity Details, Showing Values for Customer Table

Click Next.
8. In the Summary screen, click Finish.
9. Select File > Save to save your work.

You should see these files:
- Address.java and Customer.java in the SOADEMO\CustomerService\src\org\soademocustomerservice\persistance directory
3.4 Edit persistence.xml

In CustomerService\src\META-INF\persistence.xml:

- Change the name of the persistence-unit from "CustomerService" to "customerServiceUnit".
  This name is used in the CustomerServiceBean.java session bean that you will create in Section 3.5, "Create a Session Bean in the business Directory".

- Add a line to define the jta-data-source:
  `<jta-data-source>jdbc/soademoDS</jta-data-source>
  jdbc/soademoDS is the data source that you created in Oracle Application Server in Section 2.6.3, "Create a Data Source".

You can make the changes using property dialogs in JDeveloper:

1. Check that you do not have any unsaved changes in the CustomerService project. In the Application Navigator, if you see any items that appear in italics, select the item and select File > Save.

2. Select View > Structure to display the Structure window.

3. Select persistence.xml in the Application Navigator. You should see the structure of the file in the Structure window.
4. Double-click `persistence-unit` in the Structure window. This displays the persistence-unit Properties dialog.

5. In the persistence-unit Properties dialog:
   - `name`: change it to `customerServiceUnit`.

6. Click OK. The `persistence.xml` file in the Application Navigator should appear in italics, which indicates that it has unsaved changes.

7. Double-click the `persistence.xml` file in the Application Navigator. This displays the contents of the file in the editor.

   The file should look like this:

```
Example 3–1  persistence.xml

<?xml version='1.0' encoding='windows-1252' ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
  xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name='customerServiceUnit'>
```

<jta-data-source>jdb/soademoDS</jta-data-source>
</persistence-unit>
</persistence>
8. Select File > Save to save the file.
9. Close the file in the editor.

3.5 Create a Session Bean in the business Directory

In the business directory, you create a session bean to define the methods available to clients.

To create the session bean:

1. Right-click the CustomerService project, and select New. In the New Gallery, in the Categories section, expand Business Tier and select EJB. In the Items section, select "Session Bean (EJB 1.1/2.x/3.0)". Click OK.

Figure 3–13  New Gallery for Creating Session Bean

This launches the Create Session Bean wizard.

2. On the Welcome page of the Create Session Bean wizard, click Next.

3. In Step 1, EJB Name and Options:
   - EJB Name: enter CustomerService.
   - Session Type: select Stateless.
   - Transaction Type: select Container.
   - Generate Session Facade Methods: select this option.
   - Entity Implementation: select EJB 3.0 Entities.
   - Persistence Unit: select customerServiceUnit (CustomerService.jpr).
Create a Session Bean in the business Directory

4. In Step 2, Session Facade - Select EJB 3.0 Entity Methods, **deselect everything**.
   You will generate methods for the Customer entity later.

5. In Step 3, Class Definitions:
   - **Bean Class**: enter `org.soadeemo.customerservice.business.CustomerServiceBean`.
   - **Source directory**: accept the default (`CustomerService\src` directory).
Create a Session Bean in the business Directory

Creating the CustomerService Project

Figure 3–16  Create Session Bean Wizard: Step 3, Class Definitions

Click Next.

6. In Step 4, EJB Component Interfaces:

- **Implement a Remote Interface**: select this option.

  **Note**: The name of the remote interface is different from the default value.

- **Implement a Local Interface**: select this option.
  - Local Interface: `org.soademo.customerservice.business.CustomerServiceLocal`

- **Include Web Service Endpoint Interface**: select this option.

  **Note**: The name of the web service endpoint interface is different from the default value.

Note that selecting the **Include Web Service Endpoint Interface** option adds the JSR-181 library to the project. This library is needed for the project to compile. You can also add this library to the project later, if it is not already added.

In the generated WSDL, the name of the web service endpoint interface ("CustomerService") is used as a prefix for message names. It is also used as the name for the port type. You need to consider this because this affects the SOAOrderBooking project. The CustomerSvc.wsdl file in the SOAOrderBooking\bpel directory specifies "CustomerService" as the name of the port type. The port type names in the CustomerSvc.wsdl file and in the generated CustomerService WSDL must match. After you deploy the
CustomerService project, you can view the generated WSDL as described in Section 3.13, "View the WSDL for CustomerService".

Figure 3–17  Create Session Bean Wizard: Step 4, EJB Component Interfaces

Click Next.

7. In the Summary page, click Finish.

8. Select CustomerService in the Application Navigator and select File > Save to save your work.

9. JDeveloper displays the CustomerServiceBean.java file in the editor. You can close it.

In the Application Navigator, you should see the CustomerService session bean in the business directory.

Figure 3–18  Application Navigator Showing the CustomerService Session Bean in the business Directory
3.6 Define Additional Queries in Customer.java

Edit the Customer.java file, located in the CustomerService\src\org\soademo\customerservice\persistence directory, to define additional queries and table information.

When you deploy the CustomerService project, Oracle Application Server uses the annotations that define the queries to generate a WSDL for the project and enable clients to access the project as a Web Service. This WSDL is generated by Oracle Application Server; you can view it after you have deployed the CustomerService project. See Section 3.13, "View the WSDL for CustomerService".

Edit the Customer.java file so that it looks like the following:

Example 3–2 Customer.java

```java
package org.soademo.customerservice.persistence;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.TableGenerator;

// Note that the query statements between the double quotes must be on one line.
@Entity
@NamedQueries({
    @NamedQuery(name = "Customer.findAllCustomer",
        query = "select object(o) from Customer o")
    ,
    @NamedQuery(name = "Customer.findCustomerById",
        query = "select object(cust) from Customer cust where cust.custid = :custid")
    ,
    @NamedQuery(name = "Customer.findCustomerByEmail",
        query = "select object(cust) from Customer cust where cust.email = :email and cust.password = :password")
}),
@Table(name = "CUSTOMER")
@TableGenerator(name = "TABLE_ID_GENERATOR", table = "EJB_TAB_ID_GEN",
    pkColumnName = "ID_NAME", valueColumnName = "SEQ_VALUE",
    pkColumnValue = "SEQ_GEN")
public class Customer implements Serializable {
    private String creditcardnumber;
```
private String creditcardtype;
@Id
@GeneratedValue(strategy = GenerationType.TABLE,
    generator = "TABLE_ID_GENERATOR")
@Column(nullable = false)
private String custid;
private String email;
private String fname;
private String lname;
private String phonenumber;
private String status;
private String password;

@OneToMany(fetch = FetchType.LAZY, cascade = { CascadeType.ALL } )
@JoinTable(name = "CUSTOMER_ADDRESS",
    joinColumns = { @JoinColumn(name = "CUSTID")
    }, inverseJoinColumns = { @JoinColumn(name = "ADDRESSID")
    })
private List<Address> addressList;

public Customer() {
}

public String getCreditcardnumber() {
    return creditcardnumber;
}

public void setCreditcardnumber(String creditcardnumber) {
    this.creditcardnumber = creditcardnumber;
}

public String getCreditcardtype() {
    return creditcardtype;
}

public void setCreditcardtype(String creditcardtype) {
    this.creditcardtype = creditcardtype;
}

public String getCustid() {
    return custid;
}

public void setCustid(String custid) {
    this.custid = custid;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getFname() {
    return fname;
}

public void setFname(String fname) {

this.fname = fname;
}

public String getLname() {
    return lname;
}

public void setLname(String lname) {
    this.lname = lname;
}

public String getPhonenumber() {
    return phonenumber;
}

public void setPhonenumber(String phonenumber) {
    this.phonenumber = phonenumber;
}

public String getStatus() {
    return status;
}

public void setStatus(String status) {
    this.status = status;
}

public List<Address> getAddressList() {
    if (addressList == null) {
        addressList = new ArrayList();
    }
    return addressList;
}

public void setAddressList(List<Address> addressList) {
    this.addressList = addressList;
}

public Address addAddress(Address address) {
    getAddressList().add(address);
    return address;
}

public Address removeAddress(Address address) {
    getAddressList().remove(address);
    return address;
}

public void setPassword(String password) {
    this.password = password;
}

public String getPassword() {
    return password;
}
3.7 Use a Database Sequence to Generate Address IDs in Address.java

Edit Address.java so that it uses the database sequence called ADDRESS_EJB_SEQ_ID_GEN to generate address IDs.

The lines in bold mark the changes that you have to make. The rest of the file remains the same.

```java
package org.soadeMO.customerservice.persistence;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQuery;

// add these import lines
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

@Entity
@NamedQuery(name = "Address.findAllAddress",
            query = "select object(o) from Address o")
@Table(name = "ADDRESS")
public class Address implements Serializable {
private String addresstype;
@Id
@SequenceGenerator(name = "ADDRESS_ID_GEN",
                   sequenceName = "ADDRESS_EJB_SEQ_ID_GEN")
@GeneratedValue(strategy = GenerationType.SEQUENCE,
generator = "ADDRESS_ID_GEN")
@Column(nullable = false)
private String addrid;
private String city;
private String country;
private String state;
private String street;
private String zip;
...
```

3.8 Edit Session Facade for the Session Bean
(CustomerServiceBean.java)

Edit CustomerServiceBean.java, located in the CustomerService\src\org\soadeMO\customerservice\business directory, so that it adds the methods that you defined through the queries in the Customer.java file in the previous section (Section 3.6, "Define Additional Queries in Customer.java").

To add the new methods to CustomerServiceBean.java:

1. Right-click CustomerServiceBean.java and select Edit Session Facade. This displays the Specify Session Facade Options dialog.
2. In the dialog, expand Customer and select:
   - findCustomerById
3.9 Add and Modify Methods in the Session Bean (CustomerServiceBean.java)

Make the following changes to CustomerServiceBean.java:

- Section 3.9.1, "Modify queryCustomerFindCustomerByCustomerId"
- Section 3.9.2, "Modify queryCustomerFindCustomerByEmail"
- Section 3.9.3, "Add getCustomerStatus and addNewCustomer Methods"

3.9.1 Modify queryCustomerFindCustomerByCustomerId

Modify the queryCustomerFindCustomerByCustomerId method:

- Change it so that it returns a single value instead of a list.
- Modify its name to findCustomerById.

Although you can edit the file directly, it is better to make the changes using the property dialog in JDeveloper because it will synchronize the changes in other files, such as the interface files (CustomerServiceRemote.java, CustomerServiceLocal.java, and CustomerService.java) that this class implements.
1. If CustomerServiceBean.java is not showing in the editor, double-click it in the Application Navigator.

2. Select View > Structure to display the Structure window.


4. In the Structure window, right-click the queryCustomerFindCustomerById method and select Properties. This displays the Bean Method Details dialog.

5. In the Bean Method Details dialog:
   - **Name**: change to findCustomerById.
   - **Return Type**: remove java.util.List and the angle brackets so that it returns only Customer, specified in its full package name (org.soadeemo.customerservice.persistence.Customer).
   - **Parameters**: change Object to String.

The dialog should look like this:

![Bean Method Details Dialog for findCustomerById](image)

6. Click OK. You can see the changes in the editor.

7. Make these additional changes to the findCustomerById method in the editor:
   - Cast the return value to Customer.
   - Change the last method to getSingleResult (instead of getResultList).

   Example 3–3 shows what the updated method should look like.

   **Example 3–3  The Updated findCustomerById**

   ```java
   public Customer findCustomerById(String custid) {
       return (Customer)em.createNamedQuery("Customer.findCustomerById")
               .setParameter("custid", custid).getSingleResult();
   }
   ```

8. Select File > Save to save your changes.

9. In the Structure window, expand Sources. You should see the interface files listed under Sources.
10. Double-click the interface files (CustomerServiceRemote.java, CustomerServiceLocal.java, and CustomerService.java) to display them in the editor. The interface files are listed under Sources in the Structure window.

After you double-click an interface file, you need to re-select CustomerServiceBean.java in the Application Navigator. The interface files are listed as the sources for CustomerServiceBean.java.

Notice that JDeveloper has updated the methods in these interface files to synchronize with your updates.

Figure 3–21 Application Navigator and Structure Window Showing the Structure for CustomerServiceBean.java

11. Save the interface files.

3.9.2 Modify queryCustomerFindCustomerByEmail

Make similar changes to the queryCustomerFindCustomerByEmail method like you did for the queryCustomerFindCustomerById method:

- Change it so that it returns a single value instead of a list.
- Modify its name to findCustomerByEmail.

Although you can edit the file directly, it is better to make the changes using the property dialog in JDeveloper because it will synchronize the changes in other files, such as the interface files (CustomerServiceRemote.java, CustomerServiceLocal.java, and CustomerService.java) that this class implements.


2. In the Structure window, right-click the queryCustomerFindCustomerByEmail method and select Properties. This displays the Bean Method Details dialog.
Add and Modify Methods in the Session Bean (CustomerServiceBean.java)

3. In the Bean Method Details dialog:
   - **Name**: change to `findCustomerByEmail`.
   - **Return Type**: remove `java.util.List` and the angle brackets so that it returns only `Customer`, specified in its full package name (`org.soade.demo.customerservice.persistence.Customer`).
   - **Parameters**: for both parameters, change `Object` to `String`.

   The dialog should look like this:

   ![Bean Method Details Dialog for findCustomerByEmail](image)

4. Click **OK**. You can see the changes in the editor.

5. Make these additional changes to the `findCustomerByEmail` method in the editor:
   - Cast the return value to `Customer`.
   - Change the last method to `getSingleResult` (instead of `getResultList`).

   **Example 3–4** shows what the updated method should look like.

   **Example 3–4** The Updated `findCustomerByEmail`

   ```java
   public Customer findCustomerByEmail(String email, String password) {
       return (Customer)em.createNamedQuery("Customer.findCustomerByEmail").
           getSingleResult();
   }
   ```

6. Select File > Save to save your changes.

7. In the Structure window, expand **Source** and double-click the interface files (`CustomerServiceRemote.java`, `CustomerServiceLocal.java`, and `CustomerService.java`) to display them in the editor. Notice that JDeveloper has updated the methods in these files to synchronize with your updates.

8. Save the interface files.
3.9.3 Add getCustomerStatus and addNewCustomer Methods

Add the getCustomerStatus and addNewCustomer methods to CustomerServiceBean.java. Although you can simply add these methods using the editor, it is better to enter them using the Bean Method Details dialog because JDeveloper can also update the interface files with these new methods.

To create a new method:

1. Display CustomerServiceBean.java in the editor.
2. Select View > Structure to display the Structure window, if it is not already showing.
3. In the Structure window, right-click Methods and select New Method. This displays the Bean Method Details dialog.
4. In the Bean Method Details dialog:
   - **Name**: enter getCustomerStatus.
   - **Return Type**: select java.lang.String.
   - **Parameters**: enter String CustomerID.

The dialog should look like this:

![Figure 3–23 Bean Method Details Dialog for getCustomerStatus](image)

5. Click OK.
6. In the editor for CustomerServiceBean.java, enter the body of the method. See Example 3–5:

**Example 3–5 getCustomerStatus**

```java
public String getCustomerStatus(String CustomerID) {
    return findCustomerById(CustomerID).getStatus();
}
```

7. Save the file.

   Notice that JDeveloper updates the interface files with this new method.
8. Repeat the same procedure to create the addNewCustomer method. In the Bean Method Details dialog, enter these values:
Add JSR-181 Annotations to the Web Service Endpoint Interface (CustomerService.java)

- Name: enter `addNewCustomer`.
- Return Type: select `java.lang.String`.
- Parameters: enter `Customer customer`.

The dialog should look like this:

![Bean Method Details for addNewCustomer](image)

9. Click `OK` in the dialog.
10. In the editor for `CustomerServiceBean.java`, add the body for the method.

   **Example 3–6 addNewCustomer**
   
   ```java
   public String addNewCustomer(Customer customer) {
       em.persist(customer);
       return "New customer added sucessfully to customer database";
   }
   ```

12. Save the interface files.

### 3.10 Add JSR-181 Annotations to the Web Service Endpoint Interface (CustomerService.java)

The `CustomerService.java` file was generated as the web service endpoint interface for the CustomerService session bean. You add JSR-181 annotations to this file to refine the names of methods and parameters.

Edit the file so that it looks like the following. You can edit the file in the editor.

**Example 3–7 CustomerService.java**

```java
package org.soadeo.customerservice.business;

import java.rmi.RemoteException;

import javax.ejb.Remote;

import javax.jws.WebMethod;
```
import javax.jws.WebParam;
import javax.jws.WebService;
import org.soademo.customerservice.persistence.Customer;

@ServiceName = "CustomerSvc",
@targetNamespace = "http://www.globalcompany.com/ns/customer"
public interface CustomerService {

@WebMethod
Customer findCustomerById(
@WebParam(name = "custid",
@targetNamespace = "http://www.globalcompany.com/ns/customer")
String custid) throws RemoteException;

@WebMethod
String getCustomerStatus(
@WebParam(name = "CustomerID",
@targetNamespace = "http://www.globalcompany.com/ns/customer")
String CustomerID);

@WebMethod
String addNewCustomer(
@WebParam(name = "customer",
@targetNamespace = "http://www.globalcompany.com/ns/customer")
Customer customer);

@WebMethod
Customer findCustomerByEmail(
@WebParam(name = "email",
@targetNamespace = "http://www.globalcompany.com/ns/customer")
String email,
@WebParam(name = "password",
@targetNamespace = "http://www.globalcompany.com/ns/customer")
String password);
}

### 3.11 Create EJB JAR Deployment Profile for the CustomerService Project

Create a deployment profile so that you can easily deploy the CustomerService project to Oracle Application Server from JDeveloper.

To create a deployment profile:

Create EJB JAR Deployment Profile for the CustomerService Project

**Figure 3–25  New Gallery for Creating a Deployment Profile for EJB JAR File**

![New Gallery for Creating a Deployment Profile for EJB JAR File](image)

Click **OK**.

2. In the Create Deployment Profile -- EJB JAR File dialog:
   - **Deployment Profile Name**: enter **CustomerService**.
   - **Directory**: use the default (CustomerService directory).

**Figure 3–26  Create Deployment Profile - EJB JAR File Dialog**

![Create Deployment Profile - EJB JAR File Dialog](image)

Click **OK**.

3. In the EJB JAR Deployment Profile Properties dialog, accept the defaults.
4. Click OK.

You should see CustomerService.deploy under Resources in the Application Navigator.

3.12 Deploy CustomerService

1. To deploy the CustomerService project, right-click CustomerService.deploy in the Application Navigator and select Deploy To > SoademoApplicationServer, where SoademoApplicationServer specifies the connection to Oracle Application Server. JDeveloper compiles the application and creates JAR and EAR files in the CustomerService\deploy directory. If there are no errors, it displays the Configure Application dialog.
2. Click OK.
Watch the messages in the JDeveloper log area for any errors.

3.13 View the WSDL for CustomerService

You can view the WSDL for CustomerService by entering the following URL in a browser:


For hostname, enter the name of the machine running Oracle Application Server.
For port, enter the HTTP port at which Oracle HTTP Server or OC4J is listening. The default port is 8888.

After deploying CustomerService, you should see it in two places in the Application Server Control: in the Web Services tab (Figure 3–30) and the Applications tab (Figure 3–31) of the OC4J:home page.
View the WSDL for CustomerService

Figure 3–30  Web Services Tab of OC4J:home Page Showing the CustomerService Web Service

Figure 3–31  Applications Tab of OC4J:home Page Showing the CustomerService Application
View the WSDL for CustomerService
This chapter describes how to create the FulfillmentESB project. It contains these sections:

- Section 4.1, "About the FulfillmentESB Project"
- Section 4.2, "Create a New Project for FulfillmentESB"
- Section 4.3, "Create a System Called "Fulfillment""
- Section 4.4, "Create the "OrderFulfillment" Routing Service"
- Section 4.5, "Create the "Shipment" Routing Service"
- Section 4.6, "Create the "USPSShipment" Adapter (File Adapter)"
- Section 4.7, "Create the "FedexShipment" Adapter (Database Adapter)"
- Section 4.8, "Create the "FulfillmentBatch" Adapter (JMS Adapter)"
- Section 4.9, "Create Routing Rules"
- Section 4.10, "Save All Files in the FulfillmentESB Project"
- Section 4.11, "Register the FulfillmentESB Project"

### 4.1 About the FulfillmentESB Project

After an order has been approved, the SOAOrderBooking project invokes the FulfillmentESB project to determine the shipping method for the order. Currently there are two possible shipping methods: USPS and Fedex. The FulfillmentESB project sends orders under $500 to USPS, and orders $500 and over to Fedex.

Orders are sent to the USPS through a file adapter. The order information is written to a file.

For Fedex, orders are sent through a database adapter. The order information is written to a database.

The FulfillmentESB project also sends orders to a JMS adapter. For this tutorial, the order is just sent to a JMS queue. There is no consumer of the order data from the queue. This is just to show you how to send messages to a JMS adapter, if you are planning on using one.

The FulfillmentESB project is an ESB project. In JDeveloper, the completed project looks like the following:
The FulfillmentESB project consists of:

- OrderFulfillment routing service
- Shipment routing service
- USPSShipment file adapter
- FedexShipment database adapter
- FulfillmentBatch JMS adapter

4.2 Create a New Project for FulfillmentESB

Start by creating a new ESB project in JDeveloper:

1. Right-click the SOADEMO application, and select **New Project**.
2. In the New Gallery, in the Categories section, expand **General** and select **Projects**. In the Items section, select **ESB Project**.
Click OK.

3. In the Create ESB Project dialog, enter **FulfillmentESB** in the **Project Name** field. Accept the defaults for the other fields.

Click OK.

JDeveloper displays a blank page for the **FulfillmentESB.esb** file. In the Application Navigator, this file is located under **FulfillmentESB > Resources**:
4.3 Create a System Called "Fulfillment"

For an ESB project, all ESB activities belong to a system. By default, JDeveloper places the activities in a system called "DefaultSystem".

To make your ESB projects easier to maintain, you can create a new system and set the ESB activities in a project to belong to that system. During runtime, when you manage the project in the ESB Console, you can configure properties for the activities in a system.

If you do not create a system, then in the ESB Console, you will find all your activities under "DefaultSystem".

For the FulfillmentESB project, you create a system called "Fulfillment" and set all the activities in the FulfillmentESB project to belong to this system. The OrderBookingESB project, covered in Chapter 9, also has its own system.

1. In the empty FulfillmentESB.esb page, click the Create System/Group icon, located at the top of the page.

2. In the Create ESB System or Service Group dialog:
   - **System**: select this option.
   - **Name**: enter Fulfillment.
   - **Description**: leave blank or enter a brief description.
3. Click OK. In the Application Navigator, you should see a Fulfillment.esbsys file under FulfillmentESB > Resources.

4.4 Create the "OrderFulfillment" Routing Service

The OrderFulfillment routing service routes requests to two places:

- Shipment, which is another routing service
- FulfillmentBatch, which sends messages to a Java Messaging Service (JMS)

To create the OrderFulfillment routing service:

1. Copy the following file from the soademo_101310_prod.zip file to the SOADEMO\FulfillmentESB directory:
   - OrderBookingPO.xsd
   In the zip file, this file is located in the FulfillmentESB directory.
   This file is needed by the OrderFulfillment routing service.

2. If the Component Palette is not showing in JDeveloper, select View > Component Palette to display it.

3. Select ESB Services from the dropdown in the Component Palette.

4. Drag the Routing Service icon from the Component Palette and drop it anywhere on the FulfillmentESB page.

5. In the Create Routing Service dialog:
   - Name: enter OrderFulfillment.
   - System/Group: set to Fulfillment. If it is not set to Fulfillment, click the flashlight, which displays the ESB Service Group Browser dialog. Select Fulfillment in the dialog and click OK.
Create the "OrderFulfillment" Routing Service

Figure 4–7 ESB Service Group Browser

- **Generate WSDL From Schemas**: select this option.
- **Schema Location**: click **Browse**, which displays the Type Chooser dialog. In the dialog, select **Project Schema Files > OrderBookingPO.xsd > PurchaseOrder**.

Figure 4–8 Type Chooser Dialog for OrderFulfillment Routing Service

Click **OK** in the Type Chooser dialog.

- **Schema Element**: select **PurchaseOrder**.
- **Operation Name**: enter **execute**.
- **Namespace**: enter [http://www.globalcompany.com/ns/Fulfillment](http://www.globalcompany.com/ns/Fulfillment).

The Create Routing Service dialog should now look like this.
6. Click **OK** in the Create Routing Service dialog.

In JDeveloper, you should see this routing service instance:

**Figure 4–10 OrderFulfillment Routing Service in JDeveloper**

JDeveloper created the following files in the **FulfillmentESB** directory.

- **Fulfillment_OrderFulfillment.esbsvc** - this file defines the "OrderFulfillment" routing service. JDeveloper prefixes the system name to the filename.
- **Fulfillment_OrderFulfillment.wsdl** - this WSDL file is for the "OrderFulfillment" routing service.
4.5 Create the "Shipment" Routing Service

The Shipment routing service routes requests either to the FedexShipment database adapter or to the USPSshipment file adapter. You will define a filter rule in the Shipment routing service to route orders greater than or equal to $500 to the FedexShipment, while orders less than $500 are routed to the USPSshipment.

To create the Shipment routing service:

1. Drag the Routing Service icon from the Component Palette and drop it anywhere on the FulfillmentESB page.

2. In the Create Routing Service dialog:
   - **Name**: enter Shipment.
   - **System/Group**: should be set to Fulfillment. If not, click the flashlight icon to change it.
   - **Generate WSDL From Schemas**: select this option.
   - **Schema Location**: click Browse, which displays the Type Chooser dialog. In the dialog, select Project Schema Files > OrderBookingPO.xsd > PurchaseOrder.

   Figure 4–11 Type Chooser Dialog for Shipment Routing Service

   Click **OK** in the Type Chooser dialog.
   - **Schema Element**: select PurchaseOrder.
   - **Operation Name**: enter execute.
   - **Namespace**: enter http://www.globalcompany.com/ns/shipment.

The Create Routing Service dialog should now look like this:
3. Click **OK** in the Create Routing Service dialog.

In JDeveloper, you should see two routing services:

**Figure 4–13  JDeveloper Showing the OrderFulfillment and Shipment Routing Services**

JDeveloper created the following files in the *FulfillmentESB* directory.
4.6 Create the "USPSShipment" Adapter (File Adapter)

The "USPSShipment" service is one of the targets for the "Shipment" routing service. When the "Shipment" routing service sends an order to the "USPSShipment" service, the "USPSShipment" service writes the order information to a flat file in the C:\temp directory.

To create the "USPSShipment" service:

1. Copy the following file from the soademo_101310_prod.zip file to the FulfillmentESB directory:
   - USPSShipment.xsd
   In the zip file, this file is located in the FulfillmentESB directory.

2. In the Component Palette, select Adapter Services from the dropdown.

3. Drag the File Adapter icon from the Component Palette and drop it anywhere on the FulfillmentESB page.

4. In the Create File Adapter Service dialog:
   - **Name**: enter USPSShipment.
   - **System/Group**: should be set to Fulfillment. If not, click the flashlight icon to change it.
   - **WSDL File**: click the Configure Adapter Service WSDL icon (the one on the left), which launches the Adapter Configuration wizard.

   Click Next on the Welcome page of the wizard.

   Figure 4–14 Adapter Configuration Wizard, Step 1, Service Name
Create the "USPSShipment" Adapter (File Adapter)

**Step 1:**
- **Service Name:** enter USPSShipment.
- **Click Next.**

**Step 2, Operation:**

*Figure 4–15 Adapter Configuration Wizard, Step 2, Operation*

- **Operation Type:** select Write File.
- **Operation Name:** should be filled in with Write.
- **Click Next.**

**Step 3, File Configuration:**

*Figure 4–16 Adapter Configuration Wizard, Step 3, File Configuration*

- **Directory for Outgoing Files:** enter the directory where you want the adapter to write the files. For example, you can enter C:temp.
Create the "USPSShipment" Adapter (File Adapter)

**Note:** This directory is located on the machine where you are running Oracle Application Server, not the machine where you are running JDeveloper.

- **File Naming Convention:** specify how you want to name the files. Enter `shipment_%SEQ%.txt`. The `%SEQ%` indicates that the filenames will be numbered sequentially.

- **Number of Messages Equals:** You can leave it at 1, which means that each order will be written in a separate file.

- **Click Next.**

**Step 4, Messages:**

**Figure 4–17 Adapter Configuration Wizard, Step 4, Messages**

- **Native format translation is not required (Schema is Opaque):** do not select this option.

- **Schema Location.** Click **Browse.** In the Type Chooser, select **Project Schema Files > USPSShipment.xsd > shipment.**
Click **OK** in the Type Chooser.

- **Schema Element**: should be set to `shipment`.
- Click **Next**.

On the Finish page, click **Finish**.

The wizard creates the following files in the `FulfillmentESB` directory:

- `USPSShipment.wsdl` - this file contains the information you specified in the wizard.
- `fileAdapterOutboundHeader.wsdl` - this file contains generic information for writing to files.

The Create File Adapter Service dialog should now look like this. JDeveloper automatically filled in **Write_ptt** for Port Type.
5. Click OK in the Create File Adapter Service dialog.

In JDeveloper, you should see two routing services and a file adapter service:

4.7 Create the "FedexShipment" Adapter (Database Adapter)

The FedexShipment service is a database adapter. When the Shipment routing service routes an order to the FedexShipment adapter, the FedexShipment adapter writes the order information to the FEDEXSHIPMENT table in the SOADEMO schema.

To create the FedexShipment adapter:
1. Copy the following file from the soademo_101310_prod.zip file to the FulfillmentESB directory:
   - FedexShipment_table.xsd
   In the zip file, this file is located in the FulfillmentESB directory.

2. In the Component Palette, select Adapter Services from the dropdown.

3. Drag the Database Adapter icon from the Component Palette and drop it anywhere on the FulfillmentESB page.

4. In the Create Database Adapter Service dialog:
   - **Name**: enter *FedexShipment*.
   - **System/Group**: should be set to *Fulfillment*. If not, click the flashlight icon to change it.
   - **WSDL File**: click the Configure Adapter Service WSDL icon (the one on the left), which launches the Adapter Configuration wizard.
     - Click Next on the Welcome page of the wizard.
     - **Step 1, Service Name**: 
       - Service Name: enter *FedexShipment*.
       - Click Next.

   **Figure 4-21 Adapter Configuration Wizard, Step 1, Service Name**

   - **Step 2, Service Connection:**

Creating the FulfillmentESB Project 4-15
Figure 4–22  Adapter Configuration Wizard, Step 2, Service Connection

- **Connection**: select the database connection for the SOADEMO schema.

- **JNDI Name**: enter `eis/DB/soademo`. This is the JNDI location you specified when you created connection factory in Oracle Application Server (see Section 2.6.4, "Create a Database Adapter Connection Factory").

- **Click Next**.

Step 3, Operation Type:

Figure 4–23  Adapter Configuration Wizard, Step 3, Operation Type

- **Select Perform an Operation on a Table**, and select **Insert Only**. The SOA Order Booking application only needs to insert rows in the table.

- **Click Next**.
Step 4, Select Table, click **Import Tables**.

In the Import Tables dialog, click **Query**.

Select **FEDEXSHIPMENT** and click the right-arrow button to move it to the Selected box. The FEDEXSHIPMENT table is the table that the database adapter will write to.

**Figure 4–24  Import Tables Dialog for FedexShipment**

Click **OK** in the Import Tables dialog. Step 4, Select Table, now looks like this:

**Figure 4–25  Adapter Configuration Wizard, Step 4, Select Table**

Select the **SOADEMO.FEDEXSHIPMENT** table and click **Next**.

In Step 5, Relationships, click **Next**.
Create the "FedexShipment" Adapter (Database Adapter)

On the Finish page, click **Finish**.

