BEA WebLogic® Integration

Tutorial: Designing Your First Business Process

Version 10.2
Document Revised: March 2008

Contents

CHAPTER 1

Introduction

The Business Process Management (BPM) capability of WebLogic Integration (WLI) enables integration of diverse applications and human participants, and coordinated exchange of information with trading partners outside your enterprise.

This tutorial provides a tour of the features available for designing business processes in Workshop for WebLogic. It describes how to create a business process that orchestrates processing of a Request for Quote (RFQ).

Tutorial Goals

The goals of the tutorial include the following:

- Designing communication nodes in a process – that is, creating the interface between your business process and its clients and resources. Clients of business processes can be any other resources or services that invoke processes.
- Designing interactions with clients, including creating the methods that expose the functionality of a process.
- Designing interactions with resources by using controls. WLI controls make it easy to access enterprise resources – databases, Enterprise Java Beans (EJBs), web services, and other processes (including those that use RosettaNet and ebXML business processes) – from a process.
- Handling XML, non-XML, and Java data types in a process. This includes working with XML schemas and transforming data between disparate data types using the XQuery Mapper tool.

Tutorial Overview

The business process in this scenario starts when an RFQ is received from a client. The business process checks the inventory and pricing systems of the enterprise to determine whether the order can be fulfilled. Based on the shipping address provided by the client, the process also determines whether sales tax must be added to the quote. Finally, the business process compiles a single quote document from the sales tax, price, and availability data, logs the quote by writing it to the file system and sends it to the client.

The following sequence summarizes the steps in the RFQ business process and describes how the process is designed in WLI:

1. Receive an RFQ from a client.

   You design a **Client Request** node in your business process to handle the receipt of an XML document that contains the customer name, shipping address, and the identity and quantity of items for which the quote is requested. You design the business process so that it starts when it receives an RFQ message from a client.

2. Determine whether sales tax must be included in the quote.

   You design a **Decision** node to create different paths of execution based on the evaluation of a condition. The **Decision** node includes, on one path, a call to a web service that calculates sales tax. Business processes communicate with other services through controls. You design a **Control Send** node to communicate with a web service that calculates the sales tax for your quote.

3. Process the items sent in the RFQ message.

   For each item in the RFQ, the process calculates the price and determines availability of the quantities requested in the incoming XML message.

   You create the following nodes in the business process:

   – **For Each**: These nodes represent points in a business process at which a set of activities is performed repeatedly, once for each item in a list. **For Each** nodes include an iterator node (on which a list of items is specified) and a loop (in which the activities to be performed for each item in the list are defined)

   – **Parallel:** Parallel nodes represent points in a business process at which a number of activities are executed in parallel. In this tutorial, you design a **Parallel** node containing two branches: a path for calculating prices and a path for determining availability.

   – **Control**: **Control Send** and **Control Receive** nodes on each path handle asynchronous exchange of messages between the business process and web service resources.

- A pricing web service returns the price for the items in the RFQ.
- An availability web service returns information about the availability of the requested items.

4. Compile price, availability, and tax information in a quote document.

   You use **Transformation** controls to map the price, availability, and sales tax information to an XML document that is returned to the client as the quote.

5. Keep a record of the quote created by the business process.

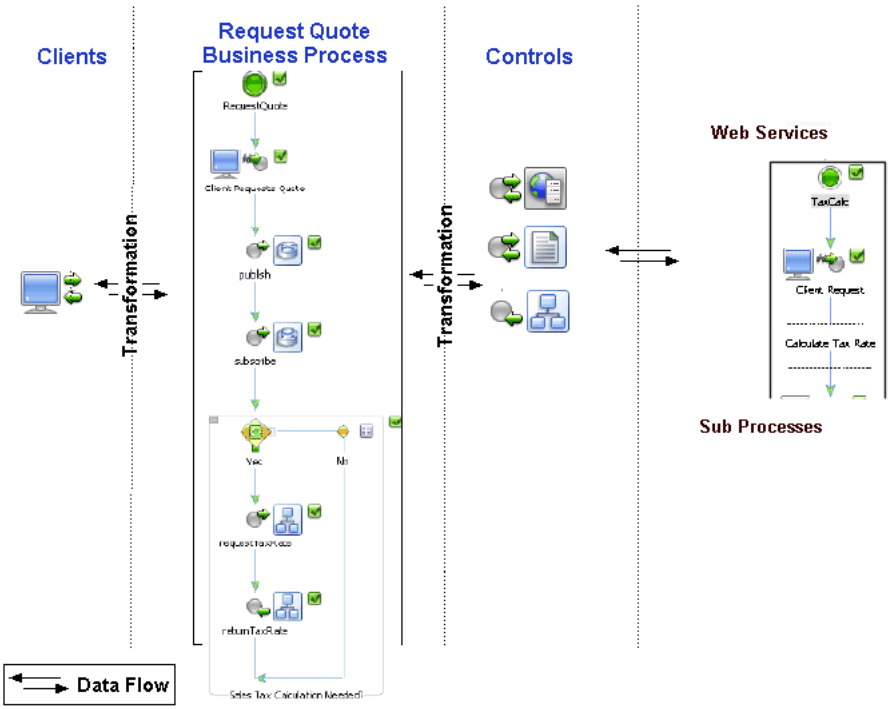
   You use a **File** control to write the quote to your file system.

6. Send the quote to the client.

   You design a **Client Response** node to send a response to the client. The response contains the data calculated by the business process.

The following figure shows the actors in the tutorial scenario.

Figure 1-1 Actors in the Tutorial Scenario



The actors in the tutorial scenario are:

- Clients that create and send RFQ messages – containing customer name, shipping address, list of items, and quantity of items required – to the RequestQuote business process.
- The RequestQuote business process that receives RFQs and returns the following for the items requested in the RFQ:
  - Price
  - Availability information
- A web service that calculates sales tax for the quote, based on the shipping address provided by the client.
- A business process that calculates the sales tax. This business process serves the same purpose as the tax calculation web service described in the preceding item. The RequestQuote business process can call either the web service or the business process to request for calculation of the sales tax for the quote.
- A pricing web service that calculates the price of the items requested by the client.
- An availability web service that determines availability of the items requested by the client.
- Transformation controls to map disparate data formats in your application.

  The business process starts when it receives an XML document from a client. Data is exchanged between resources in the application – clients, the RequestQuote business process, web services, and so on – in XML format.

Tutorial Organization

The goal of this tutorial is to create a business process that receives RFQ messages from clients, validates and processes the RFQs, and send quotes to the clients.

The tutorial is organized as follows:

- Part I, "Design and Run a Simple Process"

  In this part of the tutorial, you do the following:

  a. Create a business process.

  b. Specify how the process is started at run time.

  c. Design a **Decision** node that includes asynchronous calls to a web service.

d. Test the business process.

e. Replace the asynchronous call to the web service with a asynchronous call to another business process.

f. Design interactions between the business process and external resources.

- Part II, "Add Complex Business Logic"

  In this part of the tutorial, you add more complex business logic to the business process.

  – Create looping logic

  – Design parallel processing nodes

  – Transform the price and availability data from untyped XML data to typed XML

  – Use a **File** control to write your quote to a file system

  – Use a **Client Response** node to return the quote to the client invoking the business process.

Part I Design and Run a Simple Process

In this part of the tutorial, you create a business process, specify how the process is started at run time, design a **Decision** node that includes asynchronous calls to a web service, and test the business process.

This part consists of the following steps:

- Chapter 2, “Step 1: Create the Business Process Application”

  Describes how to create a business process.

- Chapter 3, “Step 2: Specify How the Process Is Started”

  Describes how to design the start of your business process.

- Chapter 4, “Step 3: Define Conditions for Alternate Paths of Execution”

  Describes how to design a **Decision** node and its associated conditions in the business process. The path of execution through a decision node is based on evaluation of the conditions that you specify for the node.

- Chapter 5, “Step 4: Invoke a Web Service”

  Describes how to design the interaction between the business process and a web service control.

- Chapter 6, “Step 5: Run the Business Process”

  At this point, you have created a business process, which you can test by using the test browser of WorkSpace Studio.

For information about the components available in WorkSpace Studio for designing business processes, see Appendix A, “WorkSpace Studio Views, Functions, and Shortcuts.”

CHAPTER 2

Step 1: Create the Business Process Application

In this step, you create the application in which you design the tutorial business process (**`RequestQuote.java`**).

This step includes the following tasks:

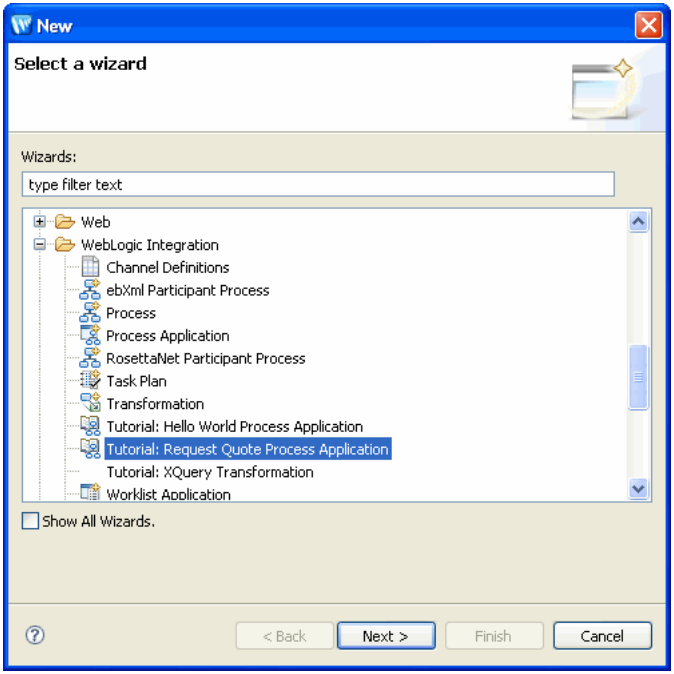
- Create Business Process Tutorial Application
- Start Designing the Process

Create Business Process Tutorial Application

1. From the WorkSpace Studio menu, choose **File > New > Other**.

   The **New** wizard is displayed.

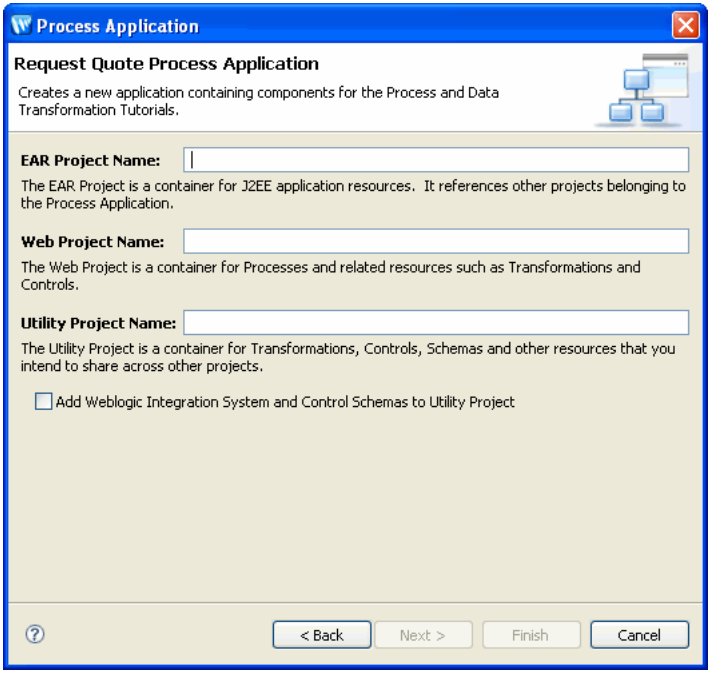
Figure 2-1 New Wizard



2. Expand **WebLogic Integration**, select **Tutorial: Request Quote Process Application**, and click **Next**.

   The **Process Application** dialog box is displayed.

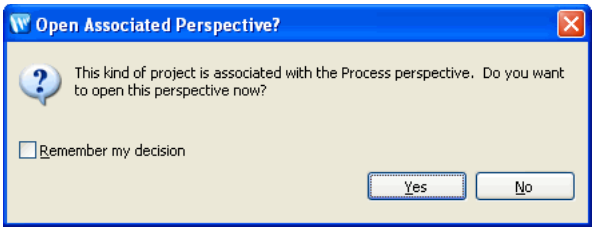
Figure 2-2 Process Application Dialog Box



3. Enter the following:
   a. **EAR Project Name**: `Tutorial_Process_Application_Ear`
   b. **Web Project Name**: `Tutorial_Process_Application_Web`
   c. **Utility Project Name**: `Tutorial_Process_Application_Utility`
4. Select the **Add WebLogic Integration System and Control Schemas to Utility Project** check box to add the system schemas to the **Schemas** folder under the **Utility** project.
5. Click **Finish**.

The **Open Associated Perspective?** dialog box is displayed.

Figure 2-3 Open Associated Perspective Dialog Box



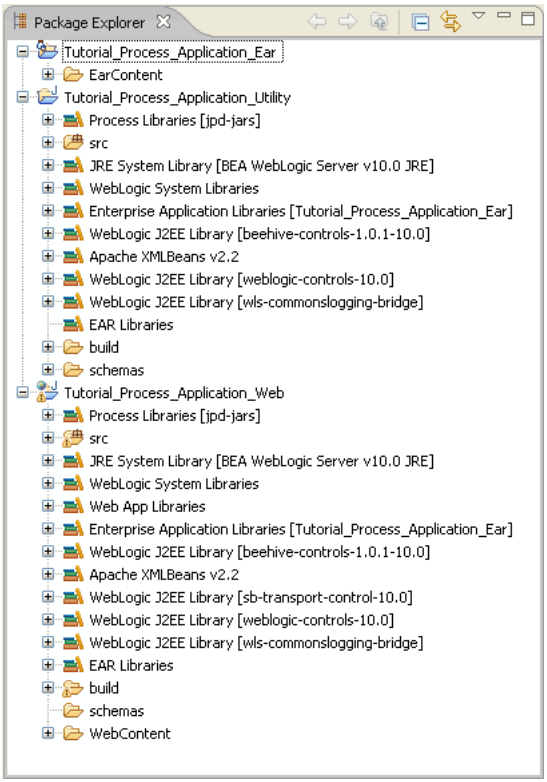
6. Click **Yes** to switch to the **Process** perspective.

   Note: Perspectives define the initial set of views and their associated layout in WorkSpace Studio. The **Process** perspective contains all the views that are necessary for creating process applications: **Node Palette**, **Data Palette**, and so on.

   Similarly, the **XQuery Transformation** perspective contains the views that are relevant for XQuery transformation: **Expression Functions, Expression Variables**, **Target Expression**, and **Constraints**, and so on.

The application is created and displayed in the **Package Explorer** view.

Figure 2-4 Package Explorer View



The **Package Explorer** view displays the files and resources available in the application:

– **Tutorial_Process_Application_Web**: A project with WLI process facet added to it. Every application contains one or more projects. Projects represent WebLogic Server (WLS) applications. In other words, when you create a project, you are creating a web application. The name of your project is included in the URL that clients use to access the application.

  The **`src/requestquote`** folder contains the business processes, transformation, XQuery files.

  • **`FileQuote.java`**: A **File** control used by the RFQ process to write the quote to the file system.

- **`PriceAvailTransformations.java`**: Contains data transformations used in **`RequestQuote.java`**.
- **`RequestQuote.java`**: This is the completed RequestQuote business process. It is provided for reference and to let you run the business process before you start recreating it. The tutorial walks you through the steps to recreate this business process.

  Note: For information about running the **`RequestQuote.java`** business process that is provided in the application folder, see Chapter 13, "Step 12: Run RequestQuote Business Process."
- **`RequestQuoteTransformation.java`** and **`TutorialJoin.java`**: Contain data transformations used in **`RequestQuote.java`**.
- **XQ files**: An XQ file is created for each transformation method on a transformation file. XQ files contain the queries (written in the XQuery language) called by the transformation files in your project.

The **`requestquote.services`** folder contains services with which your process interacts: web services, web service controls, processes, and process controls.

The **`testxml`** folder contains XML files that you can use to test the completed business process.

– **Tutorial_Process_Application_Utility**: A project that contains the XML schemas used in the application.

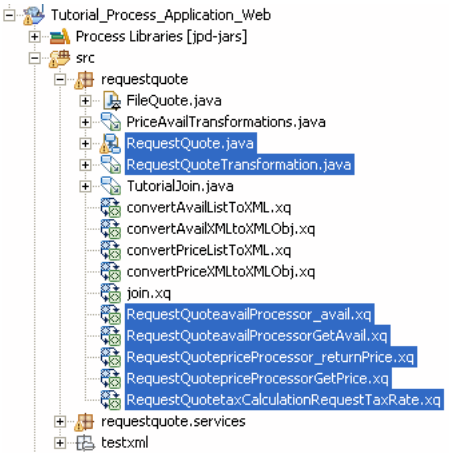
Web applications are J2EE deployment units that define a collection of web resources – business processes, web services, JSPs, servlets, and so on – and can define references to external resources such as EJBs.

