BEA AquaLogic® Integrator

Tutorial: Designing a Purchase Order Processing System by Using AquaLogic Integrator

Version 3.0
Document Revised: March 2008

AquaLogic Integrator: Introduction

This chapter provides an introduction to AquaLogic Integrator (ALINT). It also describes how to develop a Purchase Order Processing System by using the tutorial sample application.

This section discusses the following topics:

- "Overview of AquaLogic Integrator"
- "Creating an ALINT Domain"
- "How to Use This Tutorial..."

Overview of AquaLogic Integrator

AquaLogic Integrator (ALINT) is a software bundle comprising WebLogic Integration (WLI) and AquaLogic Service Bus (ALSB).

ALINT offers the following capabilities of WLI and ALSB to manage reusable services in a dynamically changing environment:

- Complex transactional process integration
- Mediation, connectivity, and embedded management

In addition, ALINT offers developer productivity enhancements in an integrated environment. It ensures a consistent and scalable approach to SOA Integration through its integration with WorkSpace Studio, which is the unified assembly platform of BEA.

Some of the key features of ALINT are:

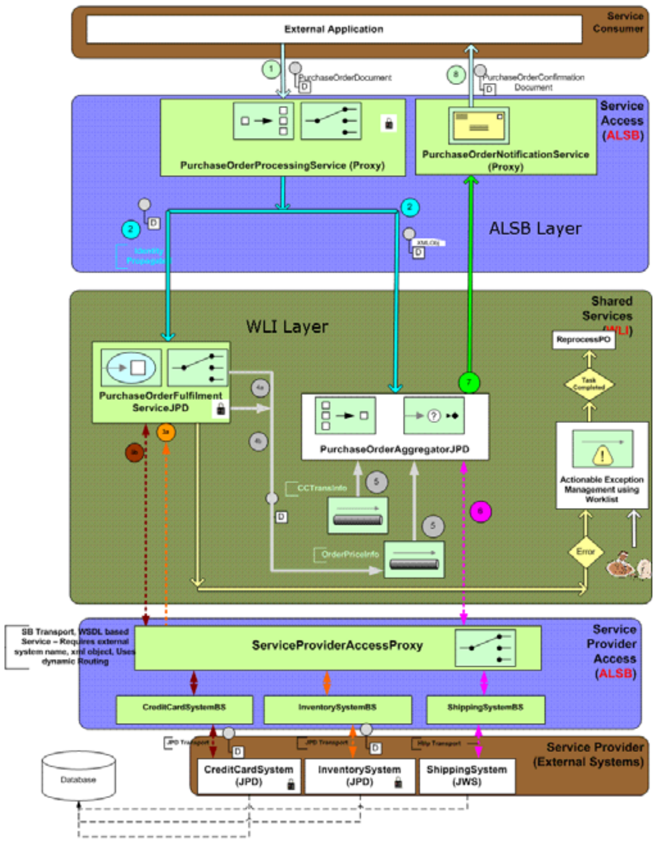
- ServiceBus (SB) Transport : For more information, see *SB Transport User Guide*.
- JPD Transport : For more information, see *JPD Transport User Guide*.
- AquaLogic Service Bus (ALSB) Control: For more information, see *Using Integration Controls*.
- Integrated Development Environment (IDE) for both WLI and ALSB

This tutorial sample application, and its workflow involved is described in "Architecture of the Sample Application."

Architecture of the Sample Application

Figure 1-1 illustrates the architecture of the Purchase Order Processing System.

Figure 1-1 Architecture


External Application
Service Consumer
PurchaseOrderDocument
PurchaseOrderConfirmation Document
Service Access (ALSB)
PurchaseOrderProcessingService (Proxy)
PurchaseOrderNotificationService (Proxy)
ALSB Layer
WLI Layer
Shared Services (WLI)
ReprocessPO
Task Completed
PurchaseOrderFulfilment ServiceJPD
PurchaseOrderAggregatorJPD
Actionable Exception Management using Worklist
Error
CCTransInfo
OrderPriceInfo
SB Transport, WSDL based Service – Requires external system name, xml object, Uses dynamic Routing
ServiceProviderAccessProxy
Service Provider Access (ALSB)
CreditCardSystemBS
InventorySystemBS
ShippingSystemBS
CreditCardSystem (JPD)
InventorySystem (JPD)
ShippingSystem (JWS)
Service Provider (External Systems)
Database

Workflow of the Sample Application

`PurchaseOrderProcessingService` is an ALSB proxy service that receives a purchase order from an external client with information, such as purchase orderID, customer name, order line item, credit card, and shipping address. The purchase order must be processed, and a confirmation has to be sent to the client.

`PurchaseOrderFulfillmentJPD` and `PurchaseOrderAggregatorJPD` business process JPDs are used for state management and process orchestration.

External Systems such as credit card system, inventory system and shipping provider systems in this tutorial sample application are modeled as JPDs, and JWS, respectively.

The workflow is described as follows:

1. `PurchaseOrderProcessingService` is implemented as a WSDL-based ALSB proxy service, which is an entry point to the application. This PurchaseOrderProcessingService receives PurchaseOrderDocument (typed XML) over HTTPS. This ALSB proxy service implements the split-route pattern and routes each of the split documents as follows:
   a. One part consisting of the order line item and credit card details, which is passed to the `PurchaseOrderFulfillmentJPD`. The `PurchaseOrderFulfillmentJPD` is a JPD that fulfills the purchase order.
   b. One part containing the Purchase Order ID, Callback location and the shipping information is published to the `PurchaseOrderAggregatorJPD` (SOAP over JMS), which aggregates the results of the `PurchaseOrderFulfillmentJPD`.
2. `PurchaseOrderFulfillmentJPD` is a WLI JPD that invokes the inventory system and the credit card system in the context of a transaction and publishes the results of the inventory and credit card systems to the corresponding message broker channels. In case of any failure, a worklist task is created with the cause of the failure, to enable actionable exception management.
3. `PurchaseOrderAggregatorJPD` that is already subscribed to these message broker channels, aggregates the results and invokes the shipping system and sends a confirmation back to the client as a callback.

Purpose of the Tutorial

The purpose of the tutorial is to demonstrate the new features like AquaLogic Service Bus (ALSB) Control, JPD transport, and Integrated IDE, using the sample application. This tutorial helps you in rebuilding the ALSB Control and ALSB Business Service End Points using JPD transport.

This tutorial focuses on demonstrating transaction context propagation, and security context propagation.

Creating an ALINT Domain

For information about creating an ALINT domain, see Installing and Using AquaLogic Integrator.

Ensure that the Administration server is running on port 7001. SSL is enabled, and the SSL port is 7002. SSL is required because the ALSB proxy service - PurchaseOrderProcessingService is HTTPS enabled for secure data transmission.

How to Use This Tutorial...

You can use this tutorial in the following ways:

Table 1-1 Using the Tutorial

Part	Description
Part I, "Using the Out of the Box Sample Application"	Describes how you can use the tutorial sample application, without having to make any changes.
Part II, "Accessing ALSB Proxy Service and Exposing a JPD"	Describes how you can rebuild two key features of ALINT namely ALSB Control and JPD Business Service.
Part III, "Developing the PO Application System"	Describes how to design the Purchase Order (PO) Processing System by creating all artifacts from scratch.

Part I Using the Out of the Box Sample Application

This part describes how you can use the tutorial sample application, without having to make any changes.

It discusses the following topics:

Running the Sample Application

The tutorial application contains the artifacts required to run the sample purchase order processing application.

This section discusses the following topics:

- "Creating the Tutorial Environment"
- "Loading the Sample Application"
- "Testing the Application"

For information about ALINT Installer, see Installing and Using AquaLogic Integrator.

You can download the tutorial sample application from dev2dev.bea.com.

You must extract the contents of this sample application to a directory on your computer. Refer to this directory as {TUTORIAL_ROOT}. There are two sub folders (applications) in this directory: **ExternalSystems** and **UsecaseApp**.

Note: After you have downloaded the sample application, ensure that these two directories are not Read-Only.

Creating the Tutorial Environment

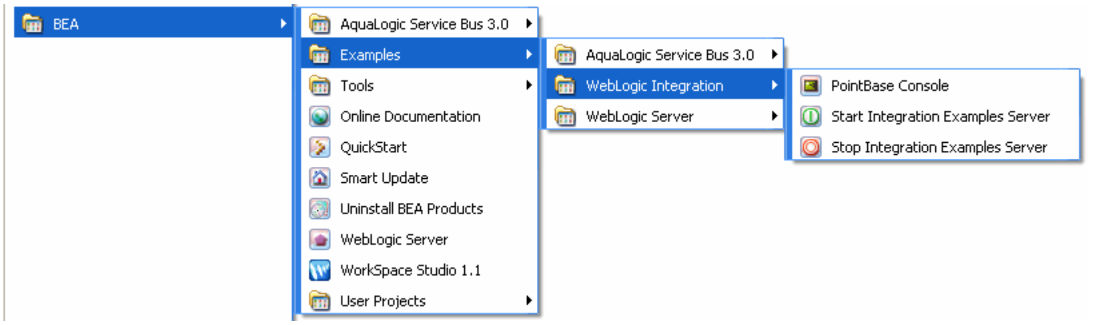
After you have created the ALINT domain, you must create the environment for the tutorial to design the *Purchase Order Processing System* sample application.

To Create the Tutorial Environment

1. Access the Pointbase Console from **Start** > **All Products** > **BEA** > **Examples** > **WebLogic Integration** > **Pointbase Console** at the **url** field of the **Connect To Database** dialog box that is displayed, type *`jdbc:pointbase:server://localhost:9093/weblogic_eval`*. Type entries into the username and password fields. For example, enter `weblogic` as the username and password.

   Note: **BEA** here refers to the BEA HOME directory where you have installed the BEA 10.2 products. Figure 2-6 shows the **Start** menu, from where you can access the Pointbase Console.

Figure 2-1 Start Menu Structure



2. In the **Pointbase Console**, run the database scripts in the `usecaseapp-dbscripts.sql` file that is available in the *{TUTORIAL_ROOT}* folder.

3. Deploy the *external services* that are used by the sample application, by performing the following steps:

   a. Click **Start** > **All Programs** > **BEA** > **WorkSpace Studio**.

   b. Create a new workspace **External Systems** in WorkSpace Studio. Select **File** > **Import**. Expand **General**, in the **Import** dialog box, select **Existing Projects into Workspace**. Click **Next**.

   c. Browse to the directory of *External Systems*. In the **Import Projects** dialog box, select all the projects displayed, and click **Finish**.

   Note: After you have imported the application, you must complete the following steps to disable EAR validation at the workspace level, and clean the ear project:

   – Right-click on the EAR (ExtSystem) project, and select **Properties**.

   - In the **Validation** pane, click **Configure Workspace Settings**. Clear the **EAR Validator** check boxes.
   - From the **BEA WorkSpace Studio** menu, click **Project** > **Clean**.

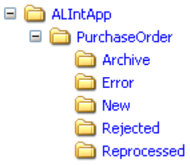
d. Expand **ExtSystemWeb** > **src**, right-click on **CreditCardSystemJPD.java**, and select **Run As** > **Run on Server**.

e. In the **BEA WebLogic Server v10.0** dialog box, browse to the domain directory, and select the ALINT domain you have created. Click **Next**.

4. In the **Add and Remove Projects** dialog box, select **ExtSystem** and **ExtServiceAccessConfig**, and click **Finish**.
5. Create the following directory structure under **C:\ ALIntApp** > **PurchaseOrder**. Create subfolders - **Archive**, **Error**, **New** and **Rejected** under the Purchase Order folder. The purchase order processed by the sample application is archived to the file system into these folders.

   Figure 2-2 shows the Purchase Order archive directory structure.

Figure 2-2 Purchase Order Archive Directory Structure

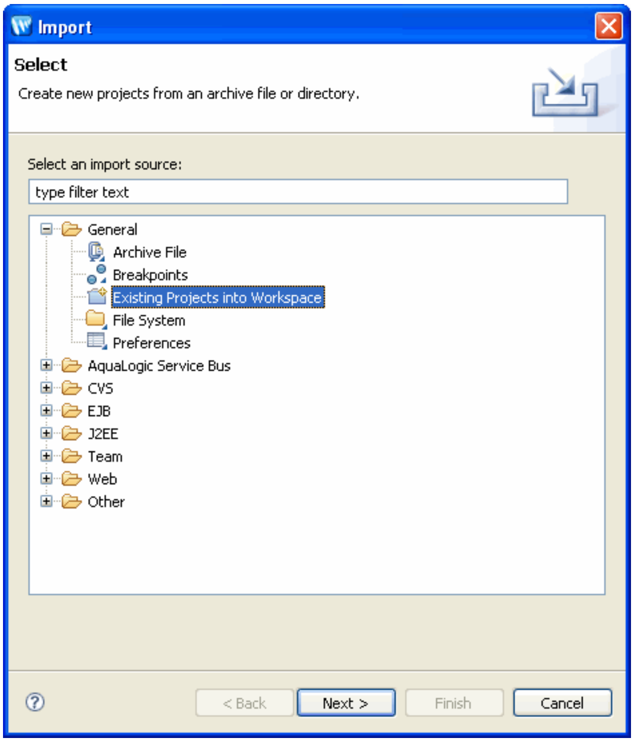


Loading the Sample Application

To Load the Sample Application

1. Click **Start** > **All Programs** > **BEA** > **WorkSpace Studio**, type a new workspace name - `UseCase`, in the **Select a workspace** dialog box, and click **OK**.
2. J2EE is the default perspective of Workspace Studio. Switch to Process perspective.
3. From the **BEA WorkSpace Studio** menu, click **File > Import > General > Existing Projects into Workspace**. Click **Next**. The **Import** dialog box is displayed.

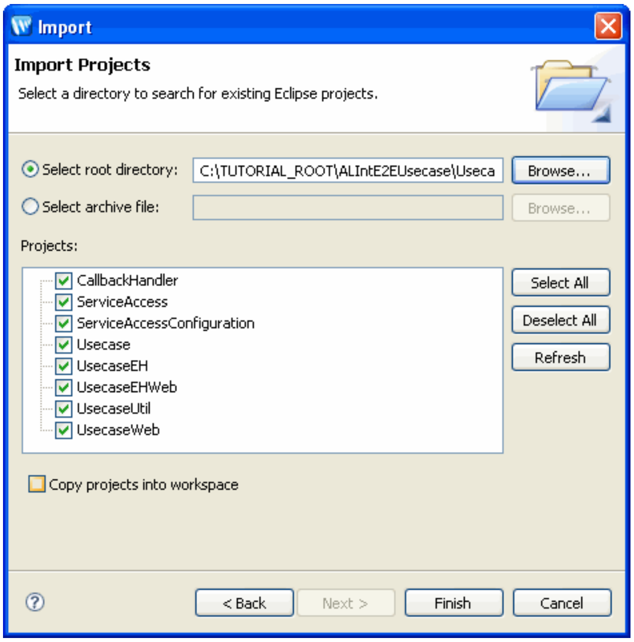
Figure 2-3 Import Projects



4. In the **Import Projects** dialog box, browse to the *{TUTORIAL_ROOT}* directory where the sample application is extracted to. Select all the files from the **Projects** pane:

   Figure 2-4 shows the **Import Projects** dialog box.

Figure 2-4 Import Projects



Select a directory to search for existing Eclipse projects.

(●) Select root directory: C:\TUTORIAL_ROOT\ALIntE2EUsecase\Useca

( ) Select archive file:

Projects:

- [✓] CallbackHandler
- [✓] ServiceAccess
- [✓] ServiceAccessConfiguration
- [✓] Usecase
- [✓] UsecaseEH
- [✓] UsecaseEHWeb
- [✓] UsecaseUtil
- [✓] UsecaseWeb

[ ] Copy projects into workspace

5. Click **Finish**.

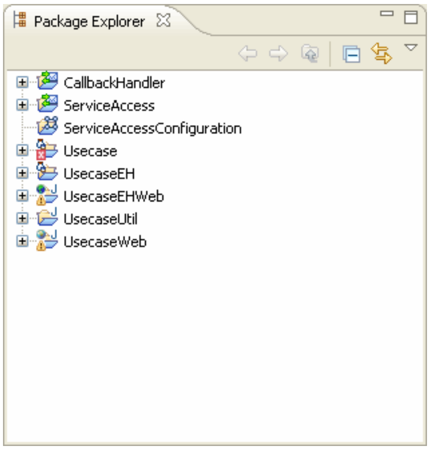
   Note: The Process perspective contains all the required views like **Node Palette**, **Data Palette**, and **Package Explorer**.

   Similarly, the ALSB perspective contains views related to ALSB, such as **Expression Functions**, **Expression Variables**, **Target Expression**, **Constraints**, and other views that are required to create a JPD.

6. The selected projects are imported and displayed in the **Package Explorer** pane. Figure 2-5 displays the **Package Explorer** pane.

   Note: Clean the Project after you have imported the files. From the **Workspace Studio** menu, click **Project** > **Clean**.

Figure 2-5 Package Explore Pane



Package Explorer contains the following:

- Service Access: ALSB project that comprises **Business Services**, and **Proxy Services**.
- Service Access Configuration: ALSB Configuration project.
- UseCase: WebLogic Integration EAR project.
- UseCaseUtil: Utility project.
- UseCaseWeb: Web project that consists of business processes and related artifacts.

Note: After you have imported the application, you must complete the following steps to disable EAR validation at the workspace level, and clean the ear project:

a. Right-click on the EAR (UseCase) project, and select **Properties**.

b. In the **Validation** pane, click **Configure Workspace Settings**. Clear the **EAR Validator** check boxes.

c. From the **BEA WorkSpace Studio** menu, click **Project** > **Clean**.

Table 2-1 lists the important artifacts that are in the **Package Explorer**, after you have imported the project files.

Table 2-1 Important Artifacts in Package Explorer

File Name	Description
`ServiceAccess/ProxyServices/PurchaseOrderProcessingService.proxy`	The WSDL-based ALSB proxy service that forms the facade for the Purchase Order processing application we are building Validates the PurchaseOrder incoming document that splits the incoming PurchaseOrder document and invokes the following services to process the order: `PurchaseOrderAggregatorJPD` and `PurchaseOrderFulfillmentJPD`.
`ServiceAccess/ProxyServices/PurchaseOrderNotificationService.proxy`	The ALSB proxy service that is required to send the PurchaseOrderStatus document as a *callback* to the client.
`ServiceAccess/BusinessServices/PurchaseOrderNotificationServiceBS.biz`	The ALSB business Service that is required to send the PurchaseOrderStatus document as *callback* to the client.
`UseCaseWeb/src/alint/process/ServiceProviderAccessControl.java`	The ALSB Control that invokes `ServiceProviderAccessProxy.proxy`.
`UseCaseWeb/src/alint/process/PurchaseOrderFulfillmentJPD.java`	The JPD that handles order fulfillment. This JPD invokes the external inventory and credit card systems using ServiceProviderAccessControl. External systems are invoked in the context of an implicit transaction, and the transaction context is propagated to external systems. The results are published to message broker channels for further processing. In case of any error, a worklist task is created.

Table 2-1 Important Artifacts in Package Explorer

`UseCaseWeb/src/alint/process/PurchaseOrderAggregatorJPD.java`	The JPD that handles purchase order processing. Aggregates the purchase order fulfillment results by subscribing to the message broker channels and confirms the order shipment. Then this JPD sends the PurchaseOrderStatus document as a *callback* to the client.
`ServiceAccess/BusinessServices/`	The business service endpoint for the two JPDs available in the UseCaseWeb folder. These business services use JPD transport.

7. Select **UseCaseWeb/alint.client.test/TestClient.java** from **Package Explorer**, right-click, and select **Run As** > **Run on Server** to publish UseCase and ServiceAccessConfiguration projects. In the **Define a New Server** dialog box, select **BEA WebLogic Server v10.0**, and click **Next**.
8. In the **BEA WebLogic Server v10.0** dialog box, browse to the directory where you have created the ALINT domain, and click **Next**. In the **Add and Remove Projects** dialog box, select **UseCase** and **ServiceAccessConfiguration** in the **Configured Projects** pane, and click **Finish**.

The test browser is displayed with the following URL: `http://localhost:7001/UseCaseWeb/alint/client/test/TestClient.jpd`. The UseCase and Service Applications are now deployed on WebLogic Server.

Note: When you publish the application, the following warning statements are displayed:

```
org.apache.commons.logging.LogConfigurationException:
org.apache.commons.logging

.LogConfigurationException: Invalid class loader hierarchy.  You have
more than

one version of 'org.apache.commons.logging.Log' visible, which is not
allowed. (

Caused by org.apache.commons.logging.LogConfigurationException:
Invalid class lo

ader hierarchy.  You have more than one version of
'org.apache.commons.logging.L

og' visible, which is not allowed.)
```

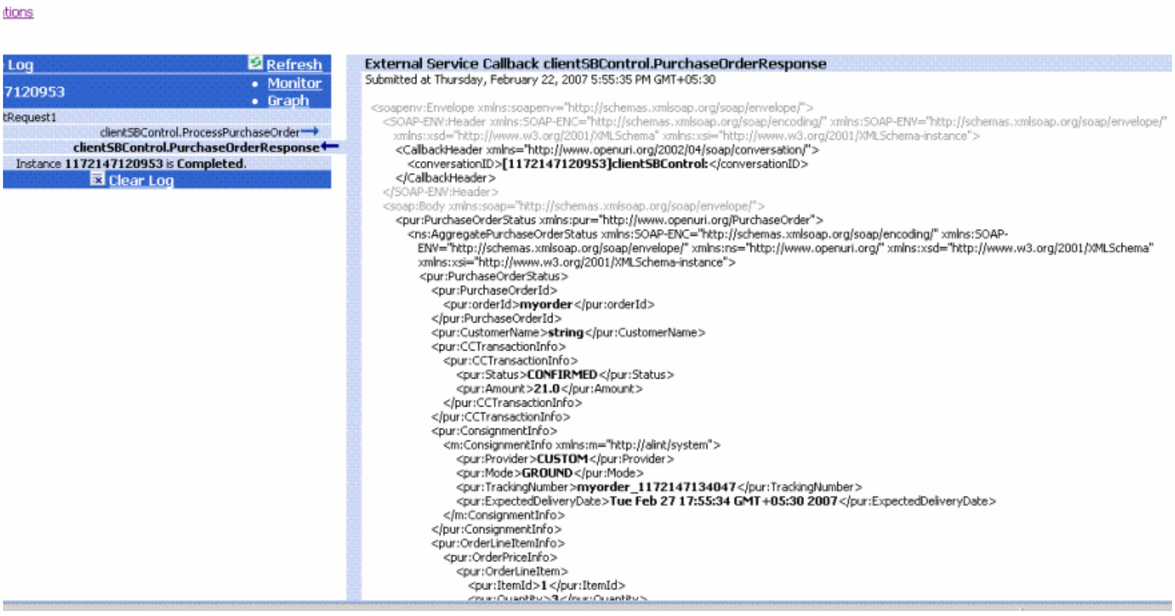
Testing the Application

The section describes how to test the sample application.

To Test the Sample Application

1. In the test browser `http://localhost:7001/UseCaseWeb/alint/client/test/TestClient.jpd`, click the **Test Soap** tab.
2. In the `input purchase order` XML file, enter the values as follows:
   - PurchaseOrderId (any purchase order id)
   - CreditCardNumber (1234567890123456)
   - ExpiryDate (08/08)
   - ItemId (2)
3. Click **clientRequest1**.
4. Click **Refresh** on the browser. The PurchaseOrderConfirmation is received as a *callback*, as shown in Figure 2-6. The processed purchase order is archived in the `C:\ALIntApp\PurchaseOrder\Archive` folder.