The wizard creates the following files in the **FulfillmentESB** directory:

- **DBAdapterOutboundHeader.wsdl** -- this file contains the generic information for connecting to a database.
- **FedexShipment.wsdl** -- this file contains the information that you specified in the wizard.
- **FedexShipment_toplink_mappings.xml** -- this file is used by TopLink
  - **src\***
  - **database\***
  - **toplink\***

The Create Database Adapter Service dialog now looks like this:
Create the "FulfillmentBatch" Adapter (JMS Adapter)

4.8 Create the "FulfillmentBatch" Adapter (JMS Adapter)

The "OrderFulfillment" routing service routes all orders to the "Shipment" routing service and to the "FulfillmentBatch" adapter. The "FulfillmentBatch" adapter sends the order information to a JMS server.

To create the FulfillmentBatch JMS adapter:

5. Click **OK** in the Create Database Adapter Service dialog.

In JDeveloper, you should see two routing services, a file adapter service, and a database adapter service.
1. In the Component Palette, select Adapter Services from the dropdown.

2. Drag the JMS Adapter icon from the Component Palette and drop it anywhere on the FulfillmentESB page.

3. In the Create JMS Adapter Service dialog:
   - Name: enter FulfillmentBatch.
   - System/Group: should be set to Fulfillment. If not, click the flashlight icon to change it.
   - WSDL File: click the Configure Adapter Service WSDL icon (the one on the left), which launches the Adapter Configuration wizard.
     
     Click Next on the Welcome page of the wizard.

     Step 1, Service Name:

     Figure 4-29  Adapter Configuration Wizard, Step 1, Service Name

     - Service Name: enter FulfillmentBatch.
     - Click Next.

     Step 2, JMS Provider:
Create the "FulfillmentBatch" Adapter (JMS Adapter)

**Creating the FulfillmentESB Project**

---

**Figure 4–30 Adapter Configuration Wizard, Step 2, JMS Provider**

- Select **Oracle Enterprise Messaging Service (OEMS)** and select **Memory/File** from the dropdown.
- Click **Next**.

**Step 3, Service Connection:**

**Figure 4–31 Adapter Configuration Wizard, Step 3, Service Connection**

- In the **Connection** dropdown, select the connection to Oracle Application Server.
- Click **Next**.

**Step 4, Operation:**
Create the "FulfillmentBatch" Adapter (JMS Adapter)

Figure 4–32  Adapter Configuration Wizard, Step 4, Operation

- **Operation Type**: select **Produce Message**.
- **Operation Name**: enter **sendMessage**.
- Click **Next**.

Step 5, Produce Operation Parameters, click **Browse**. This displays the Select Destination dialog. In the dialog, select **demoQueue (queue)**, and click **OK**.

Figure 4–33  Select Destination Dialog
The rest of the values in Step 5 are filled in for you. You can just accept these values.

Figure 4–34 Adapter Configuration Wizard, Step 5, Produce Operation Parameters

Click Next.

Step 6, Messages:

- **Native format translation is not required (Schema is Opaque):** do not select this option.

- **Schema Location:** click Browse, which displays the Type Chooser. Select Project Schema Files > OrderBookingPO.xsd > PurchaseOrder.

Figure 4–35 Type Chooser for the FulfillmentBatch JMS Adapter Service

Click OK in the Type Chooser.
Create the "FulfillmentBatch" Adapter (JMS Adapter)

- **Schema Element**: select **PurchaseOrder**.

**Figure 4–36  Adapter Configuration Wizard, Step 6, Messages**

![Image](image_url)

Click **Next**.

On the Finish page, click **Finish**.

The wizard created the following files in the **FulfillmentESB** directory:

- jmsAdapterOutboundHeader.wsdl
- FulfillmentBatch.wsdl

The Create JMS Adapter Service dialog now looks like this:

**Figure 4–37  Create JMS Adapter Service Dialog for the FulfillmentBatch JMS Adapter Service**

![Image](image_url)

4. Click **OK** in the Create JMS Adapter Service dialog.

In JDeveloper, you should see two routing services, a file adapter service, a database adapter service, and a JMS adapter service.
4.9 Create Routing Rules

To route order information from one service to another, you set up routing rules. Routing rules enable you to define filters and transformations. Filters enable you to define conditions under which a service gets to process an order, and transformations enable you to map data so that the target service is able to process the data correctly.

For the FulfillmentESB project, you create routing rules between the following services:

- Section 4.9.1, "Between OrderFulfillment and Shipment"
- Section 4.9.2, "Between OrderFulfillment and JMS Adapter"
- Section 4.9.3, "Between Shipment and USPSShipment"
- Section 4.9.4, "Between Shipment and FedexShipment"

4.9.1 Between OrderFulfillment and Shipment

The OrderFulfillment routing service routes orders to two places: Shipment and FulfillmentBatch.

To create routing rule between OrderFulfillment and Shipment:

1. Double-click OrderFulfillment in the top section of the icon. This displays the page for the Fulfillment_OrderFulfillment.esbsvc file.
2. In the Routing Rules section, click the [+] to expand it.

3. Click the green + icon to add a routing route. This displays the Browse Target Service Operation dialog. You may have to scroll to the right to see the green + icon.

4. In the Browse Target Service Operation dialog, select **Services In Project > Fulfillment > Shipment > execute.**

5. Click **OK** in the Browse Target Service Operation dialog.
   The routing rule area now looks like this:
Create Routing Rules

Figure 4–41  Routing Rule from OrderFulfillment to Shipment

6. Select File > Save to save your work.

For this routing rule, there are no filters and no transformations. The OrderFulfillment routing service routes all orders to the Shipment routing service. The data is not transformed in any way.

Click the FulfillmentESB.esb tab in JDeveloper. You should see an arrow going from OrderFulfillment to Shipment.

Figure 4–42  JDeveloper Showing Routing from OrderFulfillment to Shipment

4.9.2 Between OrderFulfillment and JMS Adapter

The OrderFulfillment routing service routes orders to two places: Shipment and FulfillmentBatch (which is a JMS adapter service).

To create routing rule from OrderFulfillment to FulfillmentBatch:

1. Double-click OrderFulfillment in the top section of the icon. This displays the page for the Fulfillment_OrderFulfillment.esbsvc file.

2. In the Routing Rules section, click the [+ ] to expand it. You should see the rule that you created in the previous section (Section 4.9.1, "Between OrderFulfillment and Shipment").
3. Click the green + icon to add another routing route. You may have to scroll to the right to see the + icon.

4. In the Browse Target Service Operation dialog, select Services In Project > Fulfillment > FulfillmentBatch > sendMessage.

5. Click OK in the Browse Target Service Operation dialog. The routing rule area now contains two rules:
Create Routing Rules

Creating the FulfillmentESB Project

4.9.3 Between Shipment and USPSShipment

The Shipment routing service routes orders that are under $500 to the USPSShipment file adapter and orders that are $500 and over to the FedexShipment database adapter.

This section shows the routing rule for USPSShipment. It uses a filter to set the $500 rule, and it uses transformation to control what information is passed to USPSShipment. USPSShipment, which is a file adapter service, then writes the information to a file in the C:\temp directory.

To create the routing rule from Shipment to USPSShipment:

1. Double-click Shipment in the top section of the icon. This displays the page for the Fulfillment_Shipment.esbsvc file.
2. In the Routing Rules section, click the [+] to expand it.

6. Select File > Save to save your work.

Click the FulfillmentESB.esb tab in JDeveloper. From OrderFulfillment, there should be two arrows: one going to Shipment and one going to FulfillmentBatch.
3. Click the green + icon to add a routing route. You may have to scroll to the right to see the + icon.

4. In the Browse Target Service Operation dialog, select Services In Project > Fulfillment > USPSShipment > Write.

Figure 4–47  Browse Target Service Operation Dialog

![Browse Target Service Operation Dialog]

5. Click OK in the Browse Target Service Operation dialog.

The routing rule area now looks like this:

Figure 4–48  Routing Rule from Shipment to USPSShipment

![Routing Rule from Shipment to USPSShipment]

6. Select File > Save to save your work.

Click the FulfillmentESB.esb tab in JDeveloper. You should see an arrow going from Shipment to USPSShipment.
7. Create the filter so that only orders under $500 are directed to USPSShipment.
   
a. Double-click Shipment in the top section of the icon to display the page for the Fulfillment_Shipment.esbsvc file.

b. In the page for the Fulfillment_Shipment.esbsvc file, click the filter icon in the routing rules area. This displays the Expression Builder dialog.

c. In the Expression Builder dialog, in the WSDL Message box, select Purchase Order_request > PurchaseOrder > inp1:PurchaseOrder > inp1:OrderInfo > inp1:OrderPrice.

   The Content Preview box shows the path:

d. Click Insert Into Expression. The path appears in the Expression box at the top of the dialog.

e. In the Expression box, append < 500 to the path, so that it now reads /inp1:PurchaseOrder/inp1:OrderInfo/inp1:OrderPrice < 500.
Create Routing Rules

Figure 4–50  Expression Builder Dialog

f. Click OK in the Expression Builder.

g. Select File > Save to save your work.

8. Create a transformation so that the USPSShipment file adapter gets the proper information in the proper fields.

a. Click the transformation icon in the Routing Rules area.

b. In the Request Transformation Map dialog, select Create New Mapper File and enter PurchaseOrder_To_USPSshipment.xsl as the filename.

Figure 4–51  Request Transformation Map Dialog

c. Click OK. This brings up the Data Mapper.
d. On the source (left) side, expand `inp1:ShipTo > inp1:Name` and `inp1:ShipTo > inp1:Address`.

e. Click and drag `inp1:First` to `imp1:fname`.

f. Click and drag `inp1:Last` to `imp1:lname`.

There should be two lines, one for First/fname and another one for Last/lname.

**Figure 4–53  Data Mapper Showing Transformation for First/fname and Last/lname**

FIGURE 4–53  DATA MAPPER SHOWING TRANSFORMATION FOR FIRST/FNAME AND LAST/LNAME

**g.** Click and drag to connect these pairs:

- `inp1:Street` to `imp1:address`
- `inp1:City` to `imp1:city`
- `inp1:State` to `imp1:state`
- `inp1:Zip` to `imp1:zipcode`
- `inp1:Country` to `imp1:country`

**h.** Select Conversion Functions from the dropdown in the Component Palette. You will need the string function for the next step.

**i.** For each of the pairs listed in step (g), drag the string function from the Component Palette and drop it on each of the lines. You should end up with a diagram that looks like this:
9. Select File > Save to save your work.

10. Click the Fulfillment_Shipment.esbsvc tab in JDeveloper. In the Routing Rules area, in the transformation section, you should see the name of the transformation file.

11. Select File > Save to save Fulfillment_Shipment.esbsvc.

12. If you click the FulfillmentESB.esb tab, you can see that the filter and transformation icons for the Shipment routing service are no longer greyed out, unlike those for the OrderFulfillment routing service.
4.9.4 Between Shipment and FedExShipment

The Shipment routing service routes orders that are under $500 to the USPSShipment file adapter and orders that are $500 and over to the FedExShipment database adapter.

This section shows the routing rule for FedExShipment. It uses a filter to set the $500 rule, and it uses transformation to control what information is passed to FedExShipment. FedExShipment, which is a database adapter service, then writes the information to the FEDEXSHIPMENT table in the SOADEMO schema.

To create the routing rule from Shipment to FedExShipment:

1. Double-click Shipment or click the tab for Fulfillment_Shipment.esbsvc, if you have it open. This displays the page for the Fulfillment_Shipment.esbsvc file.
2. In the Routing Rules section, click the [+ ] to expand it.
3. Click the green + icon to add another routing route. You may have to scroll to the right to see the + icon.
4. In the Browse Target Service Operation dialog, select Services In Project > Fulfillment > FedExShipment > insert.

Figure 4–56 Browser Target Service Operation Dialog

5. Click OK in the Browse Target Service Operation dialog.

The routing rule area now looks like this:

Figure 4–57 Routing Rule from Shipment to FedExShipment
6. Select File > Save to save Fulfillment_Shipment.esbsvc.

Click the FulfillmentESB.esb tab in JDeveloper. From Shipment, you should see two arrows: one going to USPSShipment and another one going to FedexShipment.

**Figure 4–58  JDeveloper Showing Routing from Shipment to USPSShipment and FedexShipment**

7. Create the filter that directs orders that are equal to or greater than $500 to FedexShipment.

a. Double-click Shipment to display the page for the Fulfillment_Shipment.esbsvc file.

b. In the page for the Fulfillment_Shipment.esbsvc file, click the filter icon for the FedexShipment rule in the routing rules area. This displays the Expression Builder dialog.

c. In the Expression Builder dialog, in the WSDL Message box, select **Purchase Order_request > PurchaseOrder > inp1:PurchaseOrder > inp1:OrderInfo > inp1:OrderPrice**.

The Content Preview box shows the path: /inp1:PurchaseOrder/inp1:OrderInfo/inp1:OrderPrice.

d. Click **Insert Into Expression**. The path appears in the **Expression** box at the top of the dialog.

e. In the Expression box, append >= 500 to the path, so that it now reads /inp1:PurchaseOrder/inp1:OrderInfo/inp1:OrderPrice >= 500.
f. Click OK in the Expression Builder.

g. Select File > Save to save your work.

8. Create a transformation so that the FedexShipment database adapter gets the proper information in the proper fields.

a. Click the transformation icon in the Routing Rules area.

b. In the Request Transformation Map dialog, select Create New Mapper File and enter PurchaseOrder_To_FedexshipmentCollection.xsl as the filename.

c. Click OK. This brings up the Data Mapper.

d. On the source (left) side, expand inp1:ShipTo > inp1:Name and inp1:ShipTo > inp1:Address.

e. On the target (right) side, expand top:FedexshipmentCollection > top:Fedexshipment.

f. Click and drag to connect these pairs:
   - inp1:ID to top:orderid
   - inp1:First to top:fname
Create Routing Rules

- inp1:Last to top:lname
- inp1:Street to top:street
- inp1:City to top:city
- inp1:State to top:state
- inp1:Zip to top:zipcode

Figure 4–61 Data Mapper Showing Transformation from Shipment to FedExShipment

9. Select File > Save to save PurchaseOrder_To_FedexShipmentCollection.xsl.

10. Click the Fulfillment_Shipment.esbsvc tab in JDeveloper. In the Routing Rules area, in the transformation section, you should see the name of the transformation file.
11. Select File > Save to save Fulfillment_Shipment.esbsvc.

12. If you click the FulfillmentESB.esb tab, you can see that there are two rows of icons in the Shipment routing service icon and these icons are not greyed out.

4.10 Save All Files in the FulfillmentESB Project

Browse through the FulfillmentESB files in the Application Navigator to ensure that all the files are saved. If you see a file in italics, select the file and save it.

4.11 Register the FulfillmentESB Project

In the Application Navigator, right-click the FulfillmentESB project and select Register with ESB > SoaDemoIntegServer, where SoaDemoIntegServer is the name of the connection to the integration server. When registration is complete, JDeveloper displays a confirmation dialog.

You can view the services that you created in the FulfillmentESB project in the ESB Console. To do this, enter the URL of the ESB Console in a browser:

http://host:port/esb/esb/EsbConsole.html

describes the name of the machine running Oracle Application Server, and port specifies the HTTP port at which Oracle HTTP Server or OC4J is listening.

Log in as the oc4jadmin user.

The services are grouped under Fulfillment, which is the name of the system that you created in Section 4.3, “Create a System Called "Fulfillment".”
Figure 4–63  ESB Console Showing the Services in the Fulfillment System
This chapter describes how to create the CreditService project. It contains these sections:

- Section 5.1, "About the CreditService Project"
- Section 5.2, "Create a New Project for CreditService"
- Section 5.3, "Copy the WSDL File"
- Section 5.4, "Generate Java from the WSDL"
- Section 5.5, "Display the List of Files in the Structure Window"
- Section 5.6, "Build CreditService"
- Section 5.7, "Write the Code to Perform Credit Card Validation"
- Section 5.8, "Verify Hostname and Port in CreditService.wsdl"
- Section 5.9, "Update the Context-Root"
- Section 5.10, "Rebuild CreditService"
- Section 5.11, "Deploy Credit Service to Oracle Application Server"

5.1 About the CreditService Project

The CreditService project checks whether a customer’s credit card is valid or not. In this SOA Order Booking application, the code simply checks the value of the credit card number. If the credit card number is less than 12345678, then the card is invalid. Otherwise, it is valid.

The CreditService project is developed in a "top-down" fashion: you start with a WSDL and using this WSDL file, you generate Java classes. RapidService is the opposite: it is developed in a "bottom-up" fashion, where you start with Java classes and you develop a WSDL file from the Java classes.

Note that the WSDL file for the CreditService project contains hardcoded values of localhost:8888. You need to modify these values if you meet any of these conditions:

- If you are running JDeveloper and Oracle Application Server on different machines, change the localhost value to the name of the machine running Oracle Application Server.
- If your Oracle Application Server instance listens on a port other than 8888, you need to modify the port number.
5.2 Create a New Project for CreditService

Start by creating a new project for CreditService in JDeveloper:

1. Right-click the SOADEMO application, and select **New Project**.
2. In the New Gallery, in the Categories section, expand **General** and select **Projects**. In the Items section, select **Empty Project**.

![New Gallery for the CreditService Project](image)

Click **OK**.

3. In the Create Project dialog, enter "CreditService" in the **Project Name** field.

![Create Project Dialog for CreditService](image)

Click **OK**.

In the Application Navigator, you should see an empty CreditService project located under the SOADEMO application.

4. Select **File > Save** to save your work.

5.3 Copy the WSDL File

Define the project type and copy the WSDL file from the `soademo_101310_prod.zip` file to the `CreditService` directory.
1. Right-click the CreditService project, and select **New**.

2. In the New Gallery dialog, in the Categories section, expand **Business Tier** and select **Web Services**. In the Items section, select **WSDL Document**.

   **Figure 5–3  New Gallery for Setting Up the WSDL File**

   ![New Gallery](image)

   Click **OK**. This launches the Create WSDL Document dialog.

3. In the Create WSDL Document dialog:
   - **WSDL Name**: enter **CreditService**.
   - **Directory Name**: accept the default (**SOADEMO\CreditService\src**).
   - **Target Namespace**: accept the default because you are going to overwrite the generated WSDL file anyway. In the WSDL file, the target namespace is **http://www.globalcompany.com/ns/credit**.

   **Figure 5–4  Create WSDL Document Dialog**

   ![Create WSDL Document](image)

4. Click **OK**.

5. Close (by selecting File > Close) the **CreditService.wsdl** file that JDeveloper displays in the editor.
5.4 Generate Java from the WSDL

In this step, you generate Java classes for the methods declared in the WSDL file.

1. Right-click the CreditService project, and select New. In the New Gallery, in the Categories section, expand Business Tier and select Web Services. In the Items section, select Java Web Service from WSDL.

**Figure 5–5 New Gallery for Generating Java from WSDL**

Click OK. This launches the Create J2EE 1.4 Java Web Service from WSDL wizard. Click Next in the wizard to begin.

2. In Step 1, Web Service Description, click Browse and select the CreditService.wsdl file from the SOADEMO\CreditService\src directory.

   Mapping File: leave blank.
Figure 5–6 Create J2EE 1.4 Java Web Service from WSDL Wizard: Step 1, Web Service Description

Click Next.

3. In Step 2, Default Mapping Options:
   - Package Name: enter `org.soademo.creditservice`.
   - Root Package for Generated Types: enter `org.soademo.creditservice.types`.
   - Generate Data Binding Classes: select this option.
   - Reuse Existing Type Classes: select this option.
   - Map Headers to Parameters: select this option.

Figure 5–7 Create J2EE 1.4 Java Web Service from WSDL Wizard: Step 2, Default Mapping Options

Click Next.

4. In Step 3, Specify Custom Data Type Serializer, leave blank and click Next.
5. In Step 4, Handler Details, accept the defaults and click Next.

6. In Step 5, State, do not select **Stateful Service**. Click Next.
Figure 5–10  Create J2EE 1.4 Java Web Service from WSDL Wizard: Step 5, State

7. In the Finish screen, click Finish.
   JDeveloper displays CreditService.wsdl in design view in the editor.

5.5 Display the List of Files in the Structure Window

You should see the following files in the Structure window:
1. Select View > Structure to display the Structure window.
2. In the Application Navigator, select SOADEMO > CreditService > Application Sources > org.soadeemo.creditservice > CreditService.
   In the Structure window, you should see the following files:
   ■ CreditService.wsdl
   ■ ValidateCreditCard.java
   ■ CreditCardValidationFaultMessage.java
   ■ CreditCard.java
   ■ CreditService-java-wsdl-mapping.xml
   ■ ValidateCreditCardImpl.java

5.6 Build CreditService

Right-click the CreditService project and select Rebuild.

5.7 Write the Code to Perform Credit Card Validation

Double-click ValidateCreditCardImpl.java (from the Structure window) and edit the verifyCC method so that it looks like the following:

```java
public boolean verifyCC(CreditCard creditCard)
throws CreditCardValidationFaultMessage {
    boolean validOrNot = false;
    if ((creditCard.getCcType().equals("AMEX")) ||
        (creditCard.getCcType().equals("Visa"))) {
```

Create the CreditService Project 5-7
5.8 Verify Hostname and Port in CreditService.wsdl

There are two CreditService.wsdl files:

- SOADEMO\CreditService\src\CreditService.wsdl
- SOADEMO\CreditService\public_html\WEB-INF\wsdl\CreditService.wsdl

You have to verify the hostname and port in both wsdl files.

1. Double-click the first CreditService.wsdl (in the Application Navigator, CreditService.wsdl is located in SOADEMO > CreditService > Application Sources).

2. In the editor, under Services, expand CreditService > ValidateCreditCardServiceSoapHttp > soap:address.

![Figure 5–11  JDeveloper Showing the CreditService.wsdl File in the Editor](image)

3. Right-click soap:address and select Properties. This displays the soap:address Properties dialog.

![Figure 5–12  soap:address Properties Dialog](image)

4. In the location field, edit the hostname and port in the URL as necessary. You may need to scroll all the way to the left to see the beginning of the URL.
The hostname specifies where Oracle Application Server is running, and the port specifies the HTTP port at which Oracle HTTP Server or OC4J is listening.

5. Click **OK** if you made any changes to the location. Click **Cancel** if you did not make any changes.

6. Select File > Save to save any changes you made to the WSDL file.

7. Repeat the same steps for the second wsdl file, located in CreditService > Web Content > WEB-INF\wsdl.

8. Close both files in the editor.

### 5.9 Update the Context-Root

1. Right-click **WebServices.deploy** (located in SOADEMO > CreditService > Resources) and select Properties. This displays the WAR Deployment Profile Properties dialog.

   ![WAR Deployment Profile Properties Dialog, General Section](image)

2. Select **General** on the left side. On the right side, select **Specify J2EE Web Context Root** and enter **CreditService** as its value.

3. Click **OK**.

4. Select File > Save to save your changes.

### 5.10 Rebuild CreditService

Right-click the CreditService project and select **Rebuild**.
5.11 Deploy Credit Service to Oracle Application Server

When you created the Credit Service project, JDeveloper automatically created a 
`WebServices.deploy` deployment profile for you. You can use this deployment profile 
to deploy the Credit Service project to Oracle Application Server.

1. Expand `CreditService > Resources` and right-click `WebServices.deploy`. Select 
`Deploy` to and your application server connection.

2. In the Configure Application dialog, click `OK`.

After deploying CreditService, you can enter the following URL in a browser to see the 
WSDL for CreditService:

http://hostname:port/CreditService/ValidateCreditCardServiceSoapHttp?WSDL

For `hostname`, specify the name of the machine running Oracle Application Server.

For `port`, specify the HTTP port at which Oracle Application Server is listening. This 
is either the Oracle HTTP Server port or the OC4J port.

After deploying CreditService, you should see it in two places in the Application 
Server Control: in the Web Services tab (Figure 5–14) and the Applications tab 
(Figure 5–15) of the OC4J:home page.

Figure 5–14  Web Services Tab of OC4J:home Page Showing the 
ValidateCreditCardServiceSoapHttp Web Service
Figure 5–15  Applications Tab of OC4J: home Page Showing the SOADEMO-CreditService-WS Application

This page shows the J2EE applications and application components (EJB Modules, WAR Modules, Resource Adapter Modules) deployed to this OC4J instance.

<table>
<thead>
<tr>
<th>Select Name</th>
<th>Status</th>
<th>Active Requests</th>
<th>Request Processing Time (seconds)</th>
<th>Active EJB Methods</th>
<th>Defined Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ascontrol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CustomerService</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOADEMO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOADEMO-CreditService-WS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Middleware Services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Creating the RapidService Project

This chapter describes how to create the RapidService project. It contains these sections:

- Section 6.1, "About the RapidService Project"
- Section 6.2, "Create a New Project for RapidService"
- Section 6.3, "Add JSR-181 Library to the RapidService Project"
- Section 6.4, "Create Item.java"
- Section 6.5, "Create Quote.java"
- Section 6.6, "Create RequestQuote.java"
- Section 6.7, "Check Files in the Application Navigator"
- Section 6.8, "Compile the Files"
- Section 6.9, "Publish the Project as a Web Service"
- Section 6.10, "Verify the Hostname and Port in the Generated WSDL File"
- Section 6.11, "Check Files in the Application Navigator"
- Section 6.12, "Set the Context Root to RapidService"
- Section 6.13, "Edit the Deployment Descriptor"
- Section 6.14, "Deploy the RapidService Project"
- Section 6.15, "View the WSDL for RapidService"

6.1 About the RapidService Project

The RapidService project represents a supplier that provides price quotes for customer orders. The other supplier which it competes with is SelectManufacturer. The supplier that provides the lower quote for a customer’s order gets to fulfill the order.

In the case of the RapidService and CreditService projects, the descriptors are generated at design time. The RapidService project is developed in a "bottom-up" fashion: you start with Java classes and you develop a WSDL file from the Java classes. CreditService is the opposite: it is developed in a "top-down" fashion, where you start with a WSDL and using this WSDL file, you generate Java classes.

6.2 Create a New Project for RapidService

Start by creating a new project for RapidService in JDeveloper:
1. Right-click the SOADEMO application, and select New Project.

2. In the New Gallery, in the Categories section, expand General and select Projects. In the Items section, select Empty Project.

Figure 6–1 New Gallery for RapidService Project

Click OK.

3. In the Create Project dialog, enter RapidService in the Project Name field.

Figure 6–2 Create Project Dialog for RapidService Project

Click OK.

In the Application Navigator, you should see an empty RapidService project located under the SOADEMO application.

4. Select File > Save to save your work.

6.3 Add JSR-181 Library to the RapidService Project

Before you can compile and deploy the RapidService project, you need to add the JSR-181 library to it. RequestQuote.java, which you will create in Section 6.6, "Create RequestQuote.java", contains JSR-181 annotations.

1. Right-click the RapidService project and select Project Properties. In the Project Properties dialog, select Libraries on the left side.
Add JSR-181 Library to the RapidService Project

Creating the RapidService Project

2. Click **Add Library**.

3. In the Add Library dialog, select **JSR-181 Web Services**, located under **Extension**.

4. Click **OK** in the Add Library dialog. In the Project Properties dialog, you should see the JSR-181 library added to the list.
5. Click OK in the Project Properties dialog.

6. Select the RapidService project in the Application Navigator and select File > Save to save your work.

6.4 Create Item.java

1. Right-click the RapidService project, and select New. In the New Gallery, in the Categories section, expand General and select Simple Files. In the Items section, select Java Class.
Creating the RapidService Project 6-5

Create Item.java

Figure 6–6  New Gallery for Creating a Simple Java Class

Click OK. This displays the Create Java Class dialog.

2. In the Create Java Class dialog, enter these values:

   ■ Name: enter Item.
   ■ Package: enter org.soademo.rapidservice.types.
   ■ Extends: enter java.lang.Object.
   ■ Public: select this option.
   ■ Generate Default Constructor: select this option.
   ■ Generate Main Method: do not select.

Click OK. JDeveloper displays the Item.java file in the editor.

3. Edit the file so that it contains the following lines:

   package org.soademo.rapidservice.types;

   public class Item {
      public Item() {
      }

      String itemId;
      long Quantity;

      public void setItemId(String itemId) {
         this.itemId = itemId;
      }

      public String getitemId() {
         return itemId;
      }

      public void setQuantity(long quantity) {
         this.Quantity = quantity;
      }
   }
public long getQuantity() {
    return quantity;
}
}

4. Select File > Save to save the file.

6.5 Create Quote.java

1. Right-click the RapidService project, and select New. In the New Gallery, in the Categories section, expand General and select Simple Files. In the Items section, select Java Class.

Figure 6–7 New Gallery Dialog for Creating a Simple Java Class

Click OK. This displays the Create Java Class dialog.

2. In the Create Java Class dialog, enter these values:
   - Name: enter Quote.
   - Package: enter org.soadeo.rapidservice.types.
   - Extends: enter java.lang.Object.
   - Public: select this option.
   - Generate Default Constructor: select this option.
   - Generate Main Method: do not select.
Figure 6–8  Create Java Class Dialog for Quote.java

Click OK. JDeveloper displays the Quote.java file in the editor.

3. Edit the file so that it contains the following lines:

```java
package org.soademo.rapidservice.types;

public class Quote implements java.io.Serializable {
    protected java.lang.String supplierName;
    protected java.lang.String supplierPrice;

    public Quote() {} 

    public java.lang.String getSupplierName() {
        return supplierName;
    }

    public void setSupplierName(java.lang.String supplierName) {
        this.supplierName = supplierName;
    }

    public java.lang.String getSupplierPrice() {
        return supplierPrice;
    }

    public void setSupplierPrice(java.lang.String supplierPrice) {
        this.supplierPrice = supplierPrice;
    }
}
```

4. Select File > Save to save the file.

6.6 Create RequestQuote.java

1. Right-click the RapidService project, and select New. In the New Gallery, in the Categories section, expand General and select Simple Files. In the Items section, select Java Class.
Create RequestQuote.java

6-8 Oracle SOA Suite Tutorial

Figure 6–9  New Gallery for Creating a Simple Java Class

Click OK. This displays the Create Java Class dialog.

2. In the Create Java Class dialog, enter these values:
   - **Name**: enter RequestQuote.
   - **Package**: enter org.soadeom.rapidservice. Note that this is different from the previous Java classes you created.
   - **Extends**: enter java.lang.Object.
   - **Public**: select this option.
   - **Generate Default Constructor**: select this option.
   - **Generate Main Method**: do not select.

Figure 6–10  Create Java Class Dialog for RequestQuote.java

Click OK. JDeveloper displays the file in the editor.

3. Edit the file so that it contains the following lines:
package org.soademo.rapidservice;
import javax.jws.WebMethod;
import javax.jws.WebService;
import org.soademo.rapidservice.types.Item;
import org.soademo.rapidservice.types.Quote;

@ServiceName="RapidService",
targetNamespace="http://www.globalcompany.com/ns/rapidservice")
public class RequestQuote {
    public RequestQuote() {
    }

    @WebMethod(operationName="OrderQuote")
    public Quote processRequestQuote(String productName, String item
    String partnum, String quantity, String price) {
        Quote priceQuote = new Quote();
        priceQuote.setSupplierName("RapidDistributors");
        priceQuote.setSupplierPrice("5000");
        return priceQuote;
    }

    @WebMethod(operationName="POItemsQuote")
    public Quote processQuote(Item[] items){
        Long totalPrice = new Long(0);
        for ( int i=0; i<items.length; i++ ) {
            Item localItem = items[i];
            totalPrice += localItem.getQuantity()*110;
        }
        Quote priceQuote = new Quote();
        priceQuote.setSupplierName("RapidDistributors");
        priceQuote.setSupplierPrice(totalPrice.toString());
        return priceQuote;
    }
}

4. Select File > Save to save the file.

6.7 Check Files in the Application Navigator

In the Application Navigator, you should see the following files for the RapidService project:
6.8 Compile the Files

Right-click the RapidService project and select Rebuild. Check the log window for any errors.

6.9 Publish the Project as a Web Service

1. Right-click the RapidService project, and select New. In the New Gallery, in the Categories section, expand Business Tier and select Web Services. In the Items section, select Java Web Service.

   Click OK. This displays the Create J2EE Web Service Version dialog.

2. In the Select J2EE Web Service Version dialog, select J2EE 1.4 (JAX-RPC) Web Service and click OK.
Figure 6–13 Select J2EE Web Service Version Dialog

This launches the Create Java J2EE 1.4 Web Service wizard. Click Next to start.