7. In this tutorial, you create the **`RequestQuote.java`** file from scratch.

   To proceed, delete the following files from the **`src/requestquote`** folder of the **Tutorial_Process_Application_Web** project:

   – **`RequestQuote.java`**
   – **`RequestQuoteTransformation.java`**
   – **`RequestQuoteavailProcessor_avail.xq`**
   – **`RequestQuoteavailProcessorGetAvail.xq`**
   – **`RequestQuotepriceProcessor_returnPrice.xq`**
   – **`RequestQuotepriceProcessorGetPrice.xq`**
   – **`RequestQuotetaxCalculationRequestTaxRate.xq`**

Figure 2-5 Web Project Files



Caution: Delete only the files that are listed in this step. You need all the other files to create the business process.

To delete the files, select them, and press the **Delete** key (or right-click and select **Delete**).

Note: You can select multiple files by holding down the **Ctrl** key while clicking on the file names.

The files are deleted from your application folder in the file system, and do not appear in the **Package Explorer** view.

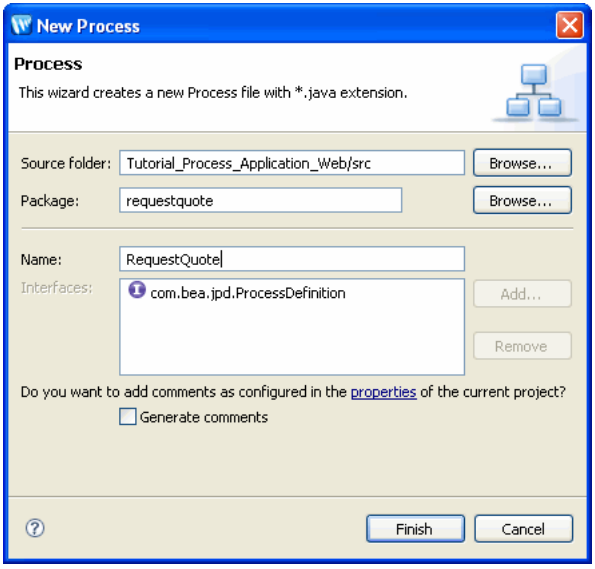
Start Designing the Process

In this step, you start the process of recreating the **`RequestQuote.java`** file.

1. In the **Package Explorer** view, go to the **`Tutorial_Process_ApplicationWeb\src`** folder, and right-click the **`requestquote`** folder.
2. Choose **New > Process**.

   The **New Process** dialog box is displayed.

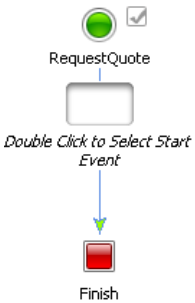
Figure 2-6 New Process Dialog Box



3. In the **Name** field, enter **`RequestQuote`**.
4. Click **Finish**.

   The new **`RequestQuote.java`** file is created and displayed in the **Design** view. At the moment, it has only a **Start** and a **Finish** node.

Figure 2-7 New Process



Note: For more information about the views in WorkSpace Studio, see Appendix A, "WorkSpace Studio Views, Functions, and Shortcuts."

Step 2: Specify How the Process Is Started

In this step, you define how your business process is started. As web services, business processes expose their functionality through methods, which clients invoke to make requests. You can also create **Process** controls from business processes. In the case of **Process** controls, other resources can interact with your process via the controls interface.

Note: You will learn more about **Process** controls in Step 6: Invoke a Business Process.

In this step, you design the **Start** node in your business process to receive a Request for Quote (RFQ) message from a client; receipt of the RFQ message is the trigger that starts the business process.You also create a variable to hold the incoming RFQ message.

In the **Design** view, interactions between a business process and a client are represented by **Client Request** and **Client Response** nodes. In this case, you add a **Client Request** node to your business process and, subsequently, create the code for this node to handle receipt of a message from a client.

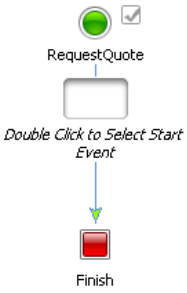
Complete the following tasks to design the **Client Request** node that starts your business process:

- Create Start Node
- Design Client Request Node

Create Start Node

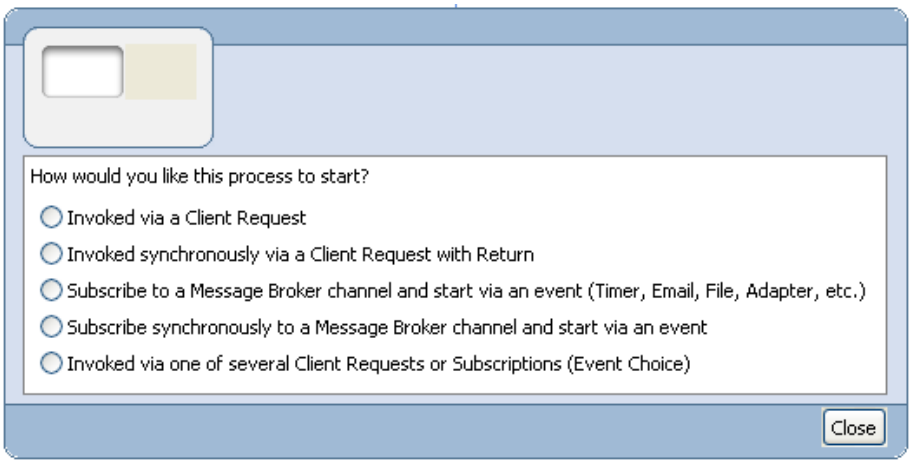
1. In the **Package Explorer** view, double-click **`RequestQuote.java`**. The **RequestQuote** business process is displayed in the **Design** view.

Figure 3-1 RequestQuote Process



2. Double-click the **Start** node to display the node builder, which displays the possible start methods.

Figure 3-2 Node Builder - Start Node



3. Select **Invoked via a Client Request**, and click **Close**.

   The empty node that was associated with the **Start** node changes to a **Client Request** node.

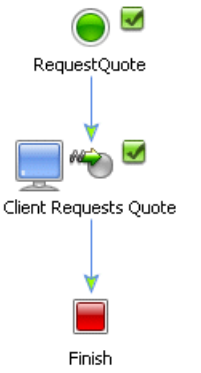
Design Client Request Node

Designing the **Client Request** node includes creating a method and parameters that the client uses to trigger the start of the business process, and designing the logic for handling receipt of requests from the client.

1. Rename the **Client Request** node.

   Select the **Client Request** node and press **F2**. Enter **`Client Requests Quote`** as the new name and press **Enter**. Your business process should now be as shown in the following figure:

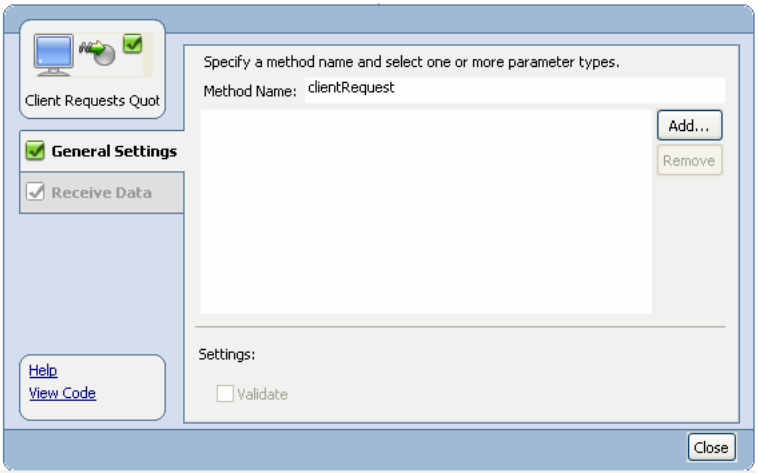
Figure 3-3 Rename Client Request Node



2. Double-click the **Client Requests Quote** node.

   The node builder is displayed.

Figure 3-4 Node Builder



Note: Node builders provide a task-driven user interface to help you design the communication between a business process and its clients and other resources. To access the node builder for any node, double-click the node in the **Design** view. A node builder that is specific for the type of node that you selected is displayed in-line in your business process.

The node builder for a **Client Request** node displays the two tabs: **General Settings** and **Receive Data**.

Specify General Settings

The following steps describe how to specify the method exposed by your business process. Clients invoke this method to start and make requests on your business process.

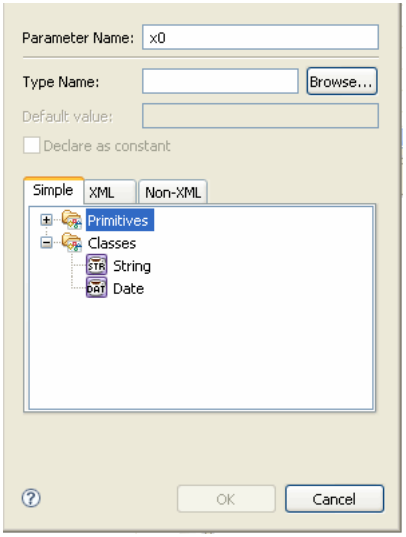
1. On the **General Settings** tab, in the **Method Name** field, change the default method name from **`clientRequest`** to **`quoteRequest`**.

   Note: When you make your business process available as a service, the name that you assign to a method on a **Client Request** node is the name of the method that is exposed through the Web Services Description Language (WSDL). It is recommended that you use a name that clearly indicates the service offered by the business process.

2. Specify a data type for the parameter of the **`quoteRequest`** method:

   a. Click **Add...** on the **General Settings** tab. A panel showing the data types is displayed.

Figure 3-5 Specify Data Type for Parameters



The RFQ is an XML message. So an **XML** data type is required at this node.

b. Select the **XML** tab.

The panel displays a list of XML schema files under **Typed** and a list of **Untyped** XML objects that are available in your project.

The XML schemas that you need for this tutorial are in the **`Schemas`** folder of the utility project. The schemas provided include **`QuoteRequest.xsd`**, **`PriceQuote.xsd`**, **`AvailQuote.xsd`**, **`Quote.xsd`**, and a system schema **`DynamicProperties.xsd`**.

Note: For XML schemas to be available to the services in your application, they must be located in **`Schemas`** folder under the web or utility project.

In this step, you use the **`QuoteRequest.xsd`** XML schema to specify the structure of documents that clients can send to trigger your business process.

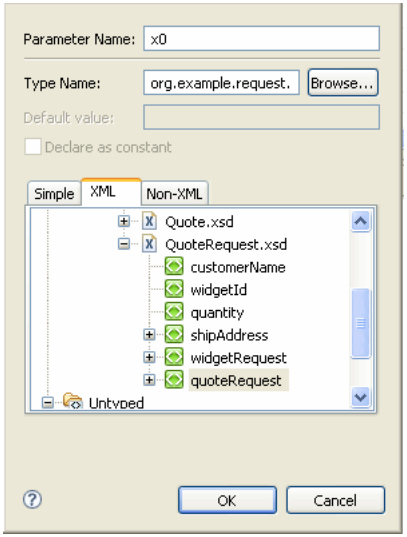
c. In the **XML** tab, progressively expand the nodes under **Typed** node, up to and including the **`QuoteRequest.xsd`** node.

A graphical representation of the **`QuoteRequest.xsd`** XML schema is displayed.

d. Select the **`quoteRequest`** node. It represents the parent element in your XML document.

The following value appears in the **Type Name** field:
`org.example.request.QuoteRequestDocument`.

Figure 3-6 Request Quote Parameter Type



e. In the **Parameter Name** field, enter **`requestXML`** in place of the default parameter name (**`x0`**).

f. Click **OK**.

The parameter type (**`QuoteRequestDocument`**) and name (**`requestXML`**) are displayed in the **General Settings** tab in the node builder.

This step completes the specification of the method exposed to clients by your business process. Messages from clients are expected to be typed XML; that is, the messages received from clients must contain XML data that is valid against an XML schema (in this case, **`QuoteRequest.xsd`**).

Note: Sample XML files (**`QuoteRequest.xml`** and **`QuoteRequest_a.xml`**) that are provided in the **`testxml`** folder are used later in the tutorial to test the process.

The **General Settings** tab is updated, as shown in the following figure, to indicate that you successfully completed the specification of a method name and parameters.

Figure 3-7 General Settings Tab - Updated

Note: indicates a completed task; indicates an incomplete task.

Specify Receive Data

1. Select the **Receive Data** tab, which lets you to specify a variable that receives an RFQ message from a client.

   The **Receive Data** tab has two modes:

   – **Variable Assignment**: You can use this mode to assign data received from a client to a variable of the same data type.

   – **Transformation:** You can use this mode to create a transformation between the data assigned to a variable and the data expected by the method parameter.

   By default, the **Receive Data** tab opens in the **Variable Assignment** panel.

   Note: It is also possible to assign typed non-XML (MFL) data directly to XML variables in the **Receive Data** tabs; no transformation is necessary. A discussion about non-XML (MFL) data is outside the scope of this tutorial. To learn about MFL files and the assignment of the data to business process variables, see Business Process Variables and Data Types in *Guide to Building Business Processes*.

   For this tutorial, we use the **Variable Assignment** mode because we want to assign the XML message received from the client directly to a variable of the same data type. In subsequent steps, you create a variable of typed XML (**`QuoteRequestDocument`**) to which your process assigns the incoming RFQs from clients.

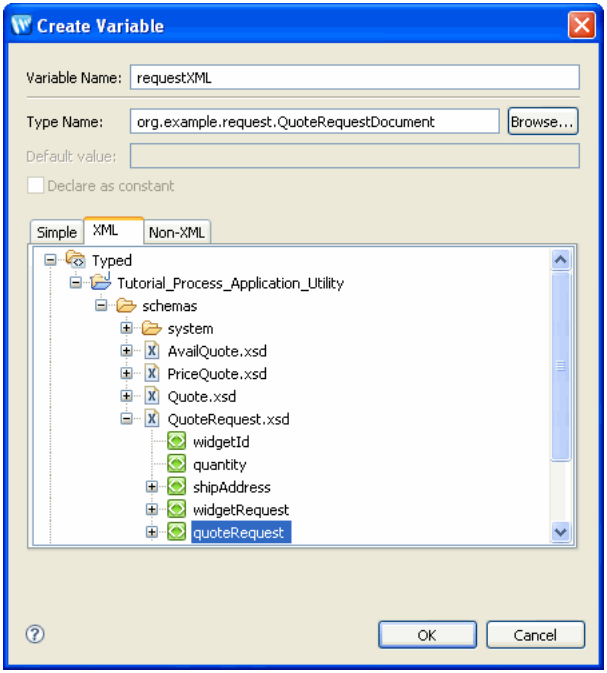
2. In the **Select variables to assign** drop-down list, select **Create new variable...**

   The **Create Variable** dialog box is displayed.

3. In the **Variable Name** field, enter **`requestXML`**.

4. In the **XML** tab, progressively expand the tree structure under the **Typed** node up to and including the **QuoteRequest.xsd** node; then, select the **quoteRequest** element as shown in the following figure.

Figure 3-8 Create Variable Dialog Box

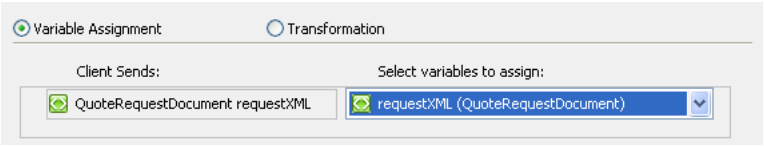


The following value appears in the **Type Name** field: **`org.example.request.QuoteRequestDocument`**.

5. Click **OK**.

   The new variable is created and displayed in the **Receive Data** tab.

Figure 3-9 Variable Assignment



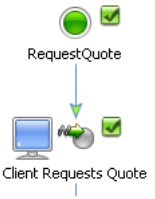
Note: The **`requestXML`** variable is shown as an **XML** variable in the **Data Palette** view.

Both tabs in the node builder (**General Settings** and **Receive Data**) are marked as complete .

6. Click **Close**.

In the **Design** view, note the completeness indicator associated with the **Client Requests Quote** node changes from ☑ to ☑, confirming that the design of the node is complete.

Figure 3-10 Completeness Indicator



7. From the WorkSpace Studio menu, choose **File > Save All**.

Related Topics

- Components of Your Application
- Designing Start Nodes
- Working With Data Types
- Interacting With Resources Using Controls

Step 3: Define Conditions for Alternate Paths of Execution

The business process is required to take a decision based on a value that the process extracts from the variable to which the XML message from the client is assigned. You design a condition, which must be evaluated at run time to determine whether the shipping address specified in the incoming RFQ XML file requires sales tax to be calculated for the quote.

- If the condition evaluates to true, then sales tax must be calculated and the flow of execution proceeds along a branch that invokes a web service to calculate the sales tax.
- If the condition evaluates to false, then no sales tax is required for the quote; the flow of execution proceeds along the default branch.

You design the decision logic by creating a **Decision** node in your business process. This step includes the following tasks:

- Add Decision Node
- Define Condition in Decision Node

Add Decision Node

1. If the **Node Palette** view is not visible, choose **Window > Show View > Node Palette**.
2. Select the **Decision** node ( ) from the **Node Palette** view, drag it to the **Design** view, and drop it just below the **Client Requests Quote** node.