Figure 2-6 Purchase Order Callback



```
clientSBControl.ProcessPurchaseOrder
clientSBControl.PurchaseOrderResponse
Instance 1172147120953 is Completed.
```

External Service Callback clientSBControl.PurchaseOrderResponse
Submitted at Thursday, February 22, 2007 5:55:35 PM GMT+05:30

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <CallbackHeader xmlns="http://www.openuri.org/2002/04/soap/conversation/">
      <conversationID>[1172147120953]clientSBControl:</conversationID>
    </CallbackHeader>
  </SOAP-ENV:Header>
  <soap:Body xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <pur:PurchaseOrderStatus xmlns:pur="http://www.openuri.org/PurchaseOrder">
      <ns:AggregatePurchaseOrderStatus xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
        ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://www.openuri.org/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <pur:PurchaseOrderStatus>
          <pur:PurchaseOrderId>
            <pur:orderId>myorder</pur:orderId>
          </pur:PurchaseOrderId>
          <pur:CustomerName>string</pur:CustomerName>
          <pur:CCTransactionInfo>
            <pur:CCTransactionInfo>
              <pur:Status>CONFIRMED</pur:Status>
              <pur:Amount>21.0</pur:Amount>
            </pur:CCTransactionInfo>
          </pur:CCTransactionInfo>
          <pur:ConsignmentInfo>
            <m:ConsignmentInfo xmlns:m="http://alint/system">
              <pur:Provider>CUSTOM</pur:Provider>
              <pur:Mode>GROUND</pur:Mode>
              <pur:TrackingNumber>myorder_1172147134047</pur:TrackingNumber>
              <pur:ExpectedDeliveryDate>Tue Feb 27 17:55:34 GMT+05:30 2007</pur:ExpectedDeliveryDate>
            </m:ConsignmentInfo>
          </pur:ConsignmentInfo>
          <pur:OrderLineItemInfo>
            <pur:OrderPriceInfo>
              <pur:OrderLineItem>
                <pur:ItemId>1</pur:ItemId>
                  <pur:Quantity>3</pur:Quantity>
```

Part II Accessing ALSB Proxy Service and Exposing a JPD

This part describes how to access ALSB proxy service using ALSB Control and exposing a JPD using the Business Service Endpoints after you have done the following:

1. Creating the workspace **UseCase**, as in “Loading the Sample Application” on page 2-3 .
2. Importing the sample application as described in “Loading the Sample Application” on page 2-3.
3. Deleting the following artifacts:

- `PurchaseOrderFulfillmentJPD.biz` from UsecaseWeb\ServiceAccess\BusinessServices
- `ServiceProviderAccessContract.wsdl` from UsecaseWeb\UsecaseWeb\src\alint\process
- `ServiceProviderAccessControl.java` from UsecaseWeb\UsecaseWeb\src\alint\process folder

To rebuild ALSB Control and the business service endpoints, see the following topics:

- Chapter 4, “Creating a PurchaseOrder Fulfillment System and Invoking External Services.”
- Chapter 5, “Creating Business Service Endpoint for PurchaseOrderFulfillmentJPD.”

Part III Developing the PO Application System

This part describes how to develop the Purchase Order (PO) Processing System by creating the required artifacts from scratch.

It discusses the following topics:

Getting Started

This chapter describes the steps you need to perform before you start developing the Purchase Order application system.

Before You Begin

Before you start developing a Purchase Order (PO) Processing System, complete the following steps:

1. Create a new workspace UseCaseNew, as described in "Loading the Sample Application."
2. Create a Process Application. In the **Workspace Studio** menu, click **File** > **New** > **Other...** > **WebLogic Integration** > **Process Application**. Enter the following labels for the projects:
   a. UseCase
   b. UseCaseUtil
   c. UseCaseWeb

   Select **Add WebLogic Integration System and Control Schemas to Utility Project** to add the system schemas to **Utility** > **Schemas** folder.

   For information about creating a Process Application, see Tutorial: Designing Your First Business Process.
3. Copy files from `ALIntE2EUsecase\UsecaseApp\UseCaseWeb\schemas` into the Process `UseCaseWeb > schemas` folder in your newly created workspace.

Note: You have downloaded and extracted the sample application to a directory on your computer. Let us refer to this directory as *<TUTORIAL_ROOT>*. You must copy a few artifacts from the downloaded application into the new workspace that you have created. All paths mentioned in the following steps are relative to *<TUTORIAL_ROOT>*.

4. Create a new package called `alint.client` under **UseCaseWeb\src** in your workspace, and copy the *<TUTORIAL_ROOT>*\ **ALIntE2EUsecase\UsecaseApp\UseCaseWeb\src\alint\client** folder into process **UseCaseWeb** > **src** folder.

5. Create a new package called `alint.process.control` under **UseCaseWeb\src** in your workspace, and copy the contents of *<TUTORIAL_ROOT>***ALIntE2EUsecase\UsecaseApp\UsecaseWeb\src\alint\process\control** folder into that package.

6. Copy `alint.channel` from *<TUTORIAL_ROOT>***ALIntE2EUsecase\UsecaseApp\UsecaseUtil\src** to **UsecaseUtil\src** folder.

7. Create an ALSB project. Enter the following labels:

   a. ServiceAccess (Project name)

   b. ServiceAccessConfiguration (ALSB Configuration project)

8. Create the following folders in the ServiceAccess folder:

   - **Resources**
   - **ProxyServices**
   - **BusinessServices**

9. Copy *<TUTORIAL_ROOT>***ALIntE2EUsecase\UsecaseApp\ServiceAccess\Resources\PurchaseOrderProcessingService.wsdl** and **PurchaseOrder.xsd** into the **Resources** folder.

10. Copy *<TUTORIAL_ROOT>***ALIntE2EUsecase\UsecaseApp\ServiceAccess\BusinessServices\SavePOToNewDir.biz** into the **BusinessServices** folder.

CHAPTER 4

Creating a PurchaseOrder Fulfillment System and Invoking External Services

This chapter describes how to create an AquaLogic Service Bus (ALSB) Control called ServiceProviderAccessControl under the **alint.process**, and consume **ServiceProviderAccessContract.wsdl** from the File System.

This chapter discusses the following topics:

Creating AquaLogic Service Bus Control

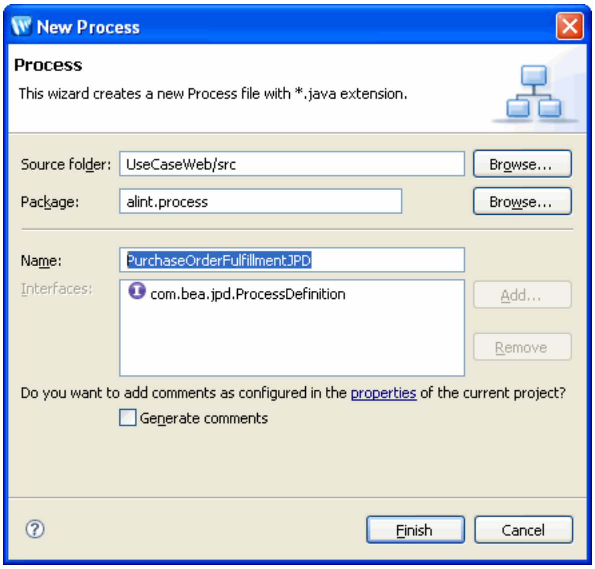
ALSB Control is based on the Beehive controls framework, which is used to invoke the ALSB proxies. The proxies have ALSB Control configured with support for security and transaction context propagation from WLI to ALSB.

External services like inventory system, credit card system, and shipping services are proxy services that can be invoked by the ALSB Control. Before you start creating the ALSB Control, create the PurchaseOrderFulfillmentJPD.

To create a Purchase OrderFulfillmentJPD and ALSB Control Using Data Palette

1. Switch to **Process** perspective from your current perspective in Workspace Studio. To select ALSB perspective, click **Windows** > **Open Perspective** > **Other...**.
2. Create a PurchaseOrderFulfillmentJPD. In **Package Explorer**, right-click on **src > alint.process**, and select **New** > **Other** > **WebLogic Integration** > **Process**. In the **New Process** dialog box, enter `PurchaseOrderFulfillmentJPD` as the **Name** of the Process. Type `alint.process` as the **Package**, if the package is not already set, as shown in Figure 4-1.

Figure 4-1 New Process



This wizard creates a new Process file with *.java extension.

Source folder: UseCaseWeb/src

Package: alint.process

Name: PurchaseOrderFulfillmentJPD

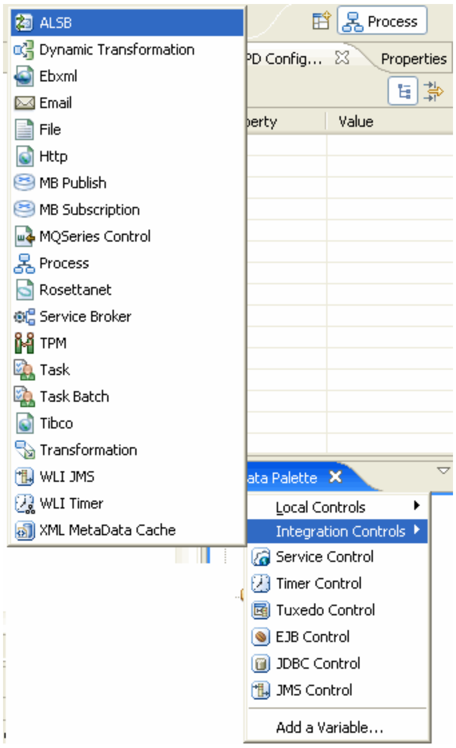
Interfaces: com.bea.jpd.ProcessDefinition

Do you want to add comments as configured in the properties of the current project?

☐ Generate comments

3. Click **Finish**.
4. In **Data Palette**, click the down arrow, and select **Integration Controls** > **ALSB**. Figure 4-2 shows how to select **ALSB** control from **Data Palette**.

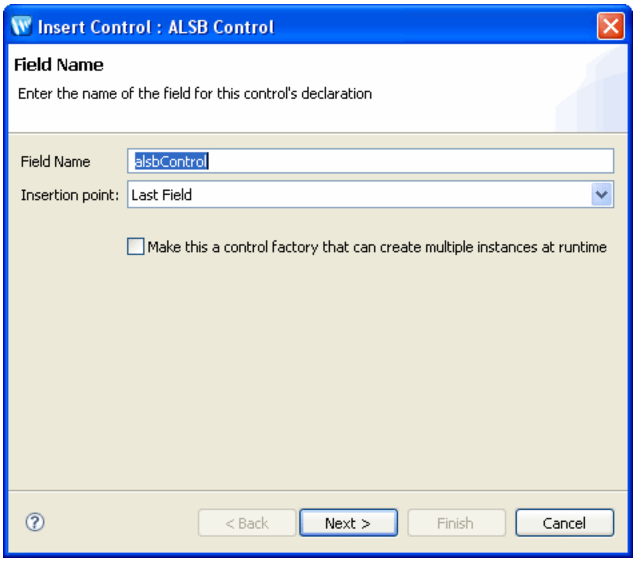
Figure 4-2 Data Palette



The **Insert Control : ALSB Control** wizard is displayed.

5. In the **Field Name** dialog box, enter the **Field Name**, as shown in Figure 4-3.

Figure 4-3 New ALSB Control



Field Name

Enter the name of the field for this control's declaration

Field Name: alsbControl

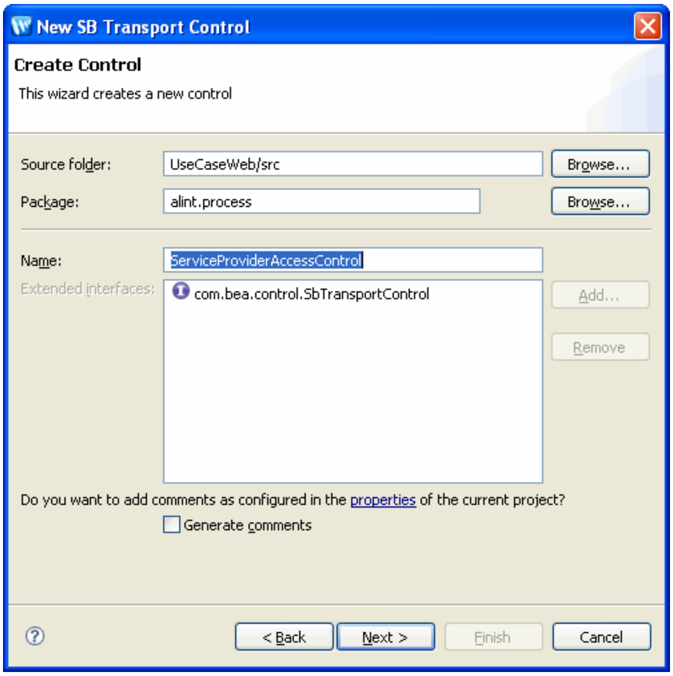
Insertion point: Last Field

☐ Make this a control factory that can create multiple instances at runtime

Click **Next**.

6. In the **Create Control** dialog box, enter the **Name** of the control as `ServiceProviderAccessControl`. Ensure that **alint.process** is the **Package**. Figure 4-4 shows the **Create Control** dialog box.

Figure 4-4 Create Control



New SB Transport Control

Create Control

This wizard creates a new control

Source folder: UseCaseWeb/src

Package: alint.process

Name: ServiceProviderAccessControl

Extended interfaces: com.bea.control.SbTransportControl

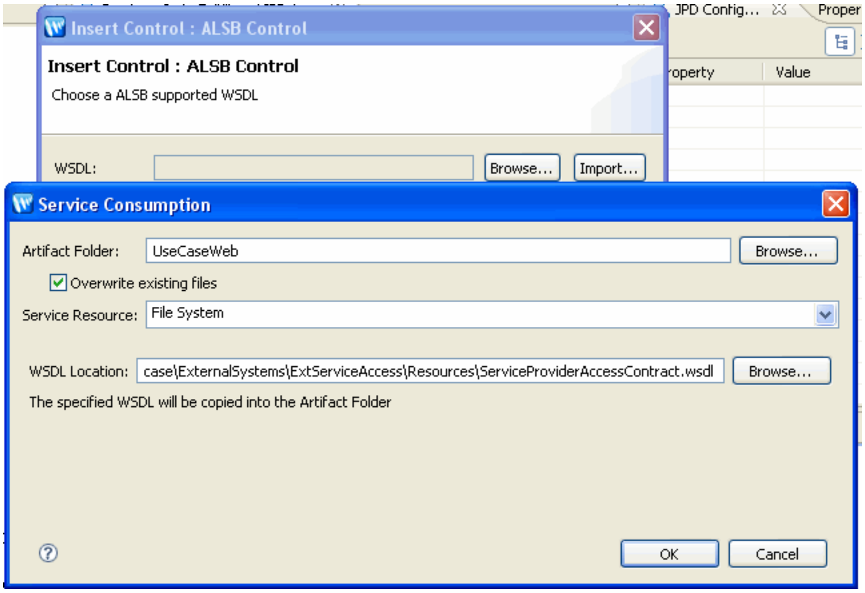
Do you want to add comments as configured in the properties of the current project?

Generate comments

Click **Next**.

7. Click **Import** to consume the WSDL from the filesystem, and perform the following:
   - **Artifact Folder**: Click **Browse** next to the **Artifact Folder**, and navigate to `UseCaseWeb\src\alint\process`.
   - **Service Resources**: Ensure that **Service Resource** is File System.
   - **WSDL Location**: Click **Browse** and navigate to `{TUTORIAL_ROOT}\ALIntE2EUsecase\ExternalSystems\ExtServiceAccess\Resources` folder and select the **ServiceProviderAccessContract.wsdl**.

Figure 4-5 Consuming WSDL


Insert Control : ALSB Control

Choose a ALSB supported WSDL

WSDL:

Service Consumption

Artifact Folder: UseCaseWeb

[✓] Overwrite existing files

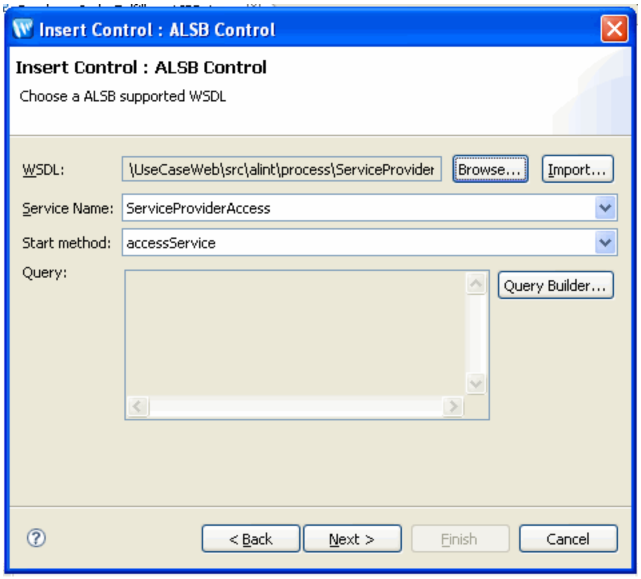
Service Resource: File System

WSDL Location: case\ExternalSystems\ExtServiceAccess\Resources\ServiceProviderAccessContract.wsdl

The specified WSDL will be copied into the Artifact Folder

Click **OK**. The values are populated, as shown in Figure 4-6.

Figure 4-6 WSDL Details


Insert Control : ALSB Control
Insert Control : ALSB Control
Choose a ALSB supported WSDL
Browse...
Import...
Query Builder...
< Back
Next >
Finish
Cancel

WSDL: \UseCaseWeb\src\alint\process\ServiceProvider

Service Name: ServiceProviderAccess

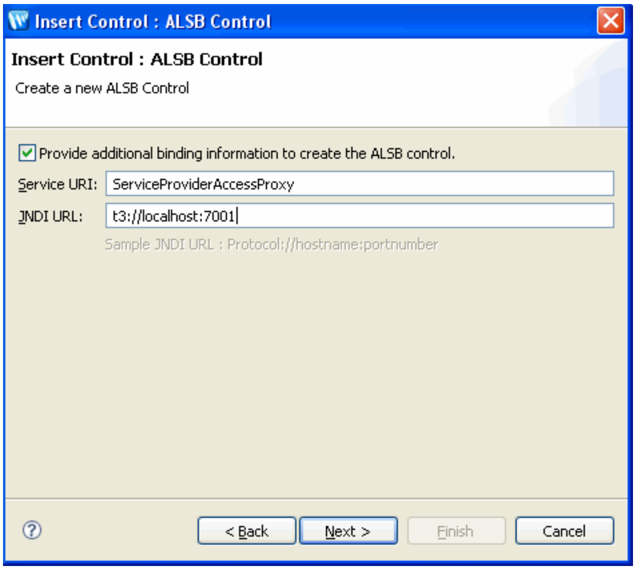
Start method: accessService

Query:

Click **Next**.

8. Select the **Provide additional binding information to create ALSB Control** check box. The other fields are enabled. In the **Service URI** field, enter `ServiceProviderAccessProxy`. On **JNDI URI**, enter `t3://localhost:7001` as shown in Figure 4-7.

Figure 4-7 Additional Binding for ALSB Control



Insert Control : ALSB Control

Create a new ALSB Control

☑ Provide additional binding information to create the ALSB control.

Service URI: ServiceProviderAccessProxy

JNDI URL: t3://localhost:7001

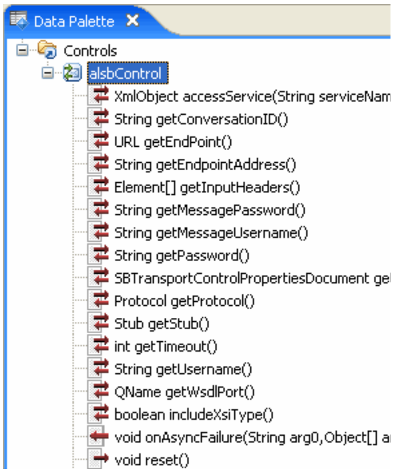
Sample JNDI URL : Protocol://hostname:portnumber

Click **Next**.

9. Select **Use XML Type Beans** in the **No Existing Types Found** dialog box. Click **Finish**.

   The ALSB Control is created in the **Controls** folder of the **Data Palette,** as shown in Figure 4-8.

Figure 4-8 ALSB Control in Data Palette



```
Controls
  alsbControl
    XmlObject accessService(String serviceNam
    String getConversationID()
    URL getEndPoint()
    String getEndpointAddress()
    Element[] getInputHeaders()
    String getMessagePassword()
    String getMessageUsername()
    String getPassword()
    SBTransportControlPropertiesDocument ge
    Protocol getProtocol()
    Stub getStub()
    int getTimeout()
    String getUsername()
    QName getWsdlPort()
    boolean includeXsiType()
    void onAsyncFailure(String arg0,Object[] a
    void reset()
```

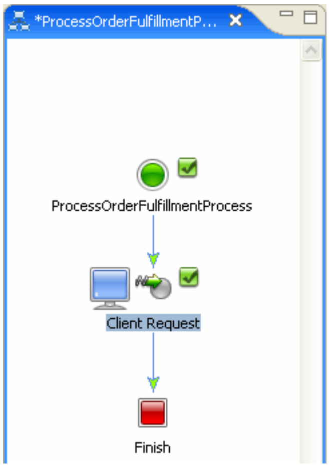
Designing PurchaseOrderFulfillmentJPD

PurchaseOrderFulfillmentJPD is used for state management and process orchestration.

To Design PurchaseOrderFulfillmentJPD

1. Drag and drop **Client Request** from **Node Palette** to the **Design** view of **PurchaseOrderFulfillment**, as shown in Figure 4-9.

Figure 4-9 Client Request Node



2. Double-click on the **Client Request** node, enter `fulfillPurchaseOrder` in **Method Name**, and click **Add... on the General Setting tab**.

3. In the dialog box that is displayed, add the following variable names:

   ```
   java.lang.String orderId
   org.openuri.purchaseOrder.OrderLineItemInfoDocument orderInfo
   org.openuri.purchaseOrder.CreditCardInfoDocument creditCardInfo
   ```

   Table 4-1 lists the parameter and types of these variable names.

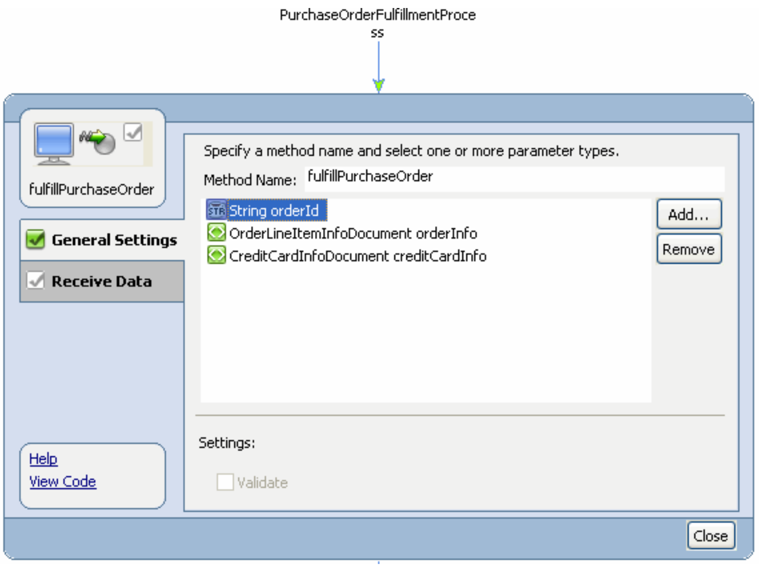
Table 4-1 Variable Names

Parameter Name	Type Name
orderId	`String`
orderInfo	`OrderLineItemInfo`
creditCardInfo	`CreditCardInfo`

For more information about creating variables for a JPD, see Creating Variables in *Guide to Building Business Process*.

4. Click **Close**. Figure 4-10 shows the parameters of the client request node.

Figure 4-10 Client Request Node - Variable Names



Specify a method name and select one or more parameter types.

Method Name: fulfillPurchaseOrder

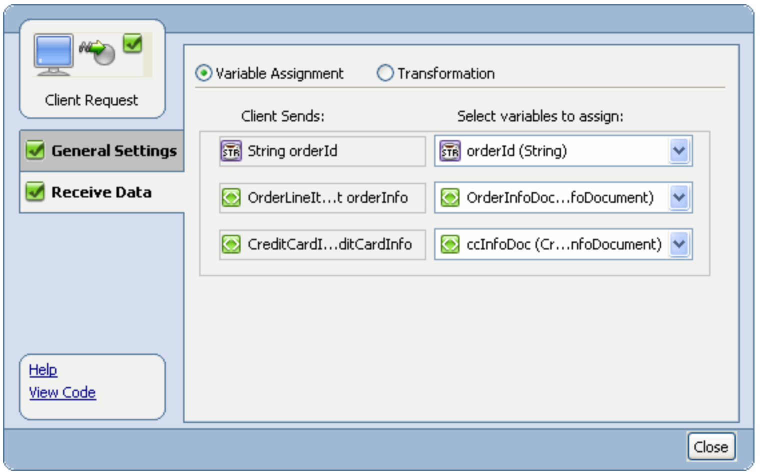
String orderId
OrderLineItemInfoDocument orderInfo
CreditCardInfoDocument creditCardInfo

Settings:

Validate

5. Click **Receive Data**, and assign the variables as shown in Figure 4-11

Figure 4-11 Assigning Variables



[✓ Variable Assignment] Transformation

Client Sends:	Select variables to assign:
String orderId	orderId (String)
OrderLineIt...t orderInfo	OrderInfoDoc...foDocument)
CreditCardI...ditCardInfo	ccInfoDoc (Cr...nfoDocument)

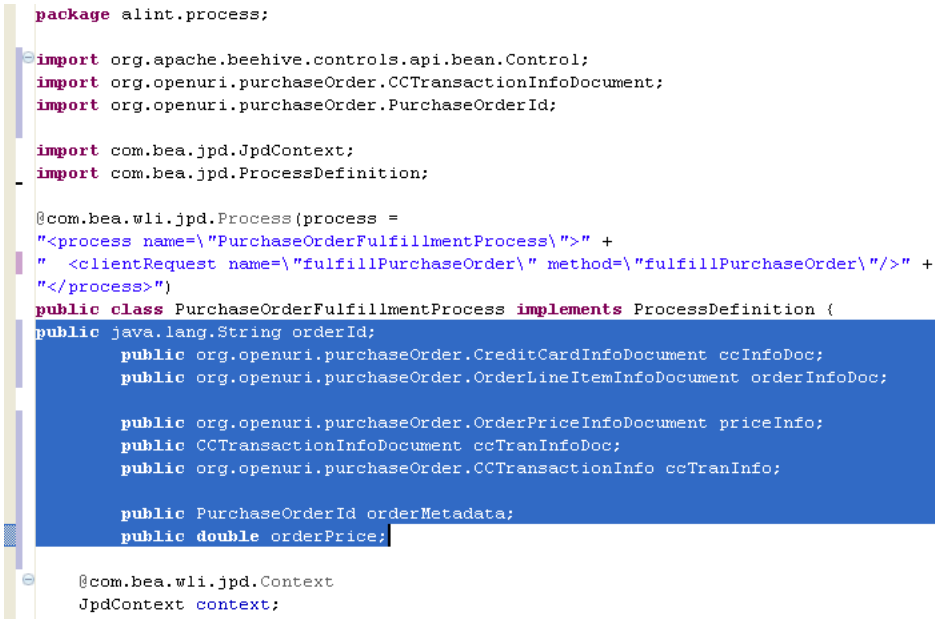
6. Create the following variables in the **Source** view of PurchaseOrderFulfillmentProcess:

```
public org.openuri.purchaseOrder.OrderPriceInfoDocument priceInfo;
public CCTransactionInfoDocument ccTranInfoDoc;
public org.openuri.purchaseOrder.CCTransactionInfo ccTranInfo;

public PurchaseOrderId orderMetadata;
public double orderPrice;
```

Figure 4-12 shows the variable names in **Source** view.

Note: In case of compilation errors, add the imports as shown in Figure 4-12 .

Figure 4-12 Create Variables in Source View



```
package alint.process;

import org.apache.beehive.controls.api.bean.Control;
import org.openuri.purchaseOrder.CCTransactionInfoDocument;
import org.openuri.purchaseOrder.PurchaseOrderId;

import com.bea.jpd.JpdContext;
import com.bea.jpd.ProcessDefinition;