3. In Step 1, Class:
   - **Web Service Name**: enter *RapidService*.
   - **Component to Publish**: select `org.soademo.rapidservice.RequestQuote` from the dropdown.
   - **Autogenerate Service Endpoint Interface**: select this option (should be automatically selected for you).
   - **Service Endpoint Interface**: select `org.soademo.rapidservice.RequestQuotePortType` (this should be entered for you automatically).
   - **SOAP 1.1 Binding**: select this option.
   - **SOAP 1.2 Binding**: do not select.
   - **WSIF Binding**: do not select.
   - **Generate annotations into class**: select this option.

Figure 6–14 Create Java J2EE 1.4 Web Service Wizard: Step 1, Class

Click Next.

4. In Step 2, Message Format:
   - **SOAP Message Format**: select Document/Wrapped.
   - **Enable REST Access to SOAP Ports**: do not select.
- **Generate Schema with Qualified Elements**: select this option.
- **Use DIME Encoding**: do not select.

Figure 6–15  Create Java J2EE 1.4 Web Service Wizard: Step 2, Message Format

Click **Next**.

5. In Step 3, Specify Custom Data Type Serializers, click **Next**.

Figure 6–16  Create Java J2EE 1.4 Web Service Wizard: Step 3, Specify Custom Data Type Serializers

6. In Step 4, Mapping, click **Next**.
7. In Step 5, Methods, select `processQuote` and `processRequestQuote`.

8. In Step 6, Handler Details, click **Next**.
9. In Step 7, State, click **Next**.

10. In Step 8, Additional Classes, click **Next**.
Verify the Hostname and Port in the Generated WSDL File

6.10 Verify the Hostname and Port in the Generated WSDL File

Verify that the hostname and port in the RapidService.wsdl file are correct.

1. If RapidService.wsdl is not displayed in the editor, double-click it in the Application Navigator. The file is located under RapidService > Web Content > WEB-INF\wsdl.

2. Under Services, expand RequestQuoteSoapHttpPort.

3. Right-click soap:address and select Properties. This displays the soap:address Properties dialog.

4. In the soap:address Properties dialog, verify that the hostname and port are correct. The hostname specifies the name of the machine running Oracle Application Server, and the port specifies the HTTP port at which Oracle HTTP Server or OC4J is listening.

Figure 6–22 RapidService.wsdl

11. In the Finish page, click Finish. JDeveloper displays the RapidService.wsdl file in the editor. RapidService.wsdl is one of the files created by the wizard.

Continue with the next section to see the files JDeveloper created.
5. Save your changes, if you made any.

6.11 Check Files in the Application Navigator

After the wizard has completed, you should see the following files in the Application Navigator and Structure window.

1. Select View > Structure to display the Structure window.

2. In the Application Navigator, select SOADEMO > RapidService > Application Sources > org.soadeemo.rapidservice > RapidService.

   In the Structure window, you should see the following files:
   - RequestQuote.java
   - RequestQuotePortType.java
   - RapidService.wsdl
   - RapidService-java-wsdl-mapping.xml

3. Expand the SOADEMO > RapidService > Web Content node in the Application Navigator to see the other files that JDeveloper created. See Figure 6–24.

6.12 Set the Context Root to RapidService

1. Double-click the RapidService project to display the Project Properties dialog.
2. On the left side, select J2EE Application.
3. Set the value of J2EE Web Application Name and J2EE Web Context Root to RapidService.

Figure 6–25  Project Properties Dialog for RapidService

![Project Properties Dialog for RapidService](image)

4. Click OK.
5. Select File > Save to save your work.

6.13 Edit the Deployment Descriptor

1. Double-click the deployment descriptor (Resources > WebServices.deploy) to display the WAR Deployment Profile Properties dialog.
2. Select General on the left side.
3. Set Enterprise Application Name to RapidService.
4. Click OK.

5. Select File > Save to save your work.

6.14 Deploy the RapidService Project

Right-click Resources > WebServices.deploy and select Deploy To > SoademoApplicationServer, where SoademoApplicationServer specifies the connection to your application server.

In the Configure Application dialog, click OK.

Figure 6–26  WAR Deployment Profile Properties Dialog, General Section

Figure 6–27  Configure Application Dialog
6.15 View the WSDL for RapidService

You can access the WSDL for RapidService by entering the following URL in a browser:

http://hostname:port/RapidService/RequestQuoteSoapHttpPort?WSDL

For hostname, enter the name of the machine running Oracle Application Server.

For port, enter the HTTP at which Oracle HTTP Server or OC4J is listening. The default port is 8888.

In the Application Server Control, RapidService appears in two places: in the Web Services tab (Figure 6–28) and the Applications tab (Figure 6–29) of the OC4J:home page.

Figure 6–28 Web Services Tab of OC4J:home Page Showing the RapidService Web Service
Figure 6-29  Applications Tab of OC4J:home Page Showing the RapidService Application

This page shows the J2EE applications and application components (JAR Modules, WAR Modules, Resource Adapter Modules) deployed to this OC4J instance.
This chapter describes how to create the SelectManufacturer project. It contains these sections:

- Section 7.1, "About the SelectManufacturer Project"
- Section 7.2, "Create a New BPEL Project for SelectManufacturer"
- Section 7.3, "Create "SelectService" Partner Link"
- Section 7.4, "Define Variables for the SelectManufacturer Project"
- Section 7.5, "Receive Order Data from the Client through a Receive Activity"
- Section 7.6, "Assign Values to be Returned"
- Section 7.7, "Return Values to the Client Using an Invoke Activity"
- Section 7.8, "Deploy the BPEL Process"

7.1 About the SelectManufacturer Project

The SelectManufacturer project represents a supplier called Select Manufacturer. When a customer places an order, the order is sent to two suppliers, Select Manufacturer and Rapid Distributors, to get quote requests from both suppliers. The supplier that responds with the lower quote gets to fulfill the customer’s order.

The SelectManufacturer project is an asynchronous BPEL process. This means that it contains a receive activity to initiate the BPEL process flow and an invoke activity to call back the client asynchronously with the results (that is, the quote) at the end of the flow.

7.2 Create a New BPEL Project for SelectManufacturer

1. Right-click the SOADEMO application in the Application Navigator and select New Project to display the New Gallery.

Create a New BPEL Project for SelectManufacturer

Figure 7–1  New Gallery for SelectManufacturer Project

Click OK. This displays the BPEL Project Creation wizard.

3. On the Project Settings page of the BPEL Project Creation wizard:
   - Name: enter SelectManufacturer.
   - Use Default Project Settings: select this option.
   - Template: select Empty BPEL Process.

Figure 7–2  BPEL Project Creation Wizard - Project Settings Page

Click Finish.
Create "SelectService" Partner Link

1. Copy the SelectManufacturer\bpel\SelectManufacturer.wsdl file from the soademo_101310_prod.zip file to the SOADEMO\SelectManufacturer\bpel directory.

2. Select View > Component Palette to display the Component Palette. You need the Component Palette to drag and drop BPEL activities and services onto the SelectManufacturer BPEL flow you are about to create.

3. Select Services from the dropdown in the Component Palette.

4. Drag the Partner Link icon from the Component Palette and drop it on a Services swimlane. You can drop it on either the left side or the right side.

5. In the Partner Link dialog:
   - Name: enter SelectService.
   - WSDL: click the Service Explorer icon (the second one from the left) to display the Service Explorer dialog. In the Service Explorer dialog, expand Project WSDL Files and select SelectManufacturer.wsdl.

   Click OK in the Service Explorer dialog.

   - Partner Link Type: select SelectService_PL (automatically filled in for you).
   - Partner Role: select SelectServiceRequester.
   - My Role: select SelectServiceProvider.
Figure 7–4 Create Partner Link Dialog for SelectService

6. Click OK in the Create Partner Link dialog.
At this point, the SelectManufacturer.bpel page should look like this:

Figure 7–5 SelectManufacturer.bpel Page in JDeveloper

7.4 Define Variables for the SelectManufacturer Project

In this step, you create two variables called inputVariable and outputVariable for the SelectManufacturer project. Variables defined at the project level can be accessed by all activities in the project.

1. Double-click the SelectManufacturer scope to display the Process dialog. You can double-click the "SelectManufacturer" text that is displayed sideways.

2. In the Process dialog, click the Variables tab.
3. Create a variable called `inputVariable`.
   a. Click `Create` to display the Create Variable dialog.
   b. In the Create Variable dialog, set these values:
      - Name: enter `inputVariable`.
      - Type: select `Message Type`, and click the flashlight icon. This displays the Type Chooser dialog.

In the Type Chooser, select `Type Explorer > Message Types > Partner Links > SelectService > SelectManufacture.wsdl > Message Types > RequestQuote_processRequestQuote`.

Click `OK` in the Type Chooser. The Create Variable dialog now looks like this:
Define Variables for the SelectManufacturer Project

**Figure 7–8  Create Variable Dialog for inputVariable**

![Create Variable Dialog for inputVariable](image1.png)

- Click OK in the Create Variable dialog. This returns you to the Process dialog. You should see the inputVariable in the dialog.

4. Create a second variable called outputVariable.
   - Click Create to display the Create Variable dialog.
   - In the Create Variable dialog, set these values:
     - **Name**: enter outputVariable.
     - **Type**: select Message Type, and click the flashlight icon. This displays the Type Chooser dialog. In the Type Chooser, select Type Explorer > Message Types > Partner Links > SelectService > SelectManufacture.wsdl > Message Types > RequestQuote_processRequestQuoteResponse.

**Figure 7–9  Type Chooser for outputVariable**

![Type Chooser for outputVariable](image2.png)
Click **OK** in the Type Chooser. The Create Variable dialog now looks like this:

**Figure 7–10  Create Variable Dialog for outputVariable**

![Create Variable Dialog](image)

**c.** Click **OK** in the Create Variable dialog.

5. In the Process dialog, you should see the two variables:

**Figure 7–11  Process Dialog Showing inputVariable and outputVariable**

![Process Dialog](image)

6. Click **OK** in the Process dialog.

7. Select File > Save to save your work.

### 7.5 Receive Order Data from the Client through a Receive Activity

Create a receive activity as the starting point for this BPEL process. The receive activity receives requests from clients (in this case, the client is the SOAOrderBooking project, which is another BPEL process), and the requests can contain parameters.

1. Select **Process Activities** from the dropdown in the Component Palette.
2. Drag the Receive activity icon from the Component Palette and drop it in the "Drop Activity Here" box on the page.

3. Drag an arrow from the newly created receive activity and drop it on the SelectService partner link. As you drag, you should see a line connecting the receive activity with partner link. When you release, JDeveloper displays the Receive dialog.

4. In the Receive dialog:
   - **Name**: enter Receive_1.
   - **Partner Link**: should be set to SelectService. This should be filled in automatically for you.
   - **Operation**: should be set to processRequestQuote. This should be filled in automatically for you.
   - **Variable**: click the Browse Variables icon (the icon on the right) to display the Variable Chooser dialog. Select inputVariable.

   ![Variable Chooser Dialog](image)

   Click **OK** in the Variable Chooser.

   - **Create Instance**: select this option.

The Receive dialog now looks like this:
5. Click OK in the Receive dialog.

The SelectManufacturer.bpel page now contains a link from the SelectService partner link to the receive activity.

6. Select File > Save to save your work.

### 7.6 Assign Values to be Returned

In this assign activity, you set the values that are returned by Select Manufacturer when it is asked for a quote. To simplify things in this SOA Order Booking application, the SelectManufacturer BPEL process returns two values:

- its name ("SelectDistributors")
- the quote for the order. Select Manufacturer simply returns 120 multiplied by the number of items in the order to get the quote.

These values are set in the `outputVariable` variable, which will be returned to the client.

1. Select **Process Activities** from the dropdown in the Component Palette.
2. Drag the Assign activity icon from the Component Palette and drop it after the Receive_1 activity.

3. Double-click the new Assign instance to display the Assign dialog.

4. Click the Copy Operation tab in the Assign dialog.

5. In the Copy Operation tab, create the first copy operation. This copy operation sets the quote value to return.
   a. Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.
   b. In the Create Copy Operation dialog:
      - On the From side, select Expression from the Type dropdown. Enter the following into the Expression box:
        \[
        120 \times \text{bpws:getVariableData('inputVariable','parameters', '/client:processRequestQuoteElement/client:param0/client:quantity')} \]
        This formula multiplies the number of items in the order by 120 to get the quote price.
      - On the To side, select Variable from the Type dropdown. Select Variables > Process > Variables > outputVariable > parameters > client:processRequestQuoteResponseElement > client:return > clientsupplierPrice. Note that the prefix ("client") may be different on your machine.

The Create Copy Variable dialog should look like this:

Figure 7–15 Create Copy Variable Dialog for Returning the Supplier Price

   c. Click OK in the Create Copy Operation dialog.

6. Create the second copy operation. This copy operation sets the name to return.
a. Select **Copy Operation** from the **Create** dropdown. This displays the Create Copy Operation dialog.

b. In the Create Copy Operation dialog:
   - On the **From** side, select **Expression** from the **Type** dropdown. Enter the following into the Expression box:
     ```java
     string('SelectDistributors')
     ```
   - On the **To** side, select **Variable** from the **Type** dropdown. Select **Variables > Process > Variables > outputVariable > parameters > client:processRequestQuoteResponseElement > client:return > client:supplierName.**

The Create Copy Variable dialog should look like this:

![Create Copy Variable Dialog](image)

Figure 7–16  **Create Copy Variable Dialog for Returning the Supplier Name**

c. Click **OK** in the Create Copy Operation dialog.

The Assign dialog should look like this:
7. Click **OK** in the Assign dialog.

8. Select File > Save to save your work.

The SelectManufacturer.bpmn page should now look like this:

### Figure 7–18 SelectManufacturer.bpmn Page Showing the Assign Activity

---

### 7.7 Return Values to the Client Using an Invoke Activity

The Invoke activity returns the values in the `outputVariable` to the client.

1. Select **Process Activities** from the dropdown in the Component Palette.

2. Drag the Invoke activity icon from the Component Palette and drop it after the `Assign_1` activity.

3. Drag an arrow from the newly created invoke activity and drop it on the `SelectService` partner link. This displays the Invoke dialog.

4. In the Invoke dialog:
• Name: enter Invoke_1.
• Partner Link: should be set to SelectService.
• Operation: select processRequestQuoteResponse. This is set for you automatically.
• Input Variable: click the Browser Variables icon (the one on the right) to display the Variable Chooser. In the Variable Chooser, select Variables > Process > Variables > outputVariable.

![Variable Chooser for the Input Variable](image)

Click OK in the Variable Chooser.
The Invoke dialog should now look like this:

![Invoke Dialog](image)

5. Click OK in the Invoke dialog.
6. Select File > Save to save your work.
The SelectManufacturer.bpmel page now looks like this:
7.8 Deploy the BPEL Process

1. Double-click `build.properties`, located under `SelectManufacturer > Resources` in the Application Navigator. This file defines values that are used by `build.xml` when you deploy the BPEL project.

2. Uncomment (by removing the `#` character) these lines in the `build.properties` file:

   ```
   platform=ias_10g
   domain=default
   rev=1.0
   admin.user=oc4jadmin
   admin.password=welcome99
   http.hostname=myAppServerMachine.mydomain.com
   http.port=8888
   j2ee.hostname=myAppServerMachine.mydomain.com
   rmi.port=23793
   opmn.requestport=6005
   oc4jinstancename=home
   ```

3. Edit the values as necessary. The values in bold italics are the typical values you need to modify.

   To determine the value for `rmi.port`, run:

   ```
   ORACLE_HOME\opmn\bin\opmnctl status -l
   ```

   `ORACLE_HOME` specifies the Oracle home for Oracle Application Server.

   To determine the value for `opmn.requestport`, see step 6 on page 2-11.

4. Select File > Save to save your changes to `build.properties`.
5. Right-click build.xml (located under SelectManufacturer > Resources) and select Run Ant.

6. In the Run Ant dialog, click the Properties tab.

7. In the Property Files section, click Add. In the Add Ant Property File dialog, select the build.properties file in the SelectManufacturer directory and click Open.

The Run Ant dialog should look like this:

Figure 7–22  Run Ant Dialog, Properties Tab, With build.properties File Loaded

8. Click OK. JDeveloper runs Ant to compile and deploy the project. If you get errors, see the next section, Section 7.8.1, "Deploying Using Ant from the Developer Prompt".

7.8.1 Deploying Using Ant from the Developer Prompt

If you get errors, check that the values you entered in the build.properties file are correct.

If the values are correct, but you still get errors, you can run ant from the Developer Prompt to deploy the project:

1. On the machine running Oracle Application Server, select Start > Programs > Oracle - instanceName > Oracle BPEL Process Manager > Developer Prompt. This displays a shell window configured for Oracle BPEL Process Manager.

   Note that you must run ant from the Developer Prompt shell window to deploy the SelectManufacturer project. Running ant from a regular operating system shell for deploying the project is not supported.

2. In the Developer Prompt window, change directory to the SOADEMO\SelectManufacturer directory, where SOADEMO refers to the directory where you created the SOA Order Booking application.

   > cd SOADEMO
   > cd SelectManufacturer
If you are running JDeveloper and Oracle Application Server on separate machines, you can copy the SelectManufacturer directory from the JDeveloper machine to the Oracle Application Server machine. You can place it anywhere on the Oracle Application Server machine. In the Developer Prompt, you can just navigate to that directory.

3. Run ant.
   
   > ant

### 7.8.2 Viewing SelectManufacturer in the Oracle BPEL Control

After deployment, SelectManufacturer appears in the Oracle BPEL Control. Enter the following URL in a browser to bring up the Oracle BPEL Control:

http://hostname:port/BPELConsole

Log in as the oc4jadmin user.

SelectManufacturer appears in the Dashboard tab of the Oracle BPEL Control:

*Figure 7–23  Oracle BPEL Control Showing SelectManufacturer*
Creating the SOAOrderBooking Project

This chapter describes how to create the SOAOrderBooking project, which is a BPEL project. It contains these sections:

- Section 8.1, "About the SOAOrderBooking Project"
- Section 8.2, "Create a New BPEL Project for SOAOrderBooking"
- Section 8.3, "Copy Files"
- Section 8.4, "Define Variables for the SOAOrderBooking Project"
- Section 8.5, "Create "client" Partner Link"
- Section 8.6, "Receive Input from the Client (Receive Activity)"
- Section 8.7, "Insert Order Information in the Database ("InsertOrderIntoDB" Scope)"
- Section 8.8, "Retrieve Information About the Customer ("CustomerService" Scope)"
- Section 8.9, "Verify the Customer’s Credit Card ("CreditService" Scope)"
- Section 8.10, "Set up Oracle Business Rules"
- Section 8.11, "Determine If an Order Requires Manual Approval ("RequiresManualApproval" Decide Activity)"
- Section 8.12, "Set Up a Form to Process Orders That Require Manual Approval ("requiresApproval" Switch)"
- Section 8.13, "Handle the Manager’s Response ("requiresApproval" Switch)"
- Section 8.14, "Choose a Supplier ("SelectSupplier" Scope)"
- Section 8.15, "Determine the Shipping Method ("PostFulfillmentReq" Scope)"
- Section 8.16, "Set the Order Status to "Completed" ("SetFinalOrderStatus" Scope)"
- Section 8.17, "Send an Email Notification to the Customer ("NotifyCustomer" Scope)"
- Section 8.18, "Call Back the Client ("callbackClient" Invoke Activity)"
- Section 8.19, "Add a Catch Branch to the Project"
- Section 8.20, "Deploy the Project"

8.1 About the SOAOrderBooking Project

The SOAOrderBooking project, which is a BPEL project, represents the main flow in the SOA Order Booking application. It sends the order information to the appropriate
services at the appropriate times. For example, it contacts CreditService to check the
customer’s credit card, and if the credit card is acceptable, it contacts the suppliers
(Select Manufacturer and Rapid Distributors) to get price quotes for the order.

The SOAOrderBooking project is a large project. This chapter begins by giving an
overview of the major blocks in the project, then it goes into detail for each block.

8.1.1 Blocks in the SOAOrderBooking Project

Table 8–1 lists the major blocks in the SOAOrderBooking project:

<table>
<thead>
<tr>
<th>Block</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>receiveInput</td>
<td>Receive activity</td>
<td>This activity receives incoming requests, which include the order information.</td>
</tr>
<tr>
<td>InsertOrderIntoDB</td>
<td>Scope</td>
<td>This scope inserts the order information into the database.</td>
</tr>
<tr>
<td>CustomerService</td>
<td>Scope</td>
<td>This scope retrieves the customer’s information from the database.</td>
</tr>
<tr>
<td>CreditService</td>
<td>Scope</td>
<td>This scope verifies that the customer has acceptable credit.</td>
</tr>
<tr>
<td>RequiresManualApproval</td>
<td>Decision</td>
<td>This scope is a decide activity: it consults the rules set up in a rules repository to determine whether or not an order needs to be approved by a manager. The manager can approve or reject the order.</td>
</tr>
<tr>
<td>requiresApproval</td>
<td>Switch</td>
<td>This switch checks whether the manager approved or rejected the order, and performs the appropriate activities. For approved orders, it just continues with the rest of the activities in this SOAOrderBooking project. For rejected orders, it throws a fault and does not continue.</td>
</tr>
<tr>
<td>SelectSupplier</td>
<td>Scope</td>
<td>This scope selects which supplier (Select Manufacturer or Rapid Distributors) gets to fulfill the order.</td>
</tr>
<tr>
<td>PostFulfillmentReq</td>
<td>Scope</td>
<td>This scope calls the FulfillmentESB project, which determines the shipping method for the order.</td>
</tr>
<tr>
<td>SetFinalOrderStatus</td>
<td>Scope</td>
<td>This scope writes the final order status in the database.</td>
</tr>
<tr>
<td>NotifyCustomer</td>
<td>Scope</td>
<td>This scope is an email service. It sends out email to the customers informing them that their orders have been processed successfully.</td>
</tr>
<tr>
<td>callbackClient</td>
<td>Invoke activity</td>
<td>This invoke activity notifies the client that it is done.</td>
</tr>
</tbody>
</table>

8.1.2 Blocks Shown in Minimized View

Figure 8–1 shows the SOAOrderBooking.bpel page in JDeveloper with the blocks minimized. Later sections in this chapter expand the blocks to show their contents and describe how to create the blocks.
Figure 8–1 Minimized View of the Blocks in SOAOrderBooking Project
8.2 Create a New BPEL Project for SOAOrderBooking

1. Right-click the SOADEMO application in the Application Navigator and select New Project to display the New Gallery.


3. On the Project Settings page of the BPEL Project Creation wizard:
   - **Name**: enter SOAOrderBooking.
   - **Namespace**: enter http://www.globalcompany.com/ns/OrderBooking.
   - **Use Default Project Settings**: select this option.
   - **Template**: select Empty BPEL Process.
8.3 Copy Files

Copy the following files from the soademo_101310_prod.zip file to the SOAOrderBooking\bpel directory:

- SOAOrderBooking.wsdl
- OrderBookingPO.xsd
- OrderBookingRules.xsd

In the soademo_101310_prod.zip file, these files are located in the SOAOrderBooking\bpel directory.

8.4 Define Variables for the SOAOrderBooking Project

In this step, you create two variables called inputVariable and outputVariable for the SOAOrderBooking project. Variables defined at the project level can be accessed by all activities in the project.

1. Double-click the SOAOrderBooking scope. You can double-click the "SOAOrderBooking" text that is displayed sideways.

   This displays the Process dialog.

2. In the Process dialog, click the Variables tab.

   Click Finish.
3. **Create a variable called inputVariable.**
   
a. Click **Create** to display the Create Variable dialog.

b. In the Create Variable dialog, set these values:
   
   - **Name**: enter `inputVariable`.
   - **Type**: select **Message Type**, and click the flashlight icon. This displays the Type Chooser dialog.

   In the Type Chooser, select **Type Explorer > Message Types > Project WSDL Files > SOAOrderBooking.wsdl > Message Types > SOAOrderBookingRequestMessage**.

Click **OK** in the Type Chooser. The Create Variable dialog now looks like this:
Define Variables for the SOAOrderBooking Project

Creating the SOAOrderBooking Project

8-7

Figure 8–6 Create Variable Dialog for inputVariable

<table>
<thead>
<tr>
<th>Create Variable Dialog for inputVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name: inputVariable</td>
</tr>
<tr>
<td>Type:</td>
</tr>
<tr>
<td>- Simple Type</td>
</tr>
<tr>
<td>- Message Type: &quot;order&quot;</td>
</tr>
</tbody>
</table>

4. Create a second variable called outputVariable.

   a. Click Create in the Process dialog to display the Create Variable dialog.

   b. In the Create Variable dialog, set these values:
      - Name: enter outputVariable.
      - Type: select Message Type, and click the flashlight icon. This displays the Type Chooser dialog.

   In the Type Chooser, select Type Explorer > Message Types > Project WSDL Files > SOAOrderBooking.wsdl > Message Types > SOAOrderBookingResponseMessage.

Figure 8–7 Type Chooser for outputVariable

<table>
<thead>
<tr>
<th>Type Chooser for outputVariable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
</tr>
<tr>
<td>- inputVariable</td>
</tr>
<tr>
<td>- outputVariable</td>
</tr>
<tr>
<td>- SOAOrderBookingResponseMessage</td>
</tr>
</tbody>
</table>

   c. Click OK in the Create Variable dialog. This returns you to the Process dialog. You should see the inputVariable in the dialog.
Click OK in the Type Chooser. The Create Variable dialog now looks like this:

Figure 8–8 Create Variable Dialog for outputVariable

![Create Variable Dialog for outputVariable](image)

c. Click OK in the Create Variable dialog.

5. In the Process dialog, you should see the two variables:

Figure 8–9 Process Dialog Showing inputVariable and outputVariable

![Process Dialog Showing inputVariable and outputVariable](image)

6. Click OK in the Process dialog.

7. Select File > Save to save your work.

8.5 Create "client" Partner Link

This client partner link represents the client, which passes data to the receiveInput activity and gets invoked at the end of the flow to receive the return value.

1. If the Component Palette is not showing, select View > Component Palette. Select Services from the dropdown in the Component Palette.
2. Drag the Partner Link icon from the Component Palette and drop it into a Services swimlane. This displays the Create Partner Link dialog.

3. In the Create Partner Link dialog, set these values:
   - Name: enter client.
   - WSDL File: click the Service Explorer icon (second icon from the left) to display the Service Explorer dialog. In the Service Explorer dialog, expand Project WSDL Files and select SOAOrderBooking.wsdl.

   **Figure 8-10 Service Explorer for client Partner Link**

   ![Service Explorer for client Partner Link]

   Click OK in the Service Explorer.

   - Partner Link Type: select SOAOrderBooking (automatically filled in for you).
   - Partner Role: select SOAOrderBookingRequester.
   - My Role: select SOAOrderBookingProvider.
   - Process: leave it blank.

   Before clicking OK, check that Name is still set to client. JDeveloper may have changed it to SOAOrderBooking when it was filling in the other fields for you.
4. Click OK in the Create Partner Link dialog.

5. Select File > Save to save your work.

8.6 Receive Input from the Client (Receive Activity)

Create a receive activity to receive the input data from the client. In the receive activity, you also define a sensor to send data to a JMS topic.

- Section 8.6.1, "Create the Receive Activity"
- Section 8.6.2, "Create a Sensor for the Receive Activity"

8.6.1 Create the Receive Activity

Create the receive activity:

1. Select Process Activities from the dropdown in the Component Palette.

2. Drag the Receive icon from the Component Palette and drop it on the page where it says "Drop Activity Here".

   The page should look like this at this point:

   Figure 8–12 SOAOrderBooking Page with Receive Activity

3. Do one of the following to display the Receive dialog:
Drag one of the arrows on the side of the Receive_1 activity and drop it on the "client" partner link. This associates the receive activity with the partner link.

Double-click the new Receive_1 activity.

4. In the Receive dialog, enter these values:
   - **Name**: enter `receiveInput`.
   - **Partner Link**: should be set to `client`. This is filled in for you if you dragged the arrow from the Receive_1 activity and dropped it on the "client" partner link. If not, click the flashlight to display the Partner Link Chooser dialog and select `client`.

![Partner Link Chooser Dialog for "receiveInput" Receive Activity](image)

Click **OK** in the Partner Link Chooser dialog.

- **Operation**: select `initiate`. This should be filled in automatically for you.
- **Variable**: click the Browse Variables icon (the icon on the right). In the Variable Chooser dialog, select `inputVariable`. 
Click OK in the Variable Chooser.

- **Create Instance**: select this option.

The Receive dialog now looks like this:

5. Click OK in the Receive dialog.
6. Select File > Save to save your work.

### 8.6.2 Create a Sensor for the Receive Activity

In this receive activity create an activity sensor for the receive activity. After this activity runs, this sensor writes the order information to a JMS topic. There is no consumer for this JMS topic; the purpose of this sensor is to show how to send order information to another destination.
1. Double-click the "receiveInput" activity to display the Receive dialog.
2. Click the Sensors tab in the Receive dialog.
3. Click Create to create a new sensor. This displays the Create Activity Sensor dialog.
4. In the Create Activity Sensor dialog, set the Name to InstanceStart.
5. Set the Evaluation Time to Completion. This specifies when the sensor fires. Completion signifies that the sensor fires after this activity has run.
6. In the Activity Variable Sensors section, click Create to display the Create Activity Variable Sensor dialog.

**Figure 8–16  Create Activity Variable Sensor Dialog**

![Create Activity Variable Sensor Dialog](image)

7. In the Create Activity Variable Sensor dialog, click the pencil icon for Variable XPath. This displays the Variable XPath Builder dialog.

**Figure 8–17  Variable XPath Builder Dialog**

![Variable XPath Builder Dialog](image)

8. Select Variables > Process > Variables > inputVariable.
9. Click OK in the Variable XPath Builder. The Create Activity Variable Sensor dialog should be filled in with these values for you (see Figure 8–16):

   - Variable XPath: $inputVariable
   - Output Namespace: http://www.globalcompany.com/ns/OrderBooking
Output Datatype: SOAOrderBookingRequestMessage

10. Click OK in the Create Activity Variable Sensor dialog. This takes you back to the Create Activity Sensor dialog (Figure 8–21).

11. In the Create Activity Sensor dialog, click the Add icon in the Sensor Actions section. This displays the Sensor Action Chooser dialog.

12. In the Sensor Action Chooser dialog, select Sensor Actions and select Sensor Action from the wand icon. This displays the Create Sensor Action dialog.

13. In the Create Sensor Action dialog:
   - **Name**: enter InstanceStart.
   - **Publish Type**: select JMS Topic.
   - **JMS Connection Factory**: enter jms/TopicConnectionFactory.
   - **Publish Target**: enter jms/demoTopic.
   - **Filter**: leave blank.
   - **Enable**: select this option.

*Figure 8–18  Sensor Action Chooser Dialog*

*Figure 8–19  Create Sensor Action Dialog*
14. Click OK in the Create Sensor Action dialog. The Sensor Action Chooser dialog now shows the **InstanceStart** sensor action.

*Figure 8–20 Sensor Action Chooser Dialog*

15. Click OK in the Sensor Action Chooser. This takes you back to the Create Activity Sensor dialog, which now looks like this:

*Figure 8–21 Create Activity Sensor Dialog*

16. Click OK in the Create Activity Sensor dialog.

   In the Receive dialog, the Sensors tab now looks like this:
17. Click **OK** in the Receive dialog.

18. Select File > Save to save your work.

The SOAOrderBooking page now looks like this:

**Figure 8–23  SOAOrderBooking.bpel Page with Receive Activity**

![Diagram showing the SOAOrderBooking.bpel page with a receive activity.]

The sensor information is stored in these files in the `SOAOrderBooking\bpel` directory:

- `sensor.xml`
- `sensorAction.xml`

### 8.7 Insert Order Information in the Database ("InsertOrderIntoDB" Scope)

In this scope, you create activities to insert the order information into the database. **Figure 8–24** shows the activities that you will create for the "InsertOrderIntoDB" scope.
8.7.1 Create a Database Adapter for Writing to the ORDERS Table

Activities in the "InsertOrderIntoDB" scope use a database adapter for writing order information to the ORDERS table in the database. During runtime, the database adapter connects to the database through the eis/DB/soademo connection that you set up in Section 2.6.4, "Create a Database Adapter Connection Factory".

