# WebLogic Integration™

## Tutorial: Building Your First Business Process

# Contents

## 1. Tutorial: Building Your First Business Process

## Part I. Build and Run a Simple Business Process

## 2. Step 1: Create Your Business Process Application

## 3. Working in the Design View

## 4. Step 2: Specify How the Process is Started

## 5. Step 3: Define Conditions for Alternate Paths of Execution

## 6. Step 4: Invoke a Web Service

## 7. Step 5: Run Your Business Process

## Part II. Call a Business Process Using a Process Control

# Tutorial: Building Your First Business Process

WebLogic Integration's business process management (BPM) functionality enables the integration of diverse applications and human participants, as well as the coordinated exchange of information between trading partners outside of the enterprise.

This tutorial provides a tour of the features available to design business processes in the BEA Workshop graphical design environment. It describes how to create a business process that orchestrates the processing of a *Request for Quote*.

## Tutorial Goals

The goal of the tutorial is to provide the steps to create and test a business process using the graphical environment provided in BEA Workshop. It includes:

- Designing communication nodes in a business 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 business processes to perform one or more operations.

- Designing the interactions with clients, including creating the methods that expose your business process's functionality.

- Designing the interactions with resources using controls. WebLogic Platform controls make it easy to access enterprise resources, such as databases, Enterprise Java Beans (EJBs), Web services, and other business processes (including those that use RosettaNet and ebXML business processes) from within your application.

- Handling XML, non-XML, and Java data types in the business process—includes working with XML schemas and transforming data between disparate data types using the Transformation tool.

- Designing business processes to publish and subscribe to message broker channels.

# Tutorial Overview

The business process in this scenario is started as a result of receiving a Request for Quote from clients. The business process checks the enterprise's inventory and pricing systems to determine whether the order can be filled. Based on the shipping address provided by the client, the process also determines whether sales tax should 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 your file system and sends it to the client.

### Designing the Request for Quote Business Process

The following sequence summarizes the steps in the request for quote process and describes how the business process is designed:

1. Receive a Request for Quote 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 the items for which the quote is requested. You design the business process so that it starts when it receives a Request for Quote message from a client.

2. Evaluate a condition to determine whether sales tax should be included in the quote.

   In this case, 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 via 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 Request for Quote message.

   The business process must calculate the price and determine the availability of the items and quantities requested in the incoming XML message. This involves the creation of the following nodes in your business process:

   - **For Each**: For Each 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 case, you design a **Parallel** node containing two branches:

one to execute the events that calculate the price for the quote, the second to execute the events that determine the availability of items for the quote.

– **Control** nodes: Control Send and Control Receive nodes on each path handle the asynchronous exchange of messages between a business process and Web service resources (via controls). A pricing Web service returns the price for the items in the Request Quote document. An availability Web service returns information about the availability of the requested items.

4. Compile price, availability, and tax information calculated by the business process into a quote document.

Your business process calculates a price quote, availability information, and sales tax rate. You design your business process to use Transformation controls that map the various pieces of data 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.

Your business process uses a File control to write the quote to your file system.

6. Send a response, containing 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.

### Actors in the Tutorial Scenario

The actors in the tutorial scenario are represented in the following figure and described in the text that follows the figure:
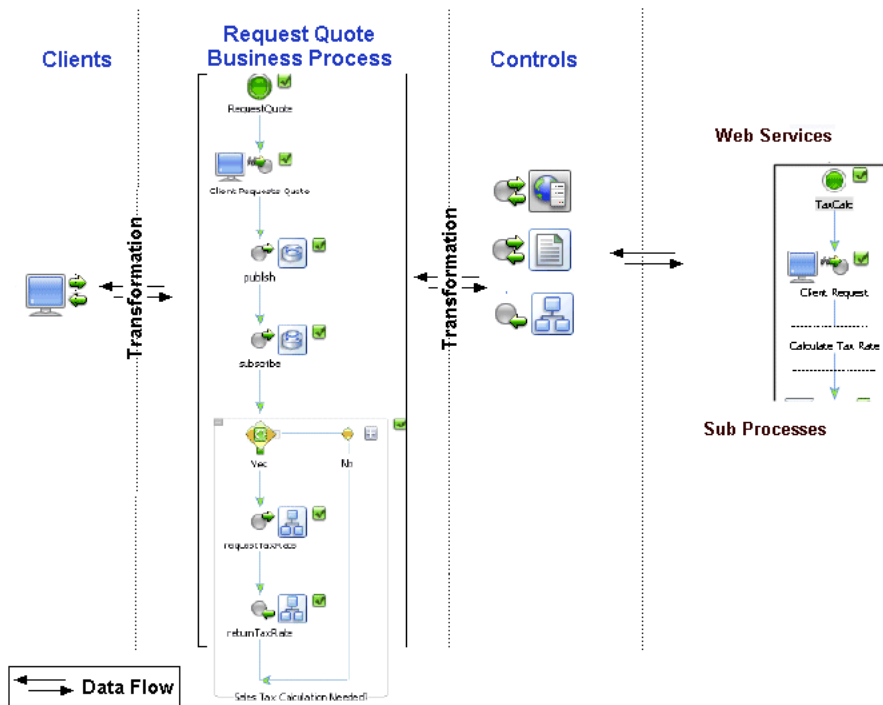
The actors in the scenario include:

- The client of your RequestQuote service. Clients of `RequestQuote` are systems that create and send Request for Quote messages. A Request for Quote message provides the business process with a customer name, shipping address, and a list of items and quantity of those items required by the client. The business process computes and returns a price and availability quote for the items requested.

- Your `RequestQuote` business process. The process receives a Request for Quote for specific items and returns a price and availability quote for the items requested.

- A tax calculation Web service designed to calculate the sales tax to include in the quote, based on the shipping address provided by a client.

- A tax calculation business process designed to calculate the sales tax. The tax calculation 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 calculation of the sales tax for the quote.

- A pricing Web service designed to calculate the price of the items requested by a client.

- An availability Web service designed to determine the availability of the quantity of items requested by a client.

- Transformation controls: The business process in this case is started when it receives an XML document from a client. Data is shared and exchanged between resources in your application (clients, your business process, Web services and so on) in XML format. Transformation controls are designed to support the mapping of disparate data formats used in your application.

- A business process that validates the Request for Quote from clients (ValidateQuote.jpd). The RequestQuote business process communicates with this ValidateQuote process via Message Broker channels. In this way, the interaction between the business processes can be loosely coupled and anonymous.

# Steps in This Tutorial

This tutorial creates a business process that meets the following requirements: receives Request for Quote messages from clients, starts the business process on receipt of the Request for Quote, validates and processes the request, and sends the status of the Request for Quote to the client.

The tutorial is organized into parts:

**Part I**

In Part I, you learn how to create a new business process, specify how the process is started at run time, and design a Decision node that includes asynchronous calls to a Web Service. Lastly, you can run and test the business process you created. To get started, proceed to Part I, "Build and Run a Simple Business Process".

**Part II**

In Part II, you learn how to replace the asynchronous call to the Web service you designed in Part I with an asynchronous call to another business process. You learn how to create a process control and how the control's framework makes it easy to change the interactions your business process makes with various resources. To learn about the specific steps to complete this part, see Part II, "Call a Business Process Using a Process Control".

**Part III**

In Part III, you add more complex business logic to the business process you created in the preceding parts. You learn how to 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, and use a Client Response node to return the quote to the client invoking the business process. At the end of this part, you can run and test the

business process you built. To learn about the specific steps to complete this part, see Part III, "Adding Looping Logic, Parallel Paths . . ."

# Part I    Build and Run a Simple Business Process

Part I of the tutorial is comprised of Steps 1 through 5. In this part, you learn how to create a new business process, specify how the process is started at run time, design a Decision node that includes asynchronous calls to a Web Service, and run and test the business process you create in this first part.

Specifically, the steps in Part I include:

**Chapter 2, "Step 1: Create Your Business Process Application"**
> Describes step-by-step instructions for creating a business process project in BEA Workshop.

**Chapter 4, "Step 2: Specify How the Process is Started"**
> Describes how to design the start of your business process. In this case, provides a step-by-step procedure to create a Client Request node, and add a method that receives the Request for Quote message from the client, which in turn causes the business process to start.

**Chapter 5, "Step 3: Define Conditions for Alternate Paths of Execution"**
> Describes how to design a decision node and its associated conditions in your business process. The path of execution through of a decision node is based on the evaluation of conditions you specify for the decision node.

**Chapter 6, "Step 4: Invoke a Web Service"**
> Describes how to design your business process's interaction with a Web Service control.

**Chapter 7, "Step 5: Run Your Business Process"**
> At this point, you have created a business process that you can run and test using the BEA Workshop Test Browser.