@com.bea.wli.jpd.Process(process =
"<process name=\"PurchaseOrderFulfillmentProcess\">" +
"  <clientRequest name=\"fulfillPurchaseOrder\" method=\"fulfillPurchaseOrder\"/>" +
"</process>")
public class PurchaseOrderFulfillmentProcess implements ProcessDefinition {
public java.lang.String orderId;
        public org.openuri.purchaseOrder.CreditCardInfoDocument ccInfoDoc;
        public org.openuri.purchaseOrder.OrderLineItemInfoDocument orderInfoDoc;

        public org.openuri.purchaseOrder.OrderPriceInfoDocument priceInfo;
        public CCTransactionInfoDocument ccTranInfoDoc;
        public org.openuri.purchaseOrder.CCTransactionInfo ccTranInfo;

        public PurchaseOrderId orderMetadata;
        public double orderPrice;

    @com.bea.wli.jpd.Context
    JpdContext context;
```

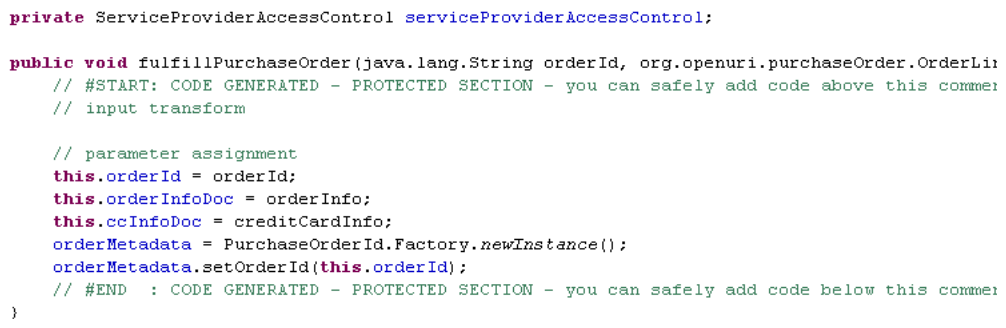
7. Edit the **fulfillPurchaseOrder** method, and add the following assignments:

```
orderMetadata = PurchaseOrderId.Factory.newInstance();
orderMetadata.setOrderId(this.orderId);
```

Note: `orderMetadata` is required to publish the results after the inventory and the credit card systems are invoked.

Figure 4-13 shows the fulfillPurchaseOrder method in **Source** view.

Figure 4-13 fulfillPurchaseOrder Method in Source View



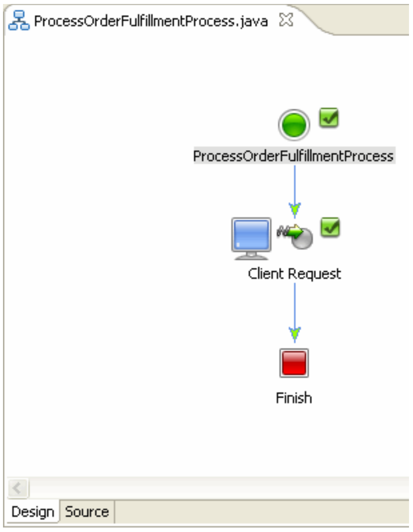
```
private ServiceProviderAccessControl serviceProviderAccessControl;

public void fulfillPurchaseOrder(java.lang.String orderId, org.openuri.purchaseOrder.OrderLi
    // #START: CODE GENERATED - PROTECTED SECTION - you can safely add code above this comme
    // input transform

    // parameter assignment
    this.orderId = orderId;
    this.orderInfoDoc = orderInfo;
    this.ccInfoDoc = creditCardInfo;
    orderMetadata = PurchaseOrderId.Factory.newInstance();
    orderMetadata.setOrderId(this.orderId);
    // #END  : CODE GENERATED - PROTECTED SECTION - you can safely add code below this comme
}
```

8. Enter the following constants for the credit card and inventory services:

```
private static final String CREDIT_CARD_SERVICE = "CreditCardService";
private static final String INVENTORY_SERVICE = "InventoryService";
```

You have now added the clientRequest node and the required variables to the ProcessOrderFulfillmentJPD, as shown in Figure 4-14.

Figure 4-14 ProcessOrderFulfillmentJPD


ProcessOrderFulfillmentProcess.java
ProcessOrderFulfillmentProcess
Client Request
Finish
Design
Source

Invoking External Services

This section describes how to invoke Inventory system and credit card systems:

- "To Invoke the Inventory System"
- "To Invoke the Credit Card System"

To Invoke the Inventory System

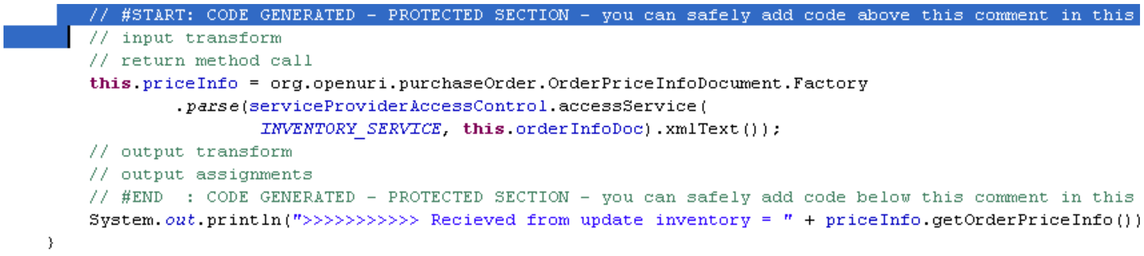
1. In the **Data Palette**, expand **alsbControl**, and drag and drop **XmlObject accessService(String serviceName_arg,XmlObject_Any_arg)** into the PurchaseOrderFulfillmentJPD **Design** view.

2. Double-click **accessService** node, and rename node to `updateInventory`. Right-click this **updateInventory** node, and select **View Code**. Copy the following code in the **Source** view:

```
public void serviceProviderAccessControlAccessService() throws
Exception {

       // #START: CODE GENERATED - PROTECTED SECTION - you can safely
add code above this comment in this method. #//

        // input transform

        // return method call

        this.priceInfo =
org.openuri.purchaseOrder.OrderPriceInfoDocument.Factory.parse(servi
ceProviderAccessControl.accessService(INVENTORY_SERVICE,
this.orderInfoDoc).xmlText());

        // output transform

        // output assignments

       // #END  : CODE GENERATED - PROTECTED SECTION - you can safely
add code below this comment in this method. #//

System.out.println(“>>>>>>>>Received from update inventory = ” +
priceInfo.getOrderPriceInfo ());

}
```

   Figure 4-15 shows the code in **Source** view.

Figure 4-15 updateInventory Code



```
    // #START: CODE GENERATED - PROTECTED SECTION - you can safely add code above this comment in this
    // input transform
    // return method call
    this.priceInfo = org.openuri.purchaseOrder.OrderPriceInfoDocument.Factory
            .parse(serviceProviderAccessControl.accessService(
                    INVENTORY_SERVICE, this.orderInfoDoc).xmlText());
    // output transform
    // output assignments
    // #END  : CODE GENERATED - PROTECTED SECTION - you can safely add code below this comment in this
    System.out.println(">>>>>>>>>>> Recieved from update inventory = " + priceInfo.getOrderPriceInfo())
}
```

To Invoke the Credit Card System

1. From **Package Explorer**, expand **alint.process**, and drag and drop **ServiceProviderAccessControl** into the **Data Palette** > **Controls** folder.

2. Expand **ServiceProviderAccessControl** from the **Data Palette**, and drag and drop **XMlObject accessService(StringserviceName_arg,xmlObject_Any_arg)** into PurchaseOrderFulfillmentProcess **Design** view.

3. Double-click **accessService** node, and rename node to `transferMoney`. Right-click this **transferMoney** node, and select **View Code**. Copy the following code in the **Source** view:

```
public void serviceProviderAccessControlAccessService1() throws
Exception {

       // #START: CODE GENERATED - PROTECTED SECTION - you can safely
add code above this comment in this method. #//

        // input transform

        // return method call

        this.opReceived =
serviceProviderAccessControl.accessService(this.CREDIT_CARD_SERVICE,
this.ccInfoDoc);

        // output transform

        // output assignments

       // #END  : CODE GENERATED - PROTECTED SECTION - you can safely
add code below this comment in this method. #//
```

```
            this.ccTranInfo = 
org.openuri.purchaseOrder.CCTransactionInfoDocument.Factory.parse(op
Received.xmlText()).getCCTransactionInfo();


            System.out.println(">>>>>>>>>>>> Received from transfer money 
= " + ccTranInfo.getStatus());
        }
```

Note: Here you are not calculating the order price that is being passed to the external credit card system. You will calculate it in Chapter 7, "Aggregating the Purchase Order Fulfillment Results.".

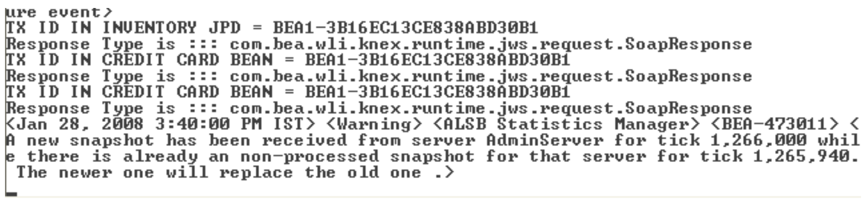
Verifying Transaction Context Propagation

To Verify Transaction Context Propagation

1. Right-click **PurchaseOrderFulfillmentJPD.java** in the **src** > **alint.process** folder in **Package Explorer**, and select **Run As** > **Run on Server**.
2. The following URL: `http://localhost:7001/UseCaseWeb/alint/process/PurchaseOrderFulfillmentJPD.jpd` is displayed in the internet browser.
3. Click the **Test Soap** tab. In the `fulfillPurchaseOrder` XML file, enter the values as follows:
   - PurchaseOrderId (any purchase order id)
   - CreditCardNumber (1234567890123456)
   - ExpiryDate (08/08)
   - ItemId (2)
4. Click **fulfillPurchaseOrder**.

You can now verify the debug messages on the server console that are printed when the purchase order is processed. Note that the Transaction ID is the same in inventory and credit card system debug statements, indicating that they are invoked in context of the same transaction. See Figure 4-16.

Figure 4-16 Debug Messages on Server Console



```
ure event>
TX ID IN INVENTORY JPD = BEA1-3B16EC13CE838ABD30B1
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
TX ID IN CREDIT CARD BEAN = BEA1-3B16EC13CE838ABD30B1
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
TX ID IN CREDIT CARD BEAN = BEA1-3B16EC13CE838ABD30B1
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
<Jan 28, 2008 3:40:00 PM IST> <Warning> <ALSB Statistics Manager> <BEA-473011> <
A new snapshot has been received from server AdminServer for tick 1,266,000 whil
e there is already an non-processed snapshot for that server for tick 1,265,940.
 The newer one will replace the old one .>
_
```

The transaction context is propagated from the PurchaseOrderFulfillmentJPD to the external systems - credit card service and inventory service.

Business processes in WebLogic Integration are transactional in nature. Every step of a process is executed within the context of a transaction. A transaction ensures that one or more operations execute as an atomic unit of work.

The ALSB control provides an option to propagate the transaction context of the JPD to AquaLogic Service Bus, which can be specified using the annotation provided for the transactional purpose.

Similarly, JPD transport supports optional transaction propagation from ALSB to WLI, which depends on the QOS parameters and propagate transaction attribute in the JPD transport configuration.

Securing External Services

ALINT supports the propagation of security context. By default, the current authenticated subject (associated with the executing thread) is propagated to ALSB when no principal/credential is specified as part of the `SBTransport` annotation in the ALSB Control (`ServiceProviderAccessControl.java`).

Note: Switch to the *External Systems* workspace to secure the external systems. To switch to a new workspace, select **File** > **Switch Workspace...**.

The external systems, credit card system and inventory system, are protected. The InventorySystem and CreditCardSystem JPDs are available in **Package Explorer** at the following locations:

- `ExtSystemWeb\src\alint\system\InventorySystemJPD.java`
- `ExtSystemWeb\src\alint\system\CreditCardSystemJPD.java`

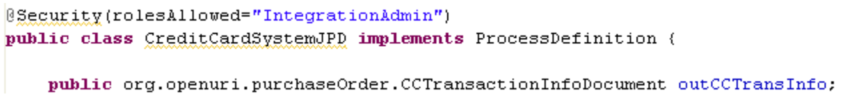
You can provide permissions to the administrator to execute the InventorySystem and CreditCardSystem JPDs.

Notes: In the Source view, `@Security(rolesAllowed="IntegrationAdmin")` is commented. To protect these JPDs from being accessed, remove the comments (`//`) from this annotation in the Source views of the JPDs.

After you have removed the comment tag, only users with `role = IntegrationAdmin` can execute the InventorySystem and CreditCardSystem JPDs.

The JPDs are protected using the `@Security` annotation, as shown in Figure 4-17.

Figure 4-17 Securing the External Services



```
@Security(rolesAllowed="IntegrationAdmin")
public class CreditCardSystemJPD implements ProcessDefinition {