To create the database adapter:

1. In the Component Palette, select Services from the dropdown.

2. Drag the Database Adapter icon from the Component Palette and drop it in a Services swimlane. This launches the Adapter Configuration wizard. Click Next on the welcome page to continue.

3. In Step 1, Service Name, enter Order as the Service Name, and click Next.
4. In Step 2, Service Connection:
   - **Connection**: select the database connection for the SOADEMO schema.
   - **JNDI Name**: enter eis/DB/soademo. This is the JNDI location you specified when you created connection factory in Oracle Application Server (see Section 2.6.4, "Create a Database Adapter Connection Factory").

5. In Step 3, Operation Type, select **Perform an Operation on a Table**, and select **Insert or Update (Merge)**. The SOA Order Booking application only needs to insert rows in the table.
6. In Step 4, Select Table, click **Import Tables**.
   
   In the Import Tables dialog, click **Query**.
   
   Select **ORDERS** and **ITEMS** and click the right-arrow button to move them to the Selected box. The database adapter will write to these tables.

   **Figure 8–28** Import Tables Dialog for FedexShipment

   Click **OK** in the Import Tables dialog. Step 4, Select Table, now looks like this:
7. Select SOADEMO.ORDERS and click Next.

8. In Step 5, Relationships, click Next.


10. JDeveloper displays the Create Partner Link dialog with the fields filled in:
Insert Order Information in the Database ("InsertOrderIntoDB" Scope)

Figure 8–31 Create Partner Link Dialog for Order Database Adapter

Click OK in the Create Partner Link dialog. You should see the "Orders" database adapter in the Services swimlane.

11. Select File > Save to save your work.

The wizard creates the following files in the SOAOrderBooking directory:

- `bpel\DBAdapterOutboundHeader.wsdl` -- this file contains generic information for connecting to a database.
- `bpel\Order.wsdl` -- this file contains the information that you specified in the wizard.
- `bpel\Order_table.xsd` -- this file contains the schema information for the ORDERS and ITEMS tables.
- `bpel\Order_toplink_mappings.xml` -- this file is used by TopLink
- `src\Order\Orders.java`
- `src\Order\Items.java`
- `database\SOADEMO\ORDERS.table`
- `database\SOADEMO\ITEMS.table`
- `toplink\Order\Order.mwp`

8.7.2 Create a Database Adapter for Retrieving the Order ID from the Database

Before you can insert order information in the ORDERS table in the database, you need to retrieve the order ID from the database. The order ID is generated from a database sequence.

1. In the Component Palette, select Services from the dropdown.

2. Drag the Database Adapter icon from the Component Palette and drop it in a Services swimlane. This launches the Adapter Configuration Wizard. Click Next on the welcome page to continue.

3. In Step 1, Service Name, enter OrderSequence as the Service Name, and click Next.
4. In Step 2, Service Connection:
   - **Connection**: select the database connection for the SOADEMO schema.
   - **JNDI Name**: enter `eis/DB/soademo`. This is the JNDI location you specified when you created a connection factory in Oracle Application Server (see Section 2.6.4, "Create a Database Adapter Connection Factory").

5. In Step 3, Operation Type, select **Execute Custom SQL**.
6. In Step 4, Custom SQL, enter the following SQL statement in the SQL box:
   `select order_seq_id_gen.nextval from dual`
   The wizard displays the appropriate statements in the XSD box.


8. JDeveloper displays the Create Partner Link dialog with the fields filled in:
Click OK in the Create Partner Link dialog. You should see the OrderSequence database adapter in the Services swimlane.

9. Select File > Save to save your work.

The wizard creates the following files in the SOAOrderBooking directory:

- `bpel\OrderSequence.wsdl` -- this file contains the information that you specified in the wizard.
- `bpel\OrderSequence.xsd` -- this file defines the format of the sequence input and output.

### 8.7.3 Create the "InsertOrderIntoDB" Scope

Create a new scope called "InsertOrderIntoDB" to contain the activities for writing order information in the database.

1. In the Component Palette, select Process Activities from the dropdown.

2. Drag the Scope icon from the Component Palette and drop it between the "receiveInput" activity and the "callbackClient" activity.

3. Double-click the new scope to display the Scope dialog.

4. In the Scope dialog, in the General tab:
   - Name: enter InsertOrderIntoDB.
   - Variable Access Serializable: do not select.

5. Click the Variables tab. You need to create three variables for this scope:
   - `orderRequest`, `orderSequenceInput`, `orderSequenceOutput`.

6. Create the `orderRequest` variable:
   - a. In the Variables tab, click Create.
   - b. In the Create Variable dialog:
     - Name: enter orderRequest.
     - Select Message Type and click the flashlight icon to display the Type Chooser. In the Type Chooser, select Type Explorer > Message Types >
Partner Links > Order > Order.wsdl > Message Types > OrdersCollection_msg.

**Figure 8–37  Type Chooser Dialog for orderRequest Variable**

![Type Chooser Dialog](image)

Click **OK** in the Type Chooser.

In the Create Variable dialog, the Message Type is set to


**Figure 8–38  Create Variable Dialog for orderRequest Variable**

![Create Variable Dialog](image)

c. Click **OK** in the Create Variable dialog. This returns you to the Scope dialog.

7. Create the **orderSequenceInput** variable:

a. In the Variables tab, click **Create**.

b. In the Create Variable dialog:
Insert Order Information in the Database ("InsertOrderIntoDB" Scope)

- Name: enter orderSequenceInput.
- Select Message Type and click the flashlight icon to display the Type Chooser. In the Type Chooser, select Type Explorer > Message Types > Partner Links > OrderSequence > OrderSequence.wsdl > Message Types > OrderSequenceInput_msg.

**Figure 8–39 Type Chooser Dialog for orderSequenceInput Variable**

Click OK in the Type Chooser.

In the Create Variable dialog, the Message Type is set to 
(http://xmlns.oracle.com/pcbpel/adapter/db/OrderSequence/)OrderSequenceInput_msg.

**Figure 8–40 Create Variable Dialog for orderSequenceInput Variable**

c. Click OK in the Create Variable dialog. This returns you to the Scope dialog.

8. Create the orderSequenceOutput variable:
a. In the Variables tab, click **Create**.

b. In the Create Variable dialog:
   - **Name**: enter `orderSequenceOutput`.
   - Select **Message Type** and click the flashlight icon to display the Type Chooser. In the Type Chooser, select **Type Explorer > Message Types > Partner Links > OrderSequence > OrderSequence.wsdl > Message Types > OrderSequenceOutputCollection_msg**.

**Figure 8–41 Type Chooser Dialog for orderSequenceOutput Variable**

Click **OK** in the Type Chooser.

In the Create Variable dialog, the Message Type is set to `http://xmlns.oracle.com/pcbpel/adapter/db/OrderSequence/OrderSequenceOutputCollection_msg`.

**Figure 8–42 Create Variable Dialog for orderSequenceOutput Variable**
c. Click OK in the Create Variable dialog. This returns you to the Scope dialog. You should see all three variables in the dialog.

**Figure 8–43 Scope Dialog for InsertOrderIntoDB, Variables Tab**

![Scope Dialog for InsertOrderIntoDB, Variables Tab]

9. Click OK in the Scope dialog.

10. Select File > Save to save your work.

**8.7.4 Retrieve the Order ID from the Database Sequence ("GetOrderId" Invoke Activity)**

This invoke activity retrieves the next value from the order_seq_id_gen database sequence and stores it in the orderSequenceOutput variable.

1. Expand the "InsertOrderIntoDB" scope.

2. Drag the Invoke icon from the Component Palette and drop it in the "InsertOrderIntoDB" scope.

3. Do one of the following to display the Invoke dialog:
   - Drag one of the arrows on the side of the "Invoke_1" activity and drop it on the "OrderSequence" database adapter. This associates the invoke activity with the database adapter.
   - Double-click the new Invoke_1 activity.

4. In the Invoke dialog, set these values:
   - **Name**: enter GetOrderId.
   - **Partner Link**: should be set to OrderSequence. If not, click the flashlight and select OrderSequence from the Partner Link Chooser.
Insert Order Information in the Database ("InsertOrderIntoDB" Scope)

Figure 8–44 Partner Link Chooser for "InsertOrder" Invoke Activity

Click OK in the Partner Link Chooser.

- **Operation**: select OrderSequence.
- **Input Variable**: click the Browse Variables icon (the second icon from the left) and select Process > Variables > Scope - InsertOrderIntoDB > Variables > orderSequenceInput.

Figure 8–45 Variable Chooser Dialog for Input Variable for "GetOrderId" Invoke Activity

Click OK in the Variable Chooser.

- **Output Variable**: click the Browse Variables icon (the second icon from the left) and select Process > Variables > Scope - InsertOrderIntoDB > Variables > orderSequenceOutput.
Click **OK** in the Variable Chooser.

The Invoke dialog should look like this:

5. Click **OK** in the Invoke dialog.
6. Select **File > Save** to save your work.

The **SOAOrderBooking.bpel** page should now look like this:
8.7.5 Prepare the Order ID and Order Status Information ("AssignOrderStatus" Assign Activity)

This assign activity prepares two values for insertion into the database:

- It invokes the order_seq_id_gen database sequence to get the order ID.
- It sets the order status to "pending".

These values are used by the "InsertOrder" invoke activity when it writes the order information to the database.

To create this assign activity:

1. Drag the Assign activity icon from the Component Palette and drop it below the "GetOrderId" activity in the "InsertOrderIntoDB" scope.
2. Double-click the new assign activity to display the Assign dialog.
3. In the Assign dialog, click the General tab, and set the Name to AssignOrderStatus.
4. Still in the Assign dialog, click the Copy Operation tab and create two copy operations: one to copy the order ID and another one to copy the order status.
5. Create the copy operation to copy the order ID:
   a. Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.
   b. In the From side, set Type to Variable, and select Variables > Process > Scope - InsertOrderIntoDB > Variables > orderSequenceOutput > OrderSequenceOutputCollection > ns4:OrderSequenceOutputCollection > ns4:OrderSequenceOutput > ns4:order_seq_id_gen.nextval.
   c. In the To side, set Type to Variable, and select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:ID.

Figure 8–48  SOAOrderBooking.bpel Page
d. Click OK in the Create Copy Operation dialog.

6. Create the copy operation to copy the order status:
   a. Select **Copy Operation** from the **Create** dropdown again to create the second copy operation. This displays the Create Copy Operation dialog.
   b. In the **From** side, set **Type** to **Expression**, and enter the following line in the **Expression** box:
      
      ```
      string('pending')
      ```
   
   c. In the **To** side, set **Type** to **Variable**, and select **Variables** > **Process** > **Variables** > **inputVariable** > **payload** > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:OrderInfo > ns1:OrderStatus.
d. Click **OK** in the Create Copy Operation dialog.

7. You should see two copy operations in the Assign dialog. Click **OK**.

8. Select File > Save to save your work.

The SOAOrderBooking.bpel page should now look like this:
8.7.6 Create the Mapping File ("TransformOrder" BPEL Service)

In this transform service, you create a file called TransformOrder.xsl to map the incoming order information to the schema defined in Order_table.xsd, which prepares it for insertion in the database. Order_table.xsd was created by the Adapter Configuration wizard that you ran in Section 8.7.1, "Create a Database Adapter for Writing to the ORDERS Table".

1. In the Component Palette, select Process Activities from the dropdown.
2. Drag the Transform icon from the Component Palette and drop it after the "AssignOrderStatus" activity.
3. Double-click the new transform activity to display the Transform dialog.
4. In the Transform dialog, click the General tab and set the Name to TransformOrder.
5. Click the Transformation tab in the dialog and set these values:
   - Source Variable: select inputVariable. The Source Part should be set to payload.
   - Target Variable: select orderRequest. The Target Part should be set to OrdersCollection.
   - Mapper File: enter TransformOrder.xsl and click the Create Mapping icon (the middle icon). This displays the Data Mapping tool for TransformOrder.xsl.

   In the Data Mapping tool, create the simple mappings shown in Table 8–2. The left column shows the source side and the right column shows the target side. You create the simple mappings by dragging and dropping the labels from the source side to the target side.
Table 8–2  Simple Mappings

<table>
<thead>
<tr>
<th>Source &gt;</th>
<th>Target &gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>client:SOAOrderBookingProcessRequest &gt; po:PurchaseOrder</td>
<td>OrdersCollection &gt; Orders</td>
</tr>
<tr>
<td>po:CustID</td>
<td>custid</td>
</tr>
<tr>
<td>po:ID</td>
<td>ordid</td>
</tr>
<tr>
<td>po:OrderInfo &gt; po:OrderDate</td>
<td>orderdate</td>
</tr>
<tr>
<td>po:OrderInfo &gt; po:OrderPrice</td>
<td>price</td>
</tr>
<tr>
<td>po:OrderInfo &gt; po:OrderStatus</td>
<td>status</td>
</tr>
</tbody>
</table>

Figure 8–53 shows the Data Mapper with the simple mappings done.

Process all the items in the order by creating a "for-each" element in the XSL file:

a. Select XSLT Constructs from the dropdown in the Component Palette in the Data Mapper.

b. On the target side, expand itemsCollection and Items.

c. Drag the for-each item from the Component Palette and drop it on Items. You want the for-each label to show up between itemsCollection and Items on the target side.

d. Expand po:OrderItems on the source side to that you can see po:Item below it. Expand po:Item as well.

e. Drag a line from po:Item on the source side to the for-each item on the target side.

f. Map the following item fields:
Insert Order Information in the Database ("InsertOrderIntoDB" Scope)

For `itemid` on the target side, you want to map it to the `position()` function. To do this in the Data Mapper:

a. Select **Node-set Functions** from the dropdown in the Component Palette.

b. Drag the `position` item from the Component Palette and drop it in the middle area (between the source and target areas).

c. Drag a line from `itemid` on the target side to the `position` item in the middle area.

d. Select File > Save to save `TransformOrder.xsl`.

Figure 8–54 shows the complete transformation in the Data Mapper.

e. Select File > Close to close `TransformOrder.xsl`. This takes you back to the main editor in JDeveloper.


### 8.7.7 Insert the Order Information into the Database ("InsertOrder" Invoke Activity)

This invoke activity inserts the data into the database.

1. In the Component Palette, select **Process Activities** from the dropdown.
2. Drag the Invoke icon from the Component Palette and drop it below the "TransformOrder" activity.

3. Do one of the following to display the Invoke dialog:
   - Drag one of the arrows on the side of the Invoke_1 activity and drop it on the "Order" database adapter. This associates the Invoke_1 activity with the database adapter.
   - Double-click the new Invoke_1 activity.

4. In the Invoke dialog, set these values:
   - **Name:** enter **InsertOrder**.
   - **Partner Link:** should be set to **Order**. If not, click the flashlight and select **Order** from the Partner Link Chooser.

![Figure 8–55 Partner Link Chooser for "InsertOrder" Invoke Activity](image)

Click **OK** in the Partner Link Chooser.

- **Operation:** select **write**.
- **Input Variable:** click the Browse Variables icon (the second icon from the left) and select **Variables > Process > Scope - InsertOrderIntoDB > Variables > orderRequest**.
Insert Order Information in the Database ("InsertOrderIntoDB" Scope)

Figure 8–56  Variable Chooser Dialog for Input Variable for "InsertOrder" Invoke Activity

Click OK in the Variable Chooser.

The Invoke dialog should look like this:

Figure 8–57  Invoke Dialog for "InsertOrder" Invoke Activity

5. Click OK in the Invoke dialog.

6. Select File > Save to save your work.

The SOAOrderBooking.bpel project should now look like this:
8.7.8 Minimize the "InsertOrderIntoDB" Scope

Click the [-] icon for the "InsertOrderIntoDB" scope to minimize it. When it is minimized, it makes it easier to create a new scope that is at the same level as the scope you minimized. Otherwise, you can accidentally drag and drop a new scope inside the existing scope. To see what a minimized scope looks like, see Figure 8–1.

8.8 Retrieve Information About the Customer ("CustomerService" Scope)

This scope invokes the CustomerService service to retrieve information about the customer. It uses the assign activity to populate variables with the returned information.

Figure 8–59 shows the activities in the "CustomerService" scope.
8.8.1 Create the "CustomerService" Partner Link

The "CustomerService" partner link provides the SOAOrderBooking project with a way to communicate with the CustomerService service deployed on Oracle Application Server. The CustomerSvc.wsdl file includes a URL that returns the WSDL for CustomerService.

1. Copy the following file from the soademo_101310_prod.zip file to the SOAOrderBooking\bpel directory.
   - CustomerSvc.wsdl
     In the soademo_101310_prod.zip file, the CustomerSvc.wsdl file is located in the SOAOrderBooking\bpel directory.

2. Verify the URL in the CustomerSvc.wsdl file.
   a. In JDeveloper, select File > Open and open the CustomerSvc.wsdl file.
   b. Click the Source tab at the bottom of the editor to view the lines in the file.
   c. The http://localhost:8888 reference in the file assumes that Oracle Application Server is running on the same machine as JDeveloper, and that Oracle Application Server is listening for requests on port 8888.
      If necessary, change localhost to the name of the machine running Oracle Application Server, and 8888 to the correct port used by your Oracle Application Server installation, for example: mypc.mydomain.com:8889.
   d. If you edited the file, save the file and close it.

3. In the Component Palette, select Services from the dropdown.

4. Drag the Partner Link icon from the Component Palette and drop it in a Services swimlane.

5. In the Create Partner Link dialog:
   - Name: enter CustomerService.
Retrieve Information About the Customer ("CustomerService" Scope)

- **WSDL File**: click the Service Explorer icon (second icon from the left) to display the Service Explorer dialog. In the Service Explorer dialog, expand `Project WSDL Files` and select `CustomerSvc.wsdl`.

![Figure 8–60 Service Explorer for "CustomerService" Partner Link](image)

Click **OK** in the Service Explorer.

- **Partner Link Type**: select `CustomerService_PL` (automatically filled in for you).
- **Partner Role**: select `CustomerService_Role`.
- **My Role**: leave it blank.

The Create Partner Link dialog should look like this:

![Figure 8–61 Create Partner Link Dialog for "CustomerService" Partner Link](image)

Before clicking **OK**, check that the **Name** is `CustomerService`. JDeveloper may have changed it to `CustomerSvc`. Click **OK** in the Create Partner Link dialog.
8.8.2 Create the "CustomerService" Scope

Create a new scope called "CustomerService".

1. In the Component Palette, select Process Activities from the dropdown.
2. Drag the Scope icon from the Component Palette and drop it below the "InsertOrderIntoDB" scope.
3. Double-click the new scope to display the Scope dialog.
4. In the Scope dialog, in the General tab:
   - **Name**: enter CustomerService.
   - **Variable Access Serializable**: do not select.
5. Click the Variables tab. You need to create a variable for this scope.
6. In the Variables tab, click Create.
7. In the Create Variable dialog:
   - **Name**: enter customerServiceRequest.
   - Select **Message Type** and click the flashlight icon to display the Type Chooser.
     - In the Type Chooser, select Type Explorer > Message Types > Partner Links > CustomerService > CustomerSvc.wsdl > Imported WSDL > CustomerService > Message Types > CustomerService_findCustomerById.

Click **OK** in the Type Chooser.

8. In the Create Variable dialog, the Message Type is set to 
   
   ```
   {http://www.globalcompany.com/ns/customer}CustomerService_findCustomerById
   ```
9. Click OK in the Create Variable dialog.

10. The `customerServiceRequest` variable appears in the Variables tab of the Scope dialog.

![Create Variable Dialog for "customerServiceRequest" Variable](image)

Click OK in the Scope dialog.

### 8.8.3 Assign Customer ID to the findCustomerById Operation ("AssignRequest" Assign Activity)

This assign activity assigns the customer ID information to the `customerServiceRequest` variable. This variable is then used as the input variable in the "GetCustInfo" invoke activity.

1. Expand the CustomerService scope.
2. Drag the Assign activity icon from the Component Palette and drop it in the CustomerService scope.
3. Double-click the new assign activity to display the Assign dialog.
4. In the Assign dialog, click the General tab, and set the Name to `AssignRequest`.

5. Still in the Assign dialog, click the Copy Operation tab.

6. Select **Copy Operation** from the **Create** dropdown. This displays the Create Copy Operation dialog.
   - In the **From** side, set **Type** to **Variable**, and select **Variables > Process > Variables > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:CustID**.
   - In the **To** side, set **Type** to **Variable**, and select **Variables > Process > Scope - Customer Service > Variables > customerServiceRequest > parameters > ns8:findCustomerById > ns8:custid**.

   **Figure 8–65 Create Copy Operation Dialog for "AssignRequest" Activity**

   ![Create Copy Operation Dialog](image)

   Click **OK** in the Create Copy Operation dialog.

7. You should see the copy operation in the Assign dialog. Click **OK**.
8.8.4 Create a Variable to Contain the Results of findCustomerById ("customerServiceResponse" Process Variable)

The `customerServiceResponse` variable is used to contain the results of the findCustomerById operation.

Create the `customerServiceResponse` variable in the SOAOrderBooking scope. The variable is created at this level so that any activity in this BPEL process can access it.

1. Double-click the SOAOrderBooking scope. You can double-click the "SOAOrderBooking" text that is sideways. This displays the Process dialog.

2. In the Process dialog, click **Create** in the Variables tab. This displays the Create Variable dialog.

3. In the Create Variable dialog:
   - **Name**: enter `customerServiceResponse`.
   - **Type**: select **Message Type**, and click the flashlight icon. This displays the Type Chooser dialog.

   In the Type Chooser, select **Type Explorer** > **Message Types** > **Partner Links** > **CustomerService** > **CustomerSvc.wsdl** > **Imported WSDL** > **Customer Service** > **Message Types** > **CustomerService_findCustomerByIdResponse**.
Click OK in the Type Chooser. The Create Variable dialog now looks like this:

4. Click OK in the Create Variable dialog. The `customerServiceResponse` variable appears in the Process dialog.
5. Click OK in the Process dialog.

6. Select File > Save to save your work.

8.8.5 Invoke findCustomerById ("GetCustInfo" Invoke Activity)

This invoke activity accesses the CustomerService partner link and invokes the findCustomerById operation. The operation is invoked with the customer ID assigned in the "AssignRequest" activity. The results of the operation is stored in the customerServiceResponse variable.

1. Drag the Invoke icon from the Component Palette and drop it below the "AssignRequest" activity.

2. Do one of the following to display the Invoke dialog:
   - Drag one of the arrows on the side of the Invoke_1 activity and drop it on the "CustomerService" partner link. This associates the invoke activity with the partner link.
   - Double-click the new Invoke_1 activity.

3. In the Invoke dialog, set these values:
   - Name: enter GetCustInfo.
   - Partner Link: should be set to CustomerService. If not, click the flashlight and select CustomerService from the Partner Link Chooser.
Click OK in the Partner Link Chooser.

- **Operation**: select `findCustomerById`

- **Input Variable**: click the Browse Variables icon (the second icon from the left) and select `Variables > Process > Scope - CustomerService > Variables > customerServiceRequest`.

Click OK in the Variable Chooser.

- **Output Variable**: click the Browse Variables icon (the second icon from the left) and select `Variables > Process > Variables > customerServiceResponse`. 

---

*Figure 8–70  Partner Link Chooser Dialog for "GetCustInfo" Invoke Activity*

*Figure 8–71  Variable Chooser Dialog for the Input Variable for the "GetCustInfo" Invoke Activity*
Click OK in the Variable Chooser. The Invoke dialog should look like this:

4. Click OK in the Invoke dialog.
5. Select File > Save to save your work.

8.8.6 Create the "AssignInitialCustomerResponse" Assign Activity

This assign activity appends the customer’s first name and last name to the payload part of the inputVariable. It uses an XML fragment to perform this append operation.

1. Drag the Assign activity icon from the Component Palette and drop it below the "GetCustInfo" activity.
2. Double-click the new assign activity to display the Assign dialog.

3. In the Assign dialog, click the General tab, and set the Name to `AssignInitialCustomerResponse`.

4. Still in the Assign dialog, click the Copy Operation tab. You will create an append operation.

5. Select Append Operation from the Create dropdown. This displays the Create Append Operation dialog.
   - In the From side, select XML Fragment from the Type dropdown, and enter the following in the XML Fragment box:
     ```xml
     <nsx:ShipTo xmlns:nsx="http://www.globalcompany.com/ns/order">
       <nsx:Name>
         <nsx:First/>
         <nsx:Last/>
       </nsx:Name>
     </nsx:ShipTo>
     ```
     Note that you need to replace the `nsx` prefix with the prefix for "http://www.globalcompany.com/ns/order". To determine the prefix, scroll to the beginning of your SOAOrderBooking.bpel file and look for this line:
     ```xml
     xmlns:prefix="http://www.globalcompany.com/ns/order"
     ```
     For example, if the line in your file looks like the following:
     ```xml
     xmlns:ns1="http://www.globalcompany.com/ns/order"
     ```
     then you need to change the prefix to `ns1` in the XML fragment.
   - In the To side, select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder.
Retrieve Information About the Customer ("CustomerService" Scope)

Creating the SOAOrderBooking Project

8-51

Figure 8–74  Create Append Operation Dialog for the "AssignInitialCustomerResponse" Activity

Click OK in the Create Append Operation dialog.

6. You should see the append operation in the Assign dialog.

Figure 8–75  Assign Dialog for the "AssignInitialCustomerResponse" Activity

\[
\text{<ns1:ShipTo xmlns:ns1="http://www.globalcompany.com/ns/order">}
\]  
\[
\text{<ns1:Name>}
\]  
\[
\text{<ns1:First/>}
\]  
\[
\text{<ns1:Last/>}
\]  
\[
\text{</ns1:Name>}
\]  
\[
\text{</ns1:ShipTo>}
\]  

*******************************************************************************

Click OK in the Create Append Operation dialog.

6. You should see the append operation in the Assign dialog.
Click OK in the Assign dialog.

7. Select File > Save to save your work.

8.8.7 Copy the Customer’s First and Last Names to the inputVariable ("AssignCustomerResponse" Assign Activity)

This assign activity assigns the first and last names of the customer (which were retrieved from the database) to the inputVariable.

1. Drag the Assign activity icon from the Component Palette and drop it below the "AssignInitialCustomerResponse" activity.

2. Double-click the new assign activity to display the Assign dialog.

3. In the Assign dialog, click the General tab, and set the Name to AssignCustomerResponse.

4. Still in the Assign dialog, click the Copy Operation tab. You will create two copy operations.

5. Create the first copy operation: Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.

   ■ In the From side, set Type to Variable, and select Variables > Process > Variables > customerServiceResponse > parameters > ns8:findCustomerByIdResponse > ns8:return > ns8:fname.

   ■ In the To side, set Type to Variable, and select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:ShipTo > ns1:Name > ns1:First.

**Figure 8-76 Create Copy Operation Dialog for "AssignCustomerResponse" Activity, First Copy Operation**

Click OK in the Create Copy Operation dialog.
6. Create the second copy operation: Select **Copy Operation** from the **Create** dropdown again. This displays the Create Copy Operation dialog.

   - In the **From** side, set **Type** to **Variable**, and select **Variables** > **Process** > **Variables** > **customerServiceResponse** > **parameters** > **ns8:findCustomerByIdResponse** > **ns8:return** > **ns8:lname**.
   - In the **To** side, set **Type** to **Variable**, and select **Variables** > **Process** > **Variables** > **inputVariable** > **payload** > **client:SOAOrderBookingProcessRequest** > **ns1:PurchaseOrder** > **ns1:ShipTo** > **ns1:Name** > **ns1:Last**.

   *Figure 8–77  Create Copy Operation Dialog for “AssignCustomerResponse” Activity, Second Copy Operation*

   ![Create Copy Operation Dialog](image)

   Click **OK** in the Create Copy Operation dialog.

7. You should see two copy operations in the Assign dialog. Click **OK**.
8. Select File > Save to save your work.

8.8.8 Minimize the "CustomerService" Scope

Click the [-] icon to minimize the "CustomerService" scope.

8.9 Verify the Customer’s Credit Card ("CreditService" Scope)

This scope verifies the credit of the customer. If a customer does not pass the credit check, the scope throws a fault and does not continue with the rest of the flow.

Figure 8–79 shows the activities in the "CreditService" scope.
8.9.1 Create "CreditValidatingService" Partner Link

The "CreditService" scope uses the CreditService created in Chapter 5, "Creating the CreditService Project". The "CreditValidatingService" partner link is the interface to the CreditService.

1. Copy the following file from the soademo_101310_prod.zip file to the SOAOrderBooking\bpel directory.
   - CreditValidatingService.wsdl

   In the soademo_101310_prod.zip file, the CreditValidatingService.wsdl file is located in the SOAOrderBooking\bpel directory.

2. Verify the URL in the CreditValidatingService.wsdl file.
   a. In JDeveloper, select File > Open and open the CreditValidatingService.wsdl file.
   b. Click the Source tab at the bottom of the editor to view the lines in the file.
   c. The http://localhost:8888 reference in the file assumes that Oracle Application Server is running on the same machine as JDeveloper, and that Oracle Application Server is listening for requests on port 8888.
If necessary, change localhost to the name of the machine running Oracle Application Server, and 8888 to the correct port used by your Oracle Application Server installation, for example: mypc.mydomain.com:8889.

d. If you edited the file, save the file and close it.

3. In the Component Palette, select Services from the dropdown.

4. Drag the Partner Link icon from the Component Palette and drop it in a Services swimlane.

5. In the Create Partner Link dialog:
   - Name: enter CreditValidatingService.
   - WSDL File: click the Service Explorer icon (second icon from the left) to display the Service Explorer dialog. In the Service Explorer dialog, expand Project WSDL Files and select CreditValidatingService.wsdl.

   Figure 8–80 Service Explorer Dialog for “CreditValidatingService” Partner Link

   Click OK in the Service Explorer.
   - Partner Link Type: select ValidateCreditCard_PL (automatically filled in for you).
   - Partner Role: select ValidateCreditCard_Role.
   - My Role: leave it blank.

   The Create Partner Link dialog should look like this:
8.9.2 Create the "CreditService" Scope

Create a new scope called "CreditService".

1. In the Component Palette, select Process Activities from the dropdown.
2. Drag the Scope icon from the Component Palette and drop it below the "CustomerService" scope.
3. Double-click the new scope to display the Scope dialog.
4. In the Scope dialog, in the General tab:
   ■ Name: enter CreditService.
   ■ Variable Access Serializable: do not select.
5. Click the Variables tab. You need to create two variables for this scope.
6. In the Variables tab, click Create.
7. In the Create Variable dialog:
   ■ Name: enter validateRequest.
   ■ Select Message Type and click the flashlight icon to display the Type Chooser. In the Type Chooser, select Type Explorer > Message Types > Partner Links > CreditValidatingService > CreditValidatingService.wsdl > Imported WSDL > ValidateCreditCardServiceSoapHttp > Message Types > CreditCardValidationRequestMessage.
8. Click OK in the Type Chooser.

In the Create Variable dialog, the Message Type is set to `{http://www.globalcompany.com/ns/credit}CreditCardValidationRequestMessage`.

9. Click OK in the Create Variable dialog.

10. Click Create again to create a second variable.

11. In the Create Variable dialog:

   - **Name**: enter `validateResponse`.
   - Select **Message Type** and click the flashlight icon to display the Type Chooser. In the Type Chooser, select **Type Explorer > Message Types > Partner Links > CreditValidatingService > CreditValidatingService.wsdl > Imported WSDL**.
Verify the Customer's Credit Card ("CreditService" Scope)

Creating the SOAOrderBooking Project

> ValidateCreditCardServiceSoapHttp > Message Types > CreditCardValidationResponseMessage.

**Figure 8–84 Type Chooser Dialog for the "validateResponse" Variable**

![Type Chooser Dialog](image)

Click **OK** in the Type Chooser.

12. In the Create Variable dialog, the Message Type is set to {http://www.globalcompany.com/ns/credit}CreditCardValidationResponseMessage.