   Note: As you drag the node to the **Design** view, targets ( ) appear, indicating possible positions at which you can insert the node. As you drag the node close to a target

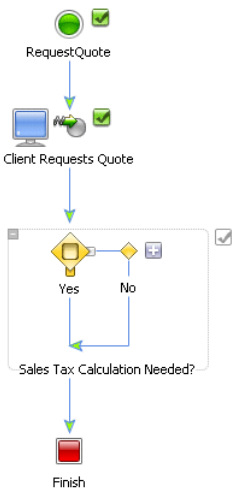
location, the target is activated ( ) and the mouse pointer changes to an arrow ( ). At this point, you can release the mouse button; the node snaps to the location indicated by the active target.

The **Decision** node includes a node for the condition (labeled **Condition**) and two paths of execution: one for actions to be executed if the condition evaluates to true and the other (**Default** path) for actions to be executed if the condition evaluates to false.

3. Change the names of the **Decision**, **Condition**, and **Default** nodes to represent the business tasks for tutorial more clearly:
   - Change the name of the **Decision** node to **Sales Tax Calculation Needed?**.
   - Change the name of the **Condition** node (true path) to **Yes**.
   - Change the name of the **Default** node (false path) to **No**.

   The business process is displayed in the **Design** view as shown in the following figure.

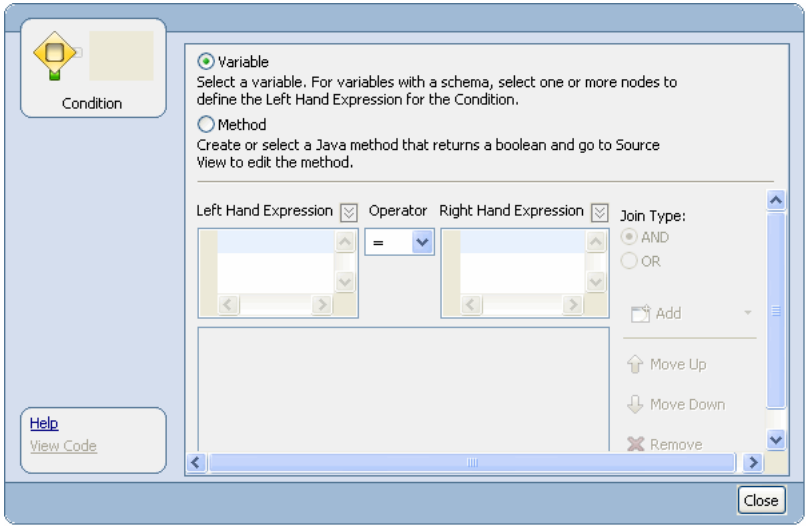
Figure 4-1 Business Process with Decision Node



Define Condition in Decision Node

1. Double-click the condition node ( ) to invoke the decision builder, which provides a task-driven user interface to help you design the decision logic.

Figure 4-2 Decision Builder



The **Variable** option button is selected by default. Do not change this selection because, in this case, you design the decision based on the value of an element in an XML document, which is valid with respect to an XML schema.

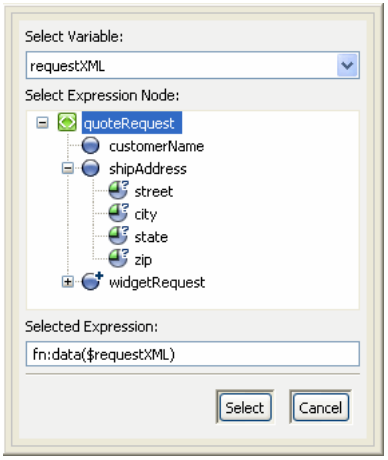
2. Select an XML element based on which the decision is to be made.

   a. In the decision builder, select a variable by clicking the icon adjacent to **Left Hand Expression**.

      A drop-down list of the variables in your project is displayed. In this case, the **`requestXML`** variable, which you created for the **Client Request** node at the start of your business process, is displayed.

      The **`quoteRequest`** XML schema is depicted in the **Select Expression Node** pane.

Figure 4-3 Selection Expression Node



The schema in our example (**`QuoteRequest.xsd`**) specifies the following elements:

- A root element: **`quoteRequest`**
- Child elements: **`customerName`** and **`shipAddress`**
- A repeating element: **`widgetRequest`**

The **`shipAddress`** element contains the attributes: **`street`**, **`city`**, **`state`** and **`zip`**.

b. Expand the **ShipAddress** element and select the **state** attribute.

With this step, you have selected the node in the XML document that represents the element based on which you want to define the condition logic.

The following expression appears in the **Selected Expression** field:

```
fn:data($requestXML/ns0:shipAddress/@state)
```

c. Click **Select**. The above expression appears in the **Left Hand Expression** field.

d. Select the **=** operator from the **Operator** list.

e. Enter **`"CA"`** in the **Right Hand Expression** field.

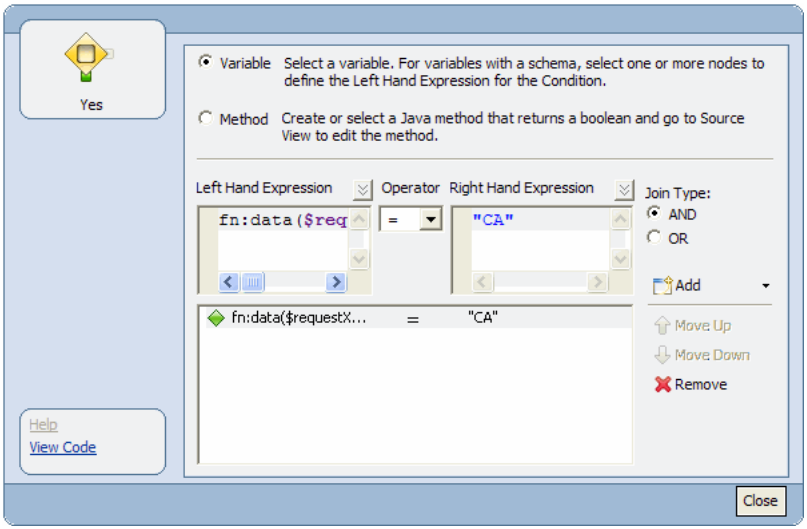
f. Click **Add** to add the condition that you just created:

```
fn:data($requestXML/ns0:shipAddress/@state) = "CA"
```

You have now finished designing the first condition for the decision node.

g. Select the expression in the condition list pane, as shown in the following figure:

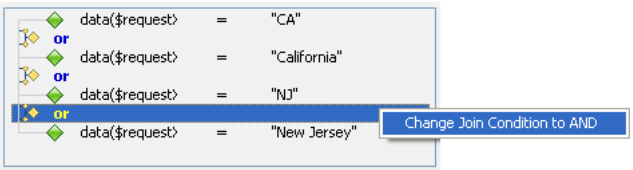
Figure 4-4 Condition List Pane



h. Change the **Join Type** to **OR**.

Note: You can change the join type for a condition even after you define the condition, by right-clicking on the join type as shown in the following figure.

Figure 4-5 Changing the Join Type



i. In the **Right Hand Expression** field, change **`CA`** to **`California`**.

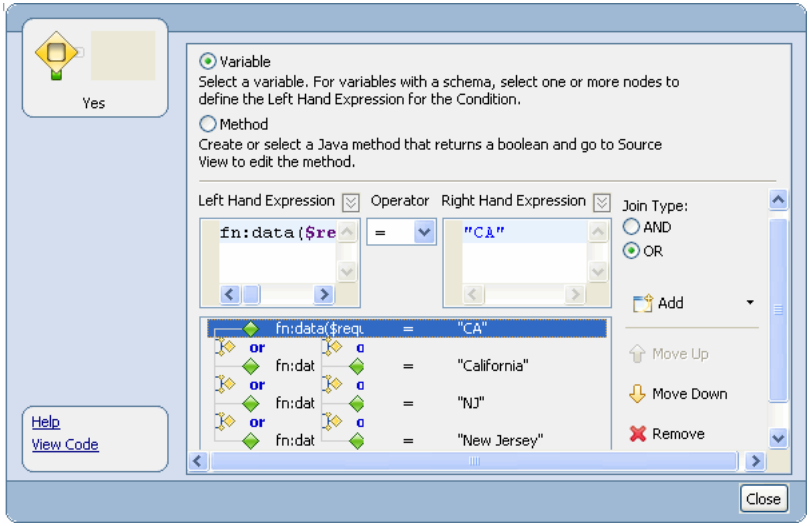
The **Add** button changes to **Update**.

j. Click the arrow adjacent to the **Update** button, and choose **Add** from the menu.

k. Similarly, add conditions for **`NJ`** and **`New Jersey`**.

The conditions that you defined are listed in the condition list pane, as shown in the following figure.

Figure 4-6 Condition List



3. Click **Close** to close the decision builder.

   The icon for the **Condition** node in the **Design** view changes from  to  , indicating that the condition defined for this node is based on the evaluation of XML data.

   This step completes the design of the condition that is evaluated when the flow transitions to the **Decision** node at run time.

   The condition logic is represented in the source code as an XQuery expression. As you define the conditions in the decision builder, BEA Workshop generates an XQuery expression.

   To view the XQuery expression, go to the **Source** view. The condition that you defined is represented by the following XQuery expression in the source code:

Listing 4-1 XQuery Expression

```
@com.bea.wli.common.XQuery(prolog =
"declare namespace ns0 = \"http://www.example.org/request\";" +
"declare function exprFunction0($requestXML) as xs:boolean {" +
```

```
"(((fn:data($requestXML/ns0:shipAddress/@state) = \"CA\") or
(fn:data($requestXML/ns0:shipAddress/@state) = \"California\")) or
(fn:data($requestXML/ns0:shipAddress/@state) = \"NJ\")) or
(fn:data($requestXML/ns0:shipAddress/@state) = \"New Jersey\")" +
"};")
```

You are now ready to define the actions on the subsequent paths in the flow.

Related Topic

Defining Conditions for Branching

Step 4: Invoke a Web Service

In the previous step, you designed a set of conditions for a decision node (**Sales Tax Calculation Needed?**). In this step, you learn how to create the activities to be performed when the condition defined in your decision node (**Sales Tax Calculation Needed?**) evaluates to true (that is, **`shipAddress/state`** in the XML document received from a client equals CA, California, NJ, or New Jersey). Specifically, you learn how to design your business process to invoke with a tax calculation web service through a web service control.

The **`TaxCalcControl.java`** web service control is created for you and included in the **`Tutorial_Process_Application_Web\requestquote\services`** folder.

Note: Java controls are server-side components managed by the Workshop framework. They encapsulate external resources and business logic for use in Workshop applications. They represent the interfaces between your business process and other resources. The underlying control implementation takes care of most of the details of the interaction for you. Controls expose Java interfaces that may be invoked directly from your business process. You can add an instance of a control to your project and then invoke its methods.

A complete description of how to create a web service and its associated control is beyond the scope of this tutorial. For more information about creating web services and creating controls from web services, see the following:

- *Tutorial: Web Services*
- "Controls and Transactions" in *Using Integration Controls*
- "Transaction Boundaries" in *Guide to Building Business Processes*

Invoking a web service control involves the following tasks:

- Create Instance of Web Service Control
- Call Web Service
- Receive Tax Rate from Web Service

Create Instance of Web Service Control

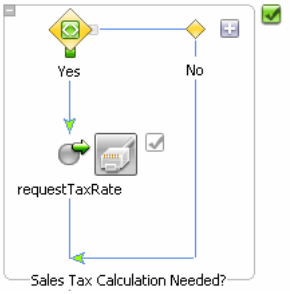
1. Select the **Design** view.
2. From the WorkSpace Studio menu, choose **Window > Show View > Data Palette**.
3. Click on the **Data Palette** tab. A drop-down menu of controls, representing the resources with which your business process can interact, is displayed.
4. Choose **Local Control > TaxCalcControl - requestQuote.services**.
5. Accept the default **Field Name**, and click **Finish**.

   Note: You can also create an instance of the control by dragging it from the **Package Explorer** view and dropping it in the **Data Palette** under the **Controls** node.

Call Web Service

1. In the **Data Palette**, expand the **taxCalcControl** node.
2. Drag the **`void requestTaxRate(String stateID_arg)`** method from the **Data Palette** view to the **Design** view, and drop it immediately below the **Yes** node of the **Sales Tax Calculation Needed?** node.

Figure 5-1 Control Send Node



A **Control Send** node is created; this node represents an asynchronous call to the tax calculation web service control. The name of the node is the same as the name of the method (**`requestTaxRate`**) that you selected from the **Data Palette** view.

Note: This interaction is designed to be asynchronous, meaning that the business process, after sending a request to the web service, does not wait for a response. It continues processing and receives a response when the web service completes the request.

3. Double-click the **requestTaxRate** node. The node builder opens on the **General Settings** tab. The control instance and target methods are already selected: **`taxCalcControl`** and **`void request TaxRate(String stateID_arg)`** respectively.

4. Select the **Send Data** tab.

   By default, the **Send Data** tab opens in the **Variable Assignment** pane. The **Control Expects** field indicates the data type (**`String stateID_arg`**) that is expected by the **`requestTaxRate()`** method exposed by the **`taxCalcControl`** web services.

   Note: As you learned in a previous step, **Send Data** tabs have two modes: **Variable Assignment** and **Transformation**.

   In this case, you must switch to the **Transformation** mode because the data type required as input for the **`taxCalcControl`** control is Java String, whereas the data type of the variable in which the RFQ message (including the value of **`shipAddress/state`**) is stored, is Typed XML (**`QuoteRequestDocument`**, valid against an XML schema).

   Note: WLI provides a tool called XQuery Mapper to transform data between heterogeneous data types. The data transformations that you create using XQuery Mapper are stored in **`xq`** files. You can think of a transformation file as another resource with which your business process interacts. The files containing the data transformations are built as controls. The controls expose methods, which business processes can invoke to transform disparate data types.

5. Select **Transformation**.

6. Click **Select Variable** to view the variables in your project, and choose **`requestXML`** (**`QuoteRequestDocument`**), which is the variable you created for the **Client Request** node at the start of your business process.

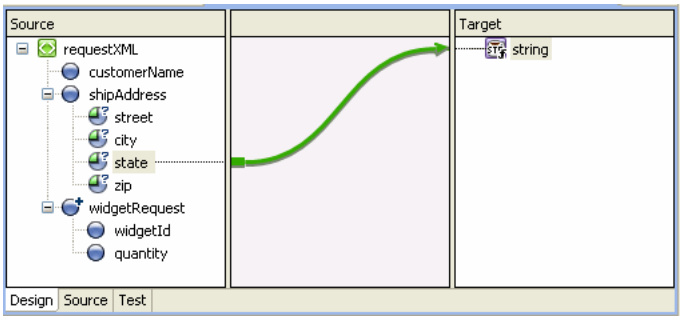
7. Click **Create Transformation**.

   An XQuery file is created and opened in the XQuery transformation perspective automatically.

   The design view of the XQuery file shows the elements of the **`QuoteRequestDocument`** XML document in the **Source** pane and a **`String`** element in the **Target** pane.

8. Select **`state`** in the **Source** pane and drag it to **`String`** in the **Target** pane. A connecting line appears between the **`state`** and **`String`** elements, as shown in the following figure. This line represents a transformation between the two data types.

Figure 5-2 Data Transformation



9. Save the **`xq`** file.

   An XQuery file (**`RequestQuotetaxCalcControlrequestTaxRate.xq`**) and a transformation control file (**`RequestQuoteTransformation.java`**) are created. Both files are displayed in the **Package Explorer** view. In addition, an instance of the transformation control is created and shown under **transformations** in the **Data Palette** (**Controls** folder).

10. Switch to the process by selecting the **RequestQuote.java** tab, and revert to the process perspective.

11. Click **Close** to close the **Request Tax Rate** node builder.

    You have now finished designing the **requestTaxRate** node.

Receive Tax Rate from Web Service

The interaction between the business process and the tax calculation control is asynchronous, which means that the business process can continue performing other work while the tax calculation service prepares its response. The tax calculation service notifies the business process when the response is ready.

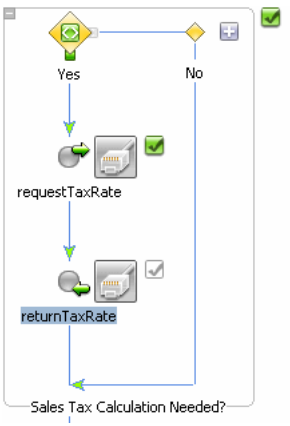
In the preceding section, you designed a call to the tax calculation web service (through a control).

To add the logic in your business process for receiving the tax rate returned by the tax calculation control, complete the following steps:

1. Switch to the **Process** perspective.
2. In the **Data Palette**, expand the **taxCalcControl** node to view the list of methods available for the control.
3. Drag the **`void returnTaxRate(float taxRate_arg)`** method from the **Data Palette** to the **Design** view and drop it in the **Sales Tax Calculation Needed?** node immediately below the **requestTaxRate** node.

   A **Control Receive** node is created representing an asynchronous response from the tax calculation web service control. The name of the node is the same as the name of the method (**`returnTaxRate`**) that you selected from the **Data Palette**.