    public org.openuri.purchaseOrder.CCTransactionInfoDocument outCCTransInfo;
```

After modifying the changes to the JPDs, do the following:

1. In the **Package Explorer**, go the **ExtSystemWeb** > **src** > **alint** > **system**, right-click on any of the JPDs, for example, **InventorySystemJPD**, and select **Run As** > **Run on Server**:
2. Accept the default values, and click **Finish**.

   Switch to the *UseCase* workspace that you were using. To switch to a new workspace, select **File** > **Switch Workspace...**.

The client is authenticated using the `basic auth` method when the `PurchaseOrderProcessingProxy` is invoked. The security context is propagated to the WLI layer (PurchaseOrderFulfillmentJPD) to ALSB (ServiceProviderAccessProxy) to the external systems (inventory and credit card systems).

You can review PurchaseOrderFulfillmentJPD to verify that you do not provide any username and password to access the external services. For information about verifying security context propagation, see “Verifying Security Context Propagation.”

Creating Business Service Endpoint for PurchaseOrderFulfillmentJPD

This chapter describes how to rebuild business service endpoints for the PurchaseOrderFulfillment JPD. This chapter discusses the following topics:

- "Creating the Business Service Endpoint for PurchaseOrderFulfillmentJPD"

Creating the Business Service Endpoint for PurchaseOrderFulfillmentJPD

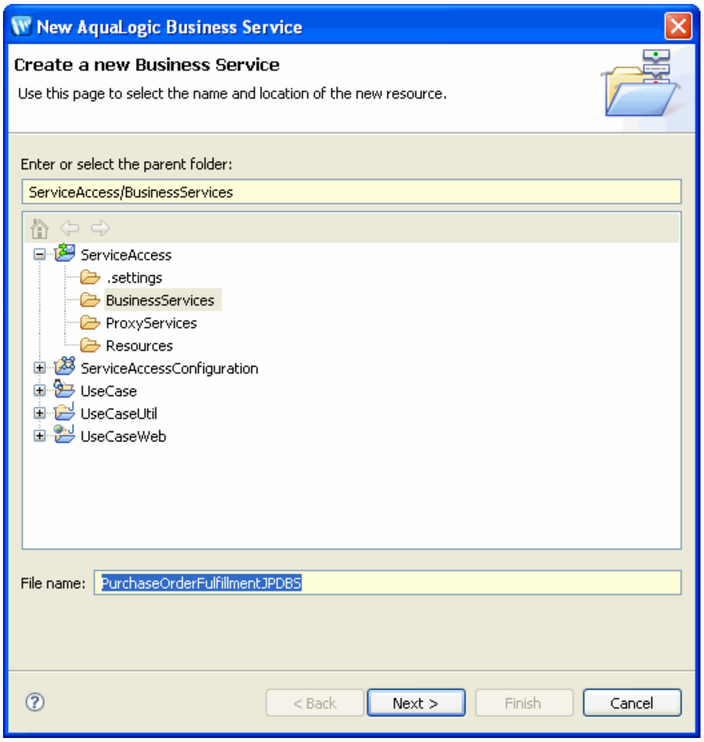
The business service end points are used by the ALSB proxy services to invoke the Purchase Order Fulfillment JPD for fulfilling the order.

To Create the Business Service Endpoint

1. Switch to ALSB perspective from your current perspective in Workspace Studio. To select ALSB perspective, in the **BEA Workshop for WebLogic** window menu, and click **Windows** > **Open Perspective** > **Other...**.

   Select **AquaLogic Service Bus** in the **Open Perspective** dialog box.
2. In the **Project Explorer** pane, select **ServiceAccess**, and expand **Business Services**.
3. Right-click **BusinessServices**, and select **New** > **Business Service**.
4. In the **Create a new Business Service** dialog box, enter the **File Name** as `PurchaseOrderFulfillmentJPDBS`, and click **Next**.

Figure 5-1 Create a Business Service


New AquaLogic Business Service

Create a new Business Service

Use this page to select the name and location of the new resource.

Enter or select the parent folder:

ServiceAccess/BusinessServices

ServiceAccess
.settings
BusinessServices
ProxyServices
Resources
ServiceAccessConfiguration
UseCase
UseCaseUtil
UseCaseWeb

File name: PurchaseOrderFulfillmentJPDBS

5. In the **Create Business Service - General Configuration (Service Access/Business Services/)** dialog box, select the **WSDL Web Service** Service Type, and click **Browse**. The **Select a WSDL** dialog box is displayed.

6. Select **Service Access**, and click **Consume**. In the **Service Consumption** dialog box, browse to **ServiceAccess** > **Resources,** and select `ServiceAccess\Resources` as the **Artifact folder**.

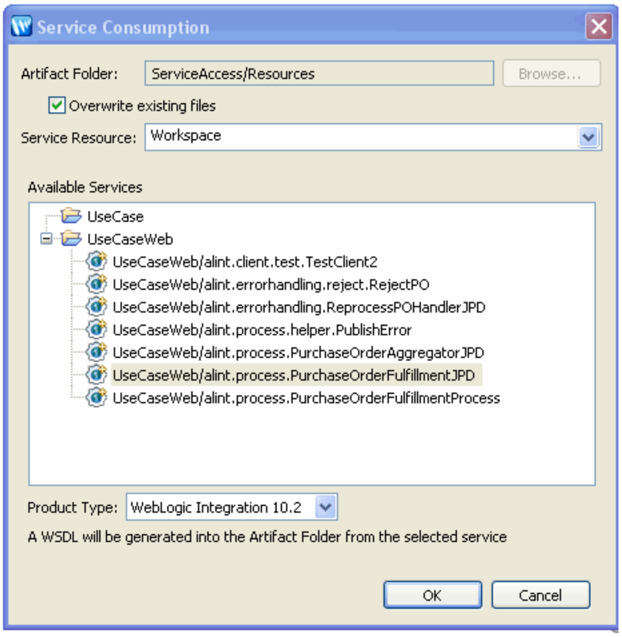
7. From the **Service Resource** drop-down list, select **Workspace**.

8. Select **WebLogic Integration 10.2** from the **Product Type** drop-down list. The **Service Consumption Status** dialog box is displayed.

9. Select **`UseCaseWeb/alint.process.PurchaseOrderFulfillmentJPD`**, and click **OK**.

   Figure 5-2 shows the **Service Consumption** dialog box.

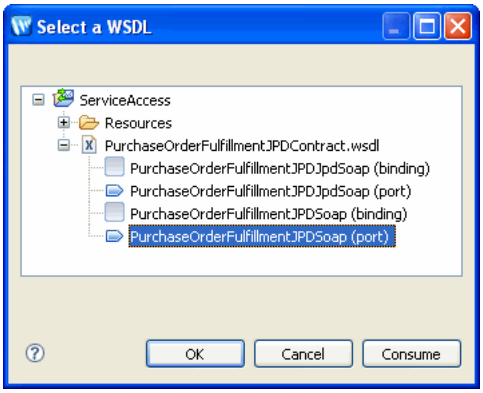
Figure 5-2 Service Consumption


Service Consumption
Artifact Folder: ServiceAccess/Resources
Browse...
Overwrite existing files
Service Resource: Workspace
Available Services
UseCase
UseCaseWeb
UseCaseWeb/alint.client.test.TestClient2
UseCaseWeb/alint.errorhandling.reject.RejectPO
UseCaseWeb/alint.errorhandling.ReprocessPOHandlerJPD
UseCaseWeb/alint.process.helper.PublishError
UseCaseWeb/alint.process.PurchaseOrderAggregatorJPD
UseCaseWeb/alint.process.PurchaseOrderFulfillmentJPD
UseCaseWeb/alint.process.PurchaseOrderFulfillmentProcess
Product Type: WebLogic Integration 10.2
A WSDL will be generated into the Artifact Folder from the selected service
OK
Cancel

Click **OK** in the **Service Consumption Status** dialog box.

10. In the **Select a WSDL** dialog box, expand **Resources**, select and expand **`PurchaseOrderFulfillmentJPDContract.wsdl`**, and select **PurchaseOrderFulfillmentJPDSoap(port)**, click **OK**, as shown in Figure 5-2.

Figure 5-3 Select a WSDL


Select a WSDL
ServiceAccess
Resources
PurchaseOrderFulfillmentJPDContract.wsdl
PurchaseOrderFulfillmentJPDJpdSoap (binding)
PurchaseOrderFulfillmentJPDJpdSoap (port)
PurchaseOrderFulfillmentJPDSoap (binding)
PurchaseOrderFulfillmentJPDSoap (port)
OK
Cancel
Consume

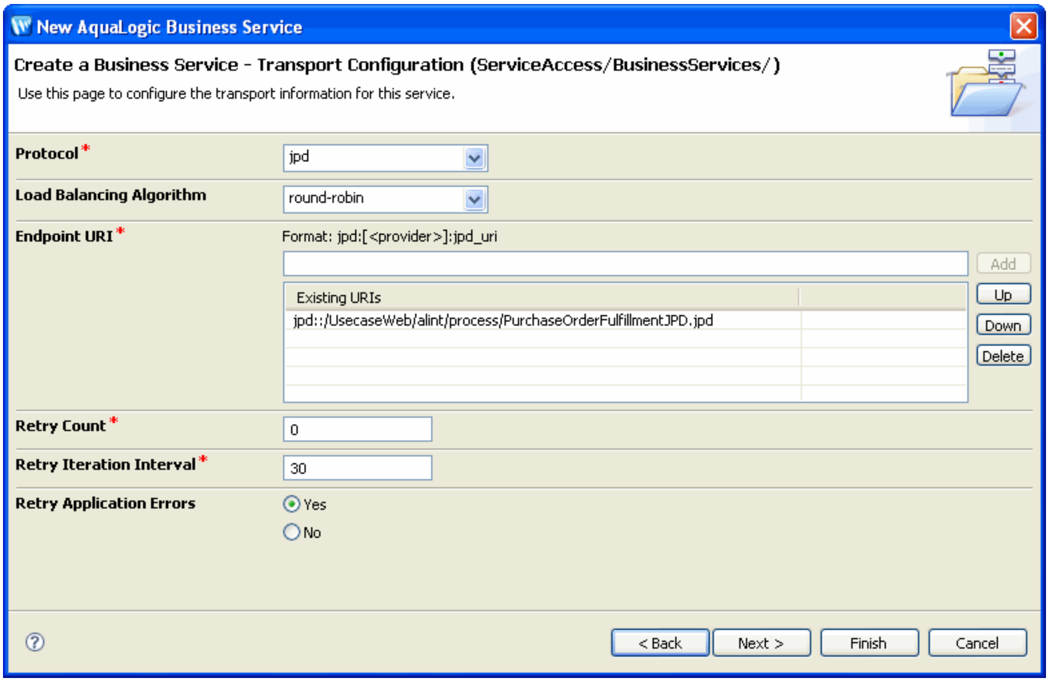
11. Click **Next** in the **Create a Business Service - Transport Configuration (ServiceAccess/Business Services)** dialog box, and ensure the following:

   – The **Protocol** is set to jpd.

   – The **Endpoint URI** is: `jpd::/UseCaseWeb/alint/process/PurchaseOrderFulfillmentJPD.jpd`. Click **Add**.

12. Click **Next**. Figure 5-4 shows the default values selected.

Figure 5-4 Selecting Protocol and Endpoint URIs



Create a Business Service - Transport Configuration (ServiceAccess/BusinessServices/)

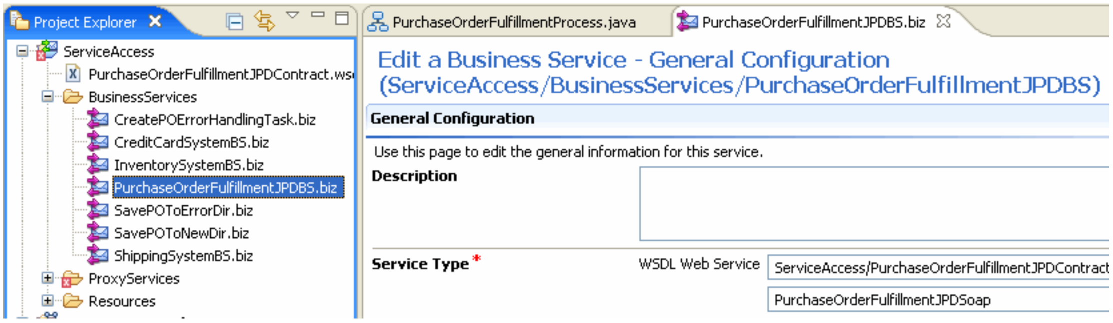
Use this page to configure the transport information for this service.

Protocol *	jpd
Load Balancing Algorithm	round-robin
Endpoint URI *	Format: jpd:[<provider>]:jpd_uri
	Existing URIs: jpd::/UsecaseWeb/alint/process/PurchaseOrderFulfillmentJPD.jpd
Retry Count *	0
Retry Iteration Interval *	30
Retry Application Errors	[✓ Yes] No

13. In the **JPD Transport Configuration** dialog box, accept the default values, Click **Finish**.

   The `PurchaseOrderFulfillmentJPDBS.biz` file is created in the **Business Services** folder in **Project Explorer**. Figure 5-5 shows the business service that is created.

Figure 5-5 Business Service in Project Explorer



Project Explorer
ServiceAccess
PurchaseOrderFulfillmentJPDContract.ws
BusinessServices
CreatePOErrorHandlingTask.biz
CreditCardSystemBS.biz
InventorySystemBS.biz
PurchaseOrderFulfillmentJPDBS.biz
SavePOToErrorDir.biz
SavePOToNewDir.biz
ShippingSystemBS.biz
ProxyServices
Resources
PurchaseOrderFulfillmentProcess.java
PurchaseOrderFulfillmentJPDBS.biz

Edit a Business Service - General Configuration (ServiceAccess/BusinessServices/PurchaseOrderFulfillmentJPDBS)

General Configuration

Use this page to edit the general information for this service.

Description

Service Type * WSDL Web Service ServiceAccess/PurchaseOrderFulfillmentJPDContract

PurchaseOrderFulfillmentJPDSoap

In this chapter, you created the business service end point to invoke the PurchaseOrderFulfillment JPD. In the next chapter, you will create a proxy service to invoke this business service end point.

CHAPTER 6

Creating Proxy Service to Invoke a JPD

This chapter describes how to create proxy services for external clients.

The PurchaseOrderProcessingService proxy invokes the PurchaseOrderFulfillmentJPD via the PurchaseOrderFulfillmentJPD business service endpoint for fulfilling the purchase order. This chapter discusses the following topics:

- "Creating the PurchaseOrderProcessingService Proxy"

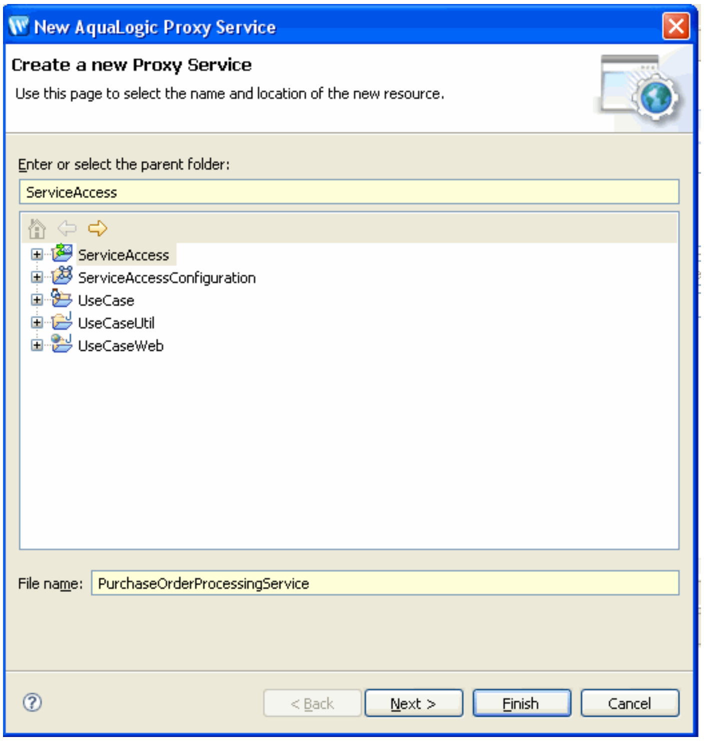
Creating the PurchaseOrderProcessingService Proxy

This section describes how to create PurchaseOrderProcessService proxy service from the **Service Access** folder in **Project Explorer**.

To Create PurchaseOrderProcessServiceProxy.proxy

1. In **Project Explorer**, select **Service Access**, right-click on **ProxyServices**, and select **New** > **Proxy Service**. The **New AquaLogic Proxy Service** wizard is displayed.
2. In the **Create a new Proxy Service** dialog box, enter `PurchaseOrderProcessingService` as the **File Name**.

Figure 6-1 Creating a Proxy Service

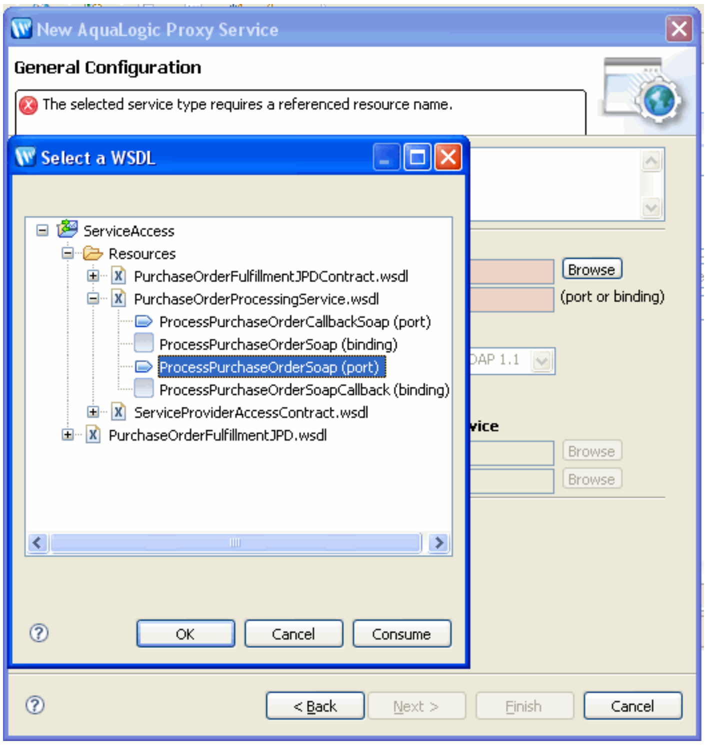

New AquaLogic Proxy Service
Create a new Proxy Service
Use this page to select the name and location of the new resource.
Enter or select the parent folder:
ServiceAccess
ServiceAccess
ServiceAccessConfiguration
UseCase
UseCaseUtil
UseCaseWeb
File name:
PurchaseOrderProcessingService
< Back
Next >
Finish
Cancel

Click **Next**.

3. In the **General Configurations** dialog box, select **WSDL Web Service**, click **Browse**, expand **ServiceAccess** > **Resources**, and select **PurchaseOrderProcessingService.wsdl**.

4. Expand **PurchaseOrderProcessingService.wsdl**, and select **ProcessPurchaseOrderSoap (port)** in the **Select a WSDL** dialog box, as shown in Figure 6-2.

Figure 6-2 Select a WSDL


New AquaLogic Proxy Service

General Configuration

The selected service type requires a referenced resource name.

Select a WSDL

ServiceAccess
Resources
PurchaseOrderFulfillmentJPDContract.wsdl
PurchaseOrderProcessingService.wsdl
ProcessPurchaseOrderCallbackSoap (port)
ProcessPurchaseOrderSoap (binding)
ProcessPurchaseOrderSoap (port)
ProcessPurchaseOrderSoapCallback (binding)
ServiceProviderAccessContract.wsdl
PurchaseOrderFulfillmentJPD.wsdl

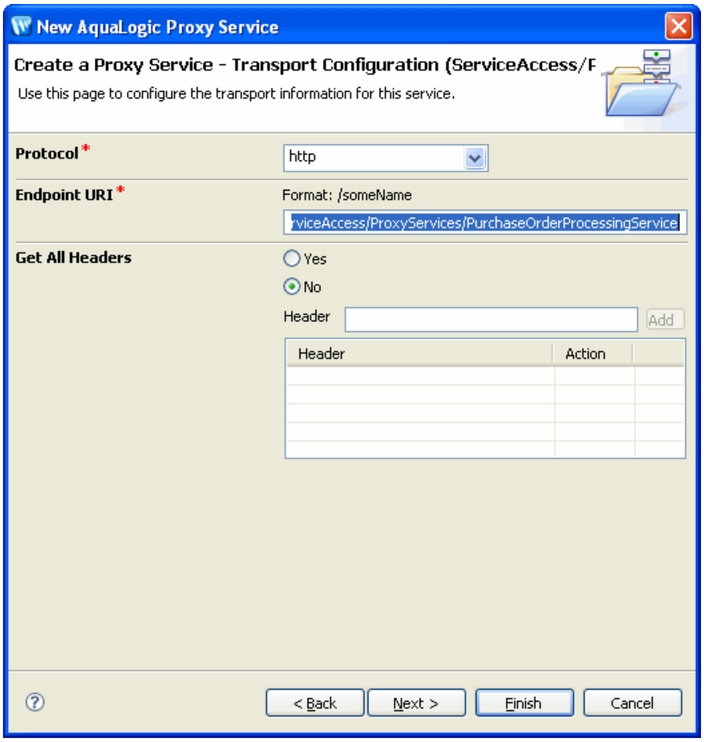
Click **OK**.

The **WSDL Web Service** and ports are selected. Click **Next**. The **Transport Configuration** dialog box is displayed.

5. Ensure that the **Protocol** is **`http`**, and the **Endpoint URI** is **`/ServiceAccess/ProxyServices/PurchaseOrderProcessingService`**, as shown in Figure 6-3.

6. Click **Next**.

Figure 6-3 Protocol and EndPoint URI


New AquaLogic Proxy Service

Create a Proxy Service - Transport Configuration (ServiceAccess/F

Use this page to configure the transport information for this service.

Protocol*: http

Endpoint URI*: Format: /someName

rviceAccess/ProxyServices/PurchaseOrderProcessingService

Get All Headers: Yes [✓ No]

Header

7. In the **Create a Proxy - HTTP Transport Configuration** dialog box, do the following:
   - Select the **HTTPS required** check box.

     This step is required to ensure secure data transmission.
   - Select **Basic** in **Authentication** pane.

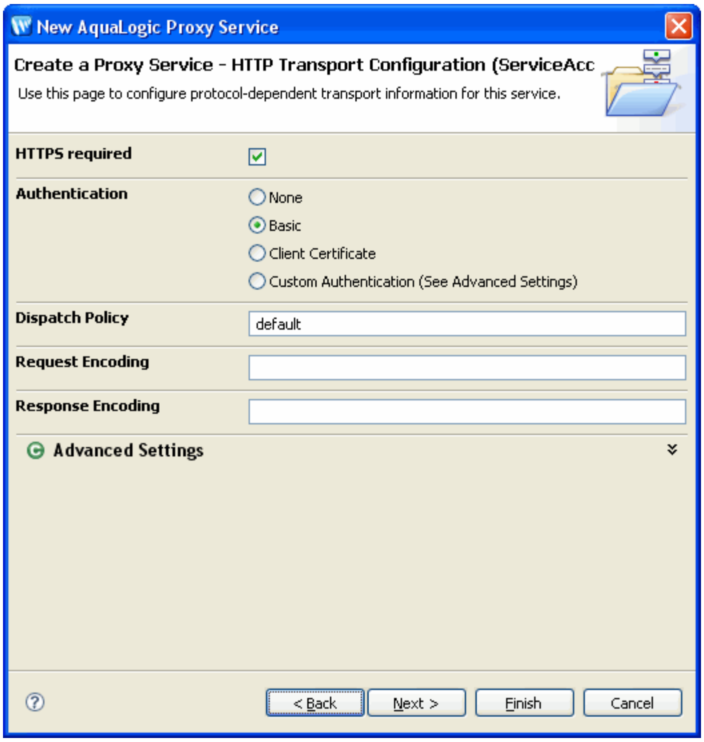
     This step ensures the proxy is protected and client needs to be authenticated to access the proxy.

     Click **Next**.

Note: The security context is propagated to the WLI layer (PurchaseOrderFulfillmentJPD), to ALSB (ServiceProviderAccessProxy), to the external systems (inventory and credit card systems), so that no authentication is required while accessing the external systems from the PurchaseOrderFulfillmentJPD.

Figure 6-4 shows the **Create a Proxy - HTTP Transport Configuration** dialog box.

Figure 6-4 Select HTTPS Options


New AquaLogic Proxy Service

Create a Proxy Service - HTTP Transport Configuration (ServiceAcc

Use this page to configure protocol-dependent transport information for this service.

HTTPS required	✓
Authentication	None [✓ Basic] Client Certificate Custom Authentication (See Advanced Settings)
Dispatch Policy	default
Request Encoding	
Response Encoding	

Advanced Settings

8. Accept the default values in the **Operation Selection Configuration** dialog box, and click **Finish**.

   PurchaseOrderProcessingServiceProxy.proxy is now available under the **ProxyServices** folder in **Project Explorer**.

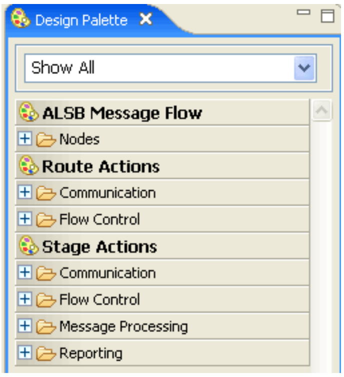
After you have created the proxy service, you must design the pipeline for the proxy.

Designing a ProxyService Pipeline

To design the proxy service pipeline, switch to the ALSB perspective. In the **Design** view of the **PurchaseOrdeProcessingService.proxy**, click **Message Flow**. The **Design Palette** is activated.

Figure 6-5 shows the **Design Palette** in ALSB perspective.

Figure 6-5 Design Palette



For more information about ALSB Design Palette, see Using the AquaLogic Service Bus IDE.

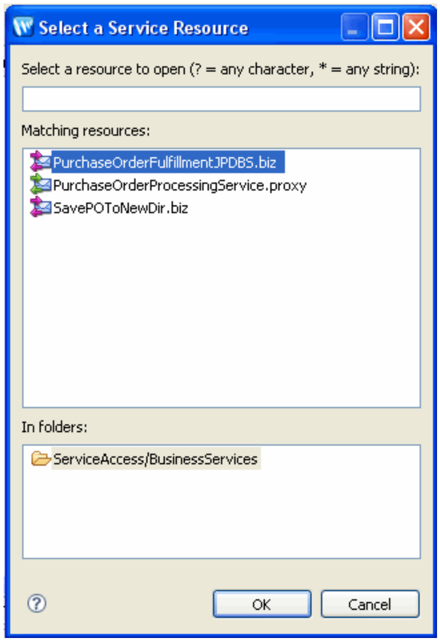
To Design the Proxy Service Pipeline

1. In the **Design** view of `PurchaseOrderProcessingService.proxy`, click **Message Flow**.
2. From the **Design Palette**, expand **Nodes**, drag and drop a **Pipleline Pair** under `PurchaseOrderProcessingService.proxy` in the **Design** view.

Note: A **PipelinePair** node consists of a **Request** pipeline and a **Response** pipeline. Pipelines can include one or more stages, which in turn include actions.

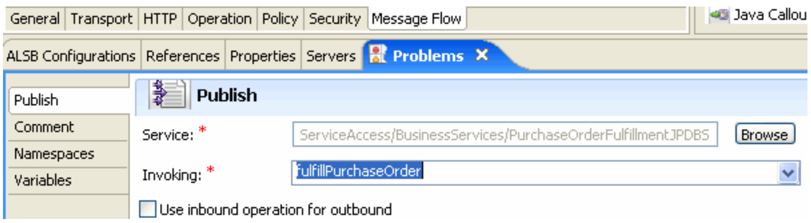
3. Expand **Nodes**, drag and drop **Stage** node, under **Request** in **PipelinePairNode**. Name this Pipeline as `Process Purchase Order`.
4. From the **Design** palette, expand **Stage Actions** > **Communications**, and select **Publish**. Drag and drop **Publish** node under **Stage** node.
5. In the **Properties** tab of the **Publish** node, click **Browse** to select a service. From the **Select a Service Resource** dialog box, select **PurchaseOrderFulfillmentJPDBS.biz**, as you are now publishing to the PurchaseOrderFulfillmentJPD. Figure 6-6 shows the **Service Resource** dialog box.

Figure 6-6 Select a Service


Select a Service Resource
Select a resource to open (? = any character, * = any string):
Matching resources:
PurchaseOrderFulfillmentJPDBS.biz
PurchaseOrderProcessingService.proxy
SavePOToNewDir.biz
In folders:
ServiceAccess/BusinessServices
OK
Cancel

Select **fulfillPurchaseOrder** from the **Invoking** drop-down list. The **Properties** tab of the Publish node is as shown in Figure 6-7.

Figure 6-7 Publish Node - Properties



Service: * ServiceAccess/BusinessServices/PurchaseOrderFulfillmentJPDBS

Invoking: * fulfillPurchaseOrder

Use inbound operation for outbound

6. From the **Design** palette, expand **Stage Actions** > **Message Processing**, and select **Assign** node. Drag and drop **Assign** node into the **Request** pipeline.

7. In the **Properties** tab of the **Assign** node, click **Expression**, and enter the following code in the **XQuery Editor** dialog box that is displayed:

```
<soap:Body xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:open="http://www.openuri.org/"
xmlns:pur="http://www.openuri.org/PurchaseOrder">
```

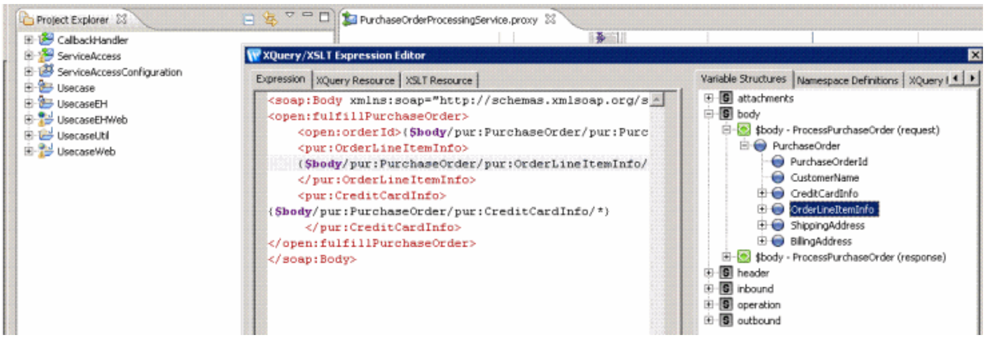
```
<open:fulfillPurchaseOrder>

<open:orderId>{$body/pur:PurchaseOrder/pur:PurchaseOrderId/text()}</ope
n:orderId>
    <pur:OrderLineItemInfo>
    {$body/pur:PurchaseOrder/pur:OrderLineItemInfo/*}
    </pur:OrderLineItemInfo>
    <pur:CreditCardInfo>
{$body/pur:PurchaseOrder/pur:CreditCardInfo/*}
     </pur:CreditCardInfo>
</open:fulfillPurchaseOrder>
</soap:Body>
```

Enter **Variable** as `body`.

You have now created the message flow to publish to `PurchaseOrderFulfillmentJPDBS.biz`. Figure 6-9 shows the message flow.

Note: Alternately, the expressions in the **XQuery/XSLT Expression Editor** can be dragged and dropped into the **Expression** tab. Figure 6-8 illustrates an example:

Figure 6-8 Expression Editor



```
<soap:Body xmlns:soap="http://schemas.xmlsoap.org/s
<open:fulfillPurchaseOrder>
    <open:orderId>{$body/pur:PurchaseOrder/pur:Purc
    <pur:OrderLineItemInfo>
    {$body/pur:PurchaseOrder/pur:OrderLineItemInfo/
    </pur:OrderLineItemInfo>
    <pur:CreditCardInfo>
{$body/pur:PurchaseOrder/pur:CreditCardInfo/*}
     </pur:CreditCardInfo>
</open:fulfillPurchaseOrder>
</soap:Body>
```

8. Add a **Stage** node under the **Response** pipeline. Drag and drop an **Assign** node under this Stage node.

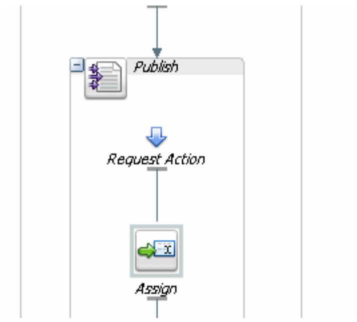
9. In the **Properties** tab of the **Assign** node, click **Expression**, and enter the following expression:

```
<SOAP-ENV:Body
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
```

```
  <ns0:ProcessPurchaseOrderResponse
xmlns:ns0="http://www.openuri.org/"/>

</SOAP-ENV:Body>
```

In the **Variable** field, type `body`.

Figure 6-9 Publishing to PurchaseOrderFulfillmentJPD



The PurchaseOrderFulfillmentJPD invokes the external systems: inventory and credit card systems, and publishes these results to the message broker channels.

The PurchaseOrderAggregatorService aggregates these results and invokes the shipping service external system.

Testing the Proxy

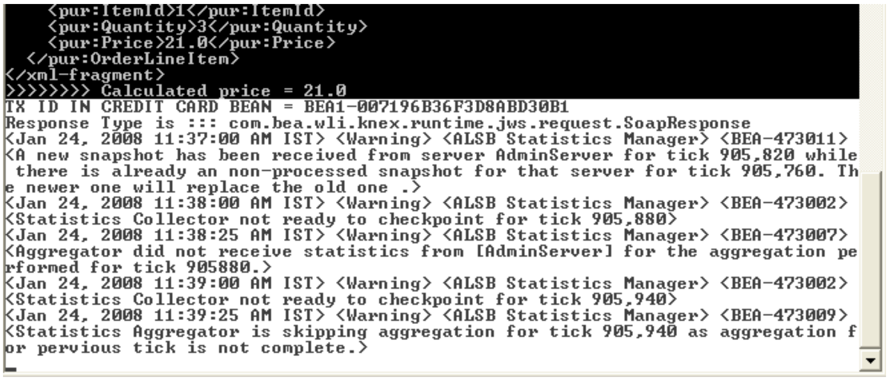
After creating the PurchaseOrderProcessingService.proxy, you can verify whether the PurchaseOrderProcessingProxy has invoked the PurchaseOrderFulfillmentJPD.

To test the PurchaseOrderProcessingProxy

You can use the **TestClient** from the **UseCaseWeb/src/alint.client.test** to test the proxy. Right-click **TestClient.java**, select **Run As** > **Run on Server**.

To check whether the external services: inventory service, credit card service are invoked, see the debug messages on the server console, which are displayed as shown in Figure 6-10. These messages are displayed only if the PurchaseOrderProcessingService.proxy has invoked the external services.

Figure 6-10 Debug Messages on Server Console



```
    <pur:ItemId>1</pur:ItemId>
    <pur:Quantity>3</pur:Quantity>
    <pur:Price>21.0</pur:Price>
  </pur:OrderLineItem>
</xml-fragment>
>>>>>>>> Calculated price = 21.0
TX ID IN CREDIT CARD BEAN = BEA1-007196B36F3D8ABD30B1
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
<Jan 24, 2008 11:37:00 AM IST> <Warning> <ALSB Statistics Manager> <BEA-473011>
<A new snapshot has been received from server AdminServer for tick 905,820 while
 there is already an non-processed snapshot for that server for tick 905,760. Th
e newer one will replace the old one .>
<Jan 24, 2008 11:38:00 AM IST> <Warning> <ALSB Statistics Manager> <BEA-473002>
<Statistics Collector not ready to checkpoint for tick 905,880>
<Jan 24, 2008 11:38:25 AM IST> <Warning> <ALSB Statistics Manager> <BEA-473007>
<Aggregator did not receive statistics from [AdminServer] for the aggregation pe
rformed for tick 905880.>
<Jan 24, 2008 11:39:00 AM IST> <Warning> <ALSB Statistics Manager> <BEA-473002>
<Statistics Collector not ready to checkpoint for tick 905,940>
<Jan 24, 2008 11:39:25 AM IST> <Warning> <ALSB Statistics Manager> <BEA-473009>
<Statistics Aggregator is skipping aggregation for tick 905,940 as aggregation f
or pervious tick is not complete.>
_
```

In this chapter, you invoked the PurchaseOrderFulfillmentJPD business service endpoint by using the PurchaseOrderProcessService proxy service. In Chapter 7, "Aggregating the Purchase Order Fulfillment Results," you will design the PurchaseOrderAggregatorJPD, which aggregates the results of the order fulfillment and handle the order shipment. You will then modify the message flow of the PurchaseOrderProcessingService proxy to invoke the PurchaseOrderAggregatorJPD.

Aggregating the Purchase Order Fulfillment Results

This chapter discusses the following topics:

- "Publishing the PurchaseOrder Fulfillment Results to Message Broker Channels"
- "Designing the PurchaseOrderAggregator JPD to Subscribe to the Message Broker Channels"
- "Creating the Business Service Endpoint for PurchaseOrderAggregatorJPD"

Publishing the PurchaseOrder Fulfillment Results to Message Broker Channels

This section describes how to create decision-making nodes in the PurchaseOrderFulfillmentJPD that you created in the "Designing PurchaseOrderFulfillmentJPD". Decision making nodes are included in the PurchaseOrderFulfillmentJPD to publish the fulfillment results to the message broker channels.

Publishing the PO Fulfillment Results

Publishing Inventory System Results

1. Switch to Process perspective. From the **BEA WorkSpace Studio** menu, select **Window** > **Open Perspective** > **Other...**, and select **Process** from the **Open Perspective** dialog box.
2. In **Package Explorer**, expand **alint.process.control**, and drag and drop **GenericPublishControl.java** to **Data Palette**.

3. In the **Source** view, replace the variable name `genericPublishControl` with `publishPriceInfoControl`.

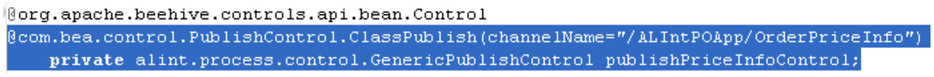
4. Add the following annotation before the variable:

```
@com.bea.control.PublishControl.ClassPublish(channelName="/ALIntPOAp
p/OrderPriceInfo")
```

Note: This PublishControl is used to publish the OrderPriceInfoDocument received from the external inventory system to the `/ALIntPOApp/OrderPriceInfo` channel. The `PurchaseOrderAggregatorJPD` will subscribe to the channel that receives the message.

Figure 7-1 shows the renamed variable, and the new annotation.

Figure 7-1 Annotation and New Variable