**Figure 8–85 Create Variable Dialog for the "validateResponse" Variable**

![Create Variable Dialog](image)

13. Click **OK** in the Create Variable dialog.

Verify the Customer’s Credit Card (“CreditService” Scope)

8.60 Oracle SOA Suite Tutorial

Figure 8–86  Scope Dialog for the “CreditService” Scope

Click OK in the Scope dialog.

15. Select File > Save to save your work.

8.9.3 Assign the Credit Card Number and Credit Card Type Information (“InitializeRequest” Assign Activity)

This assign activity copies the credit card number and type (which were retrieved from the CustomerService’s findCustomerById operation) to the validateRequest variable.

1. Expand the “CreditService” scope.

2. Drag the Assign activity icon from the Component Palette and drop it in the “CreditService” scope.

3. Double-click the new assign activity to display the Assign dialog.

4. In the Assign dialog, click the General tab, and set the Name to InitializeRequest.

5. Still in the Assign dialog, click the Copy Operation tab. You will create two copy operations.

6. Create the first copy operation: Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.

   ■ In the From side, set Type to Variable, and select Variables > Process > Variables > customerServiceResponse > parameters > ns8:findCustomerByIdResponse > ns8:return > ns8:creditcardnumber.

   ■ In the To side, set Type to Variable, and select Variables > Process > Scope - CreditService > Variables > validateRequest > CreditCard > ns11:CreditCard > ccNum.
Click OK in the Create Copy Operation dialog.

7. Create the second copy operation: Select Copy Operation from the Create dropdown again. This displays the Create Copy Operation dialog.
   - In the From side, set Type to Variable, and select Variables > Process > Variables > customerServiceResponse > parameters > ns8:findCustomerIdResponse > ns8:return > ns8:creditcardtype.
   - In the To side, set Type to Variable, and select Variables > Process > Scope - Credit Service > Variables > validateRequest > CreditCard > ns11:CreditCard > ccType.
Verify the Customer’s Credit Card ("CreditService" Scope)

Figure 8–88  Create Copy Operation Dialog for the “InitializeRequest” Activity, Second Copy Operation

Click **OK** in the Create Copy Operation dialog.

8. You should see the copy operations in the Assign dialog. Click **OK**.

**Figure 8–89**  Assign Dialog for the “InitializeRequest” Activity

8.9.4 Verify the Customer’s Credit Card ("InvokeCreditService" Invoke Activity)

This invoke activity checks whether or not the customer’s credit card is valid. The invoke activity uses the values assigned in the previous assign activity.

1. Drag the Invoke icon from the Component Palette and drop it below the "InitializeRequest" activity.

2. Do one of the following to display the Invoke dialog:
Drag one of the arrows on the side of the Invoke_1 activity and drop it on the "CreditValidatingService" partner link. This associates the invoke activity with the partner link.

Double-click the new Invoke_1 activity.

3. In the Invoke dialog, set these values:
   - **Name**: enter InvokeCreditService.
   - **Partner Link**: should be set to CreditValidatingService. If not, click the flashlight and select CreditValidatingService from the Partner Link Chooser.

   *Figure 8–90 Partner Link Chooser for the "InvokeCreditService" Activity*

   ![Partner Link Chooser](image)

   Click OK in the Partner Link Chooser.

   - **Operation**: select VerifyCC (should be filled in for you automatically).
   - **Input Variable**: click the Browse Variables icon (the second icon from the left) and select Variables > Process > Scope - CreditService > Variables > validateRequest.
Verify the Customer's Credit Card ("CreditService" Scope)

Click **OK** in the Variable Chooser.

- **Output Variable**: click the Browse Variables icon (the second icon from the left) and select Variables > Process > Scope - CreditService > Variables > validateResponse.

The Invoke dialog should look like this:
4. Click OK in the Invoke dialog.
5. Select File > Save to save your work.

### 8.9.5 Create the "OrderBookingFault" Process Variable

Create the OrderBookingFault variable in the SOAOrderBooking scope. This variable is created at this level so that any activity in this BPEL flow can access it.

1. Scroll to the top of the page and double-click the SOAOrderBooking scope. You can double-click the "SOAOrderBooking" text that is sideways. This displays the Process dialog.

2. In the Process dialog, click Create in the Variables tab. This displays the Create Variable dialog.

3. In the Create Variable dialog:
   - Name: enter OrderBookingFault.
   - Type: select Message Type, and click the flashlight icon. This displays the Type Chooser dialog.
     In the Type Chooser, select Type Explorer > Message Types > Partner Links > client > SOAOrderBooking.wsdl > Message Types > SOAOrderBookingFaultMessage.
Verify the Customer’s Credit Card (“CreditService” Scope)

Figure 8–94  Type Chooser Dialog for the “OrderBookingFault” Process Variable

Click OK in the Type Chooser. The Create Variable dialog now looks like this:

Figure 8–95  Create Variable Dialog for the “OrderBookingFault” Process Variable

4. Click OK in the Create Variable dialog. The OrderBookingFault variable appears in the Process dialog.
Verify the Customer's Credit Card ("CreditService" Scope)

Creating the SOAOrderBooking Project

8.9.6 Check the Results of the Credit Card Validation (Switch Activity)

This switch checks the result of the "InvokeCreditService" invoke activity. If the customer’s credit card is valid, the flow continues. If the customer’s credit card is not valid, the switch throws a fault.

1. Drag the Switch icon from the Component Palette and drop it below the "InvokeCreditService" activity.
2. Expand the switch.
3. Delete the <otherwise> case. This switch only handles invalid credit responses, for which it throws a fault. For credit cards that are valid, the switch does not apply to them and for those cases, they just continue to the next step in the flow.

8.9.6.1 Specify the Condition for <case>

For the <case> branch, you want to handle cases where a customer’s credit is not valid. This information is stored in the validateResponse variable.

1. Double-click the title bar of the <case> box to display the Switch Case dialog.
2. In the Switch Case dialog, click the XPath Expression Builder icon above the Expression box to display the Expression Builder dialog.
3. In the Expression Builder dialog, in the BPEL Variables box, select Variables > Process > Scope - CreditService > Variables > validateResponse > valid > ns11:valid.

The Content Preview box should show
bpws:getVariableData('validateResponse', 'valid', 'ns11:valid').

The ns11 prefix maps to the "http://www.globalcompany.com/ns/credit.xsd" namespace. It may be different on your system. If you want, you can scroll to the top of your SOAOrderBooking.bpel file (in source view) to verify the prefix.

4. Click Insert Into Expression. The Expression box should show the function with the three parameters.
5. Append '='false' to the expression in the Expression box so that the expression looks like this:

\[ \text{bpws:getVariableData('validateResponse','valid','/ns11:valid')}='false' \]

**Figure 8–97** Expression Builder Dialog Showing the Complete Expression for <case>

6. Click OK in the Expression Builder dialog.

7. Click OK in the Switch Case dialog.

### 8.9.6.2 Set the Value of the OrderBookingFault Variable ("AssignFault" Assign Activity)

This assign activity sets the value of the OrderBookingFault variable to "credit problem".

1. Drag the Assign activity icon from the Component Palette and drop it in the <case> area of the switch.

2. Double-click the new assign activity to display the Assign dialog.

3. In the Assign dialog, click the General tab, and set the Name to AssignFault.

4. Still in the Assign dialog, click the Copy Operation tab. You will create one copy operation.

5. Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.
   - In the From side, set Type to Expression, and enter the following line in the Expression box:

\[ \text{string('credit problem')} \]
- In the **To** side, set **Type** to **Variable**, and select **Variables > Process > Variables > OrderBookingFault > payload > client:SOAOrderBookingProcessFault > client:status**.

**Figure 8–98 Create Copy Operation Dialog for the "AssignFault" Activity**

Click **OK** in the Create Copy Operation dialog.

6. You should see the copy operation in the Assign dialog. Click **OK**.

**Figure 8–99 Assign Dialog for the "AssignFault" Activity**
8.9.6.3 Create the "ThrowCreditFault" Throw Activity

This throw activity throws a fault called OrderBookingFault. This fault value is stored in a variable also called OrderBookingFault. The SOAOrderBooking process terminates after executing the throw activity.

1. Drag the Throw icon from the Component Palette and drop it below the "AssignFault" activity.
2. Double-click the new throw activity to display the Throw dialog.
3. In the Throw dialog:
   - **Name**: enter ThrowCreditFault.
   - **Namespace URI**: enter http://www.globalcompany.com/ns/OrderBooking. You have to enter it manually.
   - **Local Part**: enter OrderBookingFault. You have to enter it manually.
   - **Fault Variable**: click the flashlight and select OrderBookingFault.

**Figure 8–100 Throw Dialog for the "ThrowCreditFault" Throw Activity**

4. Click OK in the Throw dialog.
5. Select File > Save to save your work.

8.9.7 Minimize the "CreditService" Scope

Click the [-] to minimize the "CreditService" scope.

8.10 Set up Oracle Business Rules

The repository for Oracle Business Rules contains the rules used by the decision service in Oracle BPEL Process Manager to determine whether or not an order needs to be approved manually by a manager. The Oracle Business Rules repository is separate from the SOA Order Booking application: you can edit the rules without modifying and redeploying the SOA Order Booking application.
The rules that you will create in the repository are:

- If a customer’s status is platinum, then a manager’s approval is not required, regardless of the amount of the order.
- If an order’s total is $1000 or more and the customer’s status is not platinum, then a manager’s approval is required.
- If an order’s total is under $1000, then a manager’s approval is not required.

8.10.1 Set up the Repository File

To create a repository file for Oracle Business Rules and to define the rules in it, you use Oracle Business Rules Rule Author ("Rule Author"), which is a Web-based tool that enables you to create and manage Oracle Business Rules repositories.

Location of the Repository File

Create the repository file in the SOAOrderBooking\bpel\rules\oracle directory.

However, if you are running JDeveloper and Oracle Application Server on different machines, you have to create the repository file on the Oracle Application Server machine (because Rule Author creates the file on the Oracle Application Server machine). After you have created the repository file and populated it with rules, you copy it from the Oracle Application Server machine to the JDeveloper machine.

1. Create the following directory where you are building the SOA Order Booking application:

   \SOADEMO\SOAOrderBooking\bpel\rules\oracle\sample_repository

   This is the directory where you will place the Oracle Business Rules repository file.

2. In a browser, go to the Rule Author page. The URL for Rule Author is:

   http://host:port/ruleauthor

   host specifies the machine running Oracle Application Server, and port specifies the HTTP port (for example, 8888).

3. Log in as the oc4jadmin user.

4. In the Repository tab:

   - Repository Type: select File.
   - File Location: enter \SOADEMO\SOAOrderBooking\bpel\rules\oracle\sample_repository. (SOADEMO refers to the directory where you are building the SOA Order Booking application.) Rule Author will create the sample_repository file.

   If you are running JDeveloper and Oracle Application Server on different machines, enter a path on the Oracle Application Server machine, for example, C:\rules\sample_repository. It does not matter where you place it, because after creating the file and defining the rules in it, you copy it from the Oracle Application Server machine to the \SOADEMO\SOAOrderBooking\bpel\rules\oracle\ directory on the JDeveloper machine.
Click Create. You should see a confirmation page. Rule Author is now connected to the new repository, and you can define rules in it.

### 8.10.2 Create a Dictionary in the Repository

A repository contains one or more dictionaries. A dictionary contains rulesets, which contain rules. A dictionary typically contains rulesets for an application.

1. Click the Create secondary tab on the Confirmation page. This displays the Create Dictionary page.
2. On the Create Dictionary page, enter **OrderBookingRules** as the dictionary name.
3. Click Create. You should get a confirmation page.

8.10.3 Copy OrderBookingRules.xsd to the Oracle Application Server Machine

This step is required only if you are running JDeveloper and Oracle Application Server on different machines. If you are running both on the same machine, you already have the OrderBookingRules.xsd file on the Oracle Application Server machine (you copied the file in Section 8.3, "Copy Files").

The OrderBookingRules.xsd file in the soademo_101310_prod.zip file describes the XML elements that provide the facts for Oracle Business Rules. The XSD file is located in the SOAOrderBooking\bpel directory in the zip file.

If you are running JDeveloper and Oracle Application Server on different machines, copy this file from the soademo_101310_prod.zip file to the Oracle Application Server machine, to the directory where you placed the repository file (for example, C:\rules). See step 4 on page 8-71.

In Section 8.10.4, "Generate JAXB Classes for the Elements in the XML Schema", you use Rule Author to generate Java classes for the XML elements defined in OrderBookingRules.xsd.
8.10.4 Generate JAXB Classes for the Elements in the XML Schema

Use Rule Author to generate Java objects for the elements in the OrderBookingRules.xsd file.

1. Click the Definitions tab, then click XMLFact on the left side. This displays the XML Fact Summary page. At this time, the OrderBookingRules dictionary does not contain any XML facts.

Figure 8–105   XML Fact Summary Page

2. Click Create.

3. On the XML Schema Selector page:

   - **XML Schema**: enter the fullpath to the OrderBookingRules.xsd file.
     - If you are running JDeveloper and Oracle Application Server on the same machine, enter SOADEMO\SOAOrderBooking\bpel\OrderBookingRules.xsd.
     - If you are running JDeveloper and Oracle Application Server on different machines, enter C:\rules\OrderBookingRules.xsd.

   - **JAXB Class Directory**: enter the fullpath to the directory where you want Rule Author to create the Java objects (.java and .class files).
     - If you are running JDeveloper and Oracle Application Server on the same machine, enter SOADEMO\SOAOrderBooking\bpel\rules.
     - If you are running JDeveloper and Oracle Application Server on different machines, enter C:\rules.

   - **Target Package Name**: enter com.oracle.demos.orderbooking. This specifies the package name for the classes that are to be created.
Set up Oracle Business Rules

Creating the SOAOrderBooking Project


Figure 8–107 shows the resulting page with the package hierarchy expanded.

In the directory that you specified in the **JAXB Class Directory** field, you should see the following files:

- Approve.java and Approve.class
- ApproveImpl.java and ApproveImpl.class
- ApproveType.java and ApproveType.class
- ApproveTypeImpl.java and ApproveTypeImpl.class
- jaxb.properties
8.10.5 Import the JAXB Classes into the Oracle Business Rules Data Model

After generating the JAXB classes, you can import them into the Oracle Business Rules data model. These JAXB classes become XML facts that you can use when you create your rules.

1. If you are not on the XML Schema Selector page, click the Definitions tab, then click XML Fact on the left side, then click Create on the XMLFact Summary page.

2. Select the check box next to com.

3. Click Import. You should see a confirmation page. The classes that have been imported are shown in bold. Also on the left side, note that there are now three XML Facts.
8.10.6 Define a Variable in the Data Model

You define variables in the data model so that if you need to make changes later, you only need to edit the value of the variable. In the case of the SOA Order Booking application, you create a variable called AUTOMATED_ORDER_LIMIT to define the dollar amount where orders above this amount would need manual approval from a manager and orders under this amount are approved automatically.

1. In the **Definitions** tab, click **Variable** on the left side.
2. In the Variable Summary page, click **Create**.
3. On the Variable page:
   - **Name**: enter AUTOMATED_ORDER_LIMIT.
   - **Alias**: enter AUTOMATED_ORDER_LIMIT.
   - **Final**: select this option.
   - **Type**: select float.
   - **Expression**: enter 1000.00.
4. Click OK. Rule Author shows the Variable Summary page. Note that the variable name is prefixed with DM (for "data model").

8.10.7 Create a Ruleset

A ruleset contains rules. Before you can create rules, you need a ruleset.

1. Click the Rulesets tab. This displays the RuleSet Summary page.
2. On the RuleSet Summary page, click Create. This displays the Ruleset page.
3. On the Ruleset page, enter ApproveOrderRequired as the Name. You can also enter a description if you like.
4. Click **OK**. You should see the RuleSet Summary page showing the new ruleset. The new ruleset also appears on the left side.

**Figure 8–113 RuleSet Summary Page**

### 8.10.8 Create Rules

You can now define your rules in the "ApproveOrderRequired" ruleset. For the SOA Order Booking application, create the following three rules:

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>belowLimit</strong></td>
<td>If an order’s total price is less than the value set in the AUTOMATED_ORDER_LIMIT variable, the order is approved automatically. See Section 8.10.8.1, &quot;Create the &quot;belowLimit&quot; Rule&quot;.</td>
</tr>
<tr>
<td><strong>overLimit</strong></td>
<td>If an order’s total price is greater than or equal to the value set in the AUTOMATED_ORDER_LIMIT variable and the customer is not a platinum customer, the order requires a manager’s approval. See Section 8.10.8.2, &quot;Create the &quot;overLimit&quot; Rule&quot;.</td>
</tr>
<tr>
<td><strong>platinumMember</strong></td>
<td>If the customer is a platinum customer, the order is approved automatically, regardless of the order amount. See Section 8.10.8.3, &quot;Create the &quot;platinumMember&quot; Rule&quot;.</td>
</tr>
</tbody>
</table>

### 8.10.8.1 Create the "belowLimit" Rule

The "belowLimit" rule states that if an order’s total price is less than the value set in the AUTOMATED_ORDER_LIMIT variable, the order is approved automatically.

1. Click the **Rulesets** tab.

2. Click the **ApproveOrderRequired** ruleset on the left side. This displays the Ruleset page.
3. In the Rules section, click Create. This displays the Rule page.

4. On the Rule page, enter belowLimit for the Name. Keep the default value for Priority.

5. In the "If" section, click New Pattern. This pops up the Pattern Definition window.

6. In the Pattern Definition window, in the Choose Pattern section:
   - Do not choose anything from the first field (that is, leave it empty).
   - Enter approve in the second field.
   - Select ApproveType from the dropdown.

7. In the Define Test for Pattern section, click Create. Then fill in the section as follows:
   - **Operand**: select approve.price.
   - **Operator**: select < (less than).
   - **Operand (choose Value or Field)**: select AUTOMATED_ORDER_LIMIT under Field. Select Fixed in the dropdown.

The Pattern Definition window should look like this:
Figure 8–115  Pattern Definition for "belowLimit" Rule

8. Click Apply in the Pattern Definition window. You should see a confirmation message at the top of the Pattern Definition window.

9. Click OK to return to the Rule page, which now looks like this:

Figure 8–116  Rule Page for the "belowLimit" Rule

10. Click Apply on the Rule page. You should see a confirmation message at the top of the Rule page.

11. In the "Then" section of the Rule page, click New Action. This pops up the Add Action window.

12. In the Add Action window:
   - Action Type: select Assign.
   - Name: select approve.approvalRequired.
   - Expression: enter false.
13. Click Apply.

14. Click OK to return to the Rule page, which now looks like this:

**Figure 8–118   Rule Page for the "belowLimit" Rule**

15. Click Apply.

16. Save your work. Click the Save Dictionary link at the top of the page, then click Save on the Save Dictionary page.

### 8.10.8.2 Create the "overLimit" Rule

The "overLimit" rule states that if an order’s total price is greater than or equal to the value set in the AUTOMATED_ORDER_LIMIT variable and the customer is not a platinum customer, the order requires a manager’s approval. If the customer is a platinum customer, then the order is approved automatically, regardless of the amount of the order.

1. Click the Rulesets tab.
2. Click the **ApproveOrderRequired** ruleset on the left side. This displays the Ruleset page. The "belowLimit" rule is already created.

![Figure 8–119  Ruleset Page for the "ApproveOrderRequired" Ruleset](image)

3. In the Rules section, click **Create**. This displays the Rule page.

4. On the Rule page, enter **overLimit** for the **Name**. Keep the default value for **Priority**.

5. In the "If" section, click **New Pattern**. This pops up the Pattern Definition window.

6. In the Pattern Definition window, in the Choose Pattern section:
   - Do not choose anything from the first field (that is, leave it empty).
   - Enter **approve** in the second field.
   - Select **ApproveType** from the dropdown.

7. In the Define Test for Pattern section, click **Create**. Then fill in the section as follows.
   - **Operand**: select **approve.price**.
   - **Operator**: select **>=** (greater than or equal to).
   - **Operand (choose Value or Field)**: select **AUTOMATED_ORDER_LIMIT** under **Field**. Select **Fixed** in the dropdown.

The Pattern Definition window should look like this:
8. Click **Create** again to define the second test. Fill in the section as follows.
   - **Operand**: select **approve.status**.
   - **Operator**: select != (not equal to).
   - **Operand (choose Value or Field)**: enter "Platinum" (include the double-quote characters) under **Value**. Select **Fixed** in the dropdown.

The Pattern Definition window should look like this:

![Pattern Definition for the "overLimit" Rule (part 2 of 2)](image)

9. Click **Apply** in the Pattern Definition window. You should see a confirmation message at the top of the Pattern Definition window.

10. Click **OK** to return to the Rule page.

11. The Rule page now looks like this:
12. Click **Apply** on the Rule page. You should see a confirmation message at the top of the Rule page.

13. In the "Then" section of the Rule page, click **New Action**. This pops up the Add Action window.

14. In the Add Action window:

   - **Action Type**: select **Assign**.
   - **Name**: select `approve.approvalRequired`.
   - **Expression**: enter `true`.

15. Click **Apply**.

16. Click **OK** to return to the Rule page, which now looks like this:
17. Click Apply.

18. Save your work. Click the Save Dictionary link at the top of the page, then click Save in the Save Dictionary page.

8.10.8.3 Create the "platinumMember" Rule

The "platinumMember" rule states that if a customer is a platinum customer, the order is approved automatically, regardless of the amount of the order.

1. Click the Rulesets tab.

2. Click the ApproveOrderRequired ruleset on the left side. This displays the Ruleset page. The "belowLimit" and "overLimit" rules are already created.

3. In the Rules section, click Create. This displays the Rule page.
4. On the Rule page, enter `platinumMember` for the **Name**. Keep the default value for **Priority**.

5. In the "If" section, click **New Pattern**. This pops up the Pattern Definition window.

6. In the Pattern Definition window, in the Choose Pattern section:
   - Do not choose anything from the first field (that is, leave it empty).
   - Enter `approve` in the second field.
   - Select **ApproveType** from the dropdown.

7. In the Define Test for Pattern section, click **Create**. Then fill in the section as follows.
   - **Operand**: select `approve.status`.
   - **Operator**: select `==` (equal to).
   - **Operand (choose Value or Field)**: enter "Platinum" (include the double-quote characters) under **Value**. Select **Fixed** in the dropdown.

   The Pattern Definition window should look like this:

   ![Pattern Definition for the “platinumMember” Rule](image)

8. Click **Apply** in the Pattern Definition window. You should see a confirmation message at the top of the Pattern Definition window.

9. Click **OK** to return to the Rule page.

10. The Rule page now looks like this:
11. Click **Apply** on the Rule page. You should see a confirmation message at the top of the Rule page.

12. In the "Then" section of the Rule page, click **New Action**. This pops up the Add Action window.

13. In the Add Action window:
   - **Action Type**: select **Assign**.
   - **Name**: select **approve.approvalRequired**.
   - **Expression**: enter **false**.

14. Click **Apply**.

15. Click **OK** to return to the Rule page, which now looks like this:
16. Click Apply.

17. Save your work. Click the Save Dictionary link at the top of the page, then click Save on the Save Dictionary page.

8.10.9 Log out of Rule Author

Click the Logout link at the top of the page.

On the Logout Confirmation page, click Save and Logout.

8.10.10 Copy the Files to the JDeveloper Machine

This section is applicable only if you are running JDeveloper and Oracle Application Server on different machines. If you are running both on the same machine, then the files are already on the JDeveloper machine.

Copy the files generated by Rule Author from the Oracle Application Server machine to the JDeveloper machine:

- Copy the `c:\rules\sample_repository` file to the `SOADEMO\SOAOrderBooking\bpel\rules\oracle` directory.

- Copy the files from `c:\rules\com\oracle\demos\orderbooking` to the `SOADEMO\SOAOrderBooking\bpel\rules\com\oracle\demos\orderbooking` directory.

8.11 Determine If an Order Requires Manual Approval ("RequiresManualApproval" Decide Activity)

This decide activity checks if an order requires to be approved by a manager. It determines this by checking the rules in the Oracle Business Rules repository. You created the rules in Section 8.10, "Set up Oracle Business Rules".
Figure 8–130 shows the activities in the "RequiresManualApproval" decide activity.

**Figure 8–130 Activities in the "RequiresManualApproval" Decide Activity**

8.11.1 Create the Activities in the "RequiresManualApproval" Decide Activity

1. In the Component Palette, select Process Activities from the dropdown.
2. Drag the Decide icon from the Component Palette and drop it after the "CreditService" scope. This displays the Edit Decide dialog.
3. In the Edit Decide dialog, enter RequiresManualApproval in the Name field.
4. For the **Decision Service** field, click the wand icon. This launches the Decision Service wizard.

5. In Step 1, Select a Ruleset or Function:
   - **Service Name**: enter **DecisionService** (which is the default name).
   - **Namespace**: enter `http://www.globalcompany.com/ns/OrderBooking/DecisionService` (which is the default).
   - **Invocation Pattern**: select **Execute Ruleset**.
   - **Ruleset**: click the flashlight icon, which displays the Rule Explorer. In the Rule Explorer, you should see `sample_repository`. Expand it, then expand **OrderBookingRules** (which is the name of the dictionary), and select **ApproveOrderRequired** (which is the name of the ruleset).

   ![Rule Explorer](image)

   **Figure 8–131  Rule Explorer**

   Click **OK** in the Rule Explorer.

6. Back in step 1 of the wizard, select **Assert Fact** and **Watch Fact** for **Approve**.

   Check here to assert all descendants from the top level element: do not select.
Click Next.

7. If the wizard complains about missing files, copy the files that it wants to the desired location. This screen is likely to appear if you are running JDeveloper and Oracle Application Server on separate machines and you copied the Oracle Business Rules repository from the Oracle Application Server machine to the JDeveloper machine.

Click Next.


9. Back to the Edit Decide dialog:
   - **Decision Service**: set to DecisionService automatically.
   - **Operation**: select Assert facts, execute rule set, retrieve results.

The Edit Decide dialog now looks like this:
Click OK.

10. Select File > Save to save your work.

JDeveloper creates the following items:

- the `SOAOrderBooking\decisionservices` directory
- the "DecisionServicePL" partner link
- the following activities in the "RequiresManualApproval" decide activity (expand "RequiresManualApproval" to see the activities):
  - Assign activity: "BPEL_Var_To_Rule_Facts"
  - Assign activity: "Facts_To_Rule_Service"
  - Assign activity: "BPEL_Header"
  - Invoke activity: "Invoke"
  - Assign activity: "Rule_Service_To_Facts"
  - Assign activity: "Facts_To_BPEL_Var"

If JDeveloper displays these activities in a somewhat random order, save and close the `SOAOrderBooking.bpmel` file. Then reopen it.

- the following variables in the "RequiresManualApproval" decide activity:
  - `com_oracle_demos_orderbooking_Approve_i`
  - `com_oracle_demos_orderbooking_Approve_o`
  - `dsIn`
  - `dsOut`

To see these variables, click the (x) icon on the left side of the "RequiresManualApproval" decide activity (Figure 8–135). This displays the Variables dialog (Figure 8–136).
Determine If an Order Requires Manual Approval ("RequiresManualApproval" Decide Activity)

**Figure 8–135** The Red Circle Highlights the Variable Icon for the "RequiresManualApproval" Decide Activity

You should see these variables:

**Figure 8–136** Variables Dialog

8.11.2 Copy Order Total and Customer Status Information ("BPEL_Var_To_Rule_Facts" Assign Activity)

The default "BPEL_Var_To_Rule_Facts" assign activity does not contain any operations. You have to define the operations yourself. For the SOA Order Booking application, you define two copy operations:

- Copy the order total price to the com_oracle_demos_orderbooking_Approve_i variable. Copy it to the approve/price element of the variable.
- Copy the status of the customer (retrieved from findCustomerById) to the com_oracle_demos_orderbooking_Approve_i variable. Copy it to the approve/status element of the variable.

1. Double-click the "BPEL_Var_To_Rule_Facts" activity to display the Assign dialog.
2. Click the Copy Operation tab. You will define two copy operations.
3. For the first copy operation, select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.
Determine If an Order Requires Manual Approval ("RequiresManualApproval" Decide Activity)

a. On the To side, set Type to Expression.

b. Click the XPath Expression Builder icon above the Expression box to display the Expression Builder dialog. You will use the Expression Builder to create the expression.


The Content Preview box should show

The ns4 prefix maps to the "http://www.globalcompany.com/ns/order" namespace. It may be different on your system. If you want, you can scroll to the top of your SOAOrderBooking.bpel file (in source view) to verify the prefix.

d. Click Insert Into Expression. The Expression box should show the function with the three parameters.

e. In the Expression box, wrap the number() function around the entire bpws:getVariableData function. The value in the Expression box should look like this:

number(bpws:getVariableData('inputVariable', 'payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:OrderInfo/ns4:OrderPrice'))

---

Figure 8–137  Expression Builder Dialog

The Expression Builder dialog is shown with the expression "number(bpws:getVariableData('inputVariable', 'payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:OrderInfo/ns4:OrderPrice'))" entered in the content preview area.
Determine If an Order Requires Manual Approval ("RequiresManualApproval" Decide Activity)

f. Click OK in the Expression Builder dialog.

g. On the From side, set Type to Variable.


Figure 8–138  Create Copy Operation Dialog

i. Click OK in the Create Copy Operation dialog. This returns you to the Assign dialog.

4. Select Copy Operation from the Create dropdown again to create the second copy operation.

a. On the To side, set Type to Expression.

b. Click the XPath Expression Builder icon above the Expression box to display the Expression Builder dialog. You will use the Expression Builder to create the expression.

c. In the Expression Builder dialog, in the BPEL Variables box, select Variables > Process > Variables > customerServiceResponse > parameters > ns8:findCustomerIdResponse > ns8:return > ns8:status.

The Content Preview box should show
<bpws:getVariableData('customerServiceResponse','parameters','/ns8:findCustomerIdResponse/ns8:return/ns8:status')>.

The ns8 prefix maps to the "http://www.globalcompany.com/ns/customer" namespace. It may be different on your system. If you want, you can scroll to the top of your SOAOrderBooking.bpel file (in source view) to verify the prefix.

d. Click Insert Into Expression. The Expression box should show the function with the three parameters.
e. In the Expression box, wrap the `string()` function around the entire `bpws:getVariableData` function. The value in the Expression box should look like this:

```java
string(bpws:getVariableData('customerServiceResponse','parameters','/ns8:findCustomerByIdResponse/ns8:return/ns8:status'))
```

**Figure 8–139 Expression Builder Dialog**

f. Click OK in the Expression Builder dialog.

f. On the From side, set Type to Variable.

i. Click OK in the Create Copy Operation dialog.

5. Click OK in the Assign dialog.

6. Select File > Save to save your work.

8.11.3 Copy the ConversationId to the dsIn Variable ("BPEL_Header" Assign Activity)

The default "BPEL_Header" assign activity comes with seven copy operations. Add another copy operation to copy the conversation ID to the dsIn variable.

1. Double-click the "BPEL_Header" activity to display the Assign dialog.

2. Click the Copy Operation tab.

3. Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.

a. On the From side, set Type to Expression.

b. Enter the following in the Expression box.

\[
\text{ora:getConversationId()}
\]

c. On the To side, set Type to Variable.

d. Select Variables > Process > Scope - RequiresManualApproval > Variables > dsIn > payload > ns12:assertExecuteWatchStateful > ns12:bpelInstance > ns14:conversationId.
e. Click OK in the Create Copy Operation dialog. This returns you to the Assign dialog.

4. Click OK in the Assign dialog.

5. Select File > Save to save your work.

8.11.4 Create the "requiresApproval" Process Variable

Create the requiresApproval variable in the SOAOrderBooking scope. This variable is created at this level so that any activity in this BPEL flow can access it. This variable is used in the switch that you will create later in Section 8.12.2, "Set the Condition for the <case>".

1. Scroll to the top of the page and double-click the SOAOrderBooking scope. You can double-click the "SOAOrderBooking" text that is sideways. This displays the Process dialog.