Figure 5-3 Control Receive Node



4. Double-click the **returnTaxRate** node.

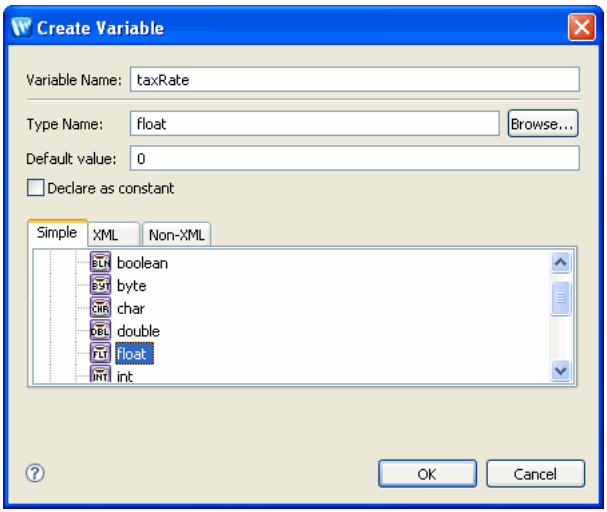
   The node builder opens in the **General Settings** tab. The control instance and target methods are already selected: **`taxCalculation`** and **`returnTaxRate(float taxRate_arg)`** respectively.
5. Select the **Receive Data** tab. The tab opens in the **Variable Assignment** mode.

   The **Control Returns** field indicates the data type and name of the parameter returned by the **`returnTaxRate()`** method of the **taxCalculation** control.
6. From the **Select variables to assign** drop-down list, select **Create new variable...**.

The **Create Variable** dialog box is displayed.

7. In the **Variable Name** field, enter **`taxRate`**.
8. On the **Simple** tab, expand **Primitives** and then select **float**.
9. In the **Default value** field, enter **`0`** (zero). This initializes the value of **`taxRate`** to zero.

Figure 5-4 Create Variable Dialog Box



10. Click **OK**.

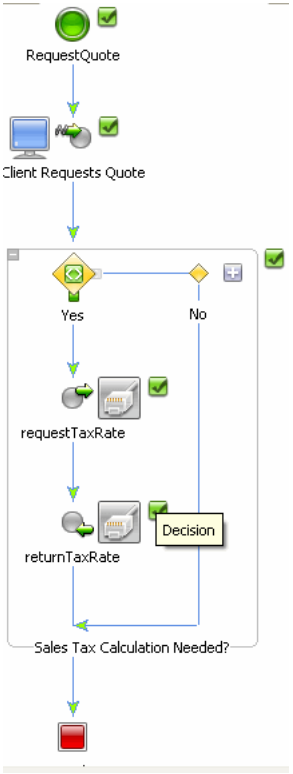
    The new variable to which the sales tax rate is assigned at run time is created and shown as a **Java** variable under **Variables** in the **Data Palette** view.

11. Click **Close** to close the node builder.

    You have now finished designing the **returnTaxRate** node and the activities to be performed by the business process when the condition in the **Decision** node evaluates to true.

    In the **Design** view, the business process appears as shown in the following figure.

Figure 5-5 Graphical View of Business Process



Note: The start node icon changed from (indicating a stateless business process) to (stateful) after you added the asynchronous call to the web service control.

To see whether the process is stateless or stateful, select the start node, and look for the **`stateless`** property in the **JPD Configuration** view.

For more information about stateful and stateless business processes, see "Building Stateless and Stateful Business Processes" in *Guide to Building Business Processes*.

To understand why the property of the process changed from stateless to stateful, see "Transaction Boundaries".

12. From the WorkSpace Studio menu, choose **File > Save All**.

No further design is required for this decision node.

- If the condition evaluates to true, the **Yes** path is executed and the tax rate for the RFQ is calculated.
- If the condition evaluates to false, the **No** path is executed and a value of zero is assigned to **`taxRate`** because the variable was initialized to zero when it was created.

Related Topics

- *Using Integration Controls*
- *Transforming Data Using the XQuery Mapper*

Step 5: Run the Business Process

To run and test the business process that you created, complete the following steps:

1. If WebLogic Server (WLS) is not already running, choose **Window > Show View > Other > Server > Servers** from the WorkSpace Studio menu, and click **OK**.

   The **Server** view is displayed. If no server is defined, the view is empty.

2. In the **Package Explorer** view, right-click on **`RequestQuote.java`**, and choose **Run As > Run On Server**.

   The **Run On Server** wizard is displayed.

3. Select **Manually define a new server** (if there is no server defined) and click **Next**.

   Note: If one or more servers are already defined, you can select **Choose an existing server**.

   The **BEA WebLogic Server v10.2** dialog box is displayed.

4. Click **Browse** adjacent to the **Domain home** field, and select the samples integration domain directory from **`BEA_HOME\wlserver_10.0\samples\domains\integration`**, where **`BEA_HOME`** represents the directory in which you installed WLI.

5. Click **Finish**.

   The server is started, and the RequestQuote application is deployed on it. The status of the server changes to **Started** in the **Servers** view.

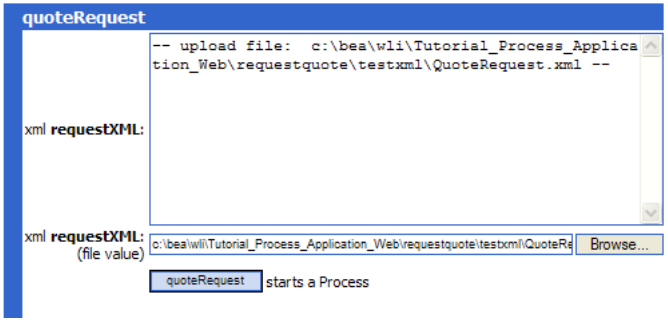
   After the application is deployed, the test browser is displayed.

6. Select the **Test Form** tab.

7. Click the **Browse...** button adjacent to the **xml requestXML (file value)** field.
8. Select the **`Tutorial_Process_Application_Web\src\testxml\QuoteRequest.xml`** file.
9. Click the **quoteRequest** button to start the business process.

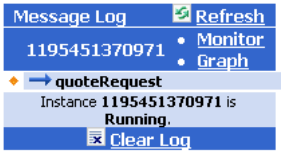
   Note: The label of the button reflects the name of the start method in the business process.

Figure 6-1 Test Form



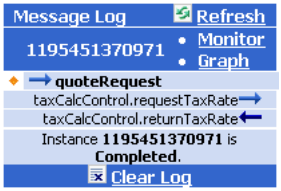
The **Test Form** tab refreshes to display a summary of your request parameters and the responses from the web service in the **Message Log**.

Figure 6-2 Message Log - Initial



10. Click **Refresh** to refresh the entries in the log until this instance of the business process completes running.

Figure 6-3 Message Log - Refreshed



11. Entries displayed in the message log correspond to the methods in the business process:
    - The **`quoteRequest`** method that starts the business process.
    - A call from your business process to the **`taxCalculation`** web service: **`taxCalcControl.requestTaxRate`**.
    - A response from the **`taxCalculation`** web service to your business process: **`taxCalcControl.returnTaxRate`**.
    - **Instance ID**: When the business process finishes, a message similar to the following is displayed in the message log:

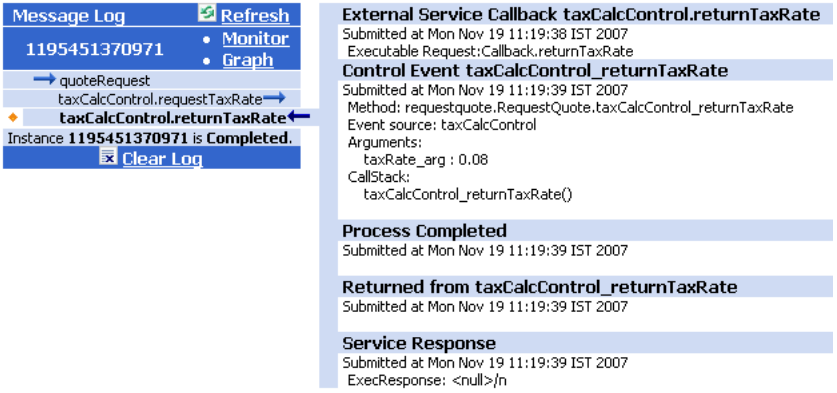
      `Instance instanceID is Completed`.

      `instanceID` is the ID that is generated when the **`quoteRequest`** method in your business process was called.

    You can click any of the methods in the message log to view the details of the call. For example, if you click **`quoteRequest`**, the **Service Request** panel displays the XML message sent by the client (you) when the method was called.

    If you click **`taxCalcControl.returnTaxRate`**, you can view the response from the **`taxCalculation`** service.

Figure 6-4 Message Log with Details



In the sample XML message that you used, **`state`** is **`NJ`**. So the process executes the **Yes** branch of the **Sales Tax Calculation Needed?** node.

The following code segment shows the rate of sales tax returned for this test XML message.

```
<returnTaxRate xmlns="http://www.openuri.org/">
<taxRate>0.08</taxRate>
</returnTaxRate>
```

By following these steps, you ran and tested a simple business process, which contains a **Start** node and a **Decision** node, and includes an asynchronous call to a web service through a control.

Subsequent steps in this tutorial build on the business process that you have created so far.

Note: For information about the other features of the test browser, see "Additional Functionality in Test Browser" on page 14-5.

CHAPTER 7

Step 6: Invoke a Business Process

Process controls are used to send requests to and receive responses from other business processes in the same domain using Java/RMI. This scenario demonstrates a typical use case for a process control – to call a subprocess from a parent business process.

In this part of the tutorial, you change the design of the business process you created in the previous part, to take advantage of a tax calculation service provided by a business process instead of using the tax calculation web service you used earlier. You can do this by first creating a process control from the tax calculation business process. Then, you simply change the control nodes in such a way that instead of communicating with the tax calculation **web service** through the web service control, the nodes communicate with the tax calculation **business process** through the new process control.

The tasks in this step include:

- Create Process Control for Tax Calculation Process
- Change Control Send Node to Interact with Process Control
- Change Control Receive Node to Interact with Process Control
- Test Request Quote Process

Create Process Control for Tax Calculation Process

1. Open **`RequestQuote.java`** in the **Package Explorer** view.
2. If the **Data Palette** view is not visible, choose **Window > Show View > Data Palette**.

   Instances of controls that are already available in your project are displayed in the **Controls** folder of the **Data Palette** view.
3. Select the **Controls** folder, and click to display a drop-down list of controls that represent the resources with which your business process can interact.
4. Choose **Integration Controls > Process**.

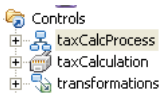
   The **Insert Control: Process** dialog box is displayed.
5. In the **Field Name** field, enter **`taxCalcProcess`** as the name for the instance of the process control that you creating, and click **Next**.
6. In the **Create Control** dialog box, enter **`TaxCalculationProcess`** in the **Name** field, and click **Next**.
7. In the **Insert Control - Process** dialog box, click **Browse...** and select **`/Tutorial_Process_Application_Web/src/requestquote.services/TaxCalcProcess.java`**.

   This Java file is a simple business process that calculates the sales tax for an RFQ.

   The start method for **`TaxCalcProcess.java`** (**requestTaxRate**) is displayed in the **Start Method** field.
8. Click **Finish**.

   The process control (**`TaxCalcControl.java`**) is created and displayed in the **Package Explorer** view. An instance of the control (**`taxCalcProcess`**) is added to the **Data Palette**. The **Controls** area in the **Data Palette** view now resembles the following figure:

Figure 7-1 Controls in Data Palette



Change Control Send Node to Interact with Process Control

1. In the **Data Palette**, click + beside **`taxCalcProcess`** under the **Controls** folder to expand the list of methods on the control.
2. Drag the `void requestTaxRate (QuoteRequestDocument quoteRequest)` method from the **Data Palette** to the **Design** view and drop it on the **requestTaxRate** node in the **`RequestQuote.java`** process.

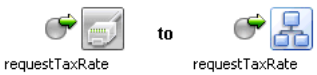
   The following message is displayed:

   ```
   This Control node is already associated with a control method. Do you
   wish to replace this control method?
   ```

3. Click **OK**.

   The **requestTaxRate** node changes to reflect the type of control with which it is now associated. The node changes as follows:

Figure 7-2 Change in Node After Associating Control



4. Double-click the **requestTaxRate** node to open its node builder.
5. In the **General Settings** tab, confirm that **`taxCalcProcess`** is selected in the **Control** field and that the following method is selected in the **Method** field:

   ```
   void requestTaxRate(QuoteRequestDocument quoteRequest)
   ```

6. Select the **Send Data** tab.

   By default, the **Variable Assignment** option is selected, and the **Control Expects** field contains **QuoteRequestDocument quoteRequest**, indicating the format and type of the message that the tax calculation process requires.

   Note: The tax calculation process requires a message of XML type **`QuoteRequestDocument`** – the same type as the **`requestXML`** variable to which the XML message sent from a client to the **`RequestQuote.java`** process is assigned. Unlike the scenario for sending data to the tax calculation web service in Chapter , "Step 4: Invoke a Web Service," no transformation is required on this node; you can create a direct variable assignment.

7. In the **Select variables to assign** field, and select **requestXML(QuoteRequest)**.
8. Click **Close**.

   This step completes the procedure to change your business process to call the tax calculation **business process** instead of the tax calculation **web service**.

Change Control Receive Node to Interact with Process Control

1. In the **Data Palette**, click + beside **`taxCalcProcess`** under the **Controls** folder to expand the list of methods on the control.
2. Drag the `void returnTaxRate(float salesTaxRate)` method from the **Data Palette** to the **Design** view and drop it on the **returnTaxRate** node in the **`RequestQuote.java`** process.

   The following message is displayed:

   `The Control node is already associated with a control method. Do you wish to replace this control method?`
3. Click **OK**.

   The **returnTaxRate** node changes to reflect the type of control with which it is now associated.
4. Double-click the **returnTaxRate** node.
5. In the **General Settings** tab, confirm that **taxCalcProcess** is selected in the **Control** field and that the following method is selected in the **Method** field:

   `void returnTaxRate(float salesTaxRate)`
6. Select the **Receive Data** tab.

   The **Variable Assignment** option is selected by default, and the **Control Returns** field contains **float salesTaxRate**, indicating the type and name of the parameter expected to be returned by the tax calculation process.
7. In the **Select variables to assign** field, and select **`taxRate (float)`**.
8. Click **Close**.

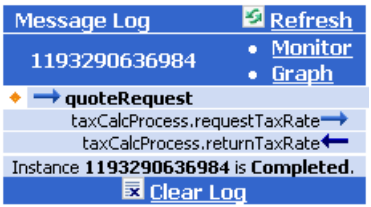
   This step completes the procedure to change the callback handler to receive a message from the tax calculation **business process** instead of from the tax calculation **web service**.
9. From the WorkSpace Studio menu, choose **File** > **Save All**.

Test Request Quote Process

You can run and test the business process, which now contains an asynchronous call to another business process in the same way that you tested the business process in the previous part. Perform steps 1 through 7 of Step 5: Run the Business Process.

When you start the operations in the **Test Form** page, the **Message Log** refreshes to display a summary of the calls to, and responses from, the tax calculation business process.

Figure 7-3 Message Log



Entries in the message log correspond to the methods in your business process:

- The **`quoteRequest`** method that starts the business process.
- A call from the **RequestQuote** business process to the **taxCalcProcess** business process: **`taxCalcProcess.requestTaxRate`**. Note that, in this case, the entire XML document (contained in the **`requestXML`** variable) is passed to the subprocess. This is different from when the business process called the tax calculation web service; in that case, only the **`state`** field was passed to the web service.
- A response from the **taxCalcProcess** business process to your **RequestQuote** business process: **`taxCalcProcess.returnTaxRate`**. Note that instead of the tax rate being returned in a web services SOAP envelope, as it was in the return from the web service, the process control returns the raw float value (**`0.08`**).
- The **Instance ID** represents the ID that was generated when the **quoteRequest** method in the business process was called.

Part II Add Complex Business Logic

In this part of the tutorial, you add more complex business logic to the business process that you designed and tested earlier. This part of the tutorial consists of the following steps:

- Chapter , “Step 7: Loop Through Items in a List”

  You extract a list of items from the RFQ document received from a client and perform a set of activities once for each item in the list.

- Chapter , “Step 8: Design Parallel Paths of Execution”

  You design your business process to execute tasks in parallel. This step also includes instructions for designing your business process to interact with resources through controls and transform the data exchanged with those controls, as required.

- Chapter , “Step 9: Create Quote Document”

  You learn how to transform the price and availability data from untyped XML data to typed XML, and then combine the price and availability data (that is returned to the business process from several external services) in a single quote document.

- Chapter , “Step 10: Write Quote to File System”

  You learn how to write business process data to a log by using a file control.

- Chapter , “Step 11: Send Quote From Business Process to Client”

- Chapter , “Step 12: Run RequestQuote Business Process”

CHAPTER 8

Step 7: Loop Through Items in a List

In this step of the tutorial, you create the logic to extract a list of items from the RFQ received from a client, and begin designing the business process to determine the price and availability of the items requested by the client.

A **For Each** node represents a point in a business process where a set of activities is performed repeatedly, once for each item in a list. A **For Each** node includes the following:

- An iterator node in which a list of items is specified
- A loop in which the activities to be performed for each item in the list are defined

An iteration variable holds the current element being processed in the **For Each** loop, for the life of the loop.

The following section gives you a brief overview of XML schemas. Read this section before you proceed with designing the **For Each** node in your business process.

Overview of XML Schemas

The business process that you create in this tutorial is designed to start when it receives an RFQ (in the form of an XML file) from a client. The RFQ must contain valid XML, that is, XML data valid against an XML schema (**`QuoteRequest.xsd`** in this case).

The **`QuoteRequest.xsd`** schema is available in your application at the following location: **`Tutorial_Process_Application_Utility\Schemas`**.

Note: To make the schemas in your project available in the business process, you must place them in a **`Schemas`** folder in **`Tutorial_Process_Application_Utility`** (utility project).

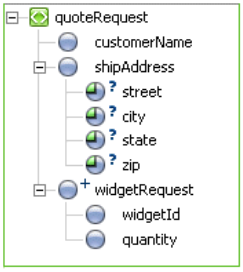
For more information, see "Creating and Importing Schema Files" in *Transforming Data Using XQuery*.

XML schemas in your application's **`Schemas`** folder are compiled to generate XML Beans. In this way, BEA Workshop generates a set of interfaces that represent different aspects of your XML schemas. XML Bean types correspond to types in the XML schema itself. XML Beans provide Java counterparts for all built-in schema types, and generate Java counterparts for any derived types in your schema.

In Step 2: Specify How the Process Is Started, you created a variable (**`requestXML`**) to which the RFQ document (that your business process receives from a client) is assigned. When you work with such variables in the **Design** view, you work with a graphical representation of the XML schema that is associated with the variable.

The following figure is a graphical representation of the **`quoteRequest`** element in the **`QuoteRequest.xsd`** schema against which the RFQ document from clients is valid:

Figure 8-1 Graphical Representation of XML Schema



Note the following characteristics of the **`QuoteRequest.xsd`** schema:

- The elements and attributes of the XML schema are represented as nodes.
- The root element is **`quoteRequest`**. It specifies the following child elements: **`customerName`**, **`shipAddress`**, and **`widgetRequest`**.
- The **`shipAddress`** element specifies the following attributes: **`street`**, **`city`**, **`state`**, and **`zip`**.

- The **`widgetRequest`** element is a repeating element (represented graphically by ). In other words, there can be one or more occurrences of the **`widgetRequest`** element in an associated XML document. The **`widgetRequest`** element, in turn, contains two elements: **`widgetId`** and **`quantity`**.

The business process in this scenario dictates that each pair of **`widgetId`** and **`quantity`** elements received in the RFQ documents from clients is processed. This processing begins with a **For Each** node. In each iteration through the **For Each** loop, one **`widgetRequest`** item must be extracted from the **`requestXML`** variable, and an activity (or a set of activities) must be performed on that item.

Design “For Each” Loop

Creating the logic for your business process to iterate over the sequence of items in the RFQ document involves the following tasks:

- Add “For Each” Node
- Select Repeating XML Element to be Iterated

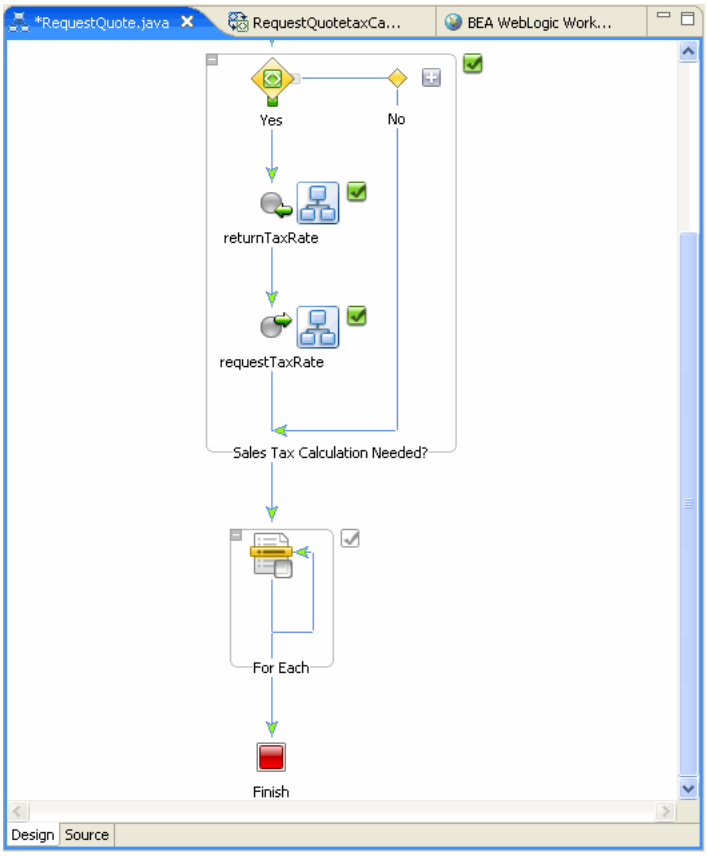
Add “For Each” Node

1. In the **Design** view, drag the **For Each** node ( ) from the **Node Palette** to the RequestQuote business process and drop it immediately after the **Sales Tax Calculation Needed?** (**Decision**) node.

2. Press **Enter** to accept the default name: **For Each**.

   The business process now has a **For Each** node, as shown in the following figure.

Figure 8-2 “For Each” Node



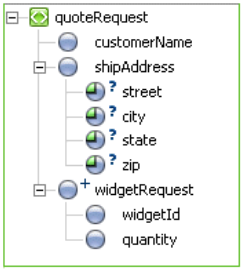
Select Repeating XML Element to be Iterated

1. Double-click the **For Each** node.
2. In the node builder, click **Select Variable**.

   A list of the variables (typed XML) in your project is displayed.
3. Select **requestXML (QuoteRequestDocument)**.

   The **`requestXML`** variable contains the repeating XML element for which you want to design the iteration logic. A representation of the XML schema in the **`requestXML`** variable is displayed in the **Select Node** pane.

Figure 8-3 XML Schema Displayed in “Select Node” Pane



4. In the **Select Node** pane, select **widgetRequest**.

   The **Repeating element** and **Iteration Variable** fields are filled with the following data:

   – **Repeating element**: Contains the following XPath expression, which when applied on the incoming XML document, returns the set of repeating XML elements:

   ```
   $requestXML/ns0:widgetRequest
   ```

   – **Iteration Variable**: Contains the name of an iteration variable: **`iter_forEach1`**.

   At run time, the current element being processed in the **For Each** loop is assigned to the iteration variable.
5. Click **Close** to close the node builder.

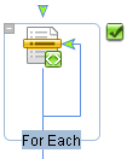
   The iteration variable, **`iter_forEach1`**, is created and added to the list of variables in the **Data Palette**. It is of XML type **WidgetRequestDocument**.**WidgetRequest**.

   For information about how the iteration variable is used in the **For Each** loop, see “Design “Create PriceList” Node” on page 10-9.

This step completes the design of the iteration logic for your **For Each** node.

The node is updated in the **Design** view to reflect the work you did to define the condition.

Figure 8-4 Updated "For Each" Node



The  icon indicates that an XML query is defined for the node.

After you create the iteration logic in the **For Each** node, you must define the activity or set of activities to be performed during each iteration.

You add activities to the **For Each** loop by creating nodes within it to support your business logic. In Step 8: Design Parallel Paths of Execution, you create a **Parallel** node and design it so that the business process executes two sets of activities in parallel: request for price, and determination of availability for the items requested by the client.

Related Topics

- Business Process Variables and Data Types
- Looping Through Items in a List
- Grouping Nodes in Your Business Process

Step 8: Design Parallel Paths of Execution

In the previous step, you created a **For Each** loop to iterate through a set of repeating elements in an RFQ document. In this step, you design the activities to be performed within each iteration of the **For Each** loop.

When your business process interacts with multiple systems, as is the case for price and availability processing in this scenario, you can use **Parallel** nodes to create two or more parallel branches of execution.

Note: Parallel branches of execution in a business process are logically parallel; physically, the branches are executed serially. Business processes benefit from this logical parallelism when communication with external systems involves waiting for responses from those external systems. While one branch of execution is waiting for a response, another branch of execution in the parallel flow can progress.

For more information, see Creating Parallel Paths of Execution in *Guide to Building Business Processes*.

In our example scenario, the business process must determine both price and availability information so that a quote can be prepared and returned to the client. This business process can benefit from parallelism because it communicates with two external systems: one each for price calculation and availability calculation. The business process expects a response from each of the external systems.

The external systems can be any resource – other business processes, web services, EJBs, databases, file systems, and so on – that returns the information your business process requires. Business processes interact with the resources through controls.

This tutorial uses two web services: one returns the price for each **`widgetID`** specified in the client's RFQ document; a second service returns availability information based on the **`widgetID`** and the **`quantity`** specified in the request document.

The controls with which the business process interacts are provided in the **`\Tutorial_Process_ApplicationWeb\requestquote.services`** folder. The controls are **`PriceProcessor.java`** and **`AvailProcessor.java`**.

Create Parallel Node

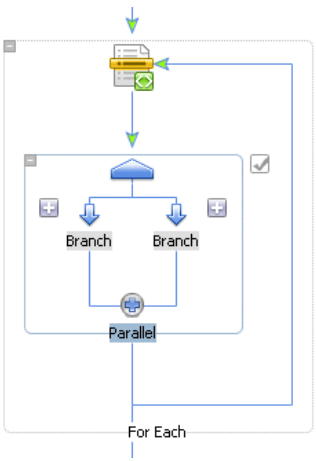
1. Make sure that your business process (**`RequestQuote.java`**) is displayed in the **Design** View.

2. Drag the **Parallel** node ( ) from the **Node Palette** to the **Design** view, and drag it inside the **For Each** loop.

3. Press **Enter** to retain the default name: **Parallel**.

   The **Design** view is updated as shown in the following figure:

Figure 9-1 Updated Design View



4. Change the names of the branches within the **Parallel** node to identify the activities that your business process executes in parallel:
   - Change the name of the left branch to **Get Price**.
   - Change the name of the right branch to **Get Availability**.

By default, **Parallel** nodes specify an AND join condition, represented by the  icon at the intersection of the parallel branch lines, indicating that the activities on all the branches must be completed for execution to proceed to the next node after the parallel node.

For this business process, since both price calculation and availability checking must be completed before proceeding to the next node, leave the AND join condition as is.

Note: If an OR join condition is specified, when the activities on one branch are completed, execution of activities on all the other branches terminates, and the flow of execution proceeds to the node following the parallel node. The OR join condition is represented by the icon in the **Design** view.

In the **Design** view, you can view and edit the **join condition** property in the **JPD Configuration** view.

Create Logic to Assemble Price and Availability Data

In this section, you learn how to do the following:

- Invoke the price and availability services (through controls) from the parallel branches you created.
- Design callbacks on your branches to wait for and handle responses from the controls.
- Construct an XML document, to which response data from the controls is appended for each iteration through the **For Each** loop.

  Note: Review your business process in **Design View**: your Parallel node is within the **For Each** loop, meaning that the flow of execution is through the **Parallel** node for each iteration through the loop.

To design the **Parallel** node to interact with the price and availability web services, complete the following tasks:

- Create Instances of PriceProcessor and AvailProcessor Controls
- Add Control Nodes in Business Process
- Design Activities on Get Price Branch
- Design Activities on Get Availability Branch

Create Instances of PriceProcessor and AvailProcessor Controls

The **`PriceProcessor.java`** and **`AvailProcessor.java`** controls are available in the **`Tutorial_Process_Application_Web\src\requestquote.services`** folder in the **Package Explorer** view.

The goal of this section is to create the appropriate controls in your application and then design the communication between the business process and the controls.

1. Click  in the **Data Palette.**

   A list of controls, which represent the resources with which your business process can interact, is displayed.

2. Choose **Local Controls** > **priceProcessorControl - requestquote.Services**.

   The **Insert Control** dialog box is displayed.

3. Enter **`priceProcessor`** in the **Field Name** field and click **Finish**.

4. Similarly, create an instance of **`availProcessor`**.

   The **`priceProcessor`** and **`availProcessor`** web service control instances are added to the **Data Palette**:

Add Control Nodes in Business Process

You learned earlier in the tutorial that you can create control nodes in your business process by dragging the methods for the appropriate control from the **Data Palette** to the business process in the **Design** view.

You can also create control nodes by dragging **Control Send**, **Control Receive**, or **Control Send with Return** from the **Node Palette** to the business process. You subsequently bind the appropriate methods to the control node you created. This is the approach that you use in this section.

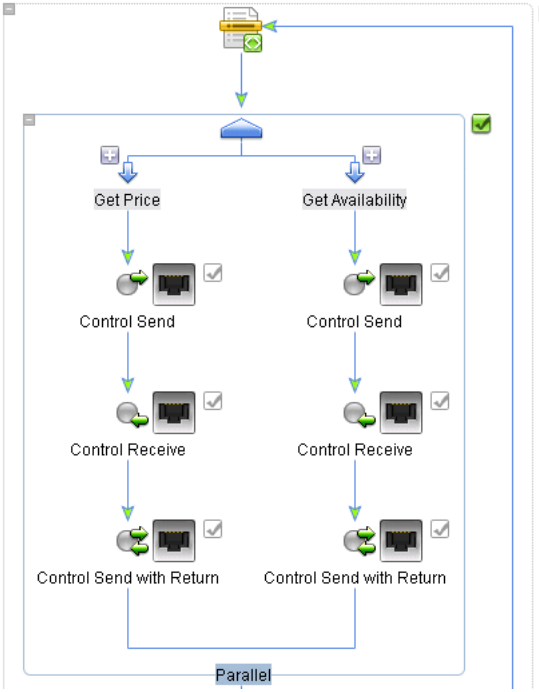
Add the following nodes from the **Node Palette** to each branch of the parallel node:

- **Control Send**
- **Control Receive**
- **Control Send with Return**

Drag each of the listed nodes from the **Node Palette** to the **Design** view, and drop the node on parallel branches.

The resulting business process is as shown in the following figure.

Figure 9-2 Parallel Node with Controls on Each Branch



In this way, each branch is designed for the following flow of execution:

1. Call a resource (through a control) from the **Control Send** node.
2. Wait for a response from the control at the **Control Receive** node.
3. Make a synchronous call to a control at the **Control Send with Return** node. At this node, you call a transformation that constructs an XML document. The response data from controls is appended to this XML document for each iteration through the **For Each** loop.

Design Activities on Get Price Branch

1. Rename the three control nodes on the **Get Price** Branch (in the order in which they are executed) to **Request Price**, **Receive Price**, and **Create PriceList** respectively.
2. Complete the following tasks:
   - Design "Request Price" Node
   - Design "Receive Price" Node
   - Design "Create PriceList" Node

Design "Request Price" Node

1. Double-click the **RequestPrice** node.

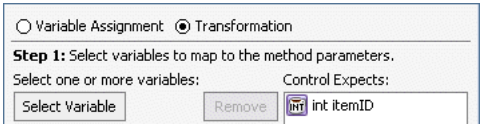
   The node builder opens on the **General Settings** tab.
2. In the **Control** field, select **priceProcessor**.

   The **Method** panel shows a list of the asynchronous send methods that you can invoke on the **priceProcessor**.
3. Select the **`void getPrice(int itemID_arg)`** method.
4. Select the **Send Data** tab.

   By default, the **Send Data** tab opens on the **Variable Assignment** pane. The **Control Expects** field contains the data type (**`int itemID`**) that is expected by the **`getPrice()`** method exposed by the **priceProcessor** web service.

   Note: The **priceProcessor** web service takes the ID of the item requested as the input and returns the price of the item.
5. Switch to the **Transformation** mode in the **Send Data** tab.

Figure 9-3 Transformation Mode



Note: This step is necessary because the data type required as input for **priceProcessor** is **`int`**, whereas the **`iter_forEach1`** variable, which holds the value of **`widgetId`** in

the **For Each** loop, is of type XML (**`WidgetRequestDocument`** is valid against an XML schema).

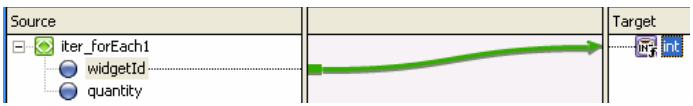
6. Click **Select Variable** and choose **iter_forEach1 (widgetRequest)**.
7. Click **Create Transformation**.

   The transformation tool opens and displays a representation of the **`iter_forEach1 (widgetRequest)`** variable in the **Source** pane, and **`int`** in the **Target** pane.

8. Click **widgetID** in the **Source** pane and drag the mouse pointer to **int** in the **Target** pane.

   A connecting line is drawn between the **widgetID** and **int** elements. The line represents the transformation between the two data types.

Figure 9-4 Data Transformation



As you draw the mapping line, the following warning is displayed:

```
The datatype of the source node: [widgetId] and target node: [int] do not
match, a type conversion will be applied.
```

Note: This transformation creates a new method under **`RequestQuoteTransformation.java`**, which was created in your project and prebuilt in the tutorial application. It is available in the **`Tutorial_Process_Application_Web\requestquote`** folder. A new XQ file called **`RequestQuotepriceProcessorGetPrice.xq`**, which contains the query for this transformation method, is also created.

9. In the **Navigation** pane, click **RequestQuote.java** to return to your process.
10. Click **Close** to close the **Request Price** node builder.

    This step completes the design of the **Request Price** node.

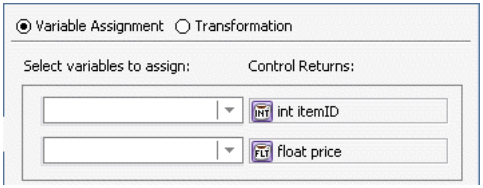
Design "Receive Price" Node

1. Double-click the **Receive Price** node to open its node builder. The node builder opens on the **General Settings** tab.
2. In the **Control** field, select **priceProcessor**.
3. The **Method** panel contains a list of the asynchronous receive methods for **priceProcessor**.

Select **`void returnPrice(int itemID_arg, float price_arg)`**.

4. Click **Receive Data**.

   The **Control Returns** field shows the data types returned by the **`returnPrice(int itemID, float price)`** method on the **`priceProcessor`** web service.

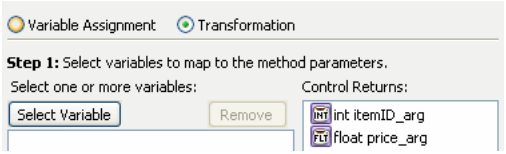


   The **`priceProcessor`** web service takes **`itemID`** (an **`int`** value) as input and returns an **`int`** value for the item ID and a **`float`** value for the price.

   Next, you must switch from the **Variable Assignment** mode to the **Transformation** mode because the data returned by the **`priceProcessor`** web service must be assigned to a typed XML variable. To do this, your process must transform the Java data types returned from the **`priceProcessor`** web service to typed XML.

5. Select **Transformation**.

   The **Receive Data** tab is displayed as shown in the following figure.



6. Click **Select Variable**, then **Create new variable...**.

   The **Create Variable** dialog box is displayed.

7. In the **Variable Name** field, enter **price**.

8. Select the **XML** tab.

9. Expand the **Typed** node (by clicking the + symbol adjacent to it), and progressively expand the nodes up to **schemas**.

10. Under the **schemas** node, expand **PriceQuote.xsd**, and select **priceRequest**.

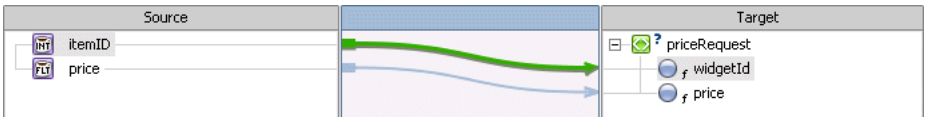
    The **Type Name** field shows **`org.example.price.PriceRequestDocument`**.

11. Click **OK**.

12. The **Create Variable** dialog box closes and the new variable is displayed in the **Receive Data** tab. It is also listed as an **XML Type** variable in the **Data Palette**.

13. On the **Receive Data** tab, click **Create Transformation**. The transformation tool opens and displays a representation of the **`itemID`** and **`price`** variables in the **Source** pane, and the **`price`** variable in the **Target** pane.

14. Map **itemID_arg** to **widgetId** and **price_arg** to **price**, as shown in the following figure.



Note: This transformation creates a new method under **`RequestQuoteTransformation.java`** in the **`Tutorial_Process_Application_Web\requestquote`** folder. A new XQ file, which contains the query for this transformation method, is also created.

15. Double-click **RequestQuote.java** in the **Package Explorer** to return to your business process.

16. Click **Close** in the **Receive Price** node builder.

    This step completes the design of the **Receive Price** node.

Design "Create PriceList" Node

In this step, you use a transformation control (**PriceAvailTransformations**) provided in your project to append the price data returned from the **priceProcessor** control (on each iteration through the **For Each** loop) to a single variable.

Previously, when you designed nodes in the business process, you created transformation methods on a transformation, as necessary, to map the data your business process sent to or received from clients and controls. In this case, you also use a transformation, but in a different way. In the case of the **Create PriceList** node, the data is not sent to a client or control. Instead, the transformation takes typed XML data as input from your business process, and returns untyped XML (**`XmlObject`**) data. The business process must append the data returned on every iteration of the **For Each** loop to a single variable, thus creating a repeating sequence of XML data. A variable that can hold this type of repeating sequence of XML data in a **For Each** loops is of type **`XmlObjectList`**. Both typed and **`XmlObject`** variables can be appended to variables of type **`XmlObjectList`**.

For more information, see "XmlObjectList Data Type" on page 10-16.

Note: This transformation is prebuilt for you in the tutorial application. It is available in the **`Tutorial_Process_Application_Web\requestquote`** folder. A description of how to create the **`PriceAvailTransform.java`** file is beyond the scope of this tutorial.

Create Instance of PriceAvailTransformations Control

1. If the **Data Palette** pane is not visible in BEA Workshop, choose **Windows > Show View > Data Palette** from the menu bar.
2. In the **Package Explorer** view, select the **PriceAvailTransformations.java** file, and drag it to the **Controls** area of the **Data Palette**. An instance of the control (**`priceAvailTransformations`**) is created and displayed in the **Data Palette**.

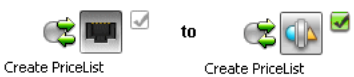
Design Interaction of Create PriceList Node with the Transformation

1. In the **Data Palette**, expand the **priceAvailTransformations** control instance.
2. Select the following method:

```
XmlObject convertPriceXMLtoXMLObj
(org.example.price.PriceRequestDocument _priceRequestDoc)
```

3. Drag the method from the **Data Palette** and drop it on the **Create PriceList** node in the **Design** view. The **Create Price List** node changes to reflect the binding of the method, as shown in the following figure.

Figure 9-5 Change in Node After Binding Method



4. Double-click the **Create PriceList** node.

   The node builder opens on the **General Settings** tab.

5. Confirm that the method that you bound to the node is selected:
6. Click **Send Data** in the node builder.

   The **Control Expects** field contains the data type and name of the parameter expected by the **`convertPriceXMLtoXMLObj()`** method on the **`priceAvailTransformations`** control: **`PriceRequestDocument _priceRequestDoc`**.

7. In the **Select variable to assign** field, select **price (PriceRequestDocument)**.

In this case, note that the data type of the **`price`** variable (**`PriceRequestDocument`**) matches that of the data expected by **`priceAvailTransformations`**.

8. Click **Receive Data** to open the third tab in the node builder.

   The **Control Returns** field contains the data type of the parameter returned by the **`convertPriceXMLtoXMLObj()`** method on the **priceAvailTransformations** control: **`XmlObject`**.

   Note: An **`XmlObject`** is a Java data type that specifies data in untyped XML format (data that is not valid against any XML schema).

9. In the **Select variable to assign** field, select **Create new variable...**.

   The **Create Variable** dialog box is displayed.

10. In the **Variable Name** field, enter **priceList**.

11. Select the **XML** tab to display a representation of the XML data types in your application.

    XmlObject is selected by default. You must change this selection in the following step.

12. Select **XmlObjectList** and click **OK**.

13. In the **Receive Data** tab, select **priceList(XmlObjectList)** from the **Select variables to assign** drop-down list.

    The **`priceList`** variable is created and assigned to receive the **`XmlObject`** data returned by the **`priceProcessor`** service.

14. Click **Close** to close the **Create PriceList** node builder.

    This step completes the design of the **Get Price** branch on the **Parallel** node in your business process.

    At run time, by executing this branch, your business process appends the **`XmlObject`**, which contains the data returned by the **`priceProcessor`** control during the current iteration through the **For Each** loop, to the **`priceList`** variable.

15. From the WorkSpace Studio menu, select **File** > **Save All**.

Design Activities on Get Availability Branch

1. Rename the three nodes on the **Get Availability** branch (in the order in which they are executed) to **Request Availability**, **Receive Availability**, and **Create AvailList** respectively.

2. Complete the following tasks:

Design “Request Availability” Node

1. Double-click the **Request Availability** node. The node builder opens on the **General Settings** tab.
2. In the **Control** field, select **availProcessor**.
3. The **Method** panel shows a list of the asynchronous send methods that you can invoke on the **`availProcessor`** control.

   Select **`void getAvail(int itemID_arg, int quantity_arg)`**.
4. Click **Send Data** in the node builder.

   By default, the **Send Data** tab opens on the **Variable Assignment** pane. The **Control Expects** field shows the data types and names of the parameters expected by the **`getAvail()`** method exposed by the **`availProcessor`** web service: **`int itemID`** and **`int quantity`**.

   Note: The **`availProcessor`** web service takes the item ID (int) and quantity (int) as input. It returns the item ID (int), quantity available (int), a boolean to indicate whether the items are available in stock, and a ship date (String).
5. Select **Transformation** to switch modes in the **Send Data** tab.

   Note: In this case, you must switch modes because the data that you provide as input to **`availProcessor`** must be transformed. The **`availProcessor`** control requires **`int`** data types as input, whereas the **`iter_forEach1`** variable, which holds the value of **`widgetId`** and **`quantity`** in the **For Each** loop, is a typed XML variable (**`WidgetRequestDocument`** is valid against an XML schema).
6. Click **Select Variable** and choose **iter_forEach1 (WidgetRequest)**.
7. Click **Create Transformation**.

   The transformation tool opens and displays a representation of the **`iter_forEach1`** variable in the **Source** pane, and the integer arguments to the **`availProcessor`** transformation method in the **Target** pane.
8. Map the elements in the **Source** pane to the elements in the **Target** pane, as follows:

- **`widgetID`** to **`itemID_arg`**
- **`quantity`** to **`quantity_arg`**

Note: This transformation creates a new method under the **`RequestQuoteTransformation.java`**. It is available in the **`Tutorial_Process_Application_Web/requestquote`** folder. A new XQ file, which contains the query for this transformation method, is also created.

9. Select **RequestQuote.java** in the **Package Explorer** to return to your process.
10. Click **Close** to close the **Request Availability** node builder.

    This step completes the design of the **Request Availability** node.

Design "Receive Availability" Node

1. Double-click the **Receive Availability** node. The node builder opens on the **General Settings** tab.
2. In the **Control** field, select **availProcessor**.
3. The **Method** panel shows a list of the asynchronous receive methods for **availProcessor**.
4. Select the following method:

   ```
   void avail(int itemID_arg, int qty_arg, boolean avail_arg, String
   date_arg)
   ```

5. Click **Receive Data** to open the second tab in the node builder.

   The **Control Returns** fields are populated with the data types and names of the parameters returned by the **`avail()`** method on the **`availProcessor`** web service.

   Note: In this case, you must switch from the **Variable Assignment** mode to the **Transformation** mode on the **Receive Data** tab because the data returned by the **`availProcessor`** web service must be assigned to a typed XML variable. For this, your process must transform the Java data types returned to typed XML data.

6. Click **Transformation**.
7. Click **Select Variable**, and then **Create new variable...**.

   The **Create Variable** dialog box is displayed.

8. In the **Variable Name** field, enter **avail**.
9. Select the **XML** tab.

10. Expand the **Typed** node (by clicking the + symbol adjacent to it), and progressively expand the nodes up to **schemas**.

11. Under the **schemas** node, expand **AvailQuote.xsd**, and select **availRequest**.

    The **Type Name** field shows `org.example.avail.AvailRequestDocument`.

12. Click **OK**.

    The **Create Variable** dialog box is closed and your new variable is created and listed as an **XML** type variable in the **Data Palette**.

13. In the **Receive Data** of the node builder, click **Create Transformation** to open the transformation tool, which displays a representation of the data types returned by `availProcessor` in the **Source** pane, and the `avail` variable in the **Target** pane.

14. Map the **Source** values to the **Target** elements as follows:

    - `itemID_arg` to `widgetId`
    - `qty_arg` to `requestedQuantity`
    - `avail_arg` to `quantityAvail`
    - `date_arg` to `shipDate`

    Note: This transformation creates a new method under the `RequestQuoteTransformation.java` in the `requestquote\Tutorial_Process_Application_Web` folder. A new XQ file, which contains the query for this transformation method, is also created.

15. Choose **File > Save All**.

16. Click **RequestQuote.java** in the **Package Explorer** to return to your business process.

17. To close the **Receive Availability** node builder, click **Close**.

    This step completes the design of the **Receive Availability** node.

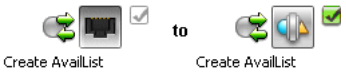
Design "Create AvailList" Node

In this step, you call a method on the `priceAvailTransformations` control to append availability data to a single variable of type `XmlObjectList`.

Note: For more information about the `XmlObjectList` data type, see "XmlObjectList Data Type" on page 10-16.

1. Expand the **`priceAvailTransformations`** control instance in the **Data Palette** and select the **`XmlObject convertAvailXMLtoXMLObj()`** method.
2. Drag the method from the **Data Palette** and drop it on the **Create AvailList** node in the **Design** view. The **Create AvailList** node changes to reflect the binding of the method.

Figure 9-6 Change in Node After Binding Method



3. Double-click the **Create AvailList** node. The node builder is displayed.
4. Confirm that the **`priceAvailTransformations`** control is selected in the **Control** field, and that the method you dragged onto the node is selected in the **Method** field:
5. Click **Send Data** to open the second tab in the node builder.

   The **Control Expects** field contains **`AvailRequestDocument`**, which is the data type expected by the **`convertAvailXMLtoXMLObj()`** method on the **`priceAvailTransformations`** control.
6. In the **Select variable to assign** field, select **avail (AvailRequestDocument)**.

   In this case, note that the data type of the **`avail`** variable (**`AvailRequest`**) matches that of the data expected by the **`priceAvailTransformations`** control.
7. Click **Receive Data** in the node builder.

   The **Control Returns** has **`XmlObject`**, which is the data type returned by the **`convertAvailXMLtoXMLObj`**() method on the **priceAvailTransformations** control.

   Note: **`XmlObject`** is a Java data type that specifies data in untyped XML format (data that is not valid against any XML schema)
8. In the **Select variable to assign** field, select **Create new variable...**.

   The **Create Variable** dialog box is displayed.
9. In the **Variable Name** field, enter **availList**.
10. Select the **XML** tab to display a representation of the XML data types in your application.
11. Select **XmlObjectList** under **Untyped**, then click **OK**.

    The **`availList`** variable is created and assigned to receive the **`XmlObject`** data returned by the **`availProcessor`** web service.

12. To close the **Create AvailList** node builder, click **Close**.

    At run time, the process appends **`XmlObject`**, which contains the data returned by the **`availProcessor`** control during the current iteration through the **For Each** loop, to the **`availList`** variable.

13. From the WorkSpace Studio menu, select **File** > **Save All**.

XmlObjectList Data Type

On each iteration through the **For Each** loop,

- The **`priceProcessor`** web service returns price data, which is assigned to the **`price`** variable.
- The **`availProcessor`** web service returns availability data, which is assigned to the **`avail`** variable.

Your business process must collect the price data returned on each iteration and create a list of price data; one item is assigned to the list for each iteration through the loop. Similarly, a list of availability data is created on the **Get Availability** branch of the **Parallel** node for each iteration through the loop.

`XmlObjectList` is a Java data type that specifies a sequence of untyped XML format data. This data type represents a sequence of XML elements (a set of repeating elements). As the final step of each iteration through the **Get Price** branch in the **Parallel** node, the business process assigns the data from the **`price`** variable to the **`priceList`** variable (type: **`XmlObjectList`**). In this way, a single variable holds the price data for each item for which the **For Each** loop iterates. Similarly, a single variable holds the availability data for each item.

For information about how the **`XmlObjectList`** variable is used, see "Design "Create PriceList" Node" on page 10-9 and "Design "Create AvailList" Node" on page 10-14.

Related Topic

Guide to Data Transformation

Step 9: Create Quote Document

As a result of the work you did when you designed the **Parallel** node, at the point at which the business process exits the **For Each** node, the price quotes are assigned to the **`priceList`** variable, and the availability information is assigned to the **`availList`** variable. Both the **`priceList`** and the **`availList`** variables are of data type **`XmlObjectList`** (untyped sequences of XML data).

In this step, you first transform the data in the **`priceList`** and **`availList`** variables from untyped XML data (**`XmlObjectList`**) to typed XML (XML that is valid against the XML schemas provided in your project). Subsequently, you combine the typed XML price and availability data to produce a single **`quote`** document, which is the response that your business process sends to the client that invoked it.

Note: WLI lets you to create transformations in the following ways:

- Use the node builders in the business process. You are already familiar with creating a transformation control and transformation methods in this way. **`RequestQuoteTransformation.java`** was created for you the first time you created a transformation from a node builder, that is, when you needed to map the data types from the RFQ message to the input of the **`taxCalculation`** control. (To review, see "Call the Tax Calculation Web Service From Your Business Process" in Step 4: Invoke a Web Service.) You subsequently created several additional transformation methods on **`RequestQuoteTransformation.java`** (and associated XQ files) on control nodes within the parallel node that you designed.
- You can choose **File > New > Other > WebLogic Integration > XQuery Transformation** from the WorkSpace Studio menu.

- You can change the perspective to **XQuery Transformation**, and then choose **File > New > XQuery Transformation** from the WorkSpace Studio menu.
- Go to the **Source** view of the transformation file, right-click, and choose **Transform > Add XQuery Transformation Method**.
- Create an XQuery file, and then choose its method from the node builder, by selecting the **Send Data** (or **Receive Data**) tab, **Transformation** option, and then the **Advanced...** button.

Transformation files can be called from your business process through control nodes. The following transformation files are provided for you in the tutorial application:

- **`PriceAvailTransformations.java`**
- **`TutorialJoin.java`**.

You used **`PriceAvailTransformations.java`** in Step 8: Design Parallel Paths of Execution.

You will use both these transformation files in this step.

A description of how to create these transformation files is outside the scope of this tutorial. For more information, see *Tutorial: Designing Your First Data Transformation*.

In this step, you design the logic in your business process to create a single quote document from the price and availability information. This involves designing control nodes that call the **`PriceAvailTransformations.java`** and **`TutorialJoin.java`** transformation files.

This step consists of the following tasks:

- Convert Price List to XML Document
- Convert Availability List to XML Quote Document
- Combine Price and Availability Quotes

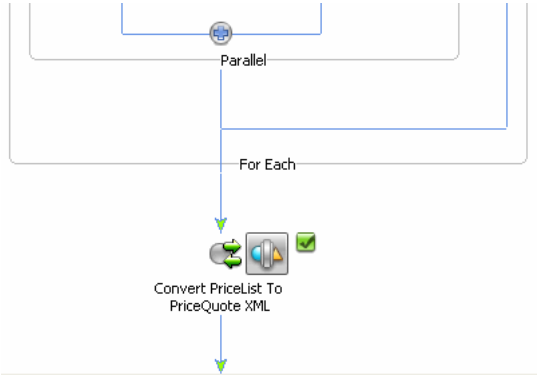
Convert Price List to XML Document

Complete the following steps to design a node for transforming the price list (created as a result of iteration through the **For Each** loop) to a typed XML variable.

1. Under the **`priceAvailTransformations`** control instance in the **Data Palette**, select the **`PriceQuoteDocument convertPriceListToXML()`** method.

2. Drag the method from the **Data Palette** and drop it on the RequestQuote business process in the **Design** view, immediately after he **For Each** block, as shown in the following figure.

Figure 10-1 Adding Control to Convert Price List to XML Document



3. Rename the node from **`convertPriceListToXML`** to **Convert `PriceList To PriceQuote XML`**.

4. Double-click the node to open its node builder.

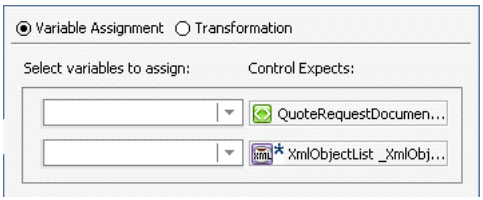
5. Verify that the **priceAvailTransformations** control and the following method are selected in the **General Settings** tab:

```
PriceQuoteDocument convertPriceListToXML
(org.example.request.QuoteRequestDocument _quoteRequestDoc,
com.bae.xml.XmlObjectList _XmlObjectListDoc)
```

6. Select the **Send Data** tab.

   The **Control Expects** field shows with the data type expected by the **`convertPriceListToXML()`** method on the **`priceAvailTransformations`** control:

Figure 10-2 "Control Expects" Field



Note: The **`convertPriceListToXML()`** method on the **`priceAvailTransformations`** control is designed to achieve two goals: First, to transform the **`XmlObjectList`**

price data to typed XML, and then to combine the customer name, shipping address, and price data in a single variable. The **`convertPriceListToXML()`** method receives the price list in a parameter of type **`XmlObjectList`**, and the customer name and shipping address in a parameter of type **`QuoteRequestDocument`**.

7. On the **Send Data** tab, under **Select variables to assign**, assign the variables that hold the data required by the **`priceAvailTransformations`** control as follows:
   - Click the arrow in the variable assignment field associated with **QuoteRequestDocument** and select **requestXML (QuoteRequestDocument)**. The **requestXML** variable holds the customer name and shipping address.
   - Click the arrow in the variable assignment field associated with **XmlObjectList**, and select **priceList (XmlObjectList)**.
8. Select the **Receive Data** tab.

   The **Control Returns** field contains **`PriceQuoteDocument`**, which is the data type returned by the **`convertPriceListToXML()`** method on the **`priceAvailTransformations`** control.
9. In the **Select variables to assign** field, select **Create new variable...**.

   The **Create Variable** dialog box is displayed.
10. In the **Variable Name** field, enter **`priceQuote`**.
11. Select the **XML** tab.
12. Expand the **Typed** node (by clicking the + symbol adjacent to it), and progressively expand the nodes up to **schemas**.
13. Under the **schemas** node, expand **PriceQuote.xsd**, and select **priceQuote**.

    The **Type Name** field shows **`org.example.price.PriceQuoteDocument`**.
14. Click **OK** to close the **Create Variable** dialog box.
15. Click **Close** to close the node builder.

    This step completes the design of the **Convert PriceList to PriceQuote XML** node. At run time, the price quotes (in typed XML format), and the customer name and shipping address are assigned to the **`priceQuote`** variable.

Note: The **`convertPriceListToXML()`** method on the **`priceAvailTransformations`** control creates the price quote XML data in the preceding step.

The input to the transformation method includes the original data sent by the client (in the **`requestXML`** variable), and the price data returned by the **`priceProcessor`** control (in the **`priceList`** variable) after the iterations in the **For Each** node.

The **`convertPriceListToXML()`** method extracts the customer name and shipping address from the **`requestXML`** variable, and a list of item IDs and prices from the **`priceList`** variable, and maps the data to the new variable (**`priceQuote`**).

You can double-click **`PriceAvailTransformations.java`** in the **Package Explorer** view and see the transformation control in the **Source** view. To view the data transformation, right-click on the **`convertAvailXMLtoXMLObj`** method, and select **Transform > Goto XQuery Document**.

Related Topics

Guide to Data Transformation

Tutorial: Building Your First Data Transformation

Convert Availability List to XML Quote Document

Complete the following steps to design a node to transform the availability list (created as a result of iteration through the **For Each** loop) to a typed XML variable.

1. Expand the **`priceAvailTransformations`** control instance in the **Data Palette**, and select the following method:

   ```
   AvailQuoteDocument convertAvailListToXML(com.bea.xml.XmlObjectList
   _XmlObjectListDoc)
   ```

2. Drag the method from the **Data Palette** and drop it on your **RequestQuote** business process in the **Design** view, placing it immediately after the **Convert PriceList to PriceQuote XML** node.

3. Rename the node to **`Convert AvailList to AvailQuote XML`**.

4. Double-click the node to open its node builder.

5. Verify that the **`priceAvailTransformations`** control and the following method are selected on the **General Settings** tab:

   ```
   AvailQuoteDocument convertAvailListToXML(com.bea.xml.XmlObjectList
   _XmlObjectListDoc)
   ```

6. Select the **Send Data** tab.

The **Control Expects** field contains **`XmlObjectList`**, which is the data type required by the **`convertAvailListToXML()`** method on the **`priceAvailTransformations`** control.

7. In the **Select variables to assign** field, select **availList (XmlObjectList)**.
8. Select the **Receive Data** tab.

   The **Control Returns** field contains **`AvailQuoteDocument`**, which is the data type returned by the **`convertAvailListToXML()`** method on the **`priceAvailTransformations`** control.

9. In the **Select variables to assign** field, select **Create new variable...**.

   The **Create Variable** dialog box is displayed.

10. In the **Variable Name** field, enter **availQuote**.
11. Select the **XML** tab.
12. Expand the **Typed** node (by clicking the + symbol adjacent to it), and progressively expand the nodes up to **schemas**.
13. Under the **schemas** node, expand **AvailQuote.xsd**, and select **availQuote**.

    The **Type Name** field shows **`org.example.avail.AvailQuoteDocument`**.

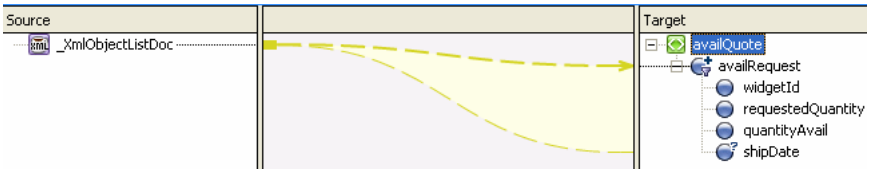
14. Click **OK** to close the **Create Variable** dialog box.
15. Click **Close** to close the node builder.

    This step completes the design of the **Convert AvailList to AvailQuote XML** node. At run time, the availability information in XML format is assigned to the **`availQuote`** variable.

Note: The **`convertAvailListToXML()`** method on the **`priceAvailTransformations`** control creates the availability XML document. The input to **`convertAvailListToXML()`** is the data returned by the **`availProcessor`** control after the iterations in the **For Each** node.

You can double-click **`PriceAvailTransformations.java`** in the **Package Explorer** view and see the transformation control in the **Source** view. To view the data transformation, right-click on the **`convertAvailListToXML`** method, and select **Transform > Goto XQuery Document**.

The following figure shows the data transformation for the **`convertAvailListToXML()`** method:



The figure shows transformation of the data in a variable of type **`XmlObjectList`**, which contains a repeating set of untyped XML data, to the repeating element in a typed XML variable.

To achieve this transformation, the repeating element in the target schema **must** be the single child of a root element. In this case, **`availRequest`** is the repeating element, and it is the single child of the **`availQuote`** element. Click the **Source** view tab in the transformation tool to see the corresponding XQuery.

Combine Price and Availability Quotes

Complete the following tasks:

- Create Instance of TutorialJoin Control
- Design Process Interaction with TutorialJoin Control

Create Instance of TutorialJoin Control

Complete the following steps to add an instance of the **`TutorialJoin.java`** control in the business process.

1. In the **Package Explorer** view, select the **`TutorialJoin.java`** file. It is available in the **`Tutorial_Process_Application_Web\requestquote`** project folder.

   Note: For information about building the **`TutorialJoin.java`** control, see *Tutorial: Building Your First Data Tansformation*.

2. Drag the **`TutorialJoin.java`** file from the **Package Explorer** view to the **Data Palette**. An instance of the control (**`tutorialJoin`**) is created and displayed in the **Data Palette**.

Design Process Interaction with TutorialJoin Control

In this step, you design the business process to call the following method on the **tutorialJoin** control:

```
join(PriceQuoteDocument _priceQuoteDoc,
AvailQuoteDocument _availQuoteDoc, float taxRate)
```

This method combines the data returned to your business process from different systems and creating a single XML response document (quote), which is subsequently returned to the client that invoked the business process.

1. Expand the **tutorialJoin** control instance in the **Data Palette** and select the following method:

   ```
   QuoteDocument join
   (org.example.price.PriceQuoteDocument _priceQuoteDoc,
   org.example.avail.AvailQuoteDocument _availQuoteDoc,float taxRate)
   ```

2. In the **Design** view, drag the method from the **Data Palette** and drop it on your **RequestQuote** business process immediately after the **Convert AvailList to AvailQuote XML** node.

3. Rename the node to **Combine Price and Avail Quotes**.

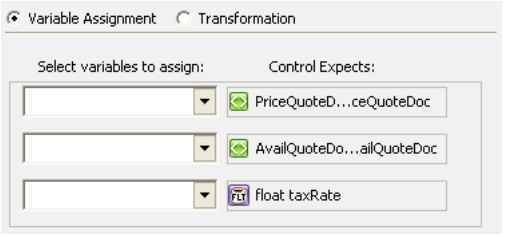
4. Double-click the **Combine Price and Avail Quotes** node. The node builder opens on the **General Settings** tab.

5. Confirm that **`tutorialJoin`** is displayed in the **Control** field, and that the following method is selected in the **Method** field:

   ```
   QuoteDocument join
   (org.example.price.PriceQuoteDocument _priceQuoteDoc,
   org.example.avail.AvailQuoteDocument _availQuoteDoc,float taxRate
   ```

6. Select the **Send Data** tab.

   The **Control Expects** field shows the data type expected by the **`join`** method on the **`tutorialJoin`** control, as shown in the following figure:



7. In the **Select variables to assign** field, select the variables such that their data types match the data type expected (**Control Expects**) in the input parameters to the **`join()`** method.
   - For **PriceQuoteDocument**, select **priceQuote (PriceQuoteDocument)**.

     `priceQuote` holds the price quote data, which is returned from the **`priceProcessor`** service in the **For Each** loop in your business process.
   - For **AvailQuoteDocument**, select **availQuote (AvailQuoteDocument)**.

     `availQuote` holds the availability quote data, which is returned from the **`availProcessor`** service in the **For Each** loop in your business process.
   - For **float taxRate**, select **taxRate (float)**.

     `taxRate` holds the rate of sales tax applied to the quote, based on the shipping address, which is returned to your business process from the **`taxCalculation`** service.
8. Select the **Receive Data** tab. The **Control Returns** field shows **`QuoteDocument`**, which is the data type returned by the **`join()`** method.
9. In the **Select variable to assign** field, select **Create new variable...**.

   The **Create Variable** dialog box is displayed.
10. In the **Variable Name** field, enter **Quote**.
11. Select the **XML** tab.
12. Expand the **Typed** node (by clicking the + symbol adjacent to it), and progressively expand the nodes up to **schemas**.
13. Under the **schemas** node, expand **Quote.xsd**, and select **quote**. The **Type Name** field shows **`org.example.quote.QuoteDocument`**.
14. Click **OK** to create the new variable. The **`Quote`** variable is displayed in the **Receive Data** tab, and also in the **Data Palette**.
15. Click **Close** to close the node builder.

    This step completes the design of the **Combine Price and Avail Quotes** node. At run time, the availability quote data in XML format is assigned to the **`Quote`** variable.
16. From the WorkSpace Studio menu, choose **File** > **Save All**.

    The only tasks that remain in this part of the tutorial are to write the quote to your file system (optional step) and create the **Client Response** node in the business process. The business process returns the quote you created to the client via the **Client Response** node.

Step 10: Write Quote to File System

In this step, you create a node for the business process to write the price and availability information to your file system. A file control enables processes to read, write, or append to a file in a file system.

This step includes the following tasks:

1. Create Instance of File Control
2. Design Control Send Node to Interact with File Control
3. (Optional) Assign File Control Properties to a Variable
4. (Optional) Use the File Control Properties

The third and fourth tasks are optional. They are provided to help you learn more about file controls, and are not required for completion of the tutorial.

Create Instance of File Control

1. Click on the **Data Palette Controls** tab and choose **Integration Controls > File**.

   The **Insert Control** dialog box is displayed.

2. Enter **`myFileQuote`** as the name of the control, and click **Next**.
3. In the **Create Control** dialog box, enter **`MyFileQuote`** in the **Name** field, and click **Next**.

a. In the **Insert Control - File** dialog box, specify the following:

- **Directory Name**: Enter the location in which you want the file control to write the file. You can use any location on your file system.
- **File name filter**: Enter a name for the file. For example, enter **`quote.xml`**.
- **Type of data**: Select **XmlObject** from the drop-down list.

b. Click **Finish**.

An instance of a file control, named **`myFileQuote`**, is created in your project and displayed under **Controls** in the **Data Palette**.

4. From the WorkSpace Studio menu, select **File** > **Save**.

Note: In the simple case, each instance of the file control allows you to manipulate a separate file. For information about using a file control to manipulate multiple files, see File Control.

Design Control Send Node to Interact with File Control

1. Expand the **myFileQuote** control instance in the **Data Palette**, and select the following method:

   `FileControlPropertiesDocument write(XmlObject someData)`

2. Drag the method and drop it in the **RequestQuote** business process immediately after the **Combine Price and Avail Quotes** node.
3. Rename the node to **Write Quote to File**.
4. Double-click the **Write Quote to File** node.
5. In the **General Settings** tab, confirm that **myFileQuote** is displayed in the **Control** field and that the following method is selected in the **Method** field:

   `FileControlPropertiesDocument write(XmlObject someData)`

6. Select the **Send Data** tab.

   The **Control Expects** field contains **`XmlObject someData`**, which is the data type that the **`write()`** method expects.

7. In the **Select variables to assign** field, select **Quote (QuoteDocument)**, which you created in "Step 9: Create Quote Document" on page 11-1.

Note: The node builder for this node contains a **Receive Data** tab. You can use this tab to specify a variable to which the data returned by the file control is assigned. For the purposes of this tutorial scenario, it is not required that you specify this variable; you can ignore the **Receive Data** tab. However, to learn how to specify a variable on the **Receive Data** tab, and a scenario in which you might subsequently use the variable, see "File Control Properties" on page 12-3.

8. Click **Close** to close the node builder.
9. From the WorkSpace Studio menu, select **File > Save**.

   This step completes the design of your file control node.

At run time, the quote document that you created in "Step 9: Create Quote Document" on page 11-1 is written to your file system in the location specified by you.

File Control Properties

The next two sections provide additional (optional) steps for further defining the **Write Quote to File** node that you created in the preceding section. These steps are not necessary for completing the tutorial. They are provided to help you understand and use the file control properties that are returned to your business process by the **`FileControlPropertiesDocument write(XmlObject someData)`** method.

When you use a file control to write a file to the file system, as you do in this step, the control returns information about the file you wrote. The information is returned in a typed XML document, **`FileControlPropertiesDocument`**, which is valid against the XML schema, **`DynamicProperties.xsd`**. The schema is available in the utility project under **`\schemas\system`**.

Note: To be able to use file control APIs, the **`DynamicProperties.xsd`** schema must be available in the **`schemas`** folder of your application's utility or web project.

Assign File Control Properties to a Variable

This section describes how to design the **Write Quote to File** node in your business process to include assigning a variable to which the File Control Properties are assigned:

Note: Before starting this section, you should have completed steps 1 through 7 in Design Control Send Node to Interact with File Control.

1. If the **Write Quote to File** node is not already open, double-click the node to open it.
2. Select the **Receive Data** tab.

The **Control Returns** field contains **`FileControlPropertiesDocument`**, which is the data type returned by the **`write()`** method.

3. In the **Select variables to assign** field, select **Create new variable...**.

   The **Create Variable** dialog box is displayed.

4. In the **Variable Name** field, enter **`fileProperties`**.

5. Select the **XML** tab.

6. Expand the **Typed** node (by clicking the + symbol adjacent to it), and progressively expand the nodes up to **schemas\system**.

7. Under **system**, expand **DynamicProperties.xsd**, and select **FileControlProperties**.

   The **Type Name** field shows **`com.bea.wli.control.dynamicProperties.FileControlPropertiesDocument`**.

8. Click **OK** to create the new variable.

   The **`fileProperties`** variable is displayed in the **Receive Data** tab, and also in the **Data Palette**.

9. Click Close to close the node builder.

10. From the WorkSpace Studio menu, select **File > Save**.

    This step completes the design of your file control node. At run time, the quote document that you create in Step 9: Create Quote Document is written to your file system in the location specified by you. Information about the file that you wrote is returned to the RequestQuote business process, and assigned to the **`fileProperties`** variable.

Use the File Control Properties

In the previous section, you assigned the data returned from the file control to a variable named **`fileProperties`**. You can now extract information about the file that you wrote from the **`fileProperties`** variable.

Click the **Source** view tab to view the code in the **`RequestQuote.java`** file.

Declaration of the **`fileProperties`** variable is shown in the following line:

```
public
com.bea.wli.control.dynamicProperties.FileControlPropertiesDocument
fileProperties;
```

The **`write()`** method on the **`myFileQuote`** control is coded as shown in the following listing:

Listing 11-1 Code for write() Method

```
public void myFileQuoteWrite() throws Exception {
   // #START: CODE GENERATED - PROTECTED SECTION
      - you can safely add code above this comment in this method. #//
   // input transform
   // return method call
   this.fileProperties = myFileQuote.write(this.Quote);
   // output transform
   // output assignments
   // #END  : CODE GENERATED - PROTECTED SECTION
      - you can safely add code below this comment in this method. #//
}
```

You can edit this method (outside the `PROTECTED SECTION`) and write code for deriving information from the **`fileProperties`** variable.

For example, the following line of code returns the file mask:

```
this.fileProperties.getFileControlProperties().getFileMask()
```

To extend this example further, edit the **`public void fileQuoteWrite()`** method in the **Source** view to include the following line of code **after** the `PROTECTED SECTION`.

```
System.out.println
("The RequestQuote Process logged the quote in the following file " +
this.fileProperties.getFileControlProperties().getFileMask());
```

In the **Design** view, the icon adjacent to the **Write Quote to File** node changes to  .

This is a visual reminder that you edited the code associated with this node in the **Source** view. When you run the business process, the name you gave the file (the file mask) is printed to the console.

For more information, see "File Control" in *Using Integration Controls*.

Step 11: Send Quote From Business Process to Client

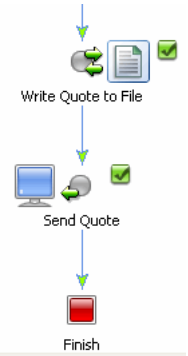
A business process must be able to send and receive messages to and from its clients. You designed your business process to receive messages from a client in "Step 2: Specify How the Process Is Started" on page 3-1.

In this step, you learn how to send messages from your business process to a client by designing **Client Response** nodes. This step includes the following tasks:

- Add Client Response Node
- Design Send Quote Node

Add Client Response Node

1. In the **Package Explorer** view, double-click **RequestQuote.java** to ensure that your business process is displayed in **Design** view.
2. In the **Node Palette,** select **Client Response**, drag it to the business process, and drop it immediately before the **Finish** node.
3. Change the name of the node to **Send Quote**.



Design Send Quote Node

This section describes how to complete the design of the interaction with clients for this business process. Specifically, at this point in the process, the business process sends a quote containing price and availability information to clients.

In this step, you specify the structure of documents that your business process sends to clients from the **Send Quote** node.

1. Double-click the **Send Quote** node in your business process.
2. In the **General Settings** tab of the node builder, change the name in the **Method Name** field from **clientResponse** to **quoteResponse**.
3. Click **Add** to display the panel of data types.

   Note: In the **Combine Price and Avail Quotes** node, you created an XML variable to hold the quote. The data assigned to this variable is valid against the **`Quote.xsd`** schema; therefore, we need a typed XML parameter at this node.
4. Select the **XML** tab.
5. Expand the **Typed** node (by clicking the + symbol adjacent to it), and progressively expand the nodes up to **schemas**.
6. Under **schemas**, expand **Quote.xsd**, and select **quote**.

   The **Type Name** field shows **`org.example.quote.QuoteDocument`**.
7. In the **Name** field, replace **x0** with **responseXML**.
8. Click **OK** to create the new variable.

The **`fileProperties`** variable is displayed in the **Receive Data** tab, and also in the **Data Palette**.

9. Click **OK**.

   The **`QuoteDocument responseXML`** parameter is added to the **General Settings** tab in the node builder and the **General Settings** tab is marked complete:.

10. Select the **Send Data** tab.

    The **Client Expects** field shows the data type and the name of the parameter that you specified in the **General Settings** tab: **`QuoteDocument responseXML`**

11. In the **Select variables to assign** field, select **Quote (quote)**.

12. Click **Close** to close the node builder.

13. From the WorkSpace Studio menu, select **File > Save**.

    This step completes the design of your RequestQuote business process.

To run the business process, proceed to "Step 12: Run RequestQuote Business Process" on page 14-1.

Step 12: Run RequestQuote Business Process

You can run and test the business process by using the browser-based interface of WorkSpace Studio. The interface lets you play the role of the client, invoking the methods on the business process and viewing the responses.

1. If WebLogic Server is not already running, from the BEA Workshop menu, choose **Window > Show View > Other > Server > Servers**, and click **OK**.

   A **Server** view is displayed in which the server and its state are shown.

2. In the **Package Explorer** view, right-click on **requestquote.java**, click **Run As**, and click **Run On Server**.

   The **Run On Server** wizard is displayed.

3. Select **Choose an existing server** (or **Manually define a server** if there is no server defined), and click **Next**.

   Note: If you select **Manually define a server**, the **Run On Server** wizard lets you specify the WebLogic Server and domain.

4. Click **Finish**.

   The server is started, and the RequestQuote application is deployed on it. The **Status** indicator in the **Server** view changes to **Started**.

   After the application is deployed, the test browser is displayed.

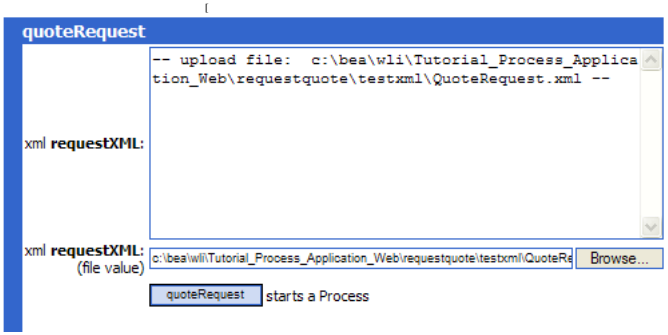
5. Select the **Test Form** tab.

6. Click the **Browse** button adjacent to the **xml requestXML (file value)** field and select the **`QuoteRequest.xml`** file from the following folder:

   `Tutorial_Process_Application_Web\src\testxml\`

7. Click **quoteRequest** to start the business process.

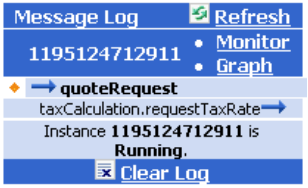
Figure 13-1 Test Form



Note: The label of the button reflects the name of the start method in the business process.

The **Message Log** area of the **Test Form** displays the status of the process.

Figure 13-2 Message Log



8. Click **Refresh** to refresh the entries in the log until this instance of the business process is completed.

Entries in the **Message Log** correspond to the methods in your business process, as shown in the following figure.

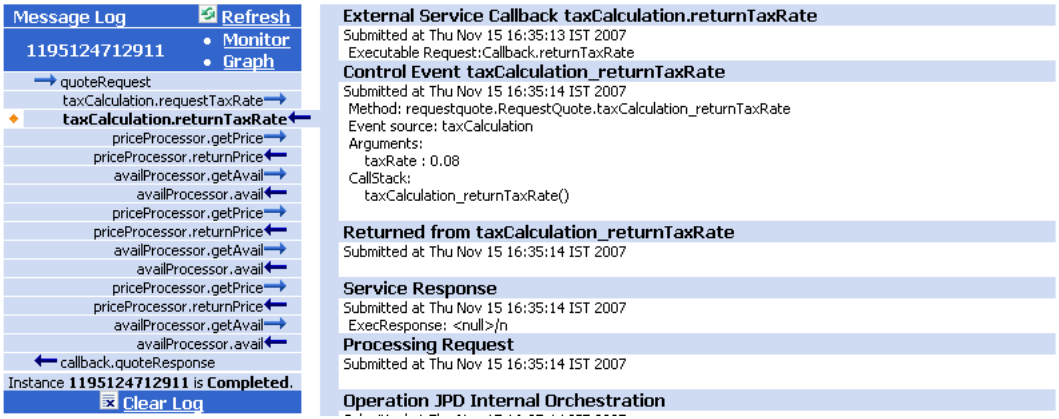
Figure 13-3 Message Log - Refreshed

- The **`quoteRequest`** method that starts the business process.
- A call from your business process to the **`taxCalculation`** process control: **`taxCalculation.requestTaxRate`**
- A response from the service to your business process: **`taxCalculation.returnTaxRate`**
- **Instance ID**: The ID generated when the **`quoteRequest`** method in your business process was called.

You can view the details of a specific call, by clicking on the corresponding methods in the **Message Log**.

For example, you can view the response from the **`taxCalculation`** service by clicking **`taxCalculation.returnTaxRate`**, as shown in the following figure.

Figure 13-4 Message Log with Details



In this case, the tax rate was calculated based on the input value (**`NJ`**) for the **`state`** element in the test XML.

You have now designed and tested a business process that performs the following activities:

1. Receives a request for quote message from a client (**Client Request** node).
2. Decides whether sales tax calculation is required (**Decision** node).
3. Requests another business process for the tax rate (**Control Send** node).
4. Receives the tax rate (**Control Receive** node).
5. Performs the following activities for each item in the Request for Quote (**For Each** and **Parallel** nodes).
   - Requests external resources for the price and availability information (**Control Send** node).
   - Receives price and availability information from the external resources (**Control Receive** node).
6. Creates an untyped XML list of the prices for all the items (**Control Send with Return** node).
7. Creates an untyped XML list of the availability information for all the items (**Control Send with Return** node).

8. Converts the price and availability information lists to typed XML documents (**Control Send with Return** node).
9. Combines the price and availability information in a single quote document (**Control Send with Return** node).
10. Writes the quote document to a file (**Control Send with Return** node).
11. Sends the quote to the client that invoked the business process (**Client Response** node).

Additional Functionality in Test Browser

The following additional links are available from the **Test Form** page in the test browser:

- **Graph**

  Click **Graph** to open the **Process Graph** tab in the test browser. The interactive instance graph is a fully expanded version of the view provided in the **Design** view.

- **Monitor**

  Click **Monitor** to open the **Process Instance Details** page of WLI Administration Console, which provides details of specific process instances.

- **Monitor all RequestQuote.jpd processes**

  Click **Monitor all RequestQuote.jpd processes** to open the **Process Instance Summary** page of WLI Administration Console.

Monitoring Processes in WLI Administration Console

You can monitor instances of your business process by using the WLI Administration Console. You can view process instances statistics, and view summary or detailed status for selected process instances. You can also suspend, resume, and terminate selected process instances.

1. Launch the WLI Administration Console in one of the following ways:
   - Click **Monitor** on the **Message Log** in the test browser's **Test Form** page.
   - Enter **`http://localhost:7001/wliconsole`** in a web browser:

   The default user name and password for the sample integration server are **`weblogic`**.
2. Select **Process Instance Monitoring** in the left navigation pane.

If you invoked the **Process Instance Monitoring** page after running the RequestQuote process, the RequestQuote and TaxCalcProcess processes are displayed in the **Process Instance Statistics** page.

3. Click the name of any business process in the **Display Name** column to go to a page that displays more information about that process.

   For example, to learn more about the instance of the TaxCalcProcess business process:

   a. Click TaxCalcProcess in the **Display Name** column.

      A **Process Instance Summary** page is displayed.

   b. This page lists all the instances of the TaxCalcProcess business process that ran or are running.

   c. To view more details about any instance, click the instance ID in the **ID** column.

4. On the **Process Instance Details** page, click **Graphical View** to view a graphical representation of this instance of the TaxCalcProcess business process.

5. In the graphical view, to see more information about a specific node, click the node.

For more information, see "Process Instance Monitoring" in *Using the WebLogic Integration Administration Console*.

APPENDIX A

WorkSpace Studio Views, Functions, and Shortcuts

This section describes the components that are available in WorkSpace Studio for designing business processes.

WorkSpace Studio Views

Ensure that you are familiar with the following WorkSpace Studio views; you will use them throughout the tutorial.

- **Package Explorer** view

  The **Package Explorer** view provides a hierarchical representation of the source files in your project, and provides a place where you can save, open, add, and delete project files.

  If the **Package Explorer** view is not visible in BEA Workshop, choose **Window > Show View > Other > Java > Package Explorer**.

- **Design** view

  The **Design** view is your primary working canvas. It displays the business process as you design it. You can drag and drop nodes, controls, and variables into the **Design** view to design your business process. There are many views such as **JPD Configuration**, **Data Palette**, **Node Palette**, **Server**, and **Problem** to help you in your tasks.

  You can also right-click a node or a group of nodes in the **Design** view to access options; different options are available depending on the type of node. The following are some of the options available from the right-click menu:

  – **Rename**, to rename the node

- **Add Exception Path,** to add an exception path to the node or group of nodes
- **Add Message Path**, to add a message path to node or group of nodes
- **Cut**, **Copy**, **Delete**, and so on.

To learn more about node groups in the **Design** view, see Grouping Nodes in Your Business Process.

- **Source** view

  This view displays the source code for the current process. As you design your process in the **Design** view, corresponding source code is written to the JPD file. You can also design and edit your JPD file in the **Source** view. For more information about the **Source** view, see Business Process Source Code.

- **Node Palette**

  This view displays the nodes that you can add to your business process. Nodes represent different types of logic in your business process.

  If the **Node Palette** is not visible in BEA Workshop, choose **Window** > **Show View** > **Node Palette**.

  As you drag a node from the **Node Palette** to the **Design** view, targets  are displayed on your business process. As you drag the node close to a target location, the target is activated  . When this happens, you can release the mouse button and the node snaps to the business process at the location indicated by the active target.

  Note: If you create a node at an invalid location (that is, if you create invalid logic in your business process) the node is marked with an error icon () in the **Design** view. You can view the error message describing the error by moving the mouse pointer over the error icon.

- **Data Palette**

  This view includes folders for **Variables** and **Controls**.

  - The **Variables** folder displays the variables created in your business process and allows you to create new variables.
  - The **Controls** folder displays the instances of controls in your business process and allows you to add new instances.

  You can use the **Data Palette** view to create variables and instances of controls in your project. You can also create variables and instances of controls in other ways as you design the process in the **Design** view.

If the **Data Palette** is not visible in BEA Workshop, choose **Window > Show View > Data Palette**.

- **JPD Configuration** view

  This view provides read and write access to the properties of the node or group of nodes that is selected in the **Design** view.

  If the **JPD Configuration** view is not visible in BEA Workshop, choose **Window > Show View > JPD Configuration**.

Functions and Shortcuts

You will use the following functions and shortcuts frequently throughout the tutorial:

 **Save**: Saves the file that is currently displayed in the **Design** or **Source** view.

Save All (Ctrl+S): Saves all the files in your application.

 **Build All**: **(Ctrl+B)**: Builds your applications. This icon is not available if automatic building is enabled by choosing **Project> Build Automatically**.

F2: To change the label (name) of a node in the business process, select the node in the **Design** view and press **F2**. Then, enter the required new label and press **Enter**.

 Use the up and down arrow keys to navigate between the nodes in your business process.

 Use the right and left arrow keys to expand and collapse a group of nodes.