```
@org.apache.beehive.controls.api.bean.Control
@com.bea.control.PublishControl.ClassPublish(channelName="/ALIntPOApp/OrderPriceInfo")
    private alint.process.control.GenericPublishControl publishPriceInfoControl;
```

5. Go to the **Source** view, and add the following lines of code:

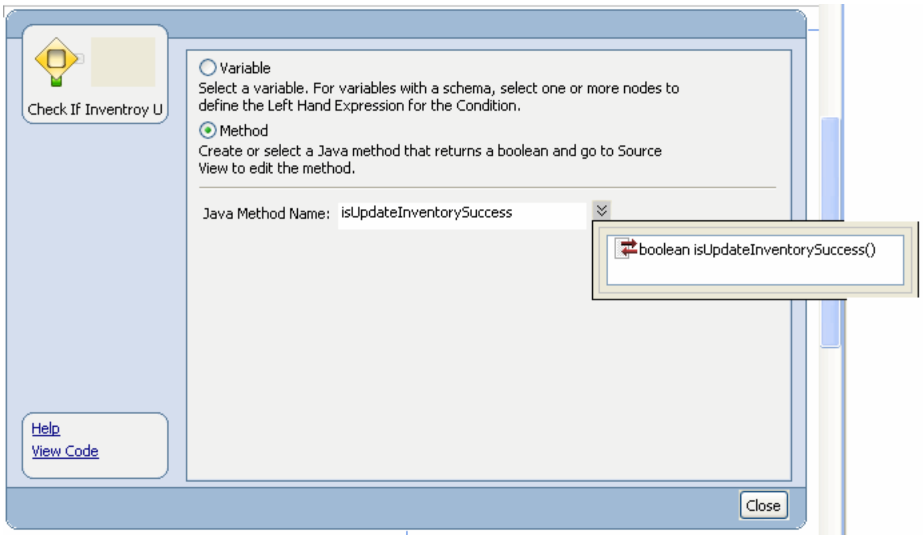
```
public boolean isUpdateInventorySuccess() {

return priceInfo.getOrderPriceInfo().xgetRejectionCode() == null;

     }
```

Note: This method tests whether the inventory update is successful, by checking the rejection code in the OrderPriceInfoDocument received from the external inventory system.

6. From the **Node Palette**, drag and drop a **Decision** node to the **Design** view after the **updateInventory** node, and rename the **Decision** node as **Check If Inventory Update is successful**.

7. Right-click the **Check If Inventory Update is successful** node, and click **Open**. In the dialog box that is displayed, select the **Java Method Name** from the drop-down list, as shown in Figure 7-2.

Figure 7-2 Selecting Java Method Name



Variable
Select a variable. For variables with a schema, select one or more nodes to define the Left Hand Expression for the Condition.
[✓ Method]
Create or select a Java method that returns a boolean and go to Source View to edit the method.

Java Method Name: isUpdateInventorySuccess

boolean isUpdateInventorySuccess()

8. Drag and drop a **Perform** node into the **Check If Inventory Update is successful** node. Double-click the **Perform** node, and rename the method to `calcOrderPrice`. Click **View Code**, and type the following code:

```
List<OrderLineItemPriceInfo> itemPriceList =
priceInfo.getOrderPriceInfo().getOrderLineItemList();

Iterator<OrderLineItemPriceInfo> priceItr = itemPriceList.iterator();

while (priceItr.hasNext()) {


orderPrice += priceItr.next().getPrice();

}
```

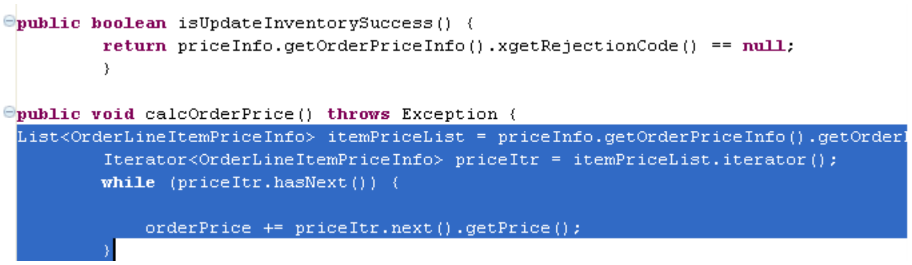
Note: In case of compilation errors, ensure that the following imports are added to the code:

```
import java.util.Iterator;

import java.util.List;

import org.openuri.purchaseOrder.OrderLineItemPriceInfo;
```

Figure 7-3 shows the code in Perform node.

Note: The `calcOrderPrice` method calculates the total order price, which is used to transfer funds from the customer credit card.

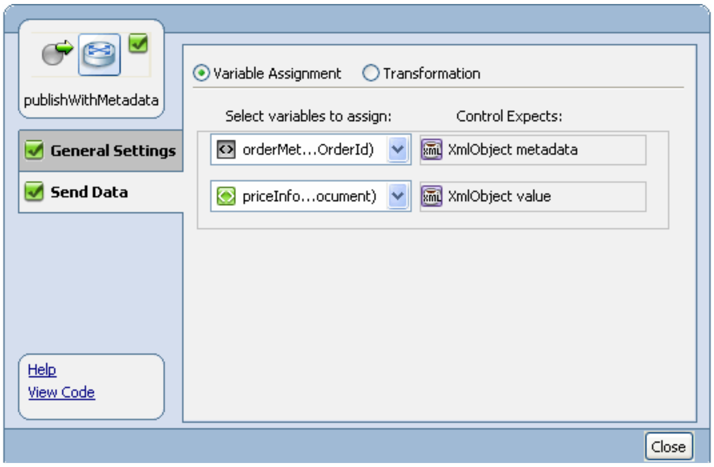
Figure 7-3 Perform Node - Source View



```
public boolean isUpdateInventorySuccess() {
        return priceInfo.getOrderPriceInfo().xgetRejectionCode() == null;
        }

public void calcOrderPrice() throws Exception {
List<OrderLineItemPriceInfo> itemPriceList = priceInfo.getOrderPriceInfo().getOrder
        Iterator<OrderLineItemPriceInfo> priceItr = itemPriceList.iterator();
        while (priceItr.hasNext()) {

            orderPrice += priceItr.next().getPrice();
        }
```

9. Expand **Controls** > **publishPriceInfoControl** in the **Data Palette.** Drag and drop **voidpublishWithMetadata** into the **Condition** node in the **Design** view. Double-click the **publishWithMetadata** node. In the **Send Data** tab, select `OrderMetadata` and `priceInfo` from the **Select variables to assign** drop-down list. Figure 7-4 shows the variable options on the **Send Data** tab.

Figure 7-4 publishWithMetadata - Send Data Tab



[✓ Variable Assignment] Transformation

Select variables to assign:	Control Expects:
orderMet...OrderId)	XmlObject metadata
priceInfo...ocument)	XmlObject value

10. Click **Close**.

11. From the **Node Palette**, drag and drop **Perform** node into the Decision **Default** node on the **Design** view. Rename the Perform node to `Throw Exception`, and rename the method to `throwEx`. Enter the following code in the **Source** view:

```
TxHelper.getUserTransaction().setRollbackOnly();

String errorInfo = ccTranInfo.xgetErrorMessage().getStringValue();
```

```
throw new
RuntimeException(ccTranInfo.xgetErrorMessage().getStringValue());
```

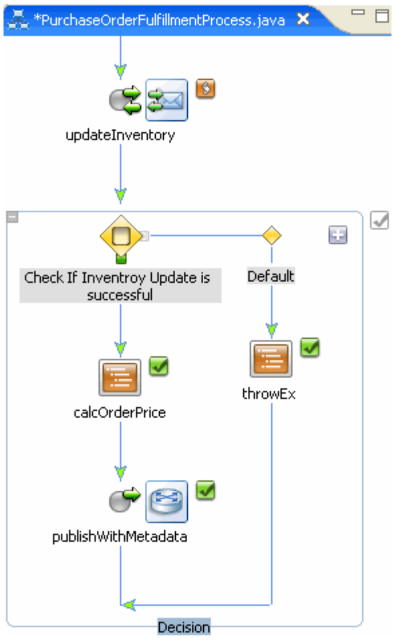
Note: In case of compilation errors, ensure to add the following imports to the code:

```
import weblogic.transaction.TxHelper;
```

You have now reviewed the response from the external inventory system and verified that the inventory update was successful, and no rejection code is set. When the inventory is updated, the OrderPriceInfo document is published to the **/ALIntPOApp/OrderPriceInfo** channel.

Figure 7-5 shows the process in **Design** view.

Figure 7-5 Inventory System


*PurchaseOrderFulfillmentProcess.java
updateInventory
Check If Inventroy Update is successful
Default
calcOrderPrice
throwEx
publishWithMetadata
Decision

Publishing Credit Card System Results

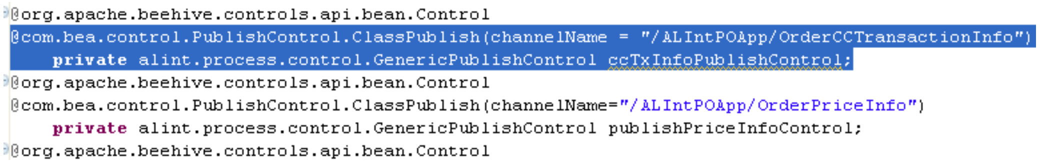
1. In **Package Explorer**, expand **alint.process.control**, and drag and drop **GenericPublishControl.java** to **Data Palette**.
2. In the **Source** view, replace the variable name **`genericPublishControl`** with **`ccTxInfoPublishControl`**.

3. Add the following annotation before the variable names:

```
@com.bea.control.PublishControl.ClassPublish(channelName =
"/ALIntPOApp/OrderCCTransactionInfo")
```

Figure 7-6 shows both the new variable and the annotation.

Figure 7-6 Annotation and New Variable



```
@org.apache.beehive.controls.api.bean.Control
@com.bea.control.PublishControl.ClassPublish(channelName = "/ALIntPOApp/OrderCCTransactionInfo")
    private alint.process.control.GenericPublishControl ccTxInfoPublishControl;
@org.apache.beehive.controls.api.bean.Control
@com.bea.control.PublishControl.ClassPublish(channelName="/ALIntPOApp/OrderPriceInfo")
    private alint.process.control.GenericPublishControl publishPriceInfoControl;
@org.apache.beehive.controls.api.bean.Control
```

4. Go to **Source** view, and add the following method:

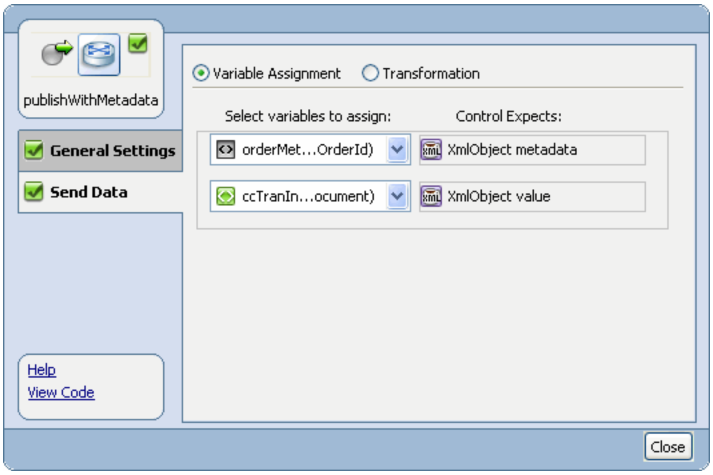
```
public boolean isMoneyTransferSuccess() {

return ccTranInfo.xgetRejectionCode() == null;

}
```

5. From the **Node Palette**, drag and drop a **Decision** node to the **Design** view after the **transferMoney** node. Rename this node to `Check If Money Transfer is Successful`.

6. Right-click on **Check If Money Transfer is Successful**, and select **Open.** From the **Java Method Name** drop-down list, select **isMoneyTransferSuccess**. Click **Close**.

7. Expand **Controls** > **ccTxInfoPublishControl** in the **Data Palette**. Drag and drop **publishWithMetadata** into the **Condition** node in the **Design** view. Double-click the **publishWithMetadata** node. In the **Send Data** tab, select `orderMetadata`, and `ccTranInfoDoc` from the **Select variables to assign** drop-down list. Figure 7-7 shows the variable options on the **Send Data** tab.

Figure 7-7 publishWithMetaData - General Settings Tab



(● Variable Assignment) ○ Transformation

Select variables to assign:	Control Expects:
orderMet...OrderId)	XmlObject metadata
ccTranIn...ocument)	XmlObject value

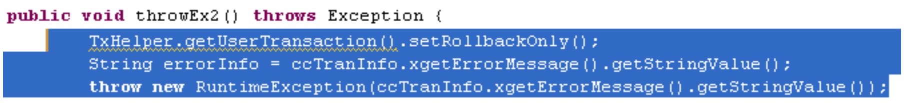
8. Click **Close**.

9. From the **Node Palette**, drag and drop the **Perform** node into the Decision **Default** node on the **Design** view. Rename the method to `throwEx2`. Enter the following code in the **Source** view:

```
TxHelper.getUserTransaction().setRollbackOnly();

String errorInfo = ccTranInfo.xgetErrorMessage().getStringValue();

throw new
RuntimeException(ccTranInfo.xgetErrorMessage().getStringValue());
```

Figure 7-8 shows the exception code in the **Source** view.

Figure 7-8 Default Perform Node - Exception Code in Source View



```
public void throwEx2() throws Exception {
        TxHelper.getUserTransaction().setRollbackOnly();
        String errorInfo = ccTranInfo.xgetErrorMessage().getStringValue();
        throw new RuntimeException(ccTranInfo.xgetErrorMessage().getStringValue());
```

You have now reviewed the response from the external credit card system and verified that the money transfer was successful and no rejection code is set. If the money transfer transaction is successful, the CreditCardTransactionInfo document is published to the **/ALIntPOApp/CCTransInfochannel**.

You have designed the PurchaseOrderFulfillmentJPD to invoke the external inventory and credit card systems, in the context of the JPD's implicit transaction. In case of an error, the transaction is rolled back, and an exception is thrown.

In the transaction succeeds, the results are published to respective message broker channels.

Designing the PurchaseOrderAggregator JPD to Subscribe to the Message Broker Channels

The PurchaseOrderAggregatorJPD aggregates the results of the PurchaseOrderFulfillment JPD. This JPD dynamically subscribes to the message broker channels, to which the PurchaseOrderFulfillment JPD will publish the results of the inventory service and credit card service external systems.

This section describes how to design the PurchaseOrderAggregatorJPD to subscribe to the message broker channels and to aggregate the purchase order fulfillment results.

This section discusses the following tasks:

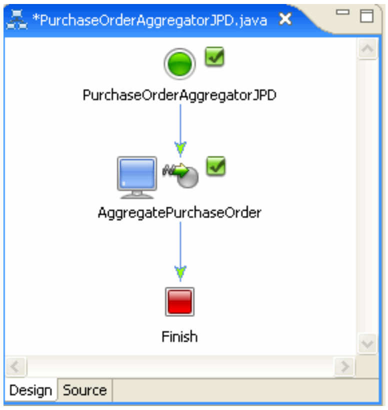
- "To Design the PurchaseOrderAggregatorJPD"
- "To Add the Subscription Message Broker Channels to the JPD"

To Design the PurchaseOrderAggregatorJPD

1. Create a PurchaseOrderAggregatorJPD. In the **Package Explorer**, right-click on **alint.process**, select **New** > **Other** > **WebLogic Integration** > **Process**. In the **New Process** dialog box, enter `PurchaseOrderAggregatorJPD` as the **Name** of the Process. Type `alint.process` as the **Package**, if the package is not already set.

2. Drag and drop **Client Request** from **Node Palette** to the **Design** view of **PurchaseOrderAggregatorJPD**, and double-click to rename it `AggregatePurchaseOrder`, as shown in Figure 7-9.

Figure 7-9 Client Request Node



3. Double-click on the **AggregatePurchaseOrder** node, enter `AggregatePurchaseOrder` in **Method Name** and click **Add...**

4. In the dialog box that is displayed, add the following variable names:

   ```
   java.lang.String orderId
   org.openuri.purchaseOrder.ShippingAddressDocument shippingAddress
   java.lang.String customerName
   ```

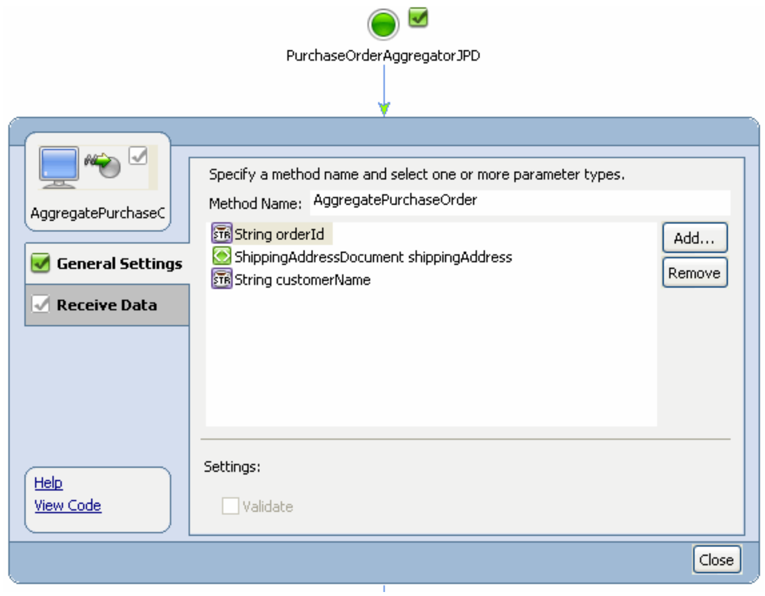
   Table 7-1 lists the parameters and types of these variable names.

Table 7-1 Variable Names

Parameter Name	Type Name
orderId	`java.lang.String`
shippingAddress	`org.openuri.purchaseOrder.ShippingAddressDocument`
customerName	`java.lang.String`

5. Click **Close**. Figure 7-10 shows the parameters in the **AggregatePurchaseOrder** node.

Figure 7-10 AggregatePurchaseOrde Node - Parameter Values


PurchaseOrderAggregatorJPD

AggregatePurchaseC

General Settings

Receive Data

Specify a method name and select one or more parameter types.

Method Name: AggregatePurchaseOrder

String orderId
ShippingAddressDocument shippingAddress
String customerName

Settings:

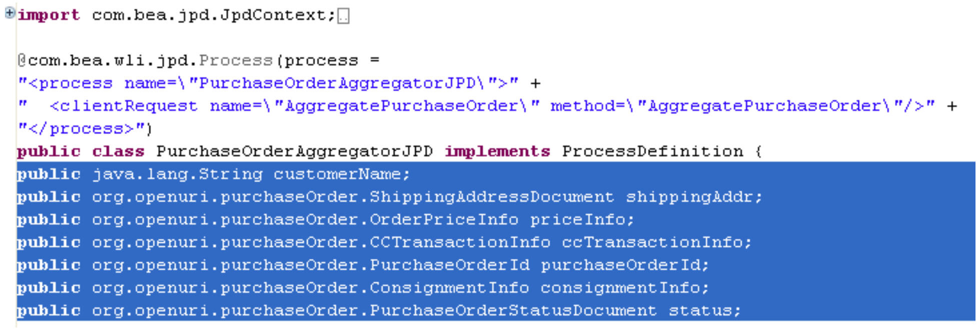
Validate

6. Create the following instance variables in the **Source** view of PurchaseOrderAggregatorJPD:

```
public java.lang.String customerName;

public java.lang.String OrderId;

public org.openuri.purchaseOrder.ShippingAddressDocument
shippingAddr;

public org.openuri.purchaseOrder.OrderPriceInfo priceInfo;

public org.openuri.purchaseOrder.CCTransactionInfo ccTransactionInfo;

public org.openuri.purchaseOrder.PurchaseOrderId purchaseOrderId;

public org.openuri.purchaseOrder.ConsignmentInfo consignmentInfo;

public org.openuri.purchaseOrder.PurchaseOrderStatusDocument status;
```

Figure 7-11 shows instance variables in **Source** view.

Figure 7-11 Instance Variables



```
import com.bea.jpd.JpdContext;

@com.bea.wli.jpd.Process(process =
"<process name=\"PurchaseOrderAggregatorJPD\">" +
"  <clientRequest name=\"AggregatePurchaseOrder\" method=\"AggregatePurchaseOrder\"/>" +
"</process>")
public class PurchaseOrderAggregatorJPD implements ProcessDefinition {
public java.lang.String customerName;
public org.openuri.purchaseOrder.ShippingAddressDocument shippingAddr;
public org.openuri.purchaseOrder.OrderPriceInfo priceInfo;
public org.openuri.purchaseOrder.CCTransactionInfo ccTransactionInfo;
public org.openuri.purchaseOrder.PurchaseOrderId purchaseOrderId;
public org.openuri.purchaseOrder.ConsignmentInfo consignmentInfo;
public org.openuri.purchaseOrder.PurchaseOrderStatusDocument status;
```

7. Add the following annotation before the `AggregatePurchaseOrder` method name:

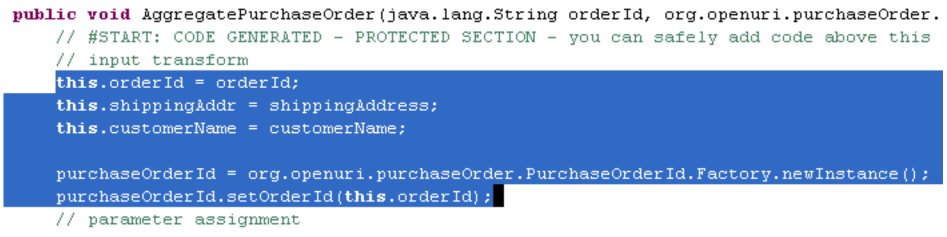
```
@Protocol(jmsSoap = true, javaCall = true)
```

8. Edit the AggregatePurchaseOrder method, and assign the parameters as follows:

```
this.orderId = orderId;

this.shippingAddr = shippingAddress;

this.customerName = customerName;


        // #END  : CODE GENERATED - PROTECTED SECTION - you can safely
add code below this comment in this method. #//

purchaseOrderId =
org.openuri.purchaseOrder.PurchaseOrderId.Factory.newInstance();


          purchaseOrderId.setOrderId(this.orderId);
```

Figure 7-12 shows the code added in **Source** view.

Figure 7-12 Assigning Values to Variables



```
public void AggregatePurchaseOrder(java.lang.String orderId, org.openuri.purchaseOrder.
    // #START: CODE GENERATED - PROTECTED SECTION - you can safely add code above this
    // input transform
    this.orderId = orderId;
    this.shippingAddr = shippingAddress;
    this.customerName = customerName;

    purchaseOrderId = org.openuri.purchaseOrder.PurchaseOrderId.Factory.newInstance();
    purchaseOrderId.setOrderId(this.orderId);
    // parameter assignment
```

To Add the Subscription Message Broker Channels to the JPD

1. From **Package Explorer**, **src** > **alint.process.control**, drag and drop **GenericSubscriptionControl** to the **Data Palette** > **Controls** folder.

2. In the **Source** view, rename this control to `orderPriceSbscrptnn`.

3. Add the following annotation before the `orderPriceSbscrptnn` line in **Source** view.

   ```
   @com.bea.controls.SubscriptionControl.ClassSubscription(channelName
   = "/ALIntPOApp/OrderPriceInfo",

   xquery = "data($metadata)",

   xqueryVersion = com.bea.wli.common.XQuery.Version.v2004)
   ```

4. Drag and drop **subscribeWithFilterValue** from **Data Palette** > **Controls** > **orderPriceSbscrptn** after the **AggregatePurchaseOrder** node in the **Design** view.

5. Double-click the **subscribeWithFilterValue** node, click **Send Data**, and select `orderId` from the **Select Variables to Assign** drop-down list.

6. Drag and drop **GenericSubscriptionControl** from **src** > **alint.process.control** into the **Data Palette**. Rename this control to `ccTxInfoSbscrptnn` in **Source** view.

7. Add the following annotation before the `ccTxInfoSbscrptnn` line in **Source** view:

   ```
   @com.bea.controls.SubscriptionControl.ClassSubscription(channelName
   = "/ALIntPOApp/OrderCCTransactionInfo",

           xquery = "data($metadata)",

           xqueryVersion = com.bea.wli.common.XQuery.Version.v2004)
   ```

8. Drag and drop **subscribeWithFilterValue** from **Data Palette** > **Controls** > **ccTxInfoSbscrptnn** after the **orderPriceSbscrptnn** node. Rename this node as **ccTxInfoSbscrptnn**.

9. Double click the **subscribeWithFilterValue** node, click **Send Data**, and select `orderId` from the **Select Variables to Assign** drop-down list.

You have now designed the PurchaseOrderAggregatorJPD to subscribe to the message broker channels to which the results of the PurchaseOrderFulfillmentJPD are published.

Designing Parallel Branch in the JPD

1. From **Node Palette**, drag and drop a **Parallel Branch** after the **ccTxInfoSbscrptnn** in the **Design** view.
2. From the **Data Palette** > **Controls** > **orderPriceSbscrptnn**, drag and drop **void onMessage(XmlObject message)** into the first Parallel Branch.
3. Right-click on this node, and select **View Code**. Add the following code in the **Source** view:

```
try

{

priceInfo =
org.openuri.purchaseOrder.OrderPriceInfo.Factory.parse(message.xmlTe
xt());

}

   catch (XmlException e) {

    throw new RuntimeException(e);

}
```

4. Similarly, from the **Data Palette** > **Controls** > **ccTxInfoSbscrptnn**, drag and drop **void onMessage(XmlObject message)** into the first Parallel Branch.
5. Again, from the **Data Palette** > **Controls** > **ccTxInfoSbscrptnn**, drag and drop **void onMessage(XmlObject message)** into the second Parallel Branch.
6. Right-click on this node, and select **View Code**. Add the following code in the source:

```
try {


          ccTransactionInfo =
org.openuri.purchaseOrder.CCTransactionInfo.Factory.parse(message.xm
lText());

} catch (XmlException e) {

throw new RuntimeException(e);

}
```

Note: You have now designed the PurchaseOrderAggregatorJPD to aggregate the results from the message broker channels.

Invoking the External Shipping System

1. From the **Package Explorer**, drag and drop **ServiceProviderAccessControl.java** into **Data Palette**.

   Note: ServiceProviderAccessControl is the ALSB Control that you created in Creating AquaLogic Service Bus Control.

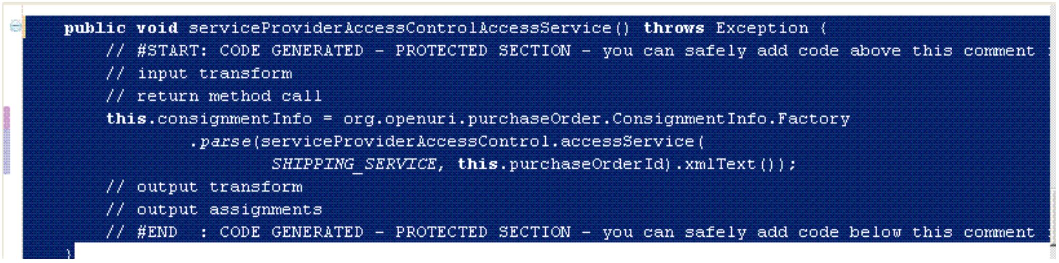
2. Drag and drop **XmlObject accessService(String ServiceName_arg,XMLObject_Any_arg)** from **ServiceProviderAccessControl** after the second Parallel Branch in the **Design** view. This is the **accessService** node.

3. Add the following code in the **Source** view of the method:

   ```
   this.consignmentInfo =
   org.openuri.purchaseOrder.ConsignmentInfo.Factory.parse(serviceProvi
   derAccessControl.accessService(“ShippingService”,
   this.purchaseOrderId).xmlText());
   ```

   Figure 7-13 shows the code you have just added.

Figure 7-13 Code for the AccessService Method



```
public void serviceProviderAccessControlAccessService() throws Exception {
    // #START: CODE GENERATED - PROTECTED SECTION - you can safely add code above this comment
    // input transform
    // return method call
    this.consignmentInfo = org.openuri.purchaseOrder.ConsignmentInfo.Factory
            .parse(serviceProviderAccessControl.accessService(
                    SHIPPING_SERVICE, this.purchaseOrderId).xmlText());
    // output transform
    // output assignments
    // #END  : CODE GENERATED - PROTECTED SECTION - you can safely add code below this comment
}
```

4. Drag and drop a **Perform** node from the **Node Palette** after the **accessService** node.

5. Double-click the **Perform** node, and type `archiveOrder` in **JavaMethodName**.

6. Right-click the **Perform** node, click **View Code,** and replace **public void archiveOrder() throws Exception {** with the following code:

   ```
   public void archiveOrder() throws Exception {


                               File orderFile = new
   File("C:/ALIntApp/PurchaseOrder/New/PO-" + orderId + ".xml");
   ```

```
                                        orderFile.renameTo(new
 File("C:/ALIntApp/PurchaseOrder/Archive/PO-" + orderId + ".xml"));


     }
```

After you have added the **Perform** node, you must unsubscribe from the subscriptions.

Unsubscribing from the Message Broker Channels

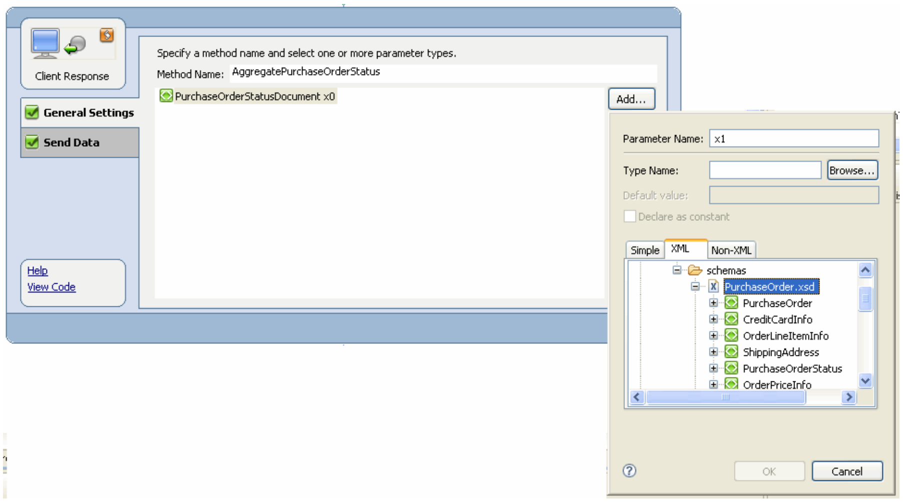
1. Drag and drop **void unsubscribe()** from **Data Palette** > **Controls** > **ccTxSbscrptnn** under the **Perform** node.
2. Similarly, drag and drop **void unsubscribe()** from **Data Palette** > **Controls** > **orderPriceSbscrptnn** under the **Perform** node.

In the next step, you send the Purchase Order status as a callback.

Sending Order Status as Callback

1. Drag and drop a **Client Response** node from the **Node Palette** to the **Design** view.
2. Double-click the **Client Response** node, and rename the **Method Name** to `AggregatePurchaseOrderStatus`.
3. Click **Add** to add the `PurchaseOrderStatusDocument x0`, as shown in Figure 7-14.

Figure 7-14 Adding Variable in the Client Response Node



Specify a method name and select one or more parameter types.

Method Name: AggregatePurchaseOrderStatus

PurchaseOrderStatusDocument x0

Parameter Name: x1

Type Name:

Default value:

Declare as constant

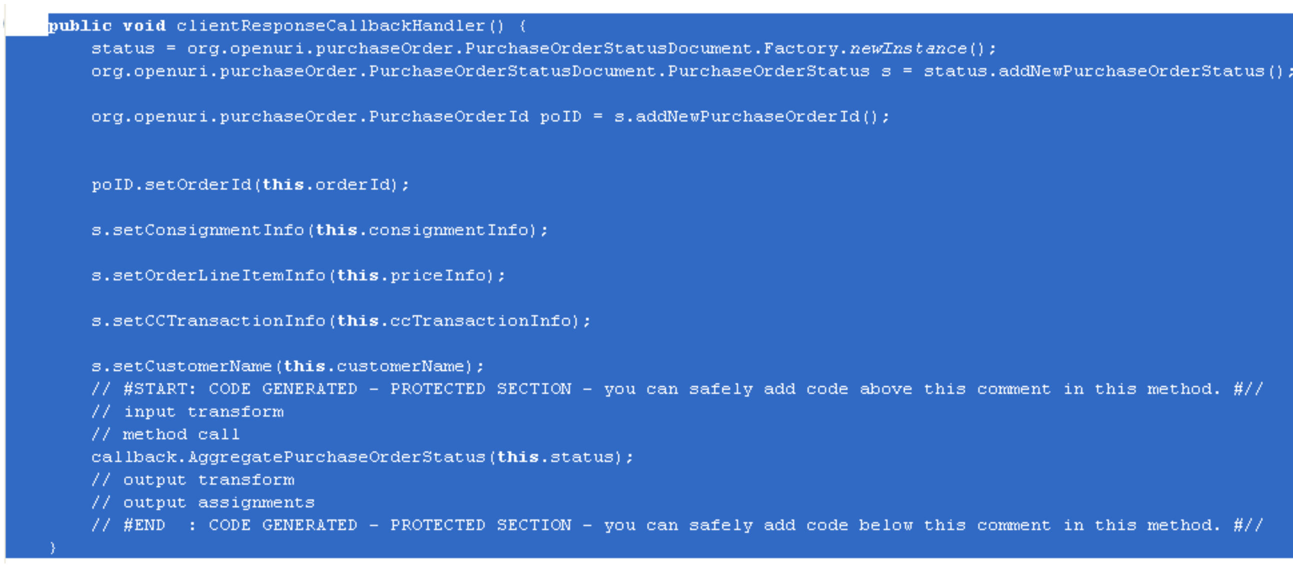
schemas
- PurchaseOrder.xsd
  - PurchaseOrder
  - CreditCardInfo
  - OrderLineItemInfo
  - ShippingAddress
  - PurchaseOrderStatus
  - OrderPriceInfo

4. Right-click on the **Client Response** node, and select **View Code**. Add the following in **Source** view.

```
status = 
org.openuri.purchaseOrder.PurchaseOrderStatusDocument.Factory.newIns
tance();


org.openuri.purchaseOrder.PurchaseOrderStatusDocument.PurchaseOrderS
tatus s = status.addNewPurchaseOrderStatus();


        org.openuri.purchaseOrder.PurchaseOrderId poID = 
s.addNewPurchaseOrderId();




        poID.setOrderId(this.orderId);


        s.setConsignmentInfo(this.consignmentInfo);


        s.setOrderLineItemInfo(this.priceInfo);


        s.setCCTransactionInfo(this.ccTransactionInfo);


        s.setCustomerName(this.customerName);
       // #START: CODE GENERATED - PROTECTED SECTION - you can safely 
add code above this comment in this method. #//
        // input transform
        // method call
        callback.AggregatePurchaseOrderStatus(this.status);
        // output transform
        // output assignments
       // #END  : CODE GENERATED - PROTECTED SECTION - you can safely 
add code below this comment in this method. #//
```

Figure 7-15 shows the **Source** view.

Figure 7-15 Client Response Method in PO Aggregator JPD



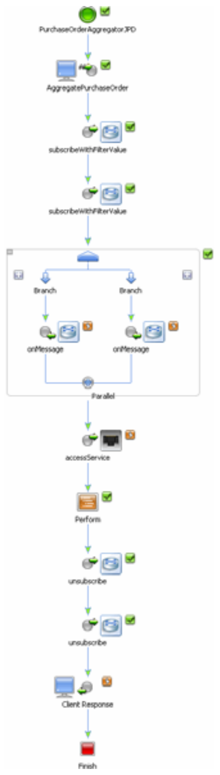
```
public void clientResponseCallbackHandler() {
    status = org.openuri.purchaseOrder.PurchaseOrderStatusDocument.Factory.newInstance();
    org.openuri.purchaseOrder.PurchaseOrderStatusDocument.PurchaseOrderStatus s = status.addNewPurchaseOrderStatus();

    org.openuri.purchaseOrder.PurchaseOrderId poID = s.addNewPurchaseOrderId();


    poID.setOrderId(this.orderId);

    s.setConsignmentInfo(this.consignmentInfo);

    s.setOrderLineItemInfo(this.priceInfo);

    s.setCCTransactionInfo(this.ccTransactionInfo);

    s.setCustomerName(this.customerName);
    // #START: CODE GENERATED - PROTECTED SECTION - you can safely add code above this comment in this method. #//
    // input transform
    // method call
    callback.AggregatePurchaseOrderStatus(this.status);
    // output transform
    // output assignments
    // #END  : CODE GENERATED - PROTECTED SECTION - you can safely add code below this comment in this method. #//
}
```

5. In **Source** view, add the following annotation before this method name:

```
@Protocol(jmsSoap = true, javaCall = true)
```

Figure 7-16 shows the Purchase Order Aggregator JPD in **Design** view.

Figure 7-16 PurchaseOrderAggregatorJPD



Creating the Business Service Endpoint for PurchaseOrderAggregatorJPD

The business service end points are used by ALSB proxy services to invoke the Purchase Order Aggregator JPD for aggregating the purchase order results.

To Create the Business Service Endpoint

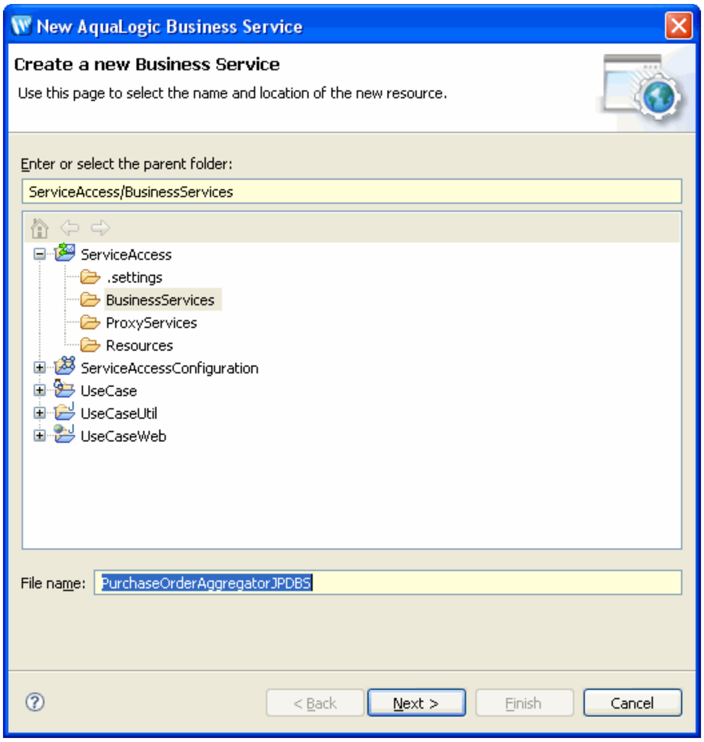
1. Switch to the ALSB perspective from your current perspective in BEA WorkSpace Studio. To select the ALSB perspective, in the **BEA WorkSpace Studio** window menu, click **Windows** > **Open Perspective** > **Other...**.

   Select **AquaLogic Service Bus** from the **Open Perspective** dialog box.

2. In the **Project Explorer** pane, select **ServiceAccess** > **Business Services** > **New**.

3. In the **Create a new Business Service** dialog box, enter the **File Name** as `PurchaseOrderAggregatorJPDBS`, and click **Next**.

Figure 7-17 Create a Business Service


New AquaLogic Business Service
Create a new Business Service
Use this page to select the name and location of the new resource.
Enter or select the parent folder:
ServiceAccess/BusinessServices
ServiceAccess
.settings
BusinessServices
ProxyServices
Resources
ServiceAccessConfiguration
UseCase
UseCaseUtil
UseCaseWeb
File name:
PurchaseOrderAggregatorJPDBS
< Back
Next >
Finish
Cancel

4. In the **Create Business Service - General Configuration (Service Access/Business Services/)** dialog box, select **WSDL Web Service** Service Type, and click **Browse**. The **Select a WSDL** dialog box is displayed.

5. Select **Service Access**, and click **Consume**. In the **Service Consumption** dialog box, browse to **ServiceAccess** > **Resources,** and select `ServiceAccess\Resources` as the **Artifact folder**.

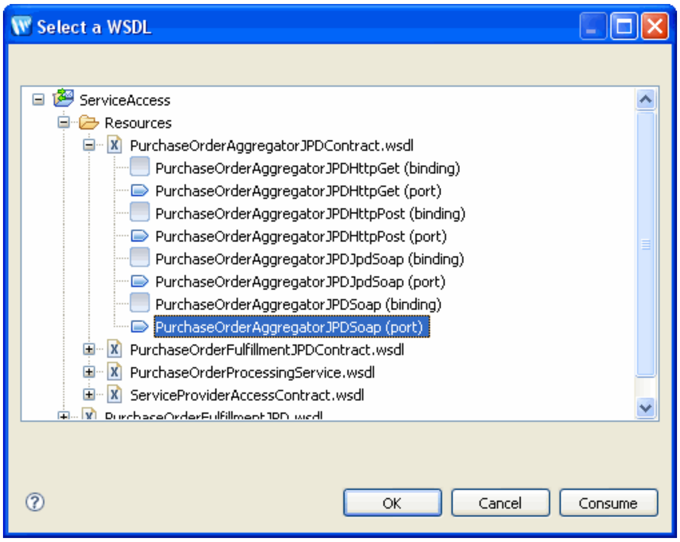
6. From the **Service Resource** drop-down list, select **Workspace**.

7. Browse to \UseCaseWeb\src\alint\process\\, select **PurchaseOrderAggregatorJPDContract.wsdl**, and click **OK**.

8. Click **OK** in the **Service Consumption Status** dialog box.

9. In the **Select a WSDL** dialog box, expand **Resources**, select and expand **`PurchaseOrderAggregatorJPDContract.wsdl`**, and then select **PurchaseOrderAggregatorJPDSoap(port)**, click **OK**, as shown in Figure 7-18.

Figure 7-18 Select a WSDL


Select a WSDL

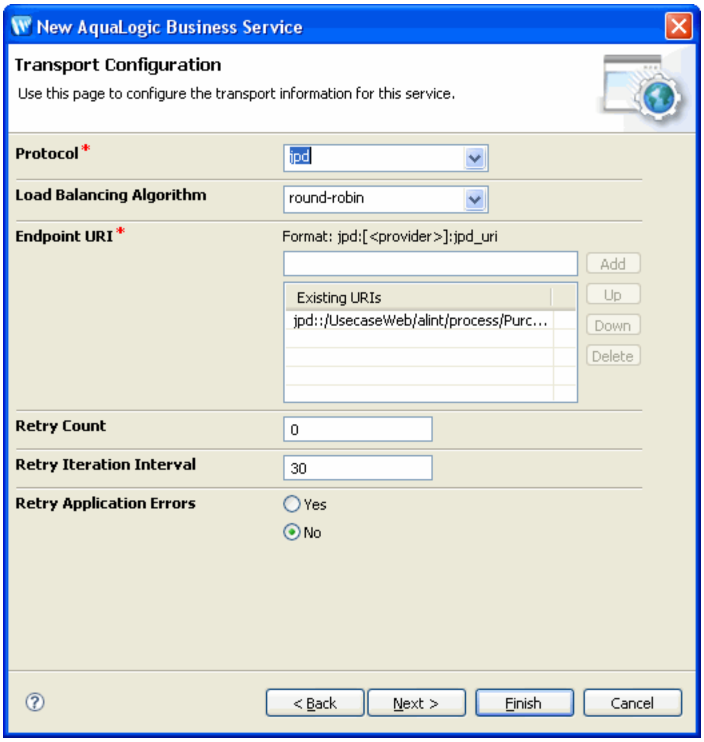
ServiceAccess
Resources
PurchaseOrderAggregatorJPDContract.wsdl
PurchaseOrderAggregatorJPDHttpGet (binding)
PurchaseOrderAggregatorJPDHttpGet (port)
PurchaseOrderAggregatorJPDHttpPost (binding)
PurchaseOrderAggregatorJPDHttpPost (port)
PurchaseOrderAggregatorJPDJpdSoap (binding)
PurchaseOrderAggregatorJPDJpdSoap (port)
PurchaseOrderAggregatorJPDSoap (binding)
PurchaseOrderAggregatorJPDSoap (port)
PurchaseOrderFulfillmentJPDContract.wsdl
PurchaseOrderProcessingService.wsdl
ServiceProviderAccessContract.wsdl

10. Click **Next** in the **Create a Business Service - Transport Configuration (ServiceAccess/Business Services)** dialog box, and ensure the following:

   – The **Protocol** is set to jpd.

   – The **Endpoint URI** is: `jpd::/UseCaseWeb/alint/process/PurchaseOrderAggregatorJPD.jpd`. Click **Add**. and click **Next**. Figure 7-19 shows the default values selected.

Figure 7-19 Selecting Protocol and Endpoint URIs


New AquaLogic Business Service

Transport Configuration

Use this page to configure the transport information for this service.

Protocol *	jpd
Load Balancing Algorithm	round-robin
Endpoint URI *	Format: jpd:[<provider>]:jpd_uri Existing URIs jpd::/UsecaseWeb/alint/process/Purc...
Retry Count	0
Retry Iteration Interval	30
Retry Application Errors	Yes [✓ No]

11. In the **JPD Transport Configuration** dialog box, accept the default values, and click **Finish**.

    The `PurchaseOrderAggregatorJPDBS.biz` file is created in the **Business Services** folder in **Project Explorer**. In **PurchaseOrderAggregatorJPDBS.biz**, click the **JPD** tab. Ensure that the **Callback Proxy Location** is `jms://localhost:7001/weblogic.jms.XAConnectionFactory/PurchaseOrderNotificationServiceRequest`.

Notes: Callbacks are only supported over JMS when JPD transport is used. Therefore, you must configure it over JMS.

While configuring the JPD transport for invoking JPD as a business service, you must specify the location to the callback proxy, which is configured for JMS (in callback pipeline). WLI sends the callback response to the queue on which callback proxy is configured (in callback pipeline). Along with the response, WLI also sends the original callback location, which is part of the original request, as a JMS transport header. You can configure the callback pipeline so that the callback is sent to this callback location.

The original callback location is sent as a JMS property namely `BEA_WLI_Target_Callback_Location`. If the original callback location contains any additional query strings, those query strings are also be sent as JMS transport headers. You must retrieve this property from the message context variable `$inbound`, and this can be used to configure the outgoing business service for sending the callback, to the actual JPD callback client.

Using JPD Transport Callback

This chapter describes how to publish to PurchaseOrderAggregatorJPD:

- "Modifying PurchaseOrderProcessingService Proxy to Invoke PO Aggregator JPD"
- "Modifying PurchaseOrderProcessingService Proxy to Invoke PurchaseOrderAggregator JPD"
- "Testing the Application"
- "Verifying Security Context Propagation"

The PurchaseOrderProcessingService proxy invokes the PurchaseOrderFulfillment JPD via the PurchaseOrderFulfillmentJPD business service endpoint for fulfilling the order.

This chapter describes how to modify the message flow of the PurchaseOrderProcessing proxy to publish to PurchaseOrderAggregatorJPD.

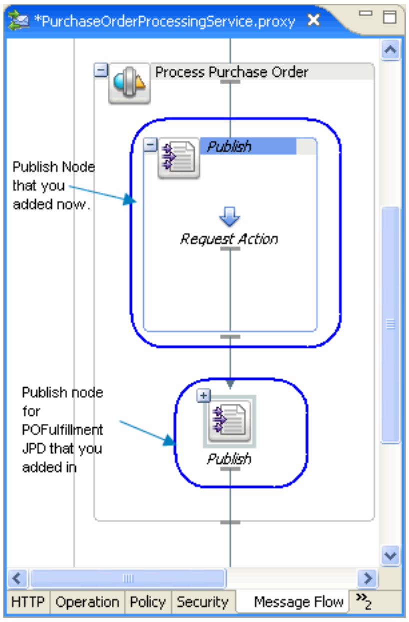
Modifying PurchaseOrderProcessingService Proxy to Invoke PO Aggregator JPD

This section describes how to invoke the PurchaseOrderAggregatorJPD from the Purchase Order Processing Proxy that you created in "Creating the PurchaseOrderProcessingService Proxy".

To Invoke PurchaseOrderAggregatorJPD

1. From the **Design Palette**, expand **Stage Actions** > **Communications**, and select **Publish**. Drag and drop **Publish** node under the **Process Purchase Order** node, as shown in Figure 8-1.

Figure 8-1 Publish Node to Invoke PurchaseOrderAggregatorJPD


*PurchaseOrderProcessingService.proxy
Process Purchase Order
Publish
Publish Node that you added now.
Request Action
Publish node for POFulfillment JPD that you added in
Publish
HTTP
Operation
Policy
Security
Message Flow

For more information about Invoking PurchaseOrderAggregatorJPD, see Chapter 6, “Creating Proxy Service to Invoke a JPD.”

2. In the **Properties** tab of **Publish**, click **Browse**. Select `PurchaseOrderAggregatorJPDBS.biz`. From the **Invoking** drop-down list, select **AggregatePurchaseOrder**.

3. From **Design Palette**, expand **Stage Actions** > **Message Processing**, and select **Assign** node. Drag and drop **Assign** node into the **Request** pipeline.

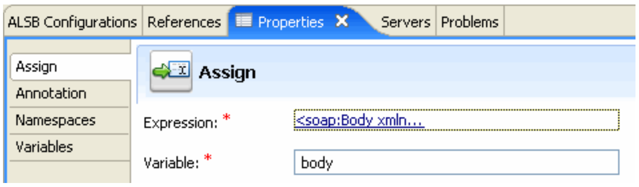
4. In the **Properties** tab of the **Assign** node, click **Expression**, and enter the following code in the **XQuery Editor** dialog box that is displayed:

```
<soap:Body xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<AggregatePurchaseOrder xmlns="http://www.openuri.org/"
xmlns:pur="http://www.openuri.org/PurchaseOrder">


<orderId>{$body/pur:PurchaseOrder/pur:PurchaseOrderId/text()}</order
Id>

    {$body/pur:PurchaseOrder/pur:ShippingAddress}
```

```
<customerName>{$body/pur:PurchaseOrder/pur:CustomerName/text()}</cus
tomerName>