2. In the Process dialog, click Create in the Variables tab. This displays the Create Variable dialog.

3. In the Create Variable dialog:
   - **Name**: enter requiresApproval.
   - **Type**: select Simple Type, and click the flashlight icon. This displays the Type Chooser dialog.
     In the Type Chooser, select boolean and click OK.

The Create Variable dialog now looks like this:
Determine If an Order Requires Manual Approval ("RequiresManualApproval" Decide Activity)

4. Click **OK** in the Create Variable dialog. The `requiresApproval` variable appears in the Process dialog.

5. Click **OK** in the Process dialog.

6. Select File > Save to save your work.

8.11.5 Copy the Result of the Decision Service to the `requiresApproval` Variable ("Facts_To_BPEL_Var" Assign Activity)

The default "Facts_To_BPEL_Var" assign activity does not come with any operations. Create a copy operation to copy the result returned by the "DecisionService" partner link to the `requiresApproval` process variable.

1. Double-click the "Facts_To_BPEL_Var" assign activity to display the Assign dialog.

2. Click the Copy Operation tab.

3. Select **Copy Operation** from the Create dropdown. This displays the Create Copy Operation dialog.
4. On the *From* side, set **Type** to **Variable**.


6. On the *To* side, set **Type** to **Variable**.

7. Select **Variables > Process > Variables > requiresApproval**.

---

**Figure 8–144  Create Copy Operation Dialog**

8. Click **OK** in the Create Copy Operation dialog.

9. You should see the copy operation in the Assign dialog. Click **OK**.

---

**Figure 8–145  Assign Dialog**
10. Select File > Save to save your work.

8.11.6 Minimize the "RequiresManualApproval" Decide Activity

Click the [-] icon to minimize the "RequiresManualApproval" decide activity.

8.12 Set Up a Form to Process Orders That Require Manual Approval ("requiresApproval" Switch)

For orders that require manual approval, the "requiresApproval" switch passes control to a human task activity. This enables a manager to approve or reject the orders.

The "requiresApproval" consists of a <case> branch only. It does not have an <otherwise> branch. For orders that do not require manual approval, this switch does not apply to them.

The <case> branch contains a human task activity and another switch activity (Figure 8–146).

Figure 8–146  "requiresApproval" Switch Contains a Human Task and a Switch (Minimized View)

Figure 8–147 shows the human task activity expanded.
Figure 8–148 shows the switch activity expanded.
Figure 8–147  <case> Branch for Switch Activity, Human Task Activity Expanded

Figure 8–148  Second Switch Expanded
8.12.1 Create the Switch

1. In the Component Palette, select **Process Activities** from the dropdown.

2. Drag the Switch icon from the Component Palette and drop it below the "RequiresManualApproval" decide activity.

3. Double-click the new switch activity and set its name to **requiresApproval** in the Switch dialog. Click **OK**.

4. Expand the switch activity.

5. Delete the <otherwise> box. For this switch, you only need to handle the case where an order requires manual approval.

8.12.2 Set the Condition for the <case>

1. Double-click the title bar of the <case> box to display the Switch Case dialog.

2. In the Switch Case dialog, click the XPath Expression Builder icon above the Expression box to display the Expression Builder dialog.

3. In the Expression Builder dialog, select **getVariableData** in the Functions box.

   The Content Preview box should show `bpws:getVariableData()`.

4. Add the first parameter to the function: In the BPEL Variables box, select **Variables > Process > Variables > requiresApproval**.

   `requiresApproval` is a process variable that you defined in Section 8.11.4, "Create the "requiresApproval" Process Variable".

   The Content Preview box should show `bpws:getVariableData('requiresApproval')`.

5. Click **Insert Into Expression**. The Expression box should show the function with the one parameter.

6. Append `='true'` to the expression in the Expression box so that the expression looks like this:

   `bpws:getVariableData('requiresApproval')='true'`

7. In the Expression box, wrap the `string()` function around the entire `bpws:getVariableData` function. The value in the Expression box should look like this:

   `string(bpws:getVariableData('requiresApproval')) = 'true'`
8. Click **OK** in the Expression Builder dialog. The Switch Case dialog now contains the expression:

*Figure 8–150  Switch Case Dialog*

9. Click **OK** in the Switch Case dialog.

10. Select **File > Save** to save your work.
8.12.3 Create a Sequence in the <case> Branch

The <case> branch will contain two activities (a human task activity and a switch activity). This means that you need a sequence activity to be the container for these two activities.

Drag the Sequence icon from the Component Palette and drop it in the <case> box. The <case> box now looks like this:

Figure 8–151  Sequence Activity in the <case> Branch

8.12.4 Create a Human Task

Create a human task activity in the <case> branch.

1. In the Component Palette, select Process Activities from the dropdown.
2. Drag the Human Task icon from the Component Palette and drop it in the Sequence area in the <case> box. This displays the Add a Human Task dialog.
3. In the Add a Human Task dialog, click the Create Task Definition icon (the second icon for the Task Definition field). This displays a different Add a Human Task dialog.
4. In the second Add a Human Task dialog, enter ApproveOrder for the Human Task Name. This sets the location to:  

   SOADEMO\SOAOrderBooking\bpel\ApproveOrder\ApproveOrder.task

Figure 8–152  Add a Human Task Dialog
5. Click OK. JDeveloper closes the Add a Human Task dialog and displays the ApproveOrder.task page.

6. In the ApproveOrder.task page, you can leave the Title empty.

7. For Parameters:
   a. Click the green + icon, which displays the Add Task Parameter dialog.
   b. In the Add Task Parameter dialog, select Element and click the flashlight icon, which displays the Type Chooser dialog. In the Type Chooser dialog, select Type Explorer > Project Schema Files > OrderBookingPO.xsd > PurchaseOrder.

   Figure 8–153 Type Chooser Dialog for ApproveOrder.task Parameter

   Click OK in the Type Chooser dialog. The Add Task Parameter dialog now looks like this:

   Figure 8–154 Add Task Parameter Dialog

   c. Click OK in the Add Task Parameter dialog.

8. For Assignment and Routing Policy:
a. Click the green + icon. This displays the Add Participant Type dialog.

**Figure 8–155 Add Participant Type Dialog**

- Click the green + icon. This displays the Add Participant Type dialog.
- For **Type**, select *Single Approver*.
- For **Label**, enter *Manager*.
- Select **By Name**, and click the flashlight for **Group Id(s)** because you want to give the approve authority to a group. This displays the Identity Lookup dialog.
- In the Identity Lookup dialog, click **Lookup**. This should produce a list of group names in the **Search Group** box.
- Select **Supervisor** in the **Search Group** box and click **Select**. This moves the name to the **Selected Group** box.
Set Up a Form to Process Orders That Require Manual Approval ("requiresApproval" Switch)

Creating the SOAOrderBooking Project

Figure 8–156  "Supervisor" Selected and Moved to the Selected Group Box

9. Click OK in the Identity Lookup dialog. "Supervisor" now appears in the Add Participant Type dialog (see Figure 8–155).

10. Select File > Save to save ApproveOrder.task.
11. Close ApproveOrder.task (select File > Close).

12. Back on the SOAOderBooking-bpel main page, double-click the "ApproveOrder_1" human task activity. This displays the Human Task dialog.

13. For the Task Title field, enter Approve Order.

14. The Task Parameters column is filled in with "PurchaseOrder", but the BPEL Variable column is blank. Click the flashlight icon in the BPEL Variable column.

15. In the Task Parameters dialog, select Variables > Process > Variables > inputVariable > payload > client:SOAOderBookingProcessRequest > ns1:PurchaseOrder.

Figure 8–158  Task Parameters Dialog

Click OK in the Task Parameters dialog.

16. The Human Task dialog now looks like the following:
8.12.5 Create a Form for the Worklist Application

When a manager wants to see the orders that are waiting for approval, the manager logs into the Worklist application. You can access this application from the Start menu of the machine running Oracle Application Server: Start > Programs > Oracle - ORACLE_HOME_NAME > Oracle BPEL Process Manager > Worklist Application.

You need to create a form that the Worklist application can use. This form enables the Worklist application to display data specific to the SOA Order Booking application.

To create this form:

1. In the Application Navigator, right-click the ApproveOrder folder (located under SOAOderBooking > Integration Content) and select Auto Generate Simple Task Form.

2. JDeveloper displays the payload-body.jsp file in the editor. You can close it without making any changes to it.

JDeveloper creates the following directory and files:

- SOADEMO\SOAOderBooking\bpel\ApproveOrder\ApproveOrder_Display.tform
- SOADEMO\SOAOderBooking\public_html

8.12.6 Accept the Default Settings for the Remaining Human Task Activities

Expand the "ApproveOrder_1" human task activity to see the activities contained in this scope (see Figure 8–147). You can just accept the default settings for these activities.
8.13 Handle the Manager’s Response ("requiresApproval" Switch)

JDeveloper created a switch for you automatically after the human task activity. This switch enables you to define the actions to take depending on whether the manager approved or rejected the order, or if the order has expired.

The switch handles these cases:

- The manager rejected the order.
- The manager approved the order.
- The order has expired.

8.13.1 Handle the Reject Case

If the manager rejected the order, you want to perform these two activities:

- set the status item to a string saying that the order has been rejected, and
- create a throw activity

8.13.1.1 Set the Status of the Order (Assign Activity)

1. Expand the switch.
2. For <case Task outcome is REJECT>, delete the default “CopyPayloadFromTask” assign activity. There are now no activities for the REJECT case.
3. In the Component Palette, select Process Activities from the dropdown.
4. Drag the Assign activity icon from the Component Palette and drop it in the REJECT case.
5. Double-click the new assign activity to display the Assign dialog.
6. In the Assign dialog, click the Copy Operation tab.
7. Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.
8. In the From side, set Type to Expression, and enter the following line in the Expression box:
   
   ```
   string('Order has been rejected by the manager')
   ```
10. Click OK in the Create Copy Operation dialog. You should see the copy operation in the Assign dialog.

11. Click OK in the Assign dialog.

**Figure 8–160  Create Copy Operation Dialog for the REJECT Case**

**Figure 8–161  Assign Dialog Showing Copy Operation for the REJECT Case**

### 8.13.1.2 Create a Throw Activity

Create a throw activity to stop the flow from continuing.

1. Drag the Throw icon from the Component Palette and drop it below the previous assign activity.

2. Double-click the new throw activity to display the Throw dialog.
3. In the Throw dialog:
   - **Name**: enter *Throw_1* (keep the default name).
   - **Namespace URI**: enter *http://www.globalcompany.com/ns/OrderBooking*. You have to enter it manually.
   - **Local Part**: enter *OrderBookingFault*. You have to enter it manually.
   - **Fault Variable**: click the flashlight and select *OrderBookingFault*.

![Figure 8–162 Throw Dialog for the REJECT Case](image)

4. Click **OK** in the Throw dialog.

### 8.13.2 Handle the Approve Case

In the Approve case, delete the default assign activity and create an empty activity by dragging the Empty icon from the Component Palette to the APPROVE case. You only need the empty activity because nothing needs to happen here. The flow just needs to continue to the next step.

### 8.13.3 Handle the Expired Case

You handle the Expired, Stale, Withdrawn, or Errored case in the same manner as for the Approve case, that is, delete the default assign activity and replace it with an empty activity. This is acceptable because in the SOA Order Booking application, orders do not expire and users will not encounter this case.

### 8.13.4 Accept the Default Settings for Each Case

JDeveloper also set the conditions for each branch in the switch. To see the conditions, double-click the titlebar on each branch. You do not need to make any changes to the conditions.

### 8.13.5 Minimize the "requiresApproval" Switch

Click the [-] icon on the "requiresApproval" switch to minimize it.
8.14 Choose a Supplier ("SelectSupplier" Scope)

This scope sends the order information to the two suppliers, Select Manufacturer and Rapid Service, and the suppliers return their bids for the orders. The scope then chooses the supplier that provided the lower bid.

Figure 8–163 shows the activities in the "SelectSupplier" scope at a high level. The scope uses a flow activity to send the order information to the two suppliers in parallel. The flow contains two scopes, one for each supplier. After the suppliers have returned their bids, a switch activity selects the lower bid.

Figure 8–164 shows the activities in the flow, and also the switch activity after the flow. The scope for Select Manufacturer includes a receive activity, which is not required for the scope for Rapid Service. This is because Select Manufacturer is invoked asynchronously. The receive activity is needed to receive the bid when Select Manufacturer returns it. Rapid Service is invoked synchronously; the invoke activity invokes the request and receives the response (that is, the bid value).

The switch activity has two branches: the <case> branch handles the case where SelectManufacturer returned the lower bid, and the <otherwise> branch handles the case where Rapid Service returned the lower bid.
Choose a Supplier ("SelectSupplier" Scope)

---

**Figure 8–164  "SelectSupplier" Scope Showing Flow and Switch Activities**

Before you can create the scope, you need to:

- create and deploy the SelectManufacturer and RapidService services, which were described in Chapter 7, “Creating the SelectManufacturer Project” and Chapter 6, "Creating the RapidService Project”.
- create partner links to SelectManufacturer and RapidService.

### 8.14.1 Create the "SelectManufacturer" Partner Link

1. In the Component Palette, select *Services* from the dropdown.

2. Drag the Partner Link icon from the Component Palette and drop it in a *Services* swimlane.

3. In the Create Partner Link dialog:
   - **Name**: enter *SelectService*.
   - **WSDL File**: click the service explorer (flashlight) icon (the second icon from the left). This displays the Service Explorer dialog.
In the Service Explorer dialog, select **Service Explorer > BPEL Services > soaDemoIntgServer > processes > default > SelectManufacturer**.

**soaDemoIntgServer** refers to your connection to the Integration Server where you deployed Select Manufacturer.

**Figure 8–165  Service Explorer Dialog for the “SelectService” Partner Link**

Click **OK** in the Service Explorer dialog.

- **Partner Link Type**: select **SelectService_PL** (automatically filled in for you).
- **Partner Role**: select **SelectServiceProvider**.
- **My Role**: select **SelectServiceRequester**.

The Create Partner Link dialog should look like this:

**Figure 8–166  Create Partner Link Dialog for the "SelectService" Partner Link**

4. Click **OK** in the Create Partner Link dialog.

5. Select File > Save to save your work.
8.14.2 Create the "RapidService" Partner Link

1. Copy the following file from the soademo_101310_prod.zip file to the SOAOrderBooking\bpel directory.
   - RapidService.wsdl
   The WSDL file is located in the SOAOrderBooking\bpel directory in the soademo_101310_prod.zip file.

2. Verify the URL in the RapidService.wsdl file.
   a. Select File > Open and open the RapidService.wsdl file.
   b. Click the Source tab at the bottom of the editor to view the lines in the file.
   c. The http://localhost:8888 reference in the file assumes that Oracle Application Server is running on the same machine as JDeveloper, and that Oracle Application Server is listening for requests on port 8888.
      If necessary, change localhost to the name of the machine running Oracle Application Server, and 8888 to the correct port used by your Oracle Application Server installation, for example: mypc.mydomain.com:8889.
   d. If you edited the file, save the file and close it.

3. Drag the Partner Link icon from the Component Palette and drop it in a partner link area.

4. In the Create Partner Link dialog:
   - **Name**: enter RapidService.
   - **WSDL File**: click the Service Explorer icon (second icon from the left) to display the Service Explorer dialog. In the Service Explorer dialog, expand Project WSDL Files and select RapidService.wsdl.

   ![Service Explorer Dialog for the "RapidService" Partner Link](image)

   Click OK in the Service Explorer.
   - **Partner Link Type**: select RapidQuote_PL (automatically filled in for you).
Choose a Supplier ("SelectSupplier" Scope)

- **Partner Role**: select `RapidQuote_Role`.
- **My Role**: leave it blank.

The Create Partner Link dialog should look like this:

*Figure 8–168 Create Partner Link Dialog for RapidService*

5. Click **OK** in the Create Partner Link dialog.
6. Select **File > Save** to save your work.

### 8.14.3 Create the "SelectSupplier" Scope

1. In the Component Palette, select **Process Activities** from the dropdown.
2. Drag the Scope icon from the Component Palette and drop it below the "requiresApproval" switch activity.
3. Double-click the new scope to display the Scope dialog.
4. In the Scope dialog, in the General tab:
   - **Name**: enter `SelectSupplier`.
   - **Variable Access Serializable**: do not select.
5. Click the Variables tab. You need to create two variables for this scope.
6. In the Variables tab, click **Create** to create the first variable.
7. In the Create Variable dialog:
   - **Name**: enter `selectManufacturerResponse`.
   - Select **Message Type** and click the flashlight icon to display the Type Chooser. In the Type Chooser, select **Type Explorer > Message Types > Partner Links > SelectService > SelectManufacturer > Message Types > RequestQuote_processRequestQuoteResponse**.
Choose a Supplier ("SelectSupplier" Scope)

8. In the Create Variable dialog, the Message Type is set to

9. Click OK in the Create Variable dialog.

10. In the Variables tab, click Create again to create the second variable.

11. In the Create Variable dialog:
   - **Name**: enter `rapidManufacturerResponse`.
   - Select **Message Type** and click the flashlight icon to display the Type Chooser. In the Type Chooser, select **Type Explorer > Message Types > Partner Links > RapidService > RapidService.wsdl > Imported WSDL**.
RequestQuoteSoapHttpPort > Message Types > RequestQuotePortType_POItemsQuoteResponse.

**Figure 8–171  Type Chooser Dialog for the "rapidManufacturerResponse" Variable**

Click OK in the Type Chooser.

12. In the Create Variable dialog, the Message Type is set to [http://www.globalcompany.com/ns/rapidservice]RequestQuotePortType_POItemsQuoteResponse.

**Figure 8–172  Create Variable Dialog for the "rapidManufacturerResponse" Variable**

13. Click OK in the Create Variable dialog.

8.14.4 Create a Flow Activity

You use a flow activity in order to send out requests to more than one supplier at the same time. In the case of this application, you send out requests to two suppliers: Select Manufacturer and Rapid Service.

1. Expand the new "SelectSupplier" scope.
2. Drag the Flow icon from the Component Palette and drop it into the new "SelectSupplier" scope.
3. Double-click the new flow instance and in the Flow dialog, set the name to CallManufacturers.

8.14.5 Set the Activities for Select Manufacturer

To get a quote from Select Manufacturer, you create these activities:

8.14.5.1 Create a Scope for SelectManufacturer

1. Expand the "CallManufacturers" flow. The flow has two parallel areas where you can define activities.
2. Drag the Scope icon from the Component Palette and drop it into the left side of the flow. The left side of the flow handles requests to and responses from Select Manufacturer.

3. Double-click the new scope instance. This displays the Scope dialog.

4. In the Scope dialog, in the General tab:
   - **Name**: enter *GetSelectMfrQuote*.
   - **Variable Access Serializable**: do not select.

5. Click the Variables tab. You need to create a variable for this scope.

6. In the Variables tab, click **Create**.

7. In the Create Variable dialog:
   - **Name**: enter *manufacturerRequest*.
   - Select **Message Type** and click the flashlight icon to display the Type Chooser. In the Type Chooser, select **Type Explorer > Message Types > Partner Links > SelectService > SelectManufacturer > Message Types > RequestQuote_processRequestQuote**.
Click OK in the Type Chooser.

8. In the Create Variable dialog, the Message Type is set to

9. Click OK in the Create Variable dialog.

10. The manufacturerRequest variable appears in the Variables tab of the Scope dialog.
8.14.5.2 Create a Transform Activity ("TransformSelectRequest" Transform Activity)

1. Expand the "GetSelectMfrQuote" scope.
2. Drag the Transform icon from the Component Palette and drop it in the "GetSelectMfrQuote" scope.
3. Double-click the transform activity to display the Transform dialog.
4. In the General tab, set the name to **TransformSelectRequest**.
5. In the Transformation tab, set the **Source Variable** to **inputVariable**, and the **Source Part** to **payload**.
   Set the **Target Variable** to **manufacturerRequest** and the **Target Part** to **parameters**.
6. Set the **Mapper File** to **SelectTransformation.xsl**. This file does not exist yet; you will use the Data Mapper to generate it.
7. Click the Create Mapping icon (second icon from the left). This displays the Data Mapper, which you will use to generate the **SelectTransformation.xsl** file.
8. In the Data Mapper, on the Source (left) side, expand **po:PurchaseOrder > po:OrderItems > po:Item**.
9. On the Target (right) side, expand **tns:param0**.
10. Drag **po:ProductName** to **tns:itemId**.
11. Drag **po:Quantity** to **tns:quantity**. The Data Mapper should look like the following:
Choose a Supplier ("SelectSupplier" Scope)

12. In the Component Palette for the Data Mapper, select **XSLT Constructs** from the dropdown.

13. Drag the for-each icon from the Component Palette and drop it on tns:param0 on the target side. You want the for-each item to appear between tns:processRequestQuoteElement and tns:param0.

14. Drag **po:Item** to the **for-each** item on the target side. The Data Mapper should now look like this:

Figure 8–179  Data Mapper for Select Manufacturer

15. Select File > Save to save SelectTransformation.xsl. This file is created in the SOAOrderBooking\bpel directory.

16. Select File > Close to close the Data Mapper for SelectTransformation.xsl.

17. Select File > Save to save SOAOrderBooking.bpel.

8.14.5.3 Create an Invoke Activity

This invoke activity invokes the SelectManufacturer partner link.

1. Drag the Invoke icon from the Component Palette and drop it below the "TransformSelectRequest" transform activity.

2. Do one of the following to display the Invoke dialog:
Drag one of the arrows on the side of the new invoke activity and drop it on the "SelectService" partner link. This associates the invoke activity with the partner link.

Double-click the new invoke activity.

3. In the Invoke dialog, set the name of the invoke activity to InvokeSelectManufacturer.

4. If the Partner Link field is not set to SelectService, click the flashlight. In the Partner Link Chooser, select SelectService and click OK.

5. For Operation, select processRequestQuote. This should be filled in automatically for you.

6. For Input Variable, click the Browse Variable icon, which displays the Variable Chooser dialog. In the dialog, select Variables > Process > Scope - SelectSupplier > Scope - GetSelectMfrQuote > Variables > manufacturerRequest.
Choose a Supplier ("SelectSupplier" Scope)

Click OK in the Variable Chooser.

7. The Invoke dialog looks like the following:

![Invoke Dialog for the "InvokeSelectManufacturer" Invoke Activity](image)

Click OK in the Invoke dialog.

8.14.5.4 Create a Receive Activity

This receive activity receives the quote from the Select Manufacturer.

1. Drag the Receive icon from the Component Palette and drop it below the "InvokeSelectManufacturer" activity.

2. Do one of the following to display the Receive dialog:
Drag one of the arrows on the side of the new receive activity and drop it on the "SelectService" partner link. This associates the receive activity with the partner link.

- Double-click the new receive activity.

3. For Name, set it to **ReceiveSelectManufacturer**.

4. **Partner Link** should be set to **SelectService**. If not, click the flashlight icon, which displays the Partner Link Chooser. Select **SelectService** and click **OK**.

![Figure 8–183 Partner Link Chooser Dialog for the "ReceiveSelectManufacturer" Receive Activity](image)

5. For **Operation**, set it to **processRequestQuoteResponse**. This should be filled in automatically for you.

6. For **Variable**, click the Browse Variable icon, which displays the Variable Chooser. Select **Variables > Process > Scope - SelectSupplier > Variables** > **selectManufacturerResponse**.
Click OK in the Variable Chooser.

7. For the Create Instance option, do not select it.

8. The Receive dialog looks like Figure 8–185. Click OK.

9. Select File > Save to save your work.

8.14.6 Set the Activities for Rapid Manufacturer

To get a quote from Rapid Manufacturer, you create these activities:
8.14.6.1 Create a Scope for Rapid Manufacturer

You use the right side of the flow activity to define the activities for Rapid Manufacturer:

1. Drag the Scope icon from the Component Palette and drop it in the right side of the flow. You will define activities in this scope to handle requests to and responses from Rapid Manufacturer.

2. Double-click the new scope activity to display the Scope dialog.

3. In the General tab, set the name of the scope to CallRapidManufacturer. Do not select Variable Access Serializable.

4. In the Variables tab, click Create, which displays the Create Variable dialog.

5. In the Create Variable dialog, enter manufacturerRequest as the variable name.

6. Select Message Type and click the flashlight, which displays the Type Chooser. In the Type Chooser, select Type Explorer > Message Types > Partner Links > RapidService > RapidService.wsdl > Imported WSDL > RequestQuoteSoapHttpPort > Message Types > RequestQuotePortType_POItemsQuote.

Click OK in the Type Chooser.

7. In the Create Variable dialog, the Message Type is set to 

\{http://www.globalcompany.com/ns/rapidservice\}RequestQuotePortType_POItemsQuote.
Choose a Supplier ("SelectSupplier" Scope)

**Figure 8–187** Create Variable Dialog for the "CallRapidManufacturer" Scope

8. Click OK in the Create Variable dialog.

9. The `manufacturerRequest` variable appears in the Variables tab of the Scope dialog.

**Figure 8–188** Scope Dialog for the "CallRapidManufacturer" Scope

Click OK in the Scope dialog.

8.14.6.2 Create a Transform Activity ("TransformRapidRequest" Transform Activity)

1. Expand the "CallRapidManufacturer" scope.

2. Drag the Transform icon from the Component Palette and drop it in the "CallRapidManufacturer" scope.

3. Double-click the transform activity to display the Transform dialog.

4. In the General tab, set the name to `TransformRapidRequest`.

5. In the Transformation tab, set the Source Variable to `inputVariable`, and the Source Part to payload.
Set the Target Variable to `manufacturerRequest` and the Target Part to `parameters`.

6. Set the Mapper File to `RapidTransformation.xsl`. This file does not exist yet; you will use the Data Mapper to generate it.

7. Click the Create Mapping icon (second icon from the left). This displays the Data Mapper, which you will use to generate the `RapidTransformation.xsl` file.

8. In the Data Mapper, on the Source (left) side, expand `po:PurchaseOrder > po:OrderItems > po:Item`.

9. On the Target (right) side, expand `tns:items`.

10. Drag `po:ProductName` to `tns:itemId`.

11. Drag `po:Quantity` to `tns:quantity`. The Data Mapper should look like the following:

![Figure 8–189 Data Mapper for Rapid Manufacturer (Not Complete Yet)](image)

12. In the Component Palette for the Data Mapper, select XSLT Constructs from the dropdown.

13. Drag the for-each icon from the Component Palette and drop it on `tns:items` on the target side. You want the for-each item to appear between `tns:POItemsQuote` and `tns:items`.

14. Drag `po:Item` to the for-each item on the target side. The Data Mapper should now look like this:
15. Select File > Save to save RapidTransformation.xsl. This file is created in the SOAOrderBooking\bpel directory.

16. Select File > Close to close the Data Mapper for RapidTransformation.xsl.

17. Select File > Save to save SOAOrderBooking.bpel.

8.14.6.3 Create an Invoke Activity

This invoke activity invokes the RapidService partner link.

1. Drag the Invoke icon from the Component Palette and drop it below the "TransformRapidRequest" transform activity.

2. Do one of the following to display the Invoke dialog:
   - Drag one of the arrows on the side of the new invoke activity and drop it on the "RapidService" partner link. This associates the invoke activity with the partner link.
   - Double-click the new invoke activity.

3. In the Invoke dialog, set the name of the invoke activity to InvokeRapidManufacturer.

4. The Partner Link field should be set to RapidService. If not, click the flashlight and in the Partner Link Chooser, select RapidService and click OK.
5. For **Operation**, select `POItemsQuote`. This should be filled in automatically for you.

6. For **Input Variable**, click the Browse Variable icon, which displays the Variable Chooser dialog. In the dialog, select `Variables > Process > Scope - SelectSupplier` > `Scope - CallRapidManufacturer` > `Variables` > `manufacturerRequest`.

Click **OK** in the Variable Chooser.
7. For **Output Variable**, click the Browse Variable icon, which displays the Variable Chooser dialog. In the dialog, select **Variables > Process > Scope - SelectSupplier > Variables > rapidManufacturerResponse**.

![Variable Chooser Dialog for the Output Variable for the "InvokeRapidManufacturer" Activity](image)

Click **OK** in the Variable Chooser.

8. The Invoke dialog looks like the following:

![Invoke Dialog for the "InvokeRapidManufacturer" Activity](image)

Click **OK** in the Invoke dialog.

9. Select **File > Save** to save your work.
8.14.7 Create a Switch to Pick the Lower-Priced Quote

Create a switch so that you can pick the manufacturer that responded with the lower-priced quote.

8.14.7.1 Create the Switch

1. Minimize the flow activity in the "SelectSupplier" scope. (But do not minimize the scope.) The page should look like this:

![Minimize Only the Flow Activity](image)

2. Drag the Switch icon from the Component Palette and drop it below the "CallManufacturers" flow activity, but still within the "SelectSupplier" scope.

3. Double-click the switch to display the Switch dialog.

4. In the Switch dialog, set the name to SelectByPrice and click OK.

8.14.7.2 Set the Condition for the Switch

For this switch activity, you just need to specify the condition for the <case> branch. For the <otherwise> branch, you do not have to set a condition. If the condition in the <case> branch is not met, then the activities in the <otherwise> branch are executed.

1. Expand the switch activity.

2. Double-click the titlebar of the <case> box to display the Switch Case dialog.

3. The <case> branch handles the case where Select Manufacturer returns a lower price than Rapid Service. In the Expression area, enter the following line:

   number(bpws:getVariableData('selectManufacturerResponse', 'parameters', '/ns15:processRequestQuoteResponseElement/ns15:return/ns15:supplierPrice')) < number(bpws:getVariableData('rapidManufacturerResponse', 'parameters', '/ns16:POItemsQuoteResponse/ns30:return/ns30:supplierPrice'))

You may need to replace the prefixes for the namespaces, as follows:

- ns15 is the prefix for http://www.globalcompany.com/ns/selectservice.

- ns16 is the prefix for http://www.globalcompany.com/ns/rapidservice.
Choose a Supplier ("SelectSupplier" Scope)

- **ns30** is the prefix for http://rapidservice.soademo.org/types/.

You can also use the Expression Builder to create the condition:

a. In the Switch Case dialog, click the XPath Expression Builder icon above the Expression box.

b. In the Expression Builder dialog, select **Variables > Process > Scope - SelectSupplier > Variables > selectManufacturerResponse > parameters > ns15:processRequestQuoteResponseElement > ns15:return > ns15:supplierPrice**.

The Content Preview box in the dialog shows the bpws:getVariableData function with parameters to get the supplierPrice data.

c. Click **Insert Into Expression**.

The Expression Builder dialog should now look like this:

**Figure 8–196 Expression Builder Dialog**

![Expression Builder Dialog](image)

- **d.** In the Expression Builder dialog, in the Expression box, type a `<` (less than) character. **Figure 8–197** shows the Expression box in the Expression Builder dialog.
e. In the Expression Builder dialog, select Variables > Process > Scope - SelectSupplier > Variables > rapidManufacturerResponse > parameters > ns16:POItemsQuoteResponse > ns16:return > ns16:supplierPrice.

f. Click Insert Into Expression.
   The Expression Builder dialog should now look like this:

Figure 8–198 Expression Builder Dialog Showing Both Parts of the Condition

The final expression should look like this:

```
number(bpws:getVariableData(....)) < number(bpws:getVariableData(....))
```

You have to manually type in `number` before the `bpws:getVariableData` function and the closing parenthesis ) after the function.

The final expression should look like this:
Choose a Supplier ("SelectSupplier" Scope)

8.14.7.3 Set the Activities for Select Manufacturer

Define the activities for Select Manufacturer if it bid the lower quote. In this case, assign the bid value to the **supplierPrice** field of **inputVariable**.

1. Drag the Assign activity icon from the Component Palette and drop it in the <case> box.

2. Double-click the new assign activity to display the Assign dialog.

3. In the Assign dialog, click the General tab, and set the **Name** to **AssignSelectManufacturer**.

4. Still in the Assign dialog, click the Copy Operation tab. You will create two copy operations.

5. Select **Copy Operation** from the Create dropdown. This displays the Create Copy Operation dialog.
   
   - In the From side, set **Type** to **Variable**, and select **Variables > Process > Scope - SelectSupplier > Variables > selectManufacturerResponse > parameters > ns15:processRequestQuoteResponseElement > ns15:return > ns15:supplierPrice**.