For information about how to work with the components and tools you use to design your business process in the BEA Workshop graphical design environment, see Chapter 3, "Working in the Design View."

# Step 1: Create Your Business Process Application

In this step, you use WebLogic Workshop to create the application, in which you build the tutorial business process (`RequestQuote.java`). The tasks in this step include:

- To Create a Business Process Tutorial Application
- To Begin the Design of Your Request for Quote Business Process

**To Create a Business Process Tutorial Application**

1. From the BEA Workshop **for WebLogic** menu, click **File** → **New** → **Other**. The **Select a Wizard** dialog box is displayed.

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

**Figure 2-1  Select a Wizard dialog box**
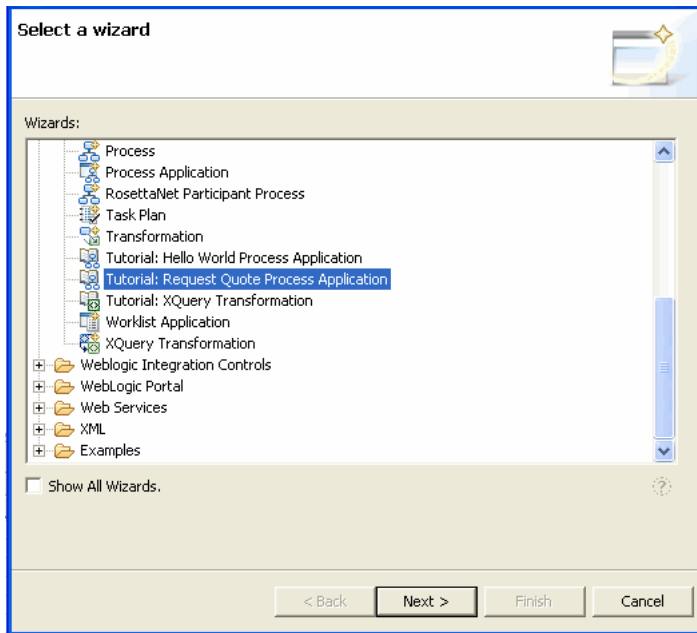


3. In the **Request Quote Process Application** dialog box, type the following:

   a. In the **Ear Project Name** field, enter `Tutorial_Process_Application_Ear`.

   b. In the **Web Project Name** field, enter `Tutorial_Process_Application_Web`.

   c. In the **Utility Project Name** field, enter `Tutorial_Process_Application_Utility`.

   d. Select the **Add Weblogic Integration System and Control Schemas in Utility Project** checkbox to add the System schemas to the **Schemas** folder under the Utility project.

**Figure 2-2  Creating Request Quote Application**
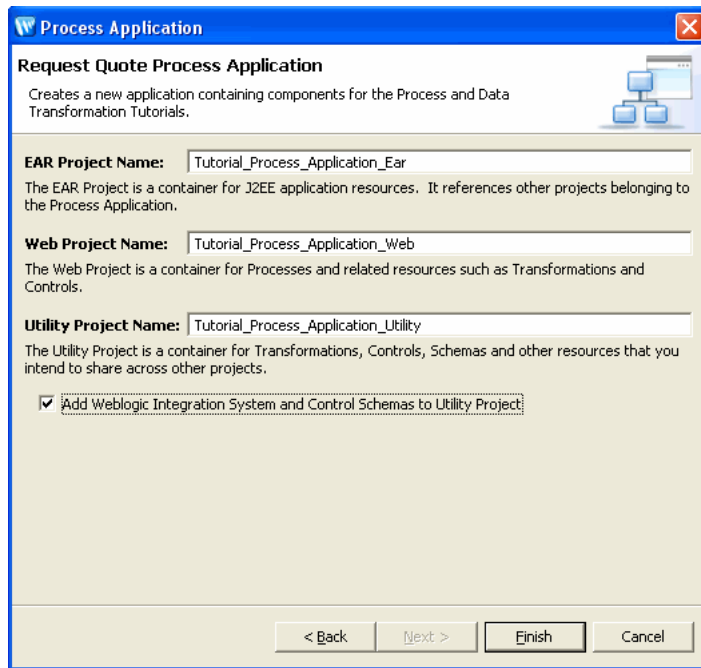


4.  Click **Finish**.

5.  In the displayed **Open Associated Perspective?** dialog box, click **Yes** to switch from **Workshop Perspective** to **Process Perspective**.

**Figure 2-3  Open Associated Perspective**
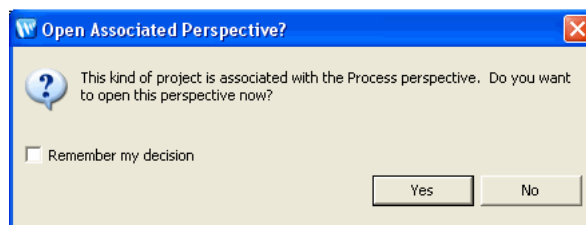


**Note:** Workshop is the default perspective of the 9.2 Workshop for WebLogic Integration. The Process perspective contains all the required views like **Node Palette**, **Data Palette**, and so on.

Similarly, XQueryTransformation perspective contains views pertaining to XQuery Transformation like **Expression Functions, Expression Variables**, **Target Expression**, and **Constraints**.

6.  The Tutorial Process Application is created and displayed in the **Package Explorer** pane.

**Figure 2-4  Package Explorer pane**
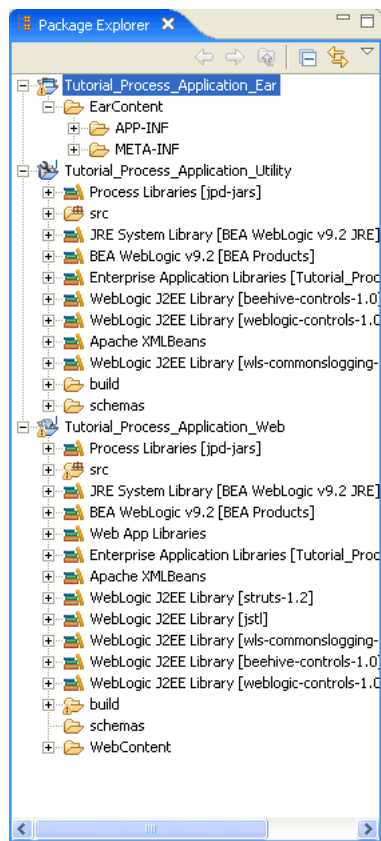


The **Package Explorer** pane displays the files and resources available in the application:

**Web Project Name**— A project with WebLogic Integration process facet added to it. Every application contains one or more projects. Projects represent WebLogic Server 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 your application.)

**Utility Project Name**—A project that contains the XML Schemas and the Message Broker channel file used in the application.

Web Applications are J2EE deployment units that define a collection of Web resources such as business processes, Web services, JSPs, servlets, HTML pages, and can define references to external resources such as EJBs.

**requestquote**—contains the business processes, transformation, xq files

– **FileQuote.java**—A File control used by your Request for Quote business process to write a file to the file system.

– **PriceAvailTransformations.java—**Contains data transformations used in `RequestQuote.java`.

– **RequestQuote.java**—The completed business process. (The tutorial walks you through rebuilding this business process. It is provided for reference, and allows you to run and test the business process before you start rebuilding it.)

– **RequestQuoteTransformation.java** and **TutorialJoin.java**)—Contains 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.

**requestquote.services** folder contains services with which your business process interacts. The **services** folder includes Web services, Web Service controls, business processes and Process controls.

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

**Note:**   If you want to run and test the `RequestQuote.java` provided for you in the application folder, complete the steps in Chapter , "Step 12: Run the Request Quote Business Process."

7. In this tutorial, you build the `RequestQuote.java` from scratch. Therefore, to proceed, you must delete the following files from your **Tutorial_Process_Application_WEB** project:

– `RequestQuote.java`

– `RequestQuoteTransformation.java` including its XQ files:

```
RequestQuoteavailProcessor_avail.xq
RequestQuoteavailProcessorGetAvail.xq
RequestQuotepriceProcessor_returnPrice.xq
RequestQuotepriceProcessorGetPrice.xq
RequestQuotetaxCalculationRequestTaxRate.xq
```

**Figure 2-5  Web Project Files**
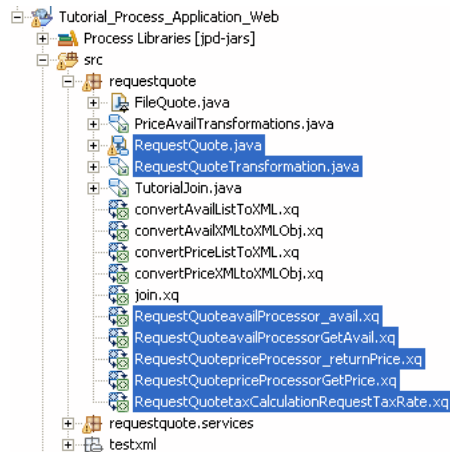


**Note:**  To delete these files, put your mouse pointer in the **Package Explorer** tab, then press **Shift** and select the files you want to delete. Right-click and select **Delete**. Click **Yes** in the Delete confirmation dialog box. Delete only the files listed in this step. You need all other files as you build the business process. Files are deleted from the **Package Explorer** pane (and from your application folder in the file system).

### To Begin the Design of Your Request for Quote Business Process

In this step you start the process of recreating the `RequestQuote.java` business process in the **requestquote** folder.

1. In the **Package Explorer** pane, under the `Tutorial_Process_ApplicationWeb\src\requestquote` folder, right-click the **requestquote** folder.

2. Select **New→Process** from the menu that is displayed. The **New Process File** dialog box is displayed.

**Figure 2-6  New Process File Dialog Box**
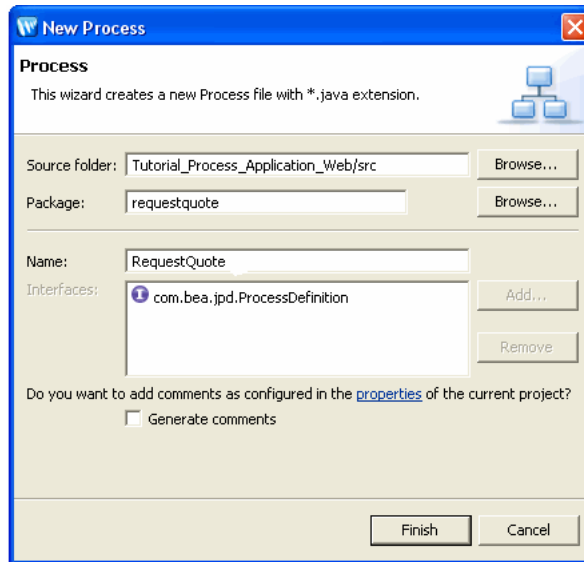


3.  In the **Name** field, enter **RequestQuote**.

4.  Click **Finish**.

    The new `RequestQuote.java` file is created and displayed in the **Design** view (which for the moment consists only of a **Start** and a **Finish** node).
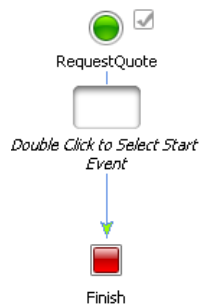


# Related Topics

Components of Your Application

# Working in the Design View

This section describes the components and tools you use to design your business process in the BEA Workshop graphical design environment. Ensure that you are familiar with the following items—you will use them throughout the tutorial.

**Package Explorer Pane**

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** pane is not visible in BEA Workshop, choose **Window→Show View→Other→Java→Package Explorer** from the BEA Workshop menu

**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, variables into the **Design** view to design your Business process. There are many views like Property View, Annotations View, Data Palette, Node Palette, Server view, Problem View, Error log available, 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 process node you are designing. Options available from the right-click menu include the following: **Rename** to rename the node, **Add Exception Path** to add an exception path to a node or a group of nodes, **Add Message Path** to add a message path to a node or group of nodes, **Cut**, **Copy**, **Delete**, and so on.

To learn more about groups of nodes in the Design View, see Grouping Nodes in Your Business Process.

**Source**

The Source displays the source code for the current business process. As you design your business process, source code is written to the JPD file in keeping with the work you do in the Design view. You can also design and edit your JPD file in the Source View. To learn more about the Source view, see Business Process Source Code.

**Node Palette**

The **Node Palette** 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 **Windows→Show View→ Node Palette** from the BEA Workshop menu.

As you drag a node from the **Node Palette** onto the **Design** view, targets ○ are displayed on your business process. As you drag the node near 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 that if you create a node at an invalid location (that is, if you create invalid logic in your business process flow) that node is marked with the following icon in the Design view: ⊗ Move your mouse pointer over the error icon to see a message that describes the error.

**Data Palette**

The Data Palette includes the following folders: **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.

Use the **Add** command on the **Data Palette** to create instances of variables and controls in your project. You can also create variables and instances of controls in other ways as you work in the **Design** view to create your process logic. As you work through the tutorial, you will employ the various methods of designing controls and variables in your business processes.

If the **Data Palette** is not visible in BEA Workshop, choose **Windows→Show View→Data Palette** from the menu bar.

**Property Editor**

Provides read and write access to the properties of a node or group of nodes selected in the **Design** view.

If the **Property Editor** is not visible in BEA Workshop, choose **Windows**→**Show View**→**Properties** from the menu bar.

**Annotation Editor**

Provides read and write access to the annotations.

If the Annotation Editor is not visible in BEA Workshop, choose **Windows**→**Show View**→**Annotations**.

# Functions and Shortcuts

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

**Save**: Saves the file currently displayed in the Design or Source View.

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

**Build All**: **(CTRL+B)**: Build your application. This icon is not available when you select **Project**→**Build Automatically** from the Workshop menu.

**F2:** To change the label (name) on a node in your business process, click **F2** when your mouse is active on the node in the **Design** view, enter the name you want to give the node, then click **Enter** on your keyboard.

Use the up and down arrows on your keyboard to navigate up and down between the nodes in your business process.

Use the right and left arrows on your keyboard to expand and collapse a *group* of nodes.

# Related Topics

Using Keyboard Shortcuts

# Step 2: Specify How the Process is Started

In this step, you specify 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 business process via the controls interface. You learn more about Process controls in Part II, "Call a Business Process Using a Process Control" of this tutorial.

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

In the **Design View**, the interactions between a business process and a client application 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 on this node to handle the receipt of a message from a client.

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

- To Create a Start Node in Your Business Process
- To Design Your Client Request Node

**To Create a Start Node in Your Business Process**

1. On the **Package Explorer** pane, double-click `RequestQuote.java`. Your **RequestQuote** business process is displayed in the **Design** view.
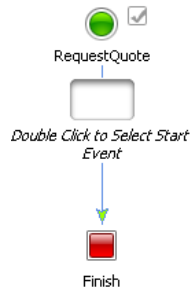
You must add a node to this **Start** node to define the start method for your business process.

2. Double-click the **Start** node to display the Start node builder.

   The node builder displays with the possible start methods.



3. In the node builder, select **Invoked via a Client Request**.

4. Click **Close**. The node builder closes and the empty node that was associated with the **Start** node is now populated with a **Client Request** node.

### To Design Your Client Request Node

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

1. Rename the **Client Request** node. To do so, click the **Client Request** node and press **F2**. Enter `Client Requests Quote` as the name to replace **Client Request** for the node. Press **Enter**. Your business process should now resemble the following figure:

2. In **Design View**, double-click the **Client Requests Quote** node. The node builder is invoked.
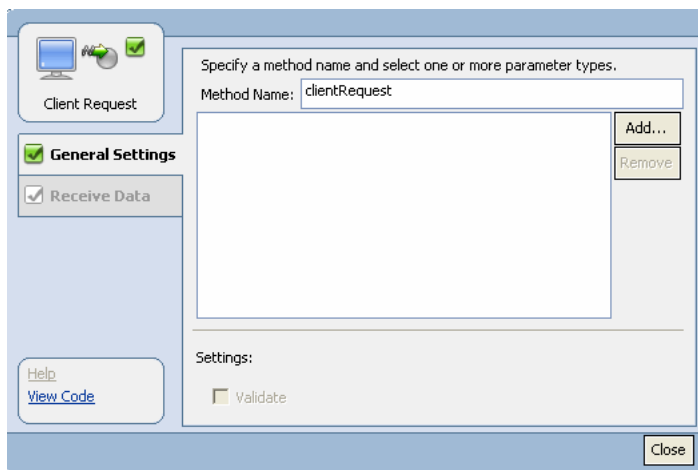


**Note:** Node builders provide a task-driven user interface that helps you design the communication between a business process and its clients and other resources. To access the node builder for any node, double-click the node in the **Design** view—a node builder specific for the node you selected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

As shown in the preceding figure, the node builder for a **Client Request** node displays the following tabs to guide your design of the communication between a client and the business process: **General Settings** and **Receive Data**.

- To Specify General Settings
- To Specify Receive Data

### To Specify General Settings
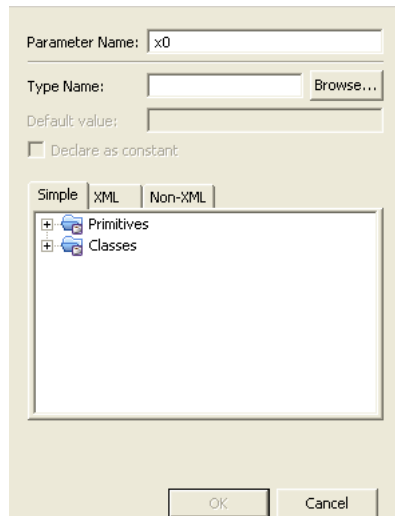
The following steps describe how to specify the method exposed by your business process to clients—clients invoke this method to start and make requests on your business process.

1. In the **Method Name** field on the **General Settings** tab, change the default method name from **clientRequest** to **quoteRequest**.

   **Note:**  When you make your business process available as a service, the name you assign to a method on a **Client Request** node is the name of the method that is exposed via the Web Services Description Language (WSDL). In general, it is recommended that you define a name that is representative of the service offered by your business process.

2. Specify a data type for the parameter to your **quoteRequest** method:

   a.  Click **Add** on the **General Settings** tab. A panel, which shows the data types is displayed:



   The Request for Quote message from clients is an XML message. Therefore, we are concerned with **XML Types** at this node.

   b.  If necessary, select **XML**. The panel is populated with a list of XML Schema files (*Typed XML*) and a list of *Untyped* XML objects available in your project.

   **Note:**  The XML Schemas you need as you build the Quote Request business process in this tutorial are provided in the `Tutorial_Process_Application_UTILITY\Schemas` folder. The Schemas provided include `QuoteRequest.xsd`, `PriceQuote.xsd`, `AvailQuote.xsd`, `Quote.xsd` and a system Schema: `DynamicProperties.xsd`.

For XML Schemas to be available to the services in your application, they must be located in a project. Projects (Web, or Utility) are represented in the **Package Explorer** pane as folders. To learn about creating projects in your applications and importing XML Schemas, including system Schemas, into your application, see Importing Files into the Project.

In this step, we use an XML Schema, specifically **QuoteRequest.xsd**, to specify the structure of documents that clients can send to start your business process.

c. In the list of **XML Types**, and expand upto **QuoteRequest.xsd** level, and click the + associated with **QuoteRequest.xsd**.

A graphical representation of the XML Schema defined by QuoteRequest.xsd is displayed in the node builder pane.

d. Click the **quoteRequest** node. (It represents the parent element in your XML document.) The **Type** field is populated with the XML type:
org.example.request.QuoteRequestDocument.

**Figure 4-1  Quote Request**
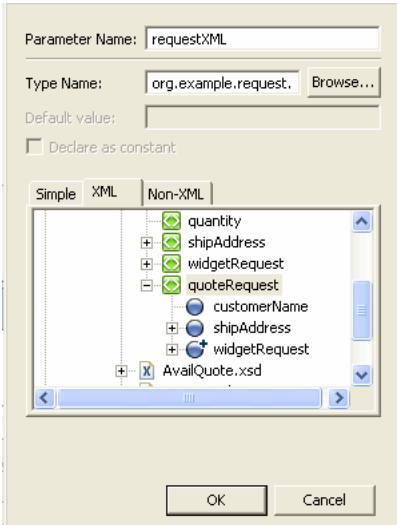


e. In the **Parameter Name** field, replace the default parameter name (**x0**) with **requestXML**.

3. Click **OK**. The parameter specifications you made (parameter type is **QuoteRequestDocument**, parameter name is **requestXML**) is displayed in **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 that is valid against an XML Schema (in this case, `QuoteRequest.xsd`).

**Note:** Example XML messages (`QuoteRequest.xml` and `QuoteRequest_a.xml`) that can be received from a client are provided in the **testxml** folder in your project. You use them later in the tutorial to test your business process.

The General Settings tab is updated to indicate that you successfully completed the specification of a method name and parameters: ✅ indicates that a task is complete; ☑ indicates that a task is not complete.



## To Specify Receive Data

1. Click the **Receive Data** tab, which allows you to specify a variable that receives a Request for Quote message from a client that is assigned at run time. By default, the **Receive Data** tab opens on the **Variable Assignment** panel.

   **Note:** **Receive Data** tabs have two modes:

   – **Variable Assignment**—Use this mode when you want to assign the data received from the client to a variable of the same data type.

   – **Transformation**—Use this mode when you want to create a transformation between the data assigned to a variable and that expected by the method parameter.

   **Note:** Note that 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 of 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 this case, 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 Request for Quote from clients.

2. Under **Select variables to assign**, click the arrow and select **Create new variable...**
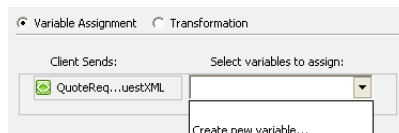
The **Create Variable** dialog box is displayed.

3. In the **Create Variable** dialog box:

   a. In the **Variable Name** field, enter **requestXML**.

   b. In the **Select Variable Type** field, in the list of **XML Types**, select the **quoteRequest** element under **QuoteRequest.xsd**.
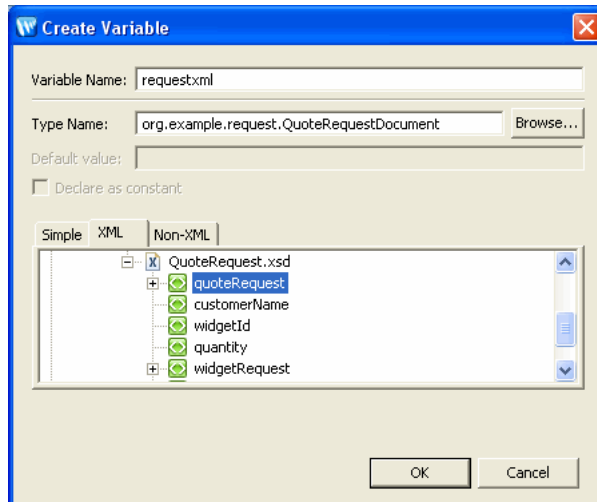


   The **Variable Type** field is populated with **org.example.request.QuoteRequestDocument**.

   c. Click **OK**. Your new variable is created and displayed in the **Receive Data** tab.
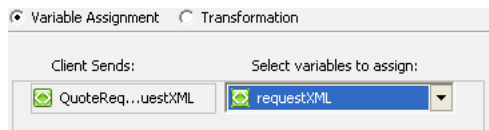


   **Note:**   The **requestXML** variable is also listed as an **XML** variable in **Outline**.
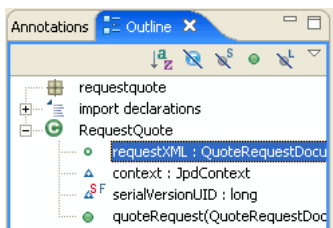


   Both tabs in the node builder (**General Settings** and **Receive Data**) are marked complete ✔.

4. Click **Close**. The **Client Requests Quote** node builder closes.

   In **Design** view, note that the completeness icon associated with the **Client Requests Quote** node changed from ☑ to ☑ indicating that the design of the node is complete.
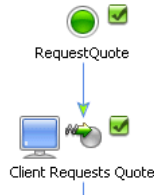


5. From the Workshop menu, select **File→Save All**.

# Related Topics

Components of Your Application

Designing Start Nodes

Working With Data Types

Interacting With Resources Using Controls

Calling Business Processes

# Step 3: Define Conditions for Alternate Paths of Execution

This step describes how you design a common pattern in business processes—one that selects one path of execution based on the evaluation of one or more conditions. You create this pattern by designing a **Decision** node in your business process.

In this part of the tutorial scenario, the business process is designed to make a decision based on a value that the business process extracts from the variable to which the XML message from the client is assigned. You design a single condition, which is evaluated at run time to determine whether the shipping address, specified in the incoming Request for Quote xml, requires that sales tax is 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 calls a Web service to calculate the sales tax. If the condition evaluates to *false*, then no sales tax is required for the quote and the flow of execution proceeds along the default branch. This step includes the following tasks:

- To Add A Decision Node To Your Business Process

- To Define a Condition in This Decision Node

### To Add A Decision Node To Your Business Process

1. If the **Node Palette** is not visible in BEA Workshop, choose **Windows→Show View→Node Palette** from the BEA Workshop menu. From the **Select Perspective** dialog box, select **Process**. The **Node Palette** is now visible.

2. Click  **Decision** in the **Node Palette**, then drag and drop the **Decision** node onto the business process, positioning it directly below the **Client Requests Quote** node that you created in Step 2: Specify How the Process is Started.

**Note:** As you drag a node from the **Node Palette** onto the **Design** view, targets ⭕ appear on your business process. As you drag the node near a target location, the target is activated ⭕ and the cursor changes to an arrow 