</AggregatePurchaseOrder>

</soap:Body>
```

Enter **Variable** as `body`.

Figure 8-2 Assign Node - Request Pipeline

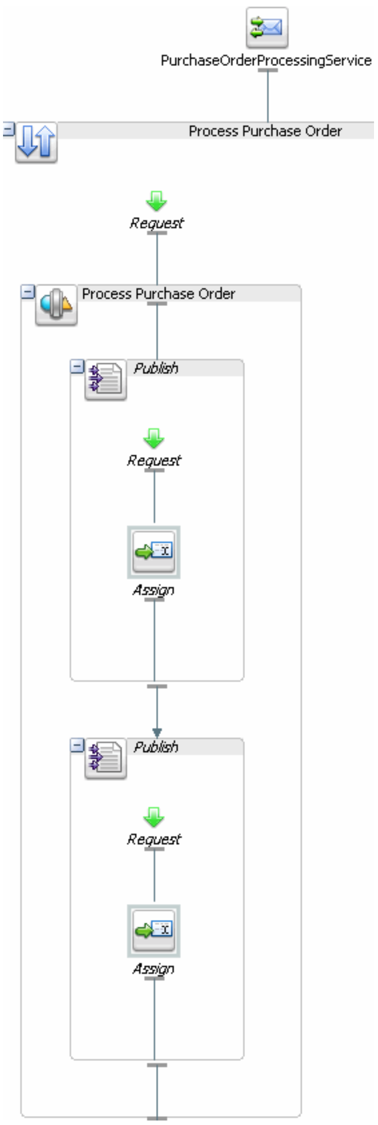


Expression: * <soap:Body xmln...

Variable: * body

You have now described the message flow to publish to `PurchaseOrderAggregatorJPDBS.biz`. Figure 8-3 shows the message flow.

Figure 8-3 Publishing to PurchaseOrderAggregatorService



The PurchaseOrderAggregatorJPD aggregates the results of the PurchaseOrderFulfillmentJPD and invokes the shipping service external system.

Modifying PurchaseOrderProcessingService Proxy to Invoke PurchaseOrderAggregator JPD

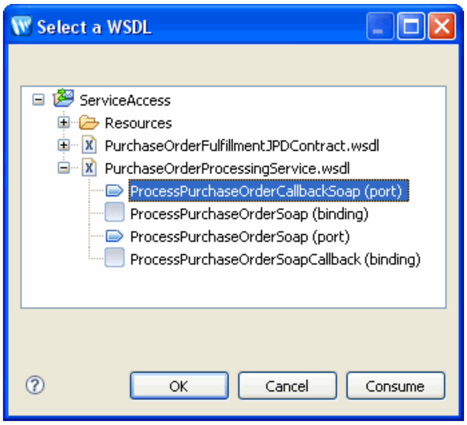
After the Purchase Order is completely processed, a callback is sent to the client using the `PurchaseOrderNotificationService.proxy`.

This section describes how to send a callback via ALSB.

To Create PurchaseOrderNotification Business Service (PurchaseOrderNotificationServiceJPDBS.biz)

1. Switch to the ALSB perspective from your current perspective in Workshop for WebLogic. To select the ALSB perspective, in the **BEA Workshop for WebLogic** window menu, click **Windows** > **Open Perspective** > **Other...**. Select **AquaLogic Service Bus** from the **Open Perspective** dialog box.
2. In **Project Explorer**, select **ServiceAccess**, and expand **Business Services**.
3. Right-click on **Business Services**, and select **New** > **Business Service**.
4. In the **Create a new Business Service** dialog box, type the **File Name** as `PurchaseOrderNotificationServiceJPDBS`, and click **Next**.
5. In the **Create Business Service - General Configuration (Service Access/Business Services)**, select **WSDL Web Service** Service Type, and click **Browse**. The **Select a WSDL** dialog box is displayed.
6. Expand **PurchaseOrderProcessingService.wsdl,** and **select ProcessPurchaseOrderSoapCallback (port)**, as shown in Figure 8-4, click **OK**.

Figure 8-4 Select a WSDL


Select a WSDL
ServiceAccess
Resources
PurchaseOrderFulfillmentJPDContract.wsdl
PurchaseOrderProcessingService.wsdl
ProcessPurchaseOrderCallbackSoap (port)
ProcessPurchaseOrderSoap (binding)
ProcessPurchaseOrderSoap (port)
ProcessPurchaseOrderSoapCallback (binding)
OK
Cancel
Consume

7. Click **Next** in the **Create a Business Service - Transport Configuration (ServiceAccess/Business Services/)** dialog box, and ensure the following:

   – **Port**: `http`

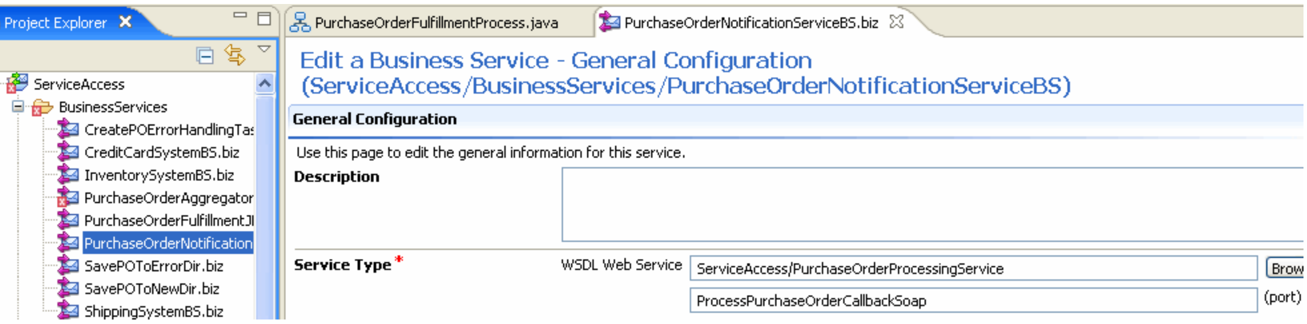
   – **Endpoint URI**: `http://localhost:7001/PurchaseOrderNotificationService`

   Note: Remove the default endpoint. In this case it is `http://examples.org`.

   Click **Next**.

   The `PurchaseOrderNotificationJPDBS.biz` is created in **Project Explorer**. Figure 8-5 shows the business service in **Design** view, and in **Project Explorer**.

Figure 8-5 PurchaseOrderNotificationJPDBS in Package Explorer



Edit a Business Service - General Configuration (ServiceAccess/BusinessServices/PurchaseOrderNotificationServiceBS)

General Configuration

Use this page to edit the general information for this service.

Description

Service Type * WSDL Web Service ServiceAccess/PurchaseOrderProcessingService

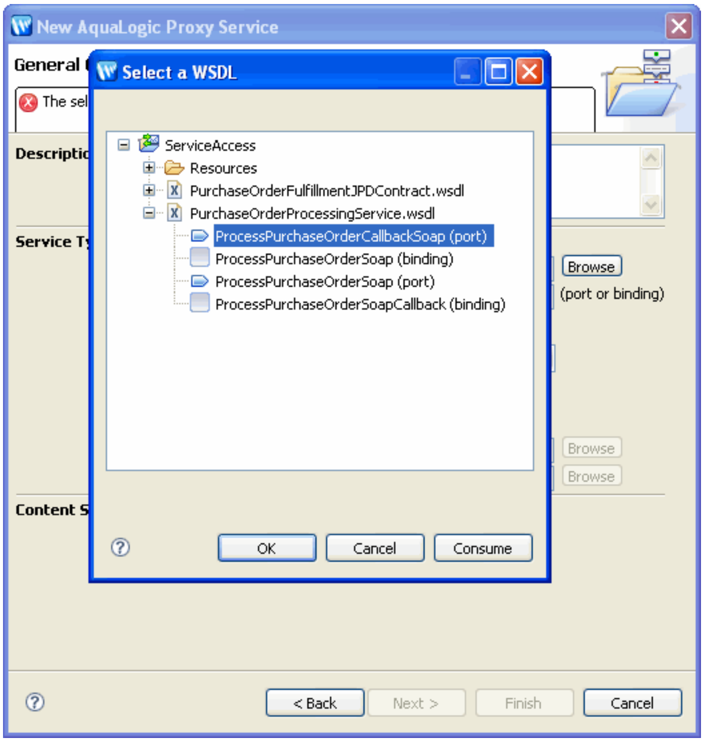
ProcessPurchaseOrderCallbackSoap (port)

Creating PurchaseOrderNotificationService Proxy

To Create PurchaseOrderNotificationService.proxy

1. In the **Project Explorer** pane, select **ServiceAccess** and right-click on **ProxyService**s, and select **New** > **Proxy Service**.
2. In the **Create a new Proxy Service** dialog box, enter `PurchaseOrderNotificationService` as the **File Name**. Click **Next**.
3. Select **WSDL Web Service**, and click **Browse**. Select **ServiceAccess** > **Resources** > **PurchaseOrderProcessingService.wsdl > ProcessPurchaseOrderSoapCallback (port)**. Click **OK**, as shown in Figure 8-6.

Figure 8-6 Select a WSDL


New AquaLogic Proxy Service
General
The sel
Descriptio
Service Ty
Content S
Browse
(port or binding)
Browse
Browse
< Back
Next >
Finish
Cancel
Select a WSDL
ServiceAccess
Resources
PurchaseOrderFulfillmentJPDContract.wsdl
PurchaseOrderProcessingService.wsdl
ProcessPurchaseOrderCallbackSoap (port)
ProcessPurchaseOrderSoap (binding)
ProcessPurchaseOrderSoap (port)
ProcessPurchaseOrderSoapCallback (binding)
OK
Cancel
Consume

Click **OK**.

4. In the **Transport Configuration** dialog box, ensure the following:

– **Protocol**: `jms`

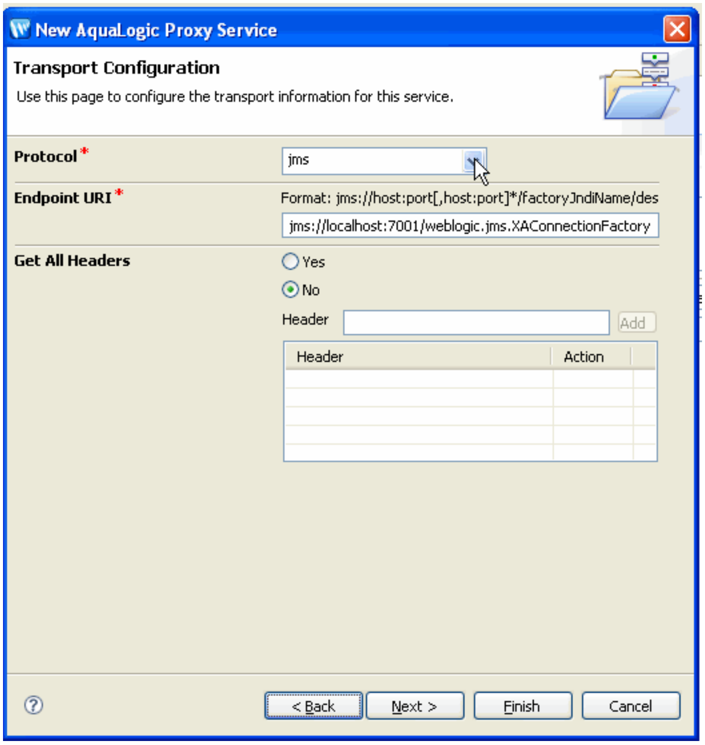
– **Endpoint URI**: `jms://localhost:7001/weblogic.jms.XAConnectionFactory/PurchaseOrderNotificationServiceRequest`

Note: Ensure that the **Endpoint URI** location matches with the **Callback Proxy Location** in the **JPD Transport Configuration** of PurchaseOrderAggregatorJPDBS.biz.

– Get All Headers is set to **Yes**. By default, this value is set to **No**.

Click **Next. Figure 8-8** shows the **Transport Configuration** dialog box.

Figure 8-7 Transport Configuration



New AquaLogic Proxy Service

Transport Configuration

Use this page to configure the transport information for this service.

Protocol *: jms

Endpoint URI *: Format: jms://host:port[,host:port]*/factoryJndiName/des

jms://localhost:7001/weblogic.jms.XAConnectionFactory

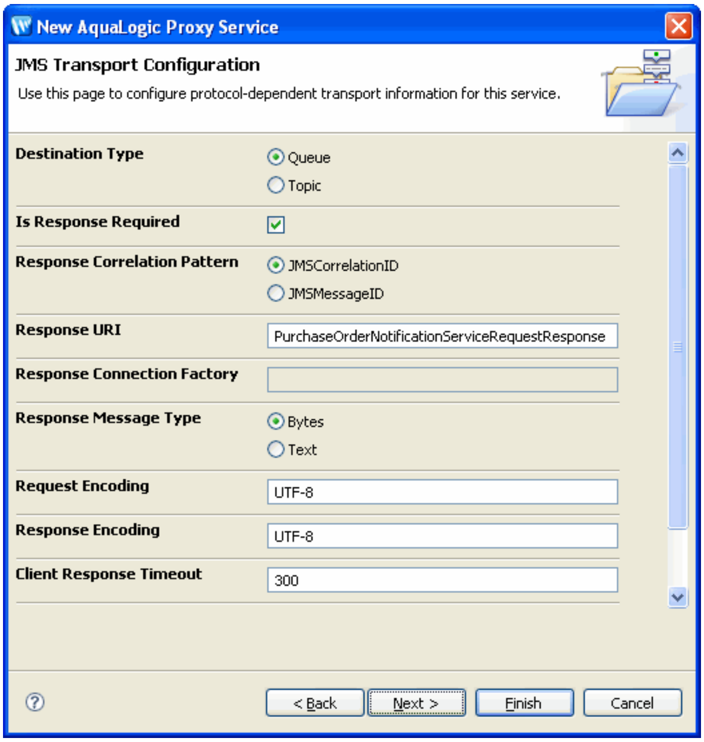
Get All Headers: Yes [✓ No]

Header

Header	Action

5. **In the JMS Transport Configuration** dialog box, select the **Is Response Required** check box, and enter **Response URI** as `jms://localhost:7001/weblogic.jms.XAConnectionFactory/PurchaseOrderNotificationServiceResponse`. Click **Next**. Figure 8-8 shows the **JMS Transport Configuration** dialog box.

Figure 8-8 JMS Transport Configuration



JMS Transport Configuration

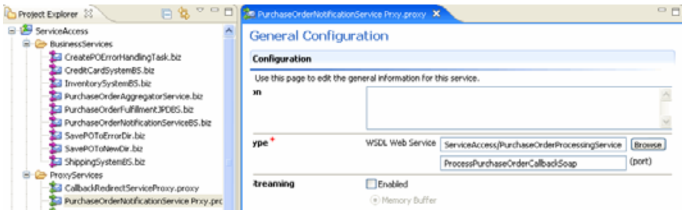
Use this page to configure protocol-dependent transport information for this service.

Destination Type	[✓ Queue] Topic
Is Response Required	✓
Response Correlation Pattern	[✓ JMSCorrelationID] JMSMessageID
Response URI	PurchaseOrderNotificationServiceRequestResponse
Response Connection Factory	
Response Message Type	[✓ Bytes] Text
Request Encoding	UTF-8
Response Encoding	UTF-8
Client Response Timeout	300

6. Accept the default values in the next dialog box, and click **Finish**.

PurchaseOrderNotificationProxy is created in **Project Explorer**, as shown in Figure 8-9.

Figure 8-9 PurchaseOrderNotificationProxy



ServiceAccess
- BusinessServices
  - CreatePOErrorHandlingTask.biz
  - CreditCardSystemBS.biz
  - InventorySystemBS.biz
  - PurchaseOrderAggregatorService.biz
  - PurchaseOrderFulfillmentJPDBS.biz
  - PurchaseOrderNotificationServiceBS.biz
  - SavePOToErrorDir.biz
  - SavePOToNewDir.biz
  - ShippingSystemBS.biz
- ProxyServices
  - CallbackRedirectServiceProxy.proxy
  - PurchaseOrderNotificationService Prxy.prc

Use this page to edit the general information for this service.

ype WSDL Web Service ServiceAccess/PurchaseOrderProcessingService

ProcessPurchaseOrderCallbackSoap

treaming Enabled

Memory Buffer

Designing the PurchaseOrderNotificationProxy Message flow

1. Click the **Message Flow** tab in **PurchaseOrderNotificationProxy**. From **Design Palette**, expand **Nodes**, drag and drop a **Pipleline Pair** under `PurchaseOrderNotificationService.proxy` in the **Design** view.

Note: A **PipelinePair** node consists of a **Request** pipeline and a **Response** pipeline. Pipelines can include one or more stages, which in turn include actions.

2. Expand **Nodes**, drag and drop **Stage** node, under **Request** in **PipelinePairNode**.
3. From the **Design** palette, expand **Stage Actions** > **Communications**, and select **Publish**. Drag and drop **Publish** node under **Stage** node.
4. In the **Properties** tab of the **Publish** node, click **Browse**, and select **PurchaseOrderNotificationServiceBS** from the **Select a Service Resource** dialog box.
5. From the **Invoking** drop-down list in the **Properties** tab, select `PurchaseOrderResponse`.
6. From the **Design** palette, expand **Stage Actions** > **Message Processing**, and select **Assign** node. Drag and drop **Assign** node into the **Publish node**.
7. In the **Properties** tab of the **Assign** node, click **Expression**, and enter the following code in the **XQuery Editor** dialog box that is displayed:

   ```
   $inbound/ctx:transport/ctx:request/tp:headers/tp:user-header[2]/@val
   ue
   ```

   Enter **Variable** as `callbackURL`.

   As discussed in Listing Notes:, the original callback location will be sent as a JMS property namely BEA_WLI_Target_Callback_Location by the JPD transport. We are retrieving it and will be using it to configure the outgoing business service for sending the callback to the actual JPD callback client in the next steps
8. From **Design Palette**, expand **Message Processing**, drag and drop **Insert** node under the **Assign** node. In the **Insert** node, enter the following:

   Expression: `<ctx:uri>{$callbackURL}</ctx:uri>`

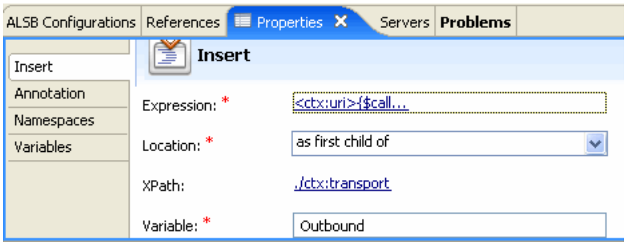
   Location: `as first child of`

   XPath: `./ctx:transport`

   Variable: `outbound`

   Figure 8-10 shows the values in the **Insert** Node.

Figure 8-10 Insert Node



Insert

Expression: * <ctx:uri>{$call...

Location: * as first child of

XPath: ./ctx:transport

Variable: * Outbound

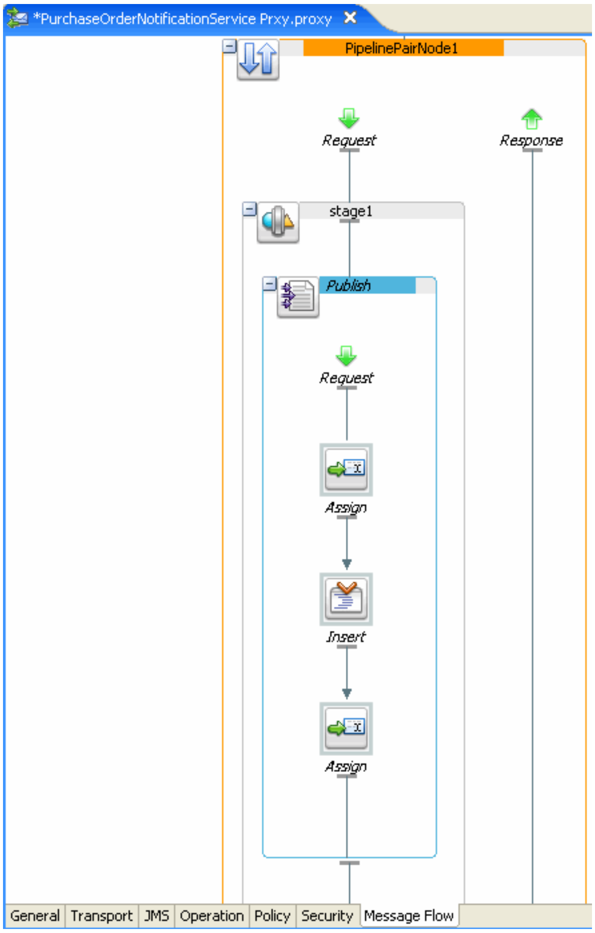
9. From **Design Palette**, expand **Message Processing**, drag and drop an **Assign** node under the **Insert** node. In the **Assign** node Expression, enter the following code:

```
<soap:Body xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

<pur:PurchaseOrderStatus
xmlns:pur="http://www.openuri.org/PurchaseOrder">

{$body/*}

</pur:PurchaseOrderStatus>

</soap:Body>
```

   Assign `body` as **Variable**.

The callback notification is sent to the client via `PurchaseOrderNotificationServiceBS.biz`. Figure 8-11 shows the message flow of callback.

Figure 8-11 Message Flow



Testing the Application

Right-click on the **TestClient.java** file in the **Package Explorer**, select **Run As** > **Run on Server** to test the application as described in "Testing the Application".

Verifying Security Context Propagation

You can review PurchaseOrderFulfillmentJPD to verify that you have not provided any username and password to accessing the external services. Do the following to verify security context propagation:

1. In the UseCase workspace, go to **Package Explorer** > **src** > **alient.client.Test**.
2. Right-click on **TestClient.java**, and select **Run As** > **Run on Server**.

The client is authenticated using the `basic auth` method when the `PurchaseOrderProcessingProxy` is invoked and the security context is propagated to the WLI layer (PurchaseOrderFulfillmentJPD) to ALSB to the external systems (inventory and credit card systems).

Actionable Exception Management

This chapter describes the use of worklist for enabling actionable exception management. This chapter explains how to modify the PurchaseOrderFulfillmentJPD to create a worklist task in case of any exception. You will be reusing the worklist artifacts from the downloaded application *{TUTORIAL_ROOT}*.

This chapter discusses the following topics:

Enabling Worklist Facets

This section describes how to enable the worklist project facets to the EAR and Webapp projects.

To Enable Worklist Facets

1. Select the EAR folder (**UseCase**) from **Package Explorer**.
2. Right-click on the **UseCase** folder, select **Properties**.
3. In the **Properties for UseCase** dialog box, select **Project Facets**.
4. Click **Add/Remove Project Facets...**.
5. Select **AquaLogic Core Facet**, and **WebLogic Integration Worklist Application Module**.
6. Similarly, right-click **UseCaseWeb**, and select **Properties**.
7. In the **Properties for UseCaseWeb** dialog box, click **Add/Remove Project Facets...**. Select **Beehive Controls**, **Struts**, and **WebLogic Integration Worklist Application Module**.

Copying Files Required to Enable Actionable Exception Management

Copy the following files into your workspace:

1. Copy **PublishError.java** from *<TUTORIAL_ROOT>***ALIntE2EUsecase\UsecaseApp\UseCaseWeb\src\alint\process\helper** into the **src\alint.process.helper** folder.
2. Copy **PublishErrorServiceBrokerControl.java** from *<TUTORIAL_ROOT>***ALIntE2EUsecase\UsecaseApp\UseCaseWeb\src\alint\process** into the **src\alint.process** folder.
3. Copy the following files:
   - **CreatePOErrorHandlingTask.java**
   - **CreatePOErrorHandlingTaskService.wsdl**
   - **CreatePOErrorHandlingTaskServiceControl.java**

   from *<TUTORIAL_ROOT>***ALIntE2EUsecase\UsecaseApp\UseCaseWeb\src\alint\errorhandling\ws** into **src\alint.errorhandler** folder.

Importing Worklist Projects

This section describes how to import Worklist projects into the workspace, in which you are building your Purchase Order Application.

Note: You have created a new workspace in "Loading the Sample Application"

To Import Worklist Projects

1. From the **BEA WorkSpace Studio** menu, click **File > Import > General > Existing Projects into Workspace**. The **Import** dialog box is displayed.
2. In the **Import Projects** dialog box, browse to the directory `<TUTORIAL_ROOT>/ALIntE2EUsecase/UsecaseApp` where the tutorial sample application is installed. Select the following files from **Project** pane:
   - Callback Handler
   - UsecaseEH
   - UsecaseEHWeb

   These files are now displayed in **Package Explorer**.
3. From **Package Explorer**, right-click on **Callback Handler**, and select **Properties**.
4. In the **Properties for Callback Handler** window, select **ALSB Configuration**. In the **ALSB Configuration** pane, select **ServiceAccessConfiguration**, and click **Apply**.

The Worklist - related files are imported to the workspace.

Modifying PurchaseOrderFulfillmentJPD for Exception Handling

1. In **Package Explorer**, go to **src** > **alint.process** > **PurchaseOrderFulfillmentJPD.java**.
2. Drag and drop the **ErrorInfoPublishControl.java** into **Data Palette**.
3. Right-click on the **PurchaseOrderFulfillmentJPD** in the **Design** view, and select **Add Exception Path** to create an Exception Path.
4. From the **Data Palette** > **Controls** > **publishErrorServiceBrokerControl**, drag and drop the **void publishError** method into the **onException** path. Right click and select **View Code**.
5. Copy the following code and paste it into the **Source** view.