**Figure 8–199 Expression Builder Dialog Showing the Final Expression**

- Click OK in the Expression Builder dialog.

**Figure 8–200 Switch Case Dialog for <case>**

- Click OK in the Switch Case dialog.
In the **To** side, set **Type** to **Variable**, and select **Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:SupplierInfo > ns1:SupplierPrice**.

**Figure 8–201  Create Copy Operation Dialog for the "AssignSelectManufacturer" Activity, First Copy Operation**

Click **OK** in the Create Copy Operation dialog.

6. Select **Copy Operation** from the **Create** dropdown again to create the second copy operation. This displays the Create Copy Operation dialog.

   - In the **From** side, set **Type** to **Variable**, and select **Variables > Process > Scope > SelectSupplier > Variables > selectManufacturerResponse > parameters > ns15:processRequestQuoteResponseElement > ns15:return > ns15:supplierName**.

   - In the **To** side, set **Type** to **Variable**, and select **Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:SupplierInfo > ns1:SupplierName**.
Choose a Supplier ("SelectSupplier" Scope)

Figure 8–202 Create Copy Operation Dialog for the "AssignSelectManufacturer" Activity, Second Copy Operation

Click OK in the Create Copy Operation dialog.

7. You should see two copy operations in the Assign dialog. Click OK.

Figure 8–203 Assign Dialog for the "AssignSelectManufacturer" Activity

8.14.7.4 Set the Activities for Rapid Distributor
Define the activities for Rapid Manufacturer if it bid the lower quote. In this case, assign the bid value to the supplierPrice field of inputVariable.

1. Drag the Assign activity icon from the Component Palette and drop it in the <otherwise> box.

2. Double-click the new assign activity to display the Assign dialog.
3. In the Assign dialog, click the General tab, and set the Name to AssignRapidManufacturer.

4. Still in the Assign dialog, click the Copy Operation tab. You will create two copy operations.

5. Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.

   ■ In the From side, set Type to Variable, and select Variables > Process > Scope - SelectSupplier > Variables > rapidManufacturerResponse > parameters > ns16:POItemsQuoteResponse > ns16:return > ns16:supplierPrice.

   ■ In the To side, set Type to Variable, and select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:SupplierInfo > ns1:SupplierPrice.

*Figure 8–204  Create Copy Operation Dialog for the “AssignRapidManufacturer” Activity, First Copy Operation*

Click OK in the Create Copy Operation dialog.

6. Select Copy Operation from the Create dropdown again to create the second copy operation. This displays the Copy Operation dialog.

   ■ In the From side, set Type to Variable, and select Variables > Process > Scope - SelectSupplier > Variables > rapidManufacturerResponse > parameters > ns16:POItemsQuoteResponse > ns16:return > ns16:supplierPrice.

   ■ In the To side, set Type to Variable, and select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:SupplierInfo > ns1:SupplierPrice.
Choose a Supplier ("SelectSupplier" Scope)

Choose a Supplier ("SelectSupplier" Scope)

8.1.4 Minimize the "SelectSupplier" Scope

Click the [-] icon on the "SelectSupplier" scope to minimize it.
8.15 Determine the Shipping Method ("PostFulfillmentReq" Scope)

This scope invokes the FulfillmentESB project, which determines how an order is to be shipped. For orders $500 and over, Fedex is the shipment method. For orders less than $500, USPS is the shipment method. These rules are defined in the FulfillmentESB project, described in Chapter 4, "Creating the FulfillmentESB Project".

Before you can create the "PostFulfillmentReq" scope, the "FulfillmentESB" project must be registered with Oracle Application Server.

Figure 8–207 shows the activities in the "PostFulfillmentReq" scope.

![Figure 8–207 Activities in the "PostFulfillmentReq" Scope]

8.15.1 Create the "OrderFulfillment" Partner Link

1. In the Component Palette, select Services from the dropdown.
2. Drag the Partner Link icon from the Component Palette and drop it in a Services swimlane. This displays the Create Partner Link dialog.
3. For Name, enter OrderFulfillment.
4. For WSDL File, click the Service Explorer icon (second icon from the left) to display the Service Explorer dialog. In the Service Explorer dialog, select Service Explorer > Registered ESB Services > soademoIntegServer > Fulfillment > OrderFulfillment. soademoIntegServer refers to the name of the Integration Server connection.
Click **OK** in the Service Explorer.

5. Back in the Create Partner Link dialog, if you get a `java.net.UnknownHostException` error:
   - Check that the proxy settings are set correctly. To view the proxy settings in JDeveloper, select Tools > Preferences. In the Preferences dialog, select **Web Browser and Proxy** on the left side.
   - If the proxy settings are correct, replace the IP address in the URL with the hostname. For example:
     
     ```
     ```
     
     Press the Tab key after you have entered the hostname.

6. For **Partner Link Type**, select **execute_pptLT**. This should be filled in for you automatically.

7. For **Partner Role**, select **execute_pptProvider**.

8. For **My Role**, leave blank.

   The Create Partner Link dialog should look like this:
8.15.2 Create the "PostFulfillmentReq" Scope

1. In the Component Palette, select Process Activities from the dropdown.
2. Drag the Scope icon from the Component Palette and drop it below the "SelectSupplier" scope.
3. Double-click the new scope to display the Scope dialog.
4. In the General tab:
   - Name: enter PostFulfillmentReq.
   - Variable Access Serializable: do not select.
5. Click the Variables tab. You need to create one variable for this scope.
6. In the Variables tab, click Create.
7. In the Create Variable dialog:
   - Name: enter orderFulfillmentRequest.
   - Select Message Type and click the flashlight icon to display the Type Chooser. In the Type Chooser, select Type Explorer > Message Types > Partner Links > OrderFulfillment > Imported WSDL > Fulfillment_OrderFulfillment.wsdl > Message Types > PurchaseOrder_request.
Determine the Shipping Method ("PostFulfillmentReq" Scope)

Click **OK** in the Type Chooser.

8. In the Create Variable dialog, the Message Type is set to `{http://www.globalcompany.com/ns/Fulfillment}PurchaseOrder_request`.

9. Click **OK** in the Create Variable dialog.

10. The `orderFulfillmentRequest` variable appears in the Variables tab of the Scope dialog.
Determine the Shipping Method ("PostFulfillmentReq" Scope)

Creating the SOAOrderBooking Project

8.15.3 Copy Order Information to Scope Variable ("initializeRequest" Assign Activity)

1. Expand the "PostFulfillmentReq" scope.
2. Drag the Assign activity icon from the Component Palette and drop it in the "PostFulfillmentReq" scope.
3. Double-click the new assign activity to display the Assign dialog.
4. In the Assign dialog, click the General tab, and set the Name to initializeRequest.
5. Still in the Assign dialog, click the Copy Operation tab. You will create a copy operation.
6. Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.
   - In the From side, set Type to Variable, and select Variables > Process > Variables > inputVariable > payload > client:SOAOderBookingProcessRequest > ns1:PurchaseOrder.
   - In the To side, set Type to Variable, and select Variables > Process > Scope - PostFulfillmentReq > Variables > orderFulfillmentRequest > PurchaseOrder > ns1:PurchaseOrder.

Click OK in the Scope dialog.

11. Select File > Save to save your work.
Determine the Shipping Method ("PostFulfillmentReq" Scope)

Figure 8–213  Create Copy Operation Dialog for the "initializeRequest" Activity

Click OK in the Create Copy Operation dialog.

7. You should see the copy operation in the Assign dialog. Click OK.

Figure 8–214  Assign Dialog for the "initializeRequest" Activity

***********************************************************************************************

8.15.4 Invoke OrderFulfillmentESB ("PostFulfillmentReq" Invoke Activity)

1. Drag the Invoke icon from the Component Palette and drop it below the "initializeRequest" assign activity.

2. Do one of the following to display the Invoke dialog:
Determine the Shipping Method ("PostFulfillmentReq" Scope)

- Drag one of the arrows on the side of the new invoke activity and drop it on the "OrderFulfillment" partner link. This associates the invoke activity with the partner link.
- Double-click the new invoke activity.

3. In the Invoke dialog, set these values:
   - **Name**: enter `PostFulfillmentReq`.
   - **Partner Link**: should be set to `OrderFulfillment`. If not, click the flashlight and select `OrderFulfillment` from the Partner Link Chooser.

   **Figure 8–215  Partner Link Chooser Dialog for the "PostFulfillmentReq" Invoke Activity**

   ![Partner Link Chooser Dialog](image)

   Click **OK** in the Partner Link Chooser.
   - **Operation**: select `execute` (should be filled in for you automatically).
   - **Input Variable**: click the Browse Variables icon (the second icon from the left) and select `Variables > Process > Scope - PostFulfillmentReq > Variables > orderFulfillmentRequest`. 
Determine the Shipping Method ("PostFulfillmentReq" Scope)

Figure 8–216 Variable Chooser Dialog for the Input Variable for the "PostFulfillmentReq" Invoke Activity

Click OK in the Variable Chooser.

The Invoke dialog should look like this:

Figure 8–217 Invoke Dialog for the "PostFulfillmentReq" Invoke Activity

4. Click OK in the Invoke dialog.
5. Select File > Save to save your work.

8.15.5 Create a Catch-All Branch for the Scope

Create a catch-all branch to catch all exceptions.

1. Click the "Add CatchAll Branch" icon on the side of the "PostFulfillmentReq" scope.
Set the Order Status to "Completed" ("SetFinalOrderStatus" Scope)

2. Expand the catch-all branch.
3. Drag the Empty icon from the Component Palette and drop it in the catchall branch. The catch just catches the exceptions without processing them.
4. Select File > Save to save your work.

8.15.6 Minimize the "PostFulfillmentReq" Scope

Click the [-] on the "PostFulfillmentReq" scope to minimize it.

8.16 Set the Order Status to "Completed" ("SetFinalOrderStatus" Scope)

This scope uses a database adapter to update the order status in the database.

Figure 8–220 shows the activities in the scope.
8.16.1 Create the "OrderStatus" Database Adapter

1. In the Component Palette, select Services from the dropdown.
2. Drag the Database Adapter icon from the Component Palette and drop it in a Services swimlane. This starts the Adapter Configuration Wizard. Click Next to start.
3. In Step 1, Service Name, set the Service Name to OrderStatus. Leave Description blank.

4. In Step 2, Service Connection, for Connection, select the name of the database connection. The JNDI Name should be set automatically to eis/DB/soademo.
Set the Order Status to “Completed” ("SetFinalOrderStatus" Scope)

Figure 8–222  Adapter Configuration Wizard: Step 2, Service Connection

5. In Step 3, Operation Type, select **Perform an Operation on a Table** and select **Update Only**.

Figure 8–223  Adapter Configuration Wizard: Step 3, Operation Type

6. In Select Table, click **Import Tables**. This displays the Import Tables dialog.

7. In the Import Tables dialog:
   - **Schema**: select SOADEMO.
   - **Name Filter**: enter \%.

Click Query.
Set the Order Status to "Completed" ("SetFinalOrderStatus" Scope)

Select the ORDERS table on the left side and click the right arrow to move it to the Selected box.

**Figure 8–224  Adapter Configuration Wizard: Import Tables**

Click **OK** in the Import Tables dialog.

8. Back in the Select Table page, select **SOADEMO.ORDERS**.

**Figure 8–225  Adapter Configuration Wizard: Step 4, Select Table**

Click **Finish** to accept the defaults for the remaining screens.

The wizard creates the following files:

- **SOAOrderBooking\bpel\OrderStatus.wsdl**
- **SOAOrderBooking\bpel\OrderStatus_table.xsd**
Set the Order Status to “Completed” (“SetFinalOrderStatus” Scope)

9. Oracle JDeveloper now displays the Partner Link dialog with some fields already filled in for you:
   Name: OrderStatus
   WSDL File: SOAOrderBooking/bpel/OrderStatus.wsdl
   Partner Link Type: OrderStatus_plt
   Partner Role: OrderStatus_role

   **Figure 8–226 Create Partner Link Dialog for the "OrderStatus" Database Adapter**

   ![Create Partner Link Dialog](image)

   Click OK.

10. Select File > Save to save your work.

---

8.16.2 Create the "SetFinalOrderStatus" Scope

1. In the Component Palette, select Process Activities from the dropdown.
2. Drag the Scope icon from the Component Palette and drop it below the “PostFulfillmentReq” scope.
3. Double-click the new scope to display the Scope dialog.
4. In the General tab:
   - Name: enter SetFinalOrderStatus.
   - Variable Access Serializable: do not select.
5. Click the Variables tab. You need to create one variable for this scope.
6. In the Variables tab, click Create.
7. In the Create Variable dialog:
   - Name: enter orderStatusRequest.
   - Select Message Type and click the flashlight icon to display the Type Chooser.

   In the Type Chooser, select **Type Explorer > Message Types > Partner Links > OrderStatus > OrderStatus.wsdl > Message Types > OrdersCollection_msg.**
Set the Order Status to "Completed" ("SetFinalOrderStatus" Scope)

8. In the Create Variable dialog, the Message Type is set to

9. Click OK in the Create Variable dialog.

10. The orderStatusRequest variable appears in the Variables tab of the Scope
dialog.
Set the Order Status to "Completed" ("SetFinalOrderStatus" Scope)

8.16.3 Prepare the Order ID and Status ("AssignOrderStatus" Assign Activity)

1. Expand the "SetFinalOrderStatus" scope.
2. Drag the Assign activity icon from the Component Palette and drop it in the "SetFinalOrderStatus" scope.
3. Double-click the new assign activity to display the Assign dialog.
4. In the Assign dialog, click the General tab, and set the Name to AssignOrderStatus.
5. Still in the Assign dialog, click the Copy Operation tab. You will create two copy operations.
6. Create the first copy operation: Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.
   - In the From side, set Type to Variable, and select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:ID.
   - In the To side, set Type to Variable, and select Variables > Process > Scope - SetFinalOrderStatus > Variables > orderStatusRequest > OrdersCollection > ns19:OrdersCollection > ns19:Orders > ns19:ordid.
Click **OK** in the Create Copy Operation dialog.

7. Create the second copy operation: Select **Copy Operation** from the Create dropdown again. This displays the Create Copy Operation dialog.

   - In the **From** side, set **Type** to **Expression**, and enter the following line in the Expression box:
     
     ```
     string('completed')
     ```
   
   - In the **To** side, set **Type** to **Variable**, and select **Variables > Process > Scope - SetFinalOrderStatus > Variables > orderStatusRequest > OrdersCollection > ns19:OrdersCollection > ns19:Orders > ns19:status**.
Set the Order Status to "Completed" ("SetFinalOrderStatus" Scope)

**Figure 8–231  Create Copy Operation Dialog for the "AssignOrderStatus" Activity, Second Copy Operation**

Click **OK** in the Create Copy Operation dialog.

8. You should see the copy operations in the Assign dialog. Click **OK**.

**Figure 8–232  Assign Dialog for the “AssignOrderStatus” Activity**

**8.16.4 Update the Order Status in the Database (“UpdateOrderStatus” Invoke Activity)**

Create the "UpdateOrderStatus" invoke activity to update the order status in the ORDERS table in the database.

1. Drag the Invoke icon from the Component Palette and drop it below the "AssignOrderStatus" assign activity.

2. Do one of the following to display the Invoke dialog:
Set the Order Status to "Completed" ("SetFinalOrderStatus" Scope)

- Drag one of the arrows on the side of the new invoke activity and drop it on the "OrderStatus" database adapter. This associates the invoke activity with the database adapter.
- Double-click the new invoke activity.
- In the Invoke dialog, set these values:
  - **Name:** enter `UpdateOrderStatus`.
  - **Partner Link:** should be set to `OrderStatus`. If not, click the flashlight and select `OrderStatus` from the Partner Link Chooser.

![Partner Link Chooser Dialog for the "UpdateOrderStatus" Invoke Activity](image)

Click **OK** in the Partner Link Chooser.

- **Operation:** select `update`.
- **Input Variable:** click the Browse Variables icon (the second icon from the left) and select `Variables > Process > Scope - SetFinalOrderStatus > Variables > orderStatusRequest`. 
Set the Order Status to “Completed” ("SetFinalOrderStatus" Scope)

Figure 8–234  Variable Chooser Dialog for the Input Variable in the "UpdateOrderStatus" Invoke Activity

Click OK in the Variable Chooser.

The Invoke dialog should look like this:

Figure 8–235  Invoke Dialog for the "UpdateOrderStatus" Invoke Activity

4. Click OK in the Invoke dialog.
5. Select File > Save to save your work.

8.16.5 Minimize the "SetFinalOrderStatus" Scope

Click the [-] on the "SetFinalOrderStatus" scope to minimize it.
8.17 Send an Email Notification to the Customer ("NotifyCustomer" Scope)

This scope uses the notification service to send an email to the customer when the order is fulfilled.

Figure 8–236 shows the activities in the "NotifyCustomer" scope.

8.17.1 Create the Notification Scope

1. Drag the Email icon from the Component Palette and drop it below the "SetFinalOrderStatus" scope. This displays the Edit Email dialog.

2. In the Edit Email dialog, enter any information you want for the email fields. Figure 8–237 shows an example.

For most of the fields, you can click the XPath Expression Builder icon to display the Expression Builder. This enables you to customize the data for the field.
Example 1: Entering the customer’s email address in the **To** field:

a. In the **To** field, click the XPath Expression Builder icon (the icon on the right).

b. In the Expression Builder dialog, select **Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns4:PurchaseOrder > ns4:UserContact > ns4:EmailAddress**.

c. Click **Insert Into Expression**. The dialog should look like this:

![Expression Builder Dialog Showing the EmailAddress Field](image)

Figure 8–238 Expression Builder Dialog Showing the EmailAddress Field

d. Click **OK**.

Example 2: Including the customer name, order ID, and order status in the **Body** of the email. The text that you want to enter looks like this:

Dear **firstName**

This is to inform you that your order number, **orderID**, has been **orderStatus**
If you have a question about your order, please contact customer service or send an email to customerservice@globalcompany.com

Thank you for doing business with Global Company. We appreciate your business!

**Global Company Customer Service**

**firstName**, **orderID**, and **orderStatus** are placeholders for data that are to be retrieved dynamically.

a. Click the XPath Expression Builder icon for **Body**.

b. You will use the **concat** function to concatenate the pieces of the body text.
Start by entering the following line in the Expression box:

```
concat(string('Dear '), )
```

c. Place the insertion point before the last closing parenthesis.


e. Click Insert Into Expression. The expression now looks like this:

```
concat(string('Dear '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ShipTo/ns4:Name/ns4:First'))
```

You cannot insert line breaks in the Expression box but you can insert line breaks in the Edit Email dialog.

g. Place the insertion point before the last closing parenthesis.

h. Add the variable for the order ID: select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns4:PurchaseOrder > ns4:ID.

i. Click Insert Into Expression. The expression now looks like this:

```
concat(string('Dear '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ShipTo/ns4:Name/ns4:First'), string('This is to inform you that your order number '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ID'))
```

j. Add the string "has been " to the message. The new part is shown in bold.

```
concat(string('Dear '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ShipTo/ns4:Name/ns4:First'), string('This is to inform you that your order number '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ID'), string('has been '), )
```

k. Place the insertion point before the last closing parenthesis.

l. Add the variable for the order status: select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns4:OrderInfo > ns4:OrderStatus.

m. Click Insert Into Expression. The expression now looks like this:

```
concat(string('Dear '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ShipTo/ns4:Name/ns4:First'), string('This is to inform you that your order number '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ID'))
```
Send an Email Notification to the Customer ("NotifyCustomer" Scope)

n. Add the final part of the message, shown in bold:

```
concat(string('Dear '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ShipTo/ns4:Name/ns4:First'), string('This is to inform you that your order number '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:ID')
, string('has been '), bpws:getVariableData('inputVariable','payload', '/client:SOAOrderBookingProcessRequest/ns4:PurchaseOrder/ns4:OrderInfo/ns4:OrderStatus'), string('If you have a question about your order, please contact customer service or send an email to customerservice@globalcompany.com Thank you for doing business with Global Company. We appreciate your business!
Global Company Customer Service'))
```

o. Click OK in the Expression Builder. The Edit Email dialog now looks like the following. Line breaks have been added to the body text to make it easier to read.

![Figure 8–239 Edit Email Dialog](image)

3. Click OK in the Edit Email dialog.

4. Select File > Save to save your work.

If you expand the "Email_1" activity, you can see that it contains two activities: an assign activity and an invoke activity. You do not have to edit these activities.

The assign activity, "EmailParamsAssign", contains the information that you provided in the Edit Email dialog.

The invoke activity, "InvokeNotificationService", calls on the notification service to send the emails.
8.17.2 Minimize the "Email_1" Scope
   Click the [-] on the "Email_1" scope to minimize it.

8.17.3 Rename the "Email_1" Scope
   To rename the "Email_1" scope, right-click the minimized scope and select **Rename**. Then type the new name, **NotifyCustomer**, and press Return.

8.18 Call Back the Client ("callbackClient" Invoke Activity)
   This invoke activity returns status to the client.

!["callbackClient" Invoke Activity](image)

8.18.1 Create the Invoke Activity
   1. Drag the Invoke icon from the Component Palette and drop it below the "NotifyCustomer" (which is the "Email_1" scope) scope.
   2. Do one of the following to display the Invoke dialog:
      - Drag one of the arrows on the side of the new invoke activity and drop it on the "client" partner link. This associates the invoke activity with the partner link.
      - Double-click the new invoke activity.
   3. In the Invoke dialog, set these values:
      - **Name**: enter **callbackClient**.
      - **Partner Link**: should be set to **client**. If not, click the flashlight and select **client** from the Partner Link Chooser.
Click **OK** in the Partner Link Chooser.

- **Operation**: select **onResult** (this should be filled in automatically for you).
- **Input Variable**: click the Browse Variables icon (the second icon from the left) and select **outputVariable**.

Click **OK** in the Variable Chooser.

The Invoke dialog should look like this:
8.18.2 Add a Sensor

You can add a sensor to the "callbackClient" invoke activity to write information for completed orders to different destination types such as databases, files, or JMS.

Recall that you also created a sensor at the beginning of the project, for the "receiveInput" receive activity described in Section 8.6, ‘Receive Input from the Client (Receive Activity)’.

1. Double-click the "callbackClient" invoke activity. This displays the Invoke dialog.
2. Click the Sensors tab in the Invoke dialog.
3. Click Create to create a new sensor. This displays the Create Activity Sensor dialog.
4. In the Create Activity Sensor dialog, set the Name to InstanceCompleted.
5. Set the Evaluation Time to Completion. This specifies when the sensor fires. Completion signifies that the sensor fires after this activity has run.
6. In the Activity Variable Sensors section, click Create to display the Create Activity Variable Sensor dialog.

7. In the Create Activity Variable Sensor dialog, click the pencil icon for Variable XPath. This displays the Variable XPath Builder dialog.
Call Back the Client ("callbackClient" Invoke Activity)

Creating the SOAOrderBooking Project

8. Select Variables > Process > Variables > inputVariable.

9. Click OK in the Variable XPath Builder dialog. The Create Activity Variable Sensor dialog should be filled in with these values for you (see Figure 8–244):

   Variable XPath: $inputVariable
   Output Namespace: http://www.globalcompany.com/ns/OrderBooking
   Output Datatype: SOAOrderBookingRequestMessage

10. Click OK in the Create Activity Variable Sensor dialog. This takes you back to the Create Activity Sensor dialog, which now looks like this.
11. In the Create Activity Sensor dialog, click the Add icon in the Sensor Actions section. This displays the Sensor Action Chooser dialog. It shows an existing sensor called InstanceStart, which you created in Section 8.6.2, "Create a Sensor for the Receive Activity".

12. In the Sensor Action Chooser dialog, select Sensor Actions and select Sensor Action from the wand icon. This displays the Create Sensor Action dialog.

13. In the Create Sensor Action dialog:
   - Name: enter InstanceCompleted.
   - Publish Type: select JMS Topic.
   - JMS Connection Factory: enter jms/TopicConnectionFactory.
   - Publish Target: enter jms/demoTopic.
Call Back the Client ("callbackClient" Invoke Activity)

- **Filter**: leave blank.
- **Enable**: select this option.

**Figure 8–248 Create Sensor Action Dialog**

14. Click **OK** in the Create Sensor Action dialog. The Sensor Action Chooser dialog now shows the **InstanceCompleted** sensor action.

**Figure 8–249 Sensor Action Chooser Dialog**

15. Click **OK** in the Sensor Action Chooser. This takes you back to the Create Activity Sensor dialog, which now looks like this:
16. Click OK in the Create Activity Sensor dialog.

In the Invoke dialog, the Sensors tab now looks like this:

17. Click OK in the Invoke dialog.

18. Select File > Save to save your work.

8.19 Add a Catch Branch to the Project

Add a catch branch to the project as a whole so that you can update the order status in the database in case an error occurs anywhere in the project.
Figure 8–252 shows the activity in the catch.

Figure 8–252 Activities in the "client:OrderBookingFault" Catch

8.19.1 Add a "client:OrderBookingFault" Catch

1. Click the triangular icon with an exclamation point at the "SOAOrderBooking" scope to add a catch branch to the project. Adding it at this level enables all activities in the project to use this catch.
2. Scroll to the right to see the new catch branch, and expand it. The new branch does not have any activity in it.

3. Double-click the catch icon to display the Catch dialog.

4. In the Catch dialog:
   - **Namespace URI**: enter `http://www.globalcompany.com/ns/OrderBooking`.
   - **Local Part**: enter `OrderBookingFault`.
   - **Fault Variable**: enter `OrderBookingFault`.
5. Click **OK** in the Catch dialog.

8.19.2 Create a Sequence

1. Expand the catch area, if you have not already done so.
2. Drag the Sequence icon from the Component Palette and drop it in the catch area. This creates a sequence in the catch area.

8.19.3 Create a Scope

Create a scope in the catch area because you need to define a variable for the activities in the catch area.

1. Drag the Scope icon from the Component Palette and drop it in the catch area.
2. Double-click the new scope to display the Scope dialog.
3. In the Scope dialog, in the General tab:
   - **Name**: enter **SetOrderStatus**.
   - **Variable Access Serializable**: do not select.
4. Click the Variables tab. You need to create a variable for this scope.
5. In the Variables tab, click **Create**.
6. In the Create Variable dialog:
   - **Name**: enter **orderStatusRequest**.
   - Select **Message Type** and click the flashlight icon to display the Type Chooser. In the Type Chooser, select **Type Explorer > Message Types > Partner Links > OrderStatus > OrderStatus.wsdl > Message Types > OrdersCollection_msg**.
Click **OK** in the Type Chooser.

7. In the Create Variable dialog, the Message Type is set to
   `{http://xmlns.oracle.com/pcbpe1/adapter/db/OrderStatus/}Order
   sCollection_msg`.

8. Click **OK** in the Create Variable dialog.

9. The `orderStatusRequest` variable appears in the Variables tab of the Scope dialog.
Click OK in the Scope dialog.

8.19.4 Create the Assign Activity

1. Expand the “SetOrderStatus” scope.
2. Drag the Assign activity icon from the Component Palette and drop it in the “SetOrderStatus” scope.
3. Double-click the new assign activity to display the Assign dialog.
4. In the Assign dialog, click the General tab, and set the Name to AssignOrderStatus.
5. Still in the Assign dialog, click the Copy Operation tab. You will create three copy operations.
6. Create the first copy operation: Select Copy Operation from the Create dropdown. This displays the Create Copy Operation dialog.
   - In the From side, set Type to Variable, and select Variables > Process > Variables > inputVariable > payload > client:SOAOrderBookingProcessRequest > ns1:PurchaseOrder > ns1:ID.
   - In the To side, set Type to Variable, and select Variables > Process > Scope - SetOrderStatus > Variables > orderStatusRequest > OrdersCollection > ns19:OrdersCollection > ns19:Orders > ns19:ordid.
Click OK in the Create Copy Operation dialog.

7. Create the second copy operation: Select Copy Operation from the Create dropdown again. This displays the Create Copy Operation dialog.
   - In the From side, set Type to Expression, and enter the following line in the Expression box:
     ```
     string('canceled')
     ```
   - In the To side, set Type to Variable, and select Variables > Process > Scope - SetOrderStatus > Variables > orderStatusRequest > OrdersCollection > ns19:OrdersCollection > ns19:Orders > ns19:status.
Click OK in the Create Copy Operation dialog.

8. Create the third copy operation: Select Copy Operation from the Create dropdown again. This displays the Create Copy Operation dialog.

   - In the From side, set Type to Variable, and select Variables > Process > Variables > OrderBookingFault > payload > client:SOAOrderBookingProcessFault > client:status.

   - In the To side, set Type to Variable, and select Variables > Process > Scope - SetOrderStatus > Variables > orderStatusRequest > OrdersCollection > ns19:OrdersCollection > ns19:Orders > ns19:comments.
Click OK in the Create Copy Operation dialog.

9. You should see the three copy operations in the Assign dialog. Click OK.

10. Select File > Save to save your work.

8.19.5 Create the Invoke Activity

1. Drag the Invoke icon from the Component Palette and drop it below the "AssignOrderStatus" assign activity in the catch area.

2. Do one of the following to display the Invoke dialog:
Drag one of the arrows on the side of the new invoke activity and drop it on the "OrderStatus" database adapter. This associates the invoke activity with the database adapter.

Double-click the new invoke activity.

3. In the Invoke dialog, set these values:
   - **Name**: enter `SetFaultedOrderStatus`.
   - **Partner Link**: should be set to `OrderStatus`. If not, click the flashlight and select `OrderStatus` from the Partner Link Chooser.

   **Figure 8–263** Partner Link Chooser Dialog for the "SetFaultedOrderStatus" Invoke Activity

   ![Partner Link Chooser](image)

   Click OK in the Partner Link Chooser.

   - **Operation**: select `update`.

   - **Input Variable**: click the Browse Variables icon (the second icon from the left) and select `Variables > Process > Scope - SetOrderStatus > Variables > orderStatusRequest`. 

Creating the SOAOrderBooking Project 8-183
Click **OK** in the Variable Chooser.

The Invoke dialog should look like this:

**Figure 8–265 Invoke Dialog for the “SetFaultedOrderStatus” Invoke Activity**

4. Click **OK** in the Invoke dialog.

5. Select File > Save to save your work.

### 8.19.6 Create a Sensor in the Invoke Activity

Create a sensor in the "SetFaultedOrderStatus" invoke activity to write order information for orders that did not complete for any reason to different destinations such as databases, files, or JMS. In this case, you write to a JMS topic.
Add a Catch Branch to the Project

Creating the SOAOrderBooking Project

1. Double-click the "SetFaultedOrderStatus" invoke activity to display the Invoke dialog.
2. Click the Sensors tab in the Invoke dialog.
3. Click Create to create a new sensor. This displays the Create Activity Sensor dialog.
4. In the Create Activity Sensor dialog, set the Name to InstanceFaulted.
5. Set the Evaluation Time to Completion. This specifies when the sensor fires. Completion signifies that the sensor fires after this activity has run.
6. In the Activity Variable Sensors section, click Create to display the Create Activity Variable Sensor dialog.

7. In the Create Activity Variable Sensor dialog, click the pencil icon for Variable XPath. This displays the Variable XPath Builder dialog.

9. Click OK in the Variable XPath Builder. The Create Activity Variable Sensor dialog should be filled in with these values for you (see Figure 8–266):

   Variable XPath: $OrderBookingFault  
   Output Namespace: http://www.globalcompany.com/ns/OrderBooking
Output Datatype: SOAOrderBookingFaultMessage

10. Click **OK** in the Create Activity Variable Sensor dialog. This takes you back to the Create Activity Sensor dialog, which now looks like this.

**Figure 8–268  Create Activity Sensor Dialog**

![Create Activity Sensor Dialog]

11. In the Create Activity Sensor dialog, click the **Add** icon in the Sensor Actions section. This displays the Sensor Action Chooser dialog. It shows existing sensors called **InstanceStart** and **InstanceCompleted**, which you created in Section 8.6.2, "Create a Sensor for the Receive Activity" and Section 8.18.2, "Add a Sensor".

**Figure 8–269  Sensor Action Chooser Dialog**

![Sensor Action Chooser Dialog]

12. In the Sensor Action Chooser dialog, select **Sensor Actions** and select **Sensor Action** from the wand icon. This displays the Create Sensor Action dialog.

13. In the Create Sensor Action dialog:
- Name: enter InstanceFaulted.
- Publish Type: select JMS Topic.
- JMS Connection Factory: enter jms/TopicConnectionFactory.
- Publish Target: enter jms/demoTopic.
- Filter: leave blank.
- Enable: select this option.

**Figure 8–270  Create Sensor Action Dialog**

14. Click OK in the Create Sensor Action dialog. The Sensor Action Chooser dialog now shows the **InstanceFaulted** sensor action.

**Figure 8–271  Sensor Action Chooser Dialog**

15. Click OK in the Sensor Action Chooser. This takes you back to the Create Activity Sensor dialog, which now looks like this:
16. Click OK in the Create Activity Sensor dialog.
   In the Invoke dialog, the Sensors tab now looks like this:

   **Figure 8–273 Invoke Dialog, Sensors Tab**

17. Click OK in the Invoke dialog.

18. Select File > Save to save your work.

### 8.20 Deploy the Project

1. Double-click `build.properties`, located under `SOAOrderBooking > Resources` in the Application Navigator. This file defines values that are used by `build.xml` when you deploy the BPEL project.
Deploy the Project

Creating the SOAOrderBooking Project

2. Uncomment (by removing the # character) these lines in the `build.properties` file:

```properties
platform=ias_10g
domain=default
rev=1.0
admin.user=oc4jadmin
admin.password=welcome99
http.hostname=myAppServerMachine.mydomain.com
http.port=8888
j2ee.hostname=myAppServerMachine.mydomain.com
rmi.port=23793
opmn.requestport=6005
oc4jinstancename=home
```

3. Edit the values as necessary. The values in bold italics are the typical values you need to modify.

   To determine the value for `rmi.port`, run:
   ```bash
   ORACLE_HOME\opmn\bin\opmnctl status -l
   ```

   *`ORACLE_HOME` specifies the Oracle home for Oracle Application Server.*

   To determine the value for `opmn.requestport`, see step 6 on page 2-11.

4. Select File > Save to save your changes to `build.properties`.

5. Right-click `build.xml`, and select Run Ant.

6. In the Run Ant dialog, click the Properties tab.

7. In the Property Files section, click Add. In the Add Ant Property File dialog, select the `build.properties` file in the SOAOrderBooking directory and click Open.

   The Run Ant dialog should look like this:

   ![Run Ant Dialog, Properties Tab, With build.properties File Loaded](image)

8. Click OK. JDeveloper runs Ant to compile and deploy the project. If you get errors, see the next section, Section 8.20.1, "Deploying Using Ant from the Developer Prompt".

Creating the SOAOrderBooking Project 8-189
8.20.1 Deploying Using Ant from the Developer Prompt

If you get errors, check that the values you entered in the build.properties file are correct.

If the values are correct, but you still get errors, you can run ant from the Developer Prompt to deploy the project:

1. Select Start > Programs > Oracle - instanceName > Oracle BPEL Process Manager > Developer Prompt. This displays a shell window configured for Oracle BPEL Process Manager.

   Note that you must run ant from the Developer Prompt shell window to deploy the SOAOrderBooking project. Running ant from a regular operating system shell for deploying the project is not supported.

2. In the Developer Prompt window, change directory to the SOADEMO\SOAOrderBooking directory, where SOADEMO refers to the directory where you created the SOA Order Booking application.

   > cd SOADEMO
   > cd SOAOrderBooking

   If you are running JDeveloper and Oracle Application Server on separate machines, you can copy the SOAOrderBooking directory from the JDeveloper machine to the Oracle Application Server machine. You can place it anywhere on the Oracle Application Server machine. In the Developer Prompt, you can just navigate to that directory.

3. Run ant.

   > ant

8.20.2 Viewing SOAOrderBooking in the Oracle BPEL Control

After deployment, SOAOrderBooking appears in the Oracle BPEL Control. Enter the following URL in a browser to bring up the Oracle BPEL Control:

http://hostname:port/BPELConsole

Log in as the oc4jadmin user.

SOAOrderBooking appears in the Dashboard tab of the Oracle BPEL Control:

Figure 8–275 Oracle BPEL Control Showing SOAOrderBooking

If you want to look at the WSDL for the SOAOrderBooking project, you can enter the following URL in a browser:
http://hostname:port/orabpel/default/SOAOrderBooking/1.0/SOAOrderBooking?wsdl
This chapter describes how to create the OrderBookingESB project. It contains these sections:

- Section 9.1, "About the OrderBookingESB Project"
- Section 9.2, "Create a New Project for OrderBookingESB"
- Section 9.3, "Create a System Called "OrderBooking""
- Section 9.4, "Create the "OrderBookingService" Routing Service"
- Section 9.5, "Invoke the SOAOrderBooking Process ("OrderBookingProcess" External Service)"
- Section 9.6, "Set up Routing Rules for OrderBookingService"
- Section 9.7, "Save All Files in the OrderBookingESB Project"
- Section 9.8, "Register the OrderBookingESB Project"

9.1 About the OrderBookingESB Project

The OrderBookingESB project is the entry point to the SOA Order Booking application. The SOADemo-Client application invokes the OrderBookingESB project when a customer clicks the Place Order button.

The OrderBookingESB project invokes the SOAOrderBooking project, which is a BPEL project that defines the main flow of the SOA Order Booking application.

The OrderBookingESB project is an ESB project. In JDeveloper, it looks like the following:

Figure 9–1 OrderBookingESB Project in JDeveloper
The OrderBookingESB project consists of:

- OrderBookingService routing service
- OrderBookingProcess external service

### 9.2 Create a New Project for OrderBookingESB

1. Right-click the SOADEMO application, and select **New Project**.
2. In the New Gallery, in the Categories section, expand **General** and select **Projects**. In the Items section, select **ESB Project**.

![Figure 9–2 New Gallery for OrderBookingESB Project](image)

Click **OK**.

3. In the Create ESB Project dialog, enter **OrderBookingESB** in the **Project Name** field. Accept the defaults for the other fields.

![Figure 9–3 Create ESB Project Dialog for OrderBookingESB](image)

Click **OK**.

---

**9-2 Oracle SOA Suite Tutorial**
JDeveloper displays a blank page for the OrderBookingESB.esb file. In the Application Navigator, this file is located under OrderBookingESB > Resources.

9.3 Create a System Called "OrderBooking"

Similar to the "Fulfillment" system that you created for the FulfillmentESB project (covered in Section 4.3, "Create a System Called "Fulfillment""), for the OrderBookingESB project you create a system called "OrderBooking". You can then set all the activities in the OrderBookingESB project to belong to this system.

If you do not create a system, JDeveloper places the activities in a system called "DefaultSystem".

To create a system called "OrderBooking":

1. In the empty OrderBookingESB.esb page, click the Create System/Group icon, located at the top of the page.

![Create System/Group Icon at the Top of the OrderBookingESB.esb Page](image)

2. In the Create ESB System or Service Group dialog:
   - **System**: select this option.
   - **Name**: enter OrderBooking.
   - **Description**: leave blank or enter a brief description.

![Create ESB System or Service Group Dialog for the "OrderBooking" System](image)
3. Click OK. In the Application Navigator, you should see an OrderBooking.esbsys file under OrderBookingESB > Resources.

9.4 Create the "OrderBookingService" Routing Service

1. Copy the following files from the soademo_101310_preview.zip file to the OrderBookingESB directory:
   - OrderBookingRequest.xsd
   - OrderBookingPO.xsd

   In the soademo_101310_preview.zip file, these xsd files are located in the OrderBookingESB directory.

2. Select View > Component Palette to display the Component Palette. In the dropdown for the Component Palette, select ESB Services.

3. Drag the Routing Service icon from the Component Palette and drop it anywhere on the OrderBookingESB.esb page.

4. In the Create Routing Service dialog:
   - Name: enter OrderBookingService.
   - System/Group: set to OrderBooking.
   - Generate WSDL From Schemas: select this option.
   - Schema Location: click Browse, which displays the Type Chooser dialog. In the dialog, select Project Schema Files > OrderBookingRequest.xsd > SOAOrderBookingProcessRequest.

   ![Type Chooser Dialog for OrderBooking Routing Service](image)

   Click OK in the Type Chooser dialog. Schema Location should be set to OrderBookingRequest.xsd.

   - Schema Element: should be set to SOAOrderBookingProcessRequest.
   - Operation Name: enter initiate.

5. Click OK in the Create Routing Service dialog.

The OrderBookingESB.esb page now looks like this:

JDeveloper also creates a WSDL file for you: OrderBooking_OrderBookingService.wsdl in the OrderBookingESB directory.

9.5 Invoke the SOAOrderBooking Process ("OrderBookingProcess" External Service)

This is the service that invokes the SOAOrderBooking process.
1. Select **ESB Services** from the dropdown in the Component Palette.

2. Drag the SOAP Service icon from the Component Palette and drop it to the right of the “OrderBookingService” routing service.

3. In the Create SOAP Invocation Service dialog:
   - **Name**: enter **OrderBookingProcess**.
   - **System/Group**: set to **OrderBooking**.
   - **WSDL File**: Click the Service Explorer (flashlight) icon and in the Service Explorer dialog, select **Service Explorer > BPEL Services > soaIntegrationServer > processes > default > SOAOrderBooking**.
     
     *soaIntegrationServer* refers to the name of your Integration Server connection.

   ![Service Explorer Dialog](image)

   Click **OK** in the Service Explorer.

   - **Port Type**: select **SOAOrderBooking**.

   The Create SOAP Invocation Service dialog should look like this:
4. Click OK in the Create SOAP Invocation Service dialog. You should see two activities in the OrderBookingESB.esb page:

**Figure 9–11 OrderBookingESB.esb Page with OrderBookingService Routing Service and OrderBookingProcess External Service**

### 9.6 Set up Routing Rules for OrderBookingService

Set up routing rules so that requests can go from OrderBookingService to OrderBookingProcess.

1. Double-click OrderBookingService in the top section of the icon. This displays the page for the OrderBooking_OrderBookingService.esbsvc file.
2. In the Routing Rules section, click the [+ ] to expand the section.
3. In the Routing Rules section, click the green + icon. You may have to scroll to the right to see the icon.
Figure 9–12 Routing Rules

4. In the Browse Target Service Operation dialog, select ESB > Services in project > OrderBooking > OrderBookingProcess > initiate.

Figure 9–13 Browse Target Service Operation Dialog

Click OK in the Browse Target Service Operation dialog. This returns you to the Routing Rules section, which should show "OrderBookingProcess::initiate" in the last column.
5. In the Routing Rules section, click the filter icon.

6. In the Expression Builder, in the WSDL Message section, select `SOAOrderBookingProcessRequest_request` > `SOAOrderBookingProcessRequest` > `tns:SOAOrderBookingProcessRequest`. Click **Insert Into Expression**. In the Expression box, you should see: `/tns:SOAOrderBookingProcessRequest`
Click **OK** in the Expression Builder.

The routing rule now includes a filter, which simply passes the order information to the OrderBookingProcess activity. The routing rule section now looks like this, with the filter information in the first column.

7. Select File > Save to save `OrderBooking(OrderBookingService.esbsvc).`
8. Click the OrderBookingESB.esb tab.

On the OrderBookingESB.esb page, you should see a line connecting the two activities:

*Figure 9–17 Final OrderBookingESB.esb Page*

9.7 Save All Files in the OrderBookingESB Project

Browse through the OrderBookingESB files in the Application Navigator to ensure that all the files are saved. If you see a file in italics, select the file and save it.

9.8 Register the OrderBookingESB Project

In the Application Navigator, right-click the OrderBookingESB project and select *Register with ESB > SoademoIntegServer*, where *SoademoIntegServer* is the name of the connection to the integration server.

You can view the services that you created in the OrderBookingESB project in the ESB Console. To do this, enter the URL of the Oracle ESB Console in a browser:

http://host:port/esb/esb/EsbConsole.html

*host* specifies the name of the machine running Oracle Application Server, and *port* specifies the HTTP port at which Oracle HTTP Server or OC4J is listening.

Log in as the *oc4jadmin* user.

The services are grouped under *OrderBooking*, which is the name of the system that you created in *Section 9.3, "Create a System Called "OrderBooking""*. 
Figure 9–18  Oracle ESB Console Showing the Services in the OrderBooking System
Interfacing the Client Application with the SOA Order Booking Application

The focus of this chapter is how to get a client application such as the SOADemo-Client application to interface with the projects in the SOA Order Booking application.

This chapter does not cover how to create the client application in detail because the client application uses the standard ADF technology, which is covered in detail in the ADF Developer’s Guide and ADF Tutorial.

This chapter includes the following topics:

- Section 10.1, "About the SOADemo-Client Application"
- Section 10.2, "Invoking Services from CustomerService"
- Section 10.3, "Invoking the OrderBookingESB Project"
- Section 10.4, "Deploying the Client Application"

10.1 About the SOADemo-Client Application

The SOADemo-Client application is a Web-based application that customers use to place orders with Global Company. When a customer logs into the application, he or she can browse products, place products into the shopping cart, and submit the order. For details on the pages in the SOADemo-Client application, see the Oracle SOA Suite Quick Start Guide.

The SOADemo-Client application uses ADF technology to simplify the application development process and to make the application easier to maintain. You can use JDeveloper to create ADF applications.

The SOADemo-Client application makes calls to these services provided by the SOA Order Booking application:

- CustomerService. In the login process, the client application uses the “find customer by email” service provided by CustomerService to validate the customer’s login ID and password.
- CustomerService. The SOADemo-Client application provides a Register button that enables new customers to add themselves to the database. The Register button invokes the “add new customer” service provided by CustomerService.
- OrderBookingESB. When the customer submits an order, it invokes the OrderBookingESB project. The OrderBookingESB project in turn invokes the SOAOderBooking BPEL process, which is the main flow for the SOA Order Booking application.
For the SOADemo-Client application to be able to invoke services provided by CustomerService and OrderBookingESB, you have to generate web service proxies for the services. In the code for the client, you call methods in the web service proxy, which then invokes the corresponding methods in CustomerService or OrderBookingESB. You can use JDeveloper to generate web service proxies.

10.2 Invoking Services from CustomerService

This section describes how to generate a web service proxy for CustomerService, and how to call it from the client application.

- Section 10.2.1, "Generate a Web Service Proxy for CustomerService"
- Section 10.2.2, "Build the Files"
- Section 10.2.3, "Write Code to Invoke the Web Service Proxy for CustomerService"

10.2.1 Generate a Web Service Proxy for CustomerService

To generate web service proxy for CustomerService, you create a project for it in the client application. The following procedure starts by creating an empty project, then invokes the Create Web Service Proxy wizard to create the proxy for CustomerService.

1. Create a project in your client application to contain the web service proxy for CustomerService. Begin by creating an empty project:

   a. Right-click the client application, and select New Project.
   b. In the New Gallery, in the Categories section, expand General and select Projects. In the Items section, select Empty Project.

   \[Figure 10-1\] New Gallery for Creating an Empty Project

   Click OK.

   c. In the Create Project dialog, enter CustomerService in the Project Name field.
Invoking Services from CustomerService

Interfacing the Client Application with the SOA Order Booking Application

Click OK.
In the Application Navigator, you should see an empty CustomerService project located under the client application.

2. Right-click the CustomerService project, and select **New**. In the New Gallery, in the Categories section, expand **Business Tier** and select **Web Services**. In the Items section, select **Web Service Proxy**.

3. In Step 1, Web Service Description:
   - **WSDL Document URL**: enter the following URL:
     
     
     `host` specifies the name of the machine running Oracle Application Server, and `port` specifies the HTTP at which Oracle HTTP Server or OC4J is listening.
   - **Mapping File**: leave it blank.
Figure 10–4  Create Web Service Proxy Wizard, Step 1: Web Service Description

Click Next. If nothing happens, click the UDDI button. This pops up another wizard. Click Cancel in that wizard. Now click Next again.

4. In Step 2, Port Endpoints, select Run against a service deployed to an external server, and click Next.

Figure 10–5  Create Web Service Proxy Wizard, Step 2: Port Endpoints

5. In Step 3, Custom Mappings, click Next.
6. In Step 4, Defined Handlers, click Next.

7. In Step 5, Default Mapping Options:
   - **Package Name**: enter oracle.soademo.view.services.
   - **Root Package for Generated Types**: enter oracle.soademo.view.services.runtime.
   - Select all the options on the page:
     - Generate Data Binding Classes
     - Reuse Existing Type Classes
     - Unwrap Wrapped Parameters
     - Map Headers to Parameters
8. In Step 6, Support Files, select **Generate JUnit Unit Test Code** only if you want to write test cases using JUnit. Note that to be able to compile the files that use JUnit classes, you need to download the JUnit library files from an external site (for example, [http://www.junit.org](http://www.junit.org)).

Click Next.

9. In the Finish step, click **Finish**.

JDeveloper creates the web service proxy files in `src` directory of the project.

### 10.2.2 Build the Files

1. If you selected **Generate JUnit Unit Test Code** in step 8 on page 10-6, then you need to download JUnit from an external site (for example, [http://www.junit.org](http://www.junit.org)).
Add the JUnit jar file to the project before building the project. To add the jar file to the project, right-click the CustomerService project in the client and select Project Properties. In the Project Properties dialog, select Libraries on the left side, and click Add Jar/Directory on the right side. Select the JUnit jar file to add to the project.

2. Right-click the CustomerService project in the client and select Rebuild.

10.2.3 Write Code to Invoke the Web Service Proxy for CustomerService

The main web service proxy file for CustomerService is $src/oracle/soademo/view/services/CustomerServiceClient.java$. This file lists the methods provided by CustomerService. Your client application can call the methods in this class, and you deploy your client application with the web service proxy.

The SOADemo-Client application invokes the web service proxy for CustomerService in two places:

- Section 10.2.3.1, "Verifying the Login"
- Section 10.2.3.2, "Registering New Customers"

10.2.3.1 Verifying the Login

In the SOADemo-Client application, when the customer clicks the Login button, it invokes the login_action method in Login.java, located in the SOADEMO-CLIENT\UserInterface\src\oracle\soademo\view\backing directory.

The login_action method calls the findCustomerByEmail method in CustomerServiceClient to verify the customer’s login. The method is passed the email and password information entered by the customer on the login page. If the customer is found, it stores the customer information in the session and returns "success".

```java
public String login_action() {
    String AUTH_USER = "Authorized_User";

    // Check credentials
    FacesContext ctx = FacesContext.getCurrentInstance();

    // Call Web service to check user credentials
    try {
        oracle.soademo.view.services.CustomerServiceClient myPort =
            new oracle.soademo.view.services.CustomerServiceClient();
        System.out.println("calling " + myPort.getEndpoint());

        // test adding new customer
        Customer newCust = null;
        newCust = myPort.findCustomerByEmail(emailId.getValue().toString(),
            pwd.getValue().toString());

        //Store customer info on session
        if (newCust != null)
            JSFUtils.storeOnSession(ctx, "custinfo", newCust);
        else {
            FacesMessage msg = new FacesMessage("Login Failed!");
            msg.setSeverity(msg.SEVERITY_ERROR);
            FacesContext.getCurrentInstance().addMessage(null, msg);
            return null;
        }
    }
```

```java
catch (Exception e) {
    throw new RuntimeException(e.getMessage());
    JSFUtils.add جميع الرسائل التي تم توليدها في الدالة،
    msg.setSeverity(msg.SEVERITY_ERROR);
    FacesContext.getCurrentInstance().addMessage(null, msg);
    return null;
}
```

```java
JSFUtils.storeOnSession(ctx, "custinfo", newCust);
```

```java
else {
    FacesMessage msg = new FacesMessage("Login Failed!");
    msg.setSeverity(msg.SEVERITY_ERROR);
    FacesContext.getCurrentInstance().addMessage(null, msg);
    return null;
}
```
10.2.3.2 Registering New Customers

The Register page enables new customers to add themselves to the database. When the customer fills in the information and clicks the Register button on the page, it invokes the register_action method in Register.java, also located in the SOADEMO-CLIENT\UserInterface\src\oracle\soademo\view\backing directory.

The register_action method collects the customer information and invokes the addNewCustomer method on CustomerService.

```java
public void register_action(ActionEvent ae) {
    String AUTH_USER = "Authorized_User";
    FacesContext ctx = FacesContext.getCurrentInstance();
    Customer newCust = new Customer();
    if (password.getValue().toString().equals(password_chk.getValue().toString())) {
        // Call Web service to register new customer
        try {
            oracle.soademo.view.services.CustomerServiceClient myPort =
                    new oracle.soademo.view.services.CustomerServiceClient();
            System.out.println("calling " + myPort.getEndpoint());

            // Adding new customer info
            Address addr = new Address();
            newCust.setFname(fname.getValue().toString());
            newCust.setLname(lname.getValue().toString());
            newCust.setEmail(email.getValue().toString());
            newCust.setPhonenumber(phone.getValue().toString());
            newCust.setPassword(password.getValue().toString());
            addr.setStreet(street.getValue().toString());
            addr.setCity(city.getValue().toString());
            addr.setState(state.getValue().toString());
            addr.setZip(zip.getValue().toString());
            newCust.setPhonenumber(phone.getValue().toString());
            addrList = new ArrayList();
            addrList.add(addr);
```
newCust.setAddressList(addrList);
myPort.addNewCustomer(newCust);

// Generate successful registration message
FacesContext.getCurrentInstance().addMessage(null,
    new FacesMessage("Registration Successful!");
}

// Generate password mismatch msg
FacesMessage msg = new FacesMessage("Registration Failed!");
msg.setSeverity(msg.SEVERITY_ERROR);
FacesContext.getCurrentInstance().addMessage(null, msg);
FacesContext.getCurrentInstance().addMessage(null,
    new FacesMessage(ex.getMessage()));
ex.printStackTrace();

// Store customer info on session
JSFUtils.storeOnSession(ctx, "custinfo", newCust);
// Set CurrentUser managed bean properties
JSFUtils.setManagedBeanValue(ctx,"Current_User.loggedin",true);
JSFUtils.setManagedBeanValue(ctx,"Current_User.userid",
    email.getValue().toString());
JSFUtils.storeOnSession(ctx, AUTH_USER, "Authorized_User");
else {
    // Generate password mismatch msg
    FacesMessage msg = new FacesMessage("Your password values do not match!");
    msg.setSeverity(msg.SEVERITY_ERROR);
    FacesContext.getCurrentInstance().addMessage(null, msg);
}

10.3 Invoking the OrderBookingESB Project
Creating the web service proxy for the OrderBookingESB project is slightly different from creating the proxy for CustomerService, because OrderBookingESB is an ESB project.

10.3.1 Retrieve the Concrete WSDL URL
Before you can run the Create Web Service Proxy wizard, you need the concrete URL that returns the WSDL for OrderBookingESB. You can do this using the ESB Console.

1. Access the ESB Console using the following URL:

   http://hostname:port/esb/esb/EsbConsole.html

   *hostname* specifies the name of the machine running Oracle Application Server, and *port* specifies the HTTP port at which Oracle HTTP Server or OC4J is listening.

2. On the left side of the ESB Console, select **OrderBookingService**. This is the starting point for the OrderBookingESB project.

3. On the right side of the ESB Console, select the Definition tab. You should see a page that looks something like Figure 10–10:
4. Copy the concrete URL so that you can paste it in the Create Web Service Proxy wizard. You can also click the URL if you want to see what the WSDL looks like.

10.3.2 Create the Project in the Client and Create the Web Service Proxy

1. Create a project in your client application to contain the web service proxy for OrderBookingESB. Begin by creating an empty project in your client application:
   a. Right-click the client application, and select New Project.
   b. In the New Gallery, in the Categories section, expand General and select Projects. In the Items section, select Empty Project.
Invoking the OrderBookingESB Project

Interfacing the Client Application with the SOA Order Booking Application

1. Click OK.

   c. In the Create Project dialog, enter **OrderService** in the **Project Name** field.

   ![New Gallery for Creating an Empty Project](image)

   **Figure 10–11** New Gallery for Creating an Empty Project

   - **Business Tier** > **Web Services** > **Web Service Proxy**

   Click OK.

   In the Application Navigator, you should see an empty OrderService project located under the client application.

2. Right-click the OrderService project, and select **New**. In the New Gallery, in the Categories section, expand **Business Tier** and select **Web Services**. In the Items section, select **Web Service Proxy**.

   ![Create Project Dialog for the OrderService Web Service Proxy Project](image)

   **Figure 10–12** Create Project Dialog for the OrderService Web Service Proxy Project

   - **OrderService**
   - **Directory Name**: clientOrderService

   Click OK.
Click **OK**. This launches the Create Web Service Proxy wizard. Click **Next** to continue.

3. In Step 1, **Web Service Description**:  
   - **WSDL Document URL**: enter the URL for OrderBookingESB. Section 10.3.1, "Retrieve the Concrete WSDL URL" describes how to obtain this URL.
   - **Mapping File**: leave it blank.
   - **Copy WSDL Into Project**: do not select this option.

Click **Next**. If nothing happens, click the **UDDI** button. This pops up another wizard. Click **Cancel** in that wizard. Now click **Next** again.
4. In Step 2, Port Endpoints, select Run against a service deployed to an external server.

In Endpoint URL, edit the URL as follows:

- Replace localhost with the name of the machine running the OrderBookingESB project.
- Verify and correct the port number, if necessary.

Click Next.

5. In Step 3, Custom Mappings, click Next.

6. In Step 4, Defined Handlers, click Next.
7. In Step 5, Default Mapping Options:
   - **Package Name**: enter `oracle.soademo.view.services`.
   - **Root Package for Generated Types**: enter `com.globalcompany.ns.order`.
   - Select all the options on the page:
     - Generate Data Binding Classes
     - Reuse Existing Type Classes
     - Unwrap Wrapped Parameters
     - Map Headers to Parameters

8. In Step 6, Support Files, select **Generate JUnit Unit Test Code** only if you want to write test cases using JUnit. Note that to be able to compile the files that use JUnit classes, you need to download the JUnit library files from an external site (such as [http://www.junit.org](http://www.junit.org)).
Click Next.

Figure 10–19  Create Web Service Proxy Wizard, Step 6, Support Files


10.3.3 Build the Files

1. If you selected Generate JUnit Unit Test Code in step 8 on page 10-14, then you need to download JUnit from an external site (for example, http://www.junit.org).

Add the JUnit jar file to the project before building the project. To add the jar file to the project, right-click the OrderService project in the client and select Project Properties. In the Project Properties dialog, select Libraries on the left side, and click Add Jar/Directory on the right side. Select the JUnit jar file to add to the project.

2. Right-click the OrderService project in the client and select Rebuild.

10.3.4 Write Code to Invoke the Web Service Proxy for OrderBookingESB

On the Shopping Cart page in the SOADemo-Client application, customers click the Place Order button to submit their order. This button invokes the PlaceOrder_ action method in ShoppingCart.java (located in the SOADemo-CLIENT\UserInterface\src\oracle\soademo\view\backing directory).

The PlaceOrder_ action method:

- creates an instance of __soap_initiate_pptClient, which is the proxy for OrderBookingESB. This class was generated by the Web Service Proxy wizard. You should check the name of the generated file in your environment. The name of the class may be slightly different (for example, __soap_ OrderBookingService_initiate_pptClient.java).
- gets the order information from the shopping cart
- gets the customer information from the session
- creates a purchase order and populates it with the order and customer information
Invoking the OrderBookingESB Project

- calls initiate on the proxy to invoke OrderBookingESB
- sets the shopping cart to empty.

The lines in bold show the places where the code invokes the web service proxy.

```java
public String PlaceOrder_action() {
    // Place order
    FacesContext ctx = FacesContext.getCurrentInstance();

    // Get order items from Cart
    Cart cartBean = (Cart) JSFUtils.getManagedObject("Shopping_Cart");
    List CartItemList = cartBean.getItemList();

    // Get Customer info from session
    Customer currentCustomer = (Customer) JSFUtils.getFromSession(ctx, "custinfo");

    try {
        oracle.soadeemo.view.services.__soap_initiate_pptClient myPort =
            new oracle.soadeemo.view.services.__soap_initiate_pptClient();
        System.out.println("calling " + myPort.getEndpoint());

        PurchaseOrderType po = new PurchaseOrderType();

        // Customer ID
        po.setCustID(currentCustomer.getCustid());

        // Need to randomly set ID for now since WS won't auto-assign
        po.setID("" + (int)(Math.random() * 1000));

        // Order Info
        OrderInfoType order = new OrderInfoType();
        order.setOrderPrice(cartBean.getOrderTotal());
        order.setOrderComments("This order was issued from the SOA Retail Client.");
        Calendar caldate = Calendar.getInstance();
        order.setOrderDate(caldate);
        order.setOrderStatus("pending");

        po.setOrderInfo(order);

        // Order Items
        OrderItemsType oitems = new OrderItemsType();
        ItemType[] items = new ItemType[CartItemList.size()];
        for (int i = 0; i < CartItemList.size(); i++) {
            items[i] = (ItemType) CartItemList.get(i);
        }
        oitems.setItem(items);
        po.setOrderItems(oitems);

        // Supplier Info
        // For now hard code fictional Supplier
        SupplierInfoType supplier = new SupplierInfoType();
        supplier.setSupplierName("Express Deliveries");

        // Supplier has 10% discount
        supplier.setSupplierPrice(cartBean.getOrderTotal().subtract(cartBean.getOrderTotal().movePointLeft(1)));
10.4 Deploying the Client Application

This section describes how to deploy the pre-built client application from the soademo_101310_prod.zip file.

To deploy the SOADEMO-Client application from the zip file:

1. In JDeveloper, open SOADEMO-CLIENT\SOADEMO-CLIENT.jws. This is the project file for the SOADEMO-Client application.

2. This step is required only if you meet any of these conditions:
   ● You are running JDeveloper and Oracle Application Server on different machines.
   ● Your Oracle Application Server installation is listening for HTTP requests on a port other than 8888.

If you meet either or both of the conditions above, you need to perform these steps:

a. Edit these files:
   - SOADEMO-CLIENT\CustomerService\src\oracle\soademo\view\services\runtime\CustomerServiceSoapHttp_Stub.java
     In JDeveloper, you can access this file from the Application Navigator as follows:
     i. In JDeveloper, select View > Structure to display the Structure window.
     ii. In the Application Navigator, select SOADEMO-CLIENT > CustomerService > Application Sources > oracle.soademo.view > services > CustomerSvcProxy.
     iii. Double-click CustomerServiceSoapHttp_Stub.java in the Structure window.
   - SOADEMO-CLIENT\OrderService\src\oracle\soademo\view\services\runtime\__soap_initiate_ppt_Stub.java
In JDeveloper, you can access this file from the Application Navigator as follows:

i. In JDeveloper, select View > Structure to display the Structure window.

ii. In the Application Navigator, select SOA DEMO-CLIENT > OrderService > Application Sources > oracle.soadem o.view > services > Initiate_pptServiceProxy.

iii. Double-click __soap_initiate_ppt_Stub.java in the Structure window.

Make the following changes in the files:

- Search for "localhost" and replace it with the fully qualified name (including the domain name) of the machine running Oracle Application Server.

- Search for "8888" and replace it with the HTTP port for your environment.

b. Delete (or rename) these files:

- SOA DEMO-CLIENT\UserInterface\deploy\soaui.war
- SOA DEMO-CLIENT\UserInterface\deploy\soaui.ear

c. Regenerate the files you just deleted.

i. Expand SOA DEMO-CLIENT > UserInterface > Resources.

ii. Right-click soaui.deploy and select Deploy to WAR File.

iii. Right-click soaui.deploy and select Deploy to EAR File.

3. Expand SOA DEMO-CLIENT > Assembly > Application Sources.

4. Right-click SOA DEMO.deploy and select Deploy To > ApplicationServerConnection, where ApplicationServerConnection specifies the connection to your Oracle Application Server installation.

5. Click OK in the Configure Application dialog.