```
public void publishErrorServiceBrokerControlPublishError() throws Exception
{
        String errorInfo =
context.getExceptionInfo().getException().getMessage();
        // #START: CODE GENERATED - PROTECTED SECTION - you can safely add
code above this comment in this method. #//
        // input transform
        // method call
        publishErrorServiceBrokerControl.publishError(this.orderMetadata,
errorInfo);
        // output transform
        // output assignments
        // #END  : CODE GENERATED - PROTECTED SECTION - you can safely add
code below this comment in this method. #//
    }
```

Modifying the Purchase Order Aggregator JPD to Handle Errors

Open the **PurchaseOrderAggregatorJPD** from **src\alint.process**.

To Modify the POAggregatorJPD

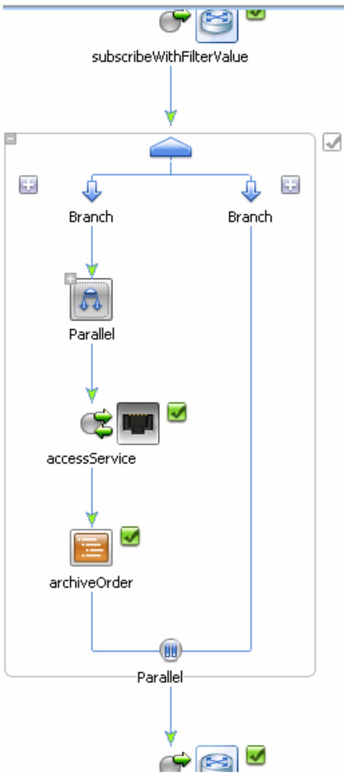
1. Drag and drop **ErrorInfoSubscriptionControl.java** from **src\alint.process.control** into the **Data Palette**.
2. In the **Source** view of the JPD, type the following annotation, before the control declaration line:

   ```
   @SubscriptionControl.ClassSubscription(channelName =
   "/ALIntPOApp/ErrorInfo",
           xquery = "data($metadata)",
           xqueryVersion = com.bea.wli.common.XQuery.Version.v2004)
   ```

3. Expand the **errorInfoSubscriptionControl.java** from the **Data Palette > Controls** folder.
4. Drag and drop the **void subsrcibeWithFilterValue**, after the **ccTxInfoSbscrptnn** node in the **Design** view.
5. Double click on this node. In the **Send Data** tab, select **order Id** from **Select variables to assign** drop-down list.

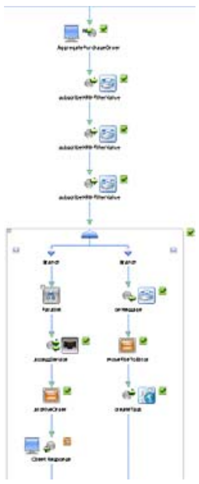
6. Drag and drop a **Parallel** node from the **Node Palette**. The **PurchaseOrderAggregatorJPD** is now updated as shown in Figure 9-1.

Figure 9-1 Parallel Nodes to the JPD



7. Drag and drop the **Parallel** node, **accessService** node, and the **archiveOrder** node under the first **Branch** on this newly created **Parallel** node. The updated JPD is as shown in Figure 9-4.

Figure 9-2 Updated POAggregatorJPD



8. From the **Data Palette** > **Controls** > **errorInfoSubscriptionControl**, drag and drop **void onMessage** method to the empty parallel branch.
9. Double click **onMessage**. In the **Receive Data** tab, select **create a new variable** from **create variables to assign** drop-down list.
10. In the **Create Variable** dialog box, enter errorMessage as the **Variable Name**. Click **OK**.

    Click **Close**.
11. Drag and drop a **Perform** node from the **Node Palette** under this **onMessage** node.
12. Rename node to `moveFileToError`.
13. Right-click on this node, and select **View Code**. Add the following code:

    ```
    File orderFile = new File("C:/ALIntApp/PurchaseOrder/New/PO-" +
    orderId + ".xml");

    orderFile.renameTo(new File("C:/ALIntApp/PurchaseOrder/Error/PO-" +
    orderId + ".xml"));
    ```
14. Drag and drop **CreatePOErrorHandlingTaskServiceControl** from **src** > **alint.errorhandling.ws** into the **Data Palette**.
15. From the **createPoErrorHandlingTaskServiceControl**, drag and drop **void CreateTask** method to the **Design** view.

16. Right-click on **createTask**, and select **View Code**. Edit the arguments of the method:

```
createPOErrorHandlingTaskServiceControl.createTask(this.customerName
,this.orderId, this.errorMessage,
callback.getEndPoint().toExternalForm(),
context.getService().getConversationID());
```

You have now created a worklist task using a webservice wrapper.

Modifying the PurchaseOrderProcessingServiceProxy to Save the Incoming Messages for Future Reference

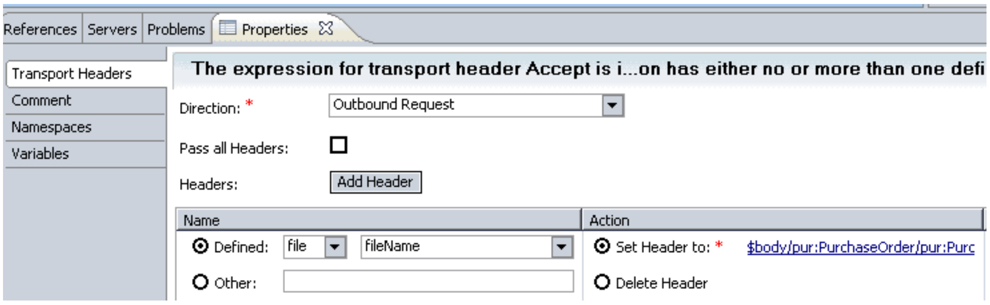
To Modify the PurchaseOrderProcessing Proxy

1. Open the message flow for PurchaseOrderProcessingProxy.
2. Drag and drop a publish node, so that it is the first node in the pipe line you have already created.

   Note: The pipeline has already been configured to publish to PurchaseOrderAggregatorJPD and PurchaseOrderFulfillmentJPD. The new publish node you are adding will be before these two.
3. Drag and drop **Stage Actions** > **Transport Headers** into the Publish node.
4. In the **Transport Header Properties** pane, ensure **Direction** is `Outbound Request`.
5. Click on **Add Header**.
6. Select **file** and **filename** from drop down lists in the **Name** pane.
7. For the expression, **Set Header To** give the following value:

```
$body/pur:PurchaseOrder/pur:PurchaseOrderId/text()
```

Figure 9-3 Properties of Transport Header



References | Servers | Problems | Properties

Transport Headers
Comment
Namespaces
Variables

The expression for transport header Accept is i...on has either no or more than one defi

Direction: * Outbound Request

Pass all Headers: ☐

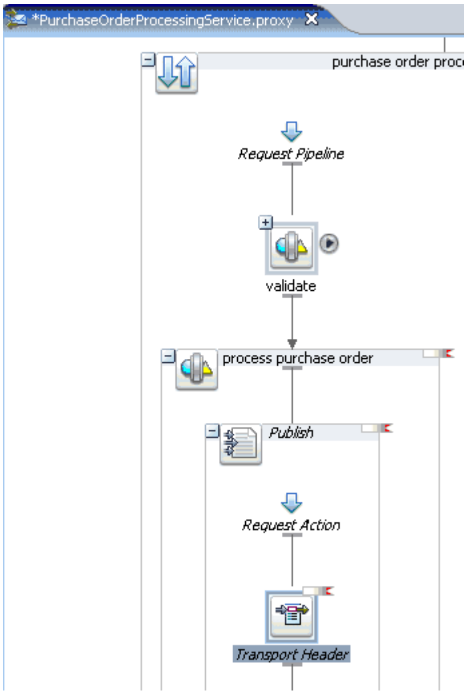
Headers: Add Header

Name	Action
● Defined: file fileName	● Set Header to: * $body/pur:PurchaseOrder/pur:Purc
○ Other:	○ Delete Header

8. Save the proxy.

You have now modified the proxy message flow to save the incoming purchase order to your file system for future reference. The modified proxy, with the transport header is shown in

Figure 9-4 Modified PO Processing Proxy



Providing Anonymous Access to Task Plan

The `ReprocessPOHandlerJPD.java` located at **`UsecaseApp\UsecaseEHWeb\src\alint\errorhandling`** needs to access task data and reprocess the Purchase Order.

To Provide Anonymous Access

1. Log on to Worklist Console: `http://localhost7001/worklistconsole`.

   For example, you can provide **User Name**/ **Password** as `weblogic/weblogic`.

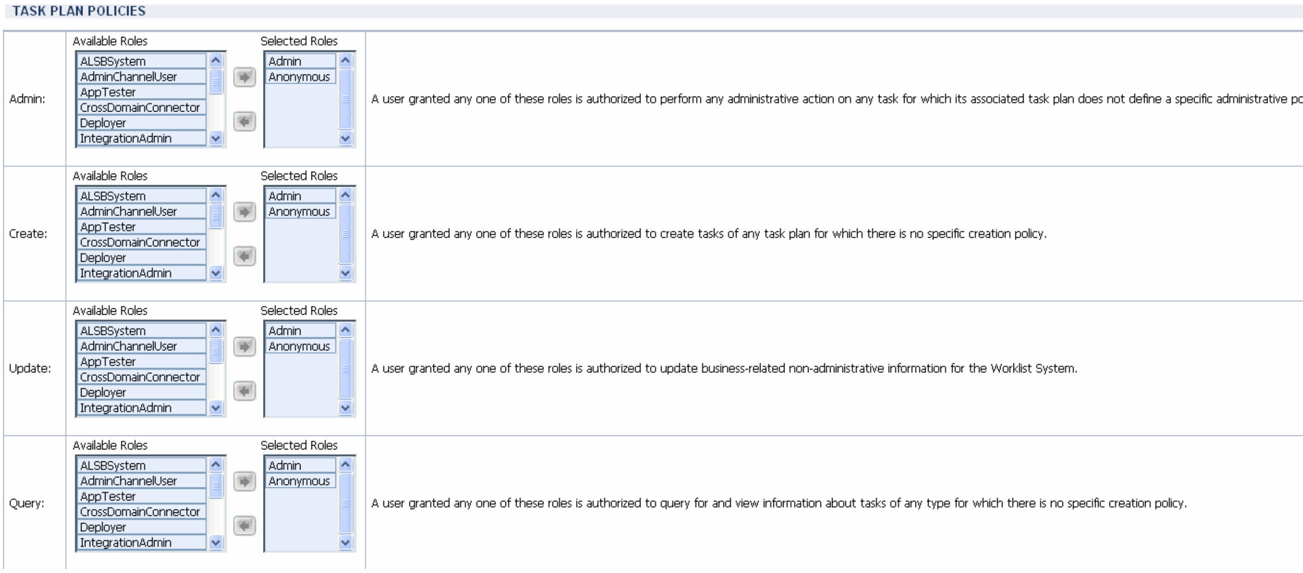
2. From the **Worklist System Instance**, expand **UsecaseEH**.

3. Click **`/task_plans/POExceptionHandlingTaskPlan`**.

4. Click **Edit** in **Task Plan Policies** pane.

5. From the **Task Plan Policies** pane, select add Admin, and Anonymous roles to the **Selected Roles** column from the **Available Roles** for **Admin**, **Update**, **Create**, and **Query** as shown in Figure 9-5.

Figure 9-5 Selecting Roles



TASK PLAN POLICIES

	Available Roles	Selected Roles	
Admin:	ALSBSystem, AdminChannelUser, AppTester, CrossDomainConnector, Deployer, IntegrationAdmin	Admin, Anonymous	A user granted any one of these roles is authorized to perform any administrative action on any task for which its associated task plan does not define a specific administrative pc
Create:	ALSBSystem, AdminChannelUser, AppTester, CrossDomainConnector, Deployer, IntegrationAdmin	Admin, Anonymous	A user granted any one of these roles is authorized to create tasks of any task plan for which there is no specific creation policy.
Update:	ALSBSystem, AdminChannelUser, AppTester, CrossDomainConnector, Deployer, IntegrationAdmin	Admin, Anonymous	A user granted any one of these roles is authorized to update business-related non-administrative information for the Worklist System.
Query:	ALSBSystem, AdminChannelUser, AppTester, CrossDomainConnector, Deployer, IntegrationAdmin	Admin, Anonymous	A user granted any one of these roles is authorized to query for and view information about tasks of any type for which there is no specific creation policy.

6. Click **Submit**.

The Task Plan policies get updated.

Testing Actionable Exception Management

After you have published the sample application, enter invalid data in the sample application browser. For example in the Soap body of the TestClient, provide a wrong PurchaseOrder ID. To verify actionable exception management, log on to the Worklist User Portal, and reprocess the error. Also, for more information about testing the sample application, see "Testing the Application."

The following steps in this section describes how to verify Actionable Exception Management:

- "To Enter Values in the TestClient"
- "To Verify the Values in Worklist User Portal"
- "To Modify the .XML File"
- "To Reprocess the Order in Worklist User Portal"

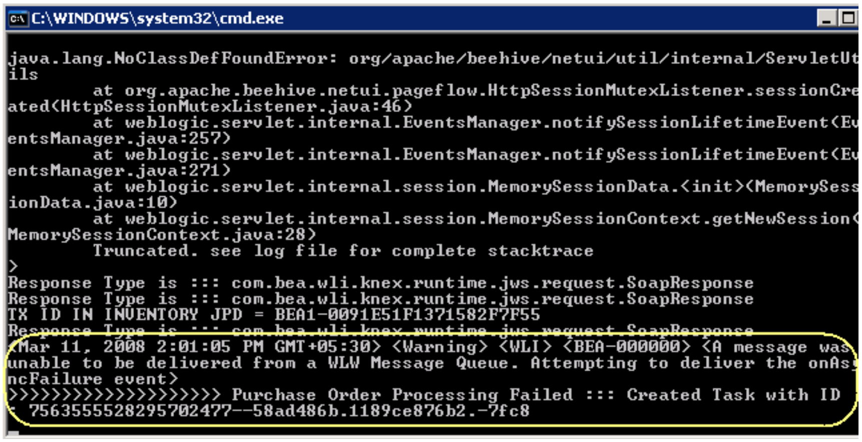
To Enter Values in the TestClient

1. In the `http://localhost:7001/UseCaseWeb/alint/client/test/TestClient.jpd`, click **TestSoap**.
2. In the `SOAP body`, enter the values as follows:
   - PurchaseOrderId (POMarch11)
   - CreditCardNumber (1234567890123456)
   - ExpiryDate (08/08)
   - ItemId (XYZ)

   Note: The PurchaseOrderId, and the ItemId are different from the values in the database. See "Testing the Application."
3. Click **clientRequest1**.
4. Click **Refresh** on the browser.

   An exception is thrown, and this can be viewed on the WebLogic Server console as shown in Figure 9-6:

Figure 9-6 Failure Message on WLS Console

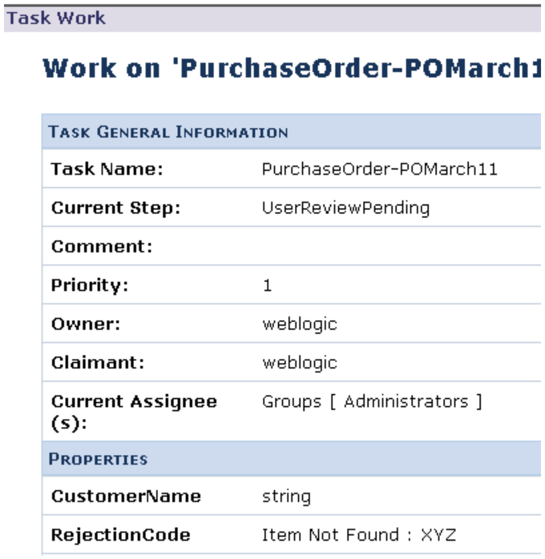


```
java.lang.NoClassDefFoundError: org/apache/beehive/netui/util/internal/ServletUt
ils
        at org.apache.beehive.netui.pageflow.HttpSessionMutexListener.sessionCre
ated(HttpSessionMutexListener.java:46)
        at weblogic.servlet.internal.EventsManager.notifySessionLifetimeEvent(Ev
entsManager.java:257)
        at weblogic.servlet.internal.EventsManager.notifySessionLifetimeEvent(Ev
entsManager.java:271)
        at weblogic.servlet.internal.session.MemorySessionData.<init>(MemorySess
ionData.java:10)
        at weblogic.servlet.internal.session.MemorySessionContext.getNewSession(
MemorySessionContext.java:28)
        Truncated. see log file for complete stacktrace
>
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
TX ID IN INVENTORY JPD = BEA1-0091E51F1371582F7F55
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
<Mar 11, 2008 2:01:05 PM GMT+05:30> <Warning> <WLI> <BEA-000000> <A message was
unable to be delivered from a WLW Message Queue. Attempting to deliver the onAsy
ncFailure event>
>>>>>>>>>>>>>>>>>>>>> Purchase Order Processing Failed ::: Created Task with ID
: 756355528295702477--58ad486b.1189ce876b2.-7fc8
```

To Verify the Values in Worklist User Portal

1. Logon to `http://localhost:7001/UsecaseEHWeb/user.portal`.
2. The Purchase Order (**PurchaseOrder-POMarch11**) you created is listed in the list of **Upcoming Tasks**. Click **PurchaseOrder-POMarch11**.
3. In **Properties** pane of the **Task Work** window, note that the **Rejection Code** is the ItemId (XYZ), you specified "To Enter Values in the TestClient", as shown in Figure 9-7.

Figure 9-7 Properties Pane



Task Work

Work on 'PurchaseOrder-POMarch1

TASK GENERAL INFORMATION	
Task Name:	PurchaseOrder-POMarch11
Current Step:	UserReviewPending
Comment:	
Priority:	1
Owner:	weblogic
Claimant:	weblogic
Current Assignee (s):	Groups [Administrators]
PROPERTIES	
CustomerName	string
RejectionCode	Item Not Found : XYZ

In the next steps, you can correct the values in the Worklist User Portal and reprocess the order, without any exceptions, after modifying the `.xml` file that is now created in the `ALIntApp\Purchase Order\Error` folder. See "Creating the Tutorial Environment," for creating these folders.

To Modify the .XML File

1. Select the `PO-POMarch11.xml` that is created in `ALIntApp\Purchase Order\Error folder`.
2. Edit the ItemId in `PO-POMarch11.xml` file by providing the following values, by entering the IdemId as `1`.
3. Save the `PO-POMarch11.xml` file.

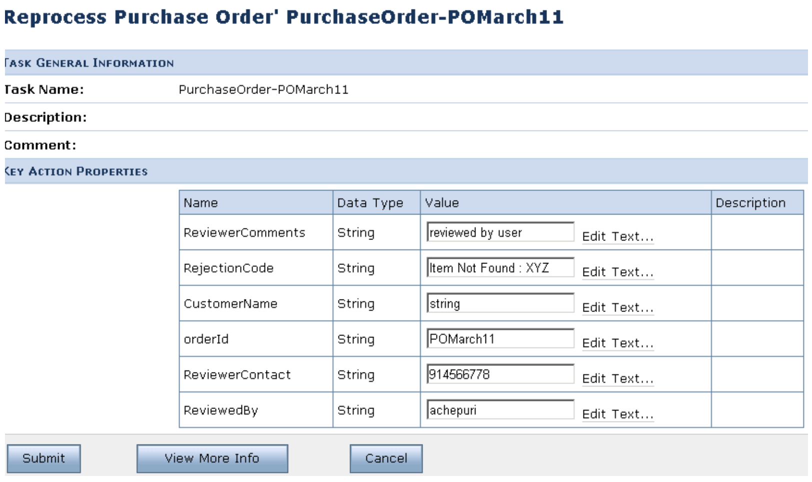
To Reprocess the Order in Worklist User Portal

1. Logon to `http://localhost:7001/UsecaseEHWeb/user.portal`.
2. Click **Show all tasks in this view** from the **Upcoming Tasks** pane.
3. Click **PurchaseOrder-POMarch11** from **Task Name** column.

   The **Task Work** window is displayed.

4. Click **Work**.
5. In the next screen, click **Reprocess Purchase Order,** under **Actions** pane. Click **Next**.
6. Enter the desired values for **Reviewer Comments**, **ReviewerContact**, **ReviewedBy**, as shown in Figure 9-8.

Figure 9-8 Reprocessing Purchase Order



Reprocess Purchase Order' PurchaseOrder-POMarch11

Task General Information

Task Name:	PurchaseOrder-POMarch11
Description:	
Comment:	

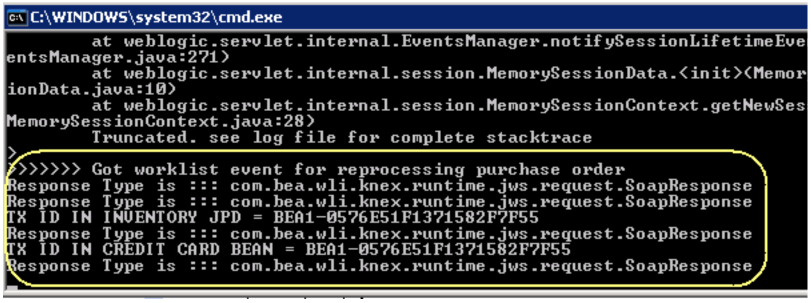
Key Action Properties

Name	Data Type	Value	Description
ReviewerComments	String	reviewed by user Edit Text...	
RejectionCode	String	Item Not Found : XYZ Edit Text...	
CustomerName	String	string Edit Text...	
orderId	String	POMarch11 Edit Text...	
ReviewerContact	String	914566778 Edit Text...	
ReviewedBy	String	achepuri Edit Text...	

7. Click **Submit**.

   The Purchase Order is reprocessed with the correct values, and the transactions are displayed on the WLS console as shown in Figure 9-9.

Figure 9-9 Transaction Completed Message on WLS Console



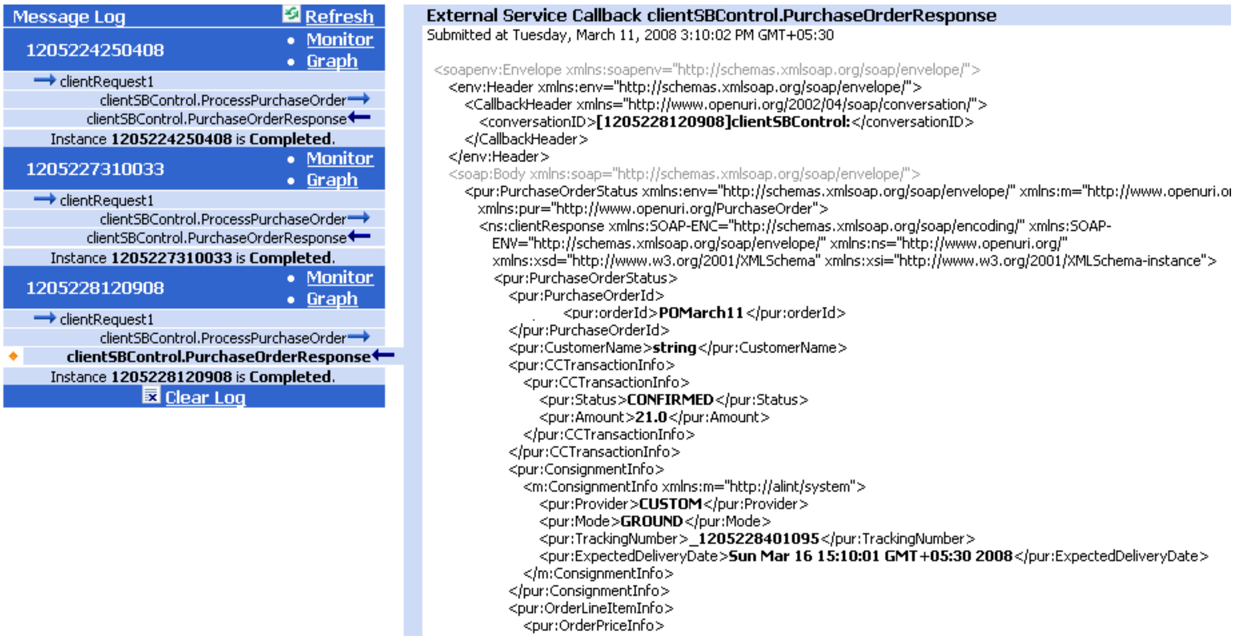
C:\WINDOWS\system32\cmd.exe

```
        at weblogic.servlet.internal.EventsManager.notifySessionLifetimeEve
entsManager.java:271)
        at weblogic.servlet.internal.session.MemorySessionData.<init>(Memor
ionData.java:10)
        at weblogic.servlet.internal.session.MemorySessionContext.getNewSes
MemorySessionContext.java:28)
        Truncated. see log file for complete stacktrace
>
>>>>>> Got worklist event for reprocessing purchase order
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
TX ID IN INVENTORY JPD = BEA1-0576E51F1371582F7F55
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
TX ID IN CREDIT CARD BEAN = BEA1-0576E51F1371582F7F55
Response Type is ::: com.bea.wli.knex.runtime.jws.request.SoapResponse
```

8. In the `http://localhost:7001/UseCaseWeb/alint/client/test/TestClient.jpd` browser, click **Refresh**.

   The callback message is displayed as shown in Figure 9-9.

Figure 9-10 Callback Message



```
1205224250408
clientRequest1
clientSBControl.ProcessPurchaseOrder
clientSBControl.PurchaseOrderResponse
Instance 1205224250408 is Completed.
1205227310033
clientRequest1
clientSBControl.ProcessPurchaseOrder
clientSBControl.PurchaseOrderResponse
Instance 1205227310033 is Completed.
1205228120908
clientRequest1
clientSBControl.ProcessPurchaseOrder
clientSBControl.PurchaseOrderResponse
Instance 1205228120908 is Completed.
```

External Service Callback clientSBControl.PurchaseOrderResponse

Submitted at Tuesday, March 11, 2008 3:10:02 PM GMT+05:30

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
    <CallbackHeader xmlns="http://www.openuri.org/2002/04/soap/conversation/">
      <conversationID>[1205228120908]clientSBControl:</conversationID>
    </CallbackHeader>
  </env:Header>
  <soap:Body xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <pur:PurchaseOrderStatus xmlns:env="http://schemas.xmlsoap.org/soap/envelope/" xmlns:m="http://www.openuri.o
      xmlns:pur="http://www.openuri.org/PurchaseOrder">
      <ns:clientResponse xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
        ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:ns="http://www.openuri.org/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <pur:PurchaseOrderStatus>
          <pur:PurchaseOrderId>
                <pur:orderId>POMarch11</pur:orderId>
          </pur:PurchaseOrderId>
          <pur:CustomerName>string</pur:CustomerName>
          <pur:CCTransactionInfo>
            <pur:CCTransactionInfo>
              <pur:Status>CONFIRMED</pur:Status>
              <pur:Amount>21.0</pur:Amount>
            </pur:CCTransactionInfo>
          </pur:CCTransactionInfo>
          <pur:ConsignmentInfo>
            <m:ConsignmentInfo xmlns:m="http://alint/system">
              <pur:Provider>CUSTOM</pur:Provider>
              <pur:Mode>GROUND</pur:Mode>
              <pur:TrackingNumber>_1205228401095</pur:TrackingNumber>
              <pur:ExpectedDeliveryDate>Sun Mar 16 15:10:01 GMT+05:30 2008</pur:ExpectedDeliveryDate>
            </m:ConsignmentInfo>
          </pur:ConsignmentInfo>
          <pur:OrderLineItemInfo>
            <pur:OrderPriceInfo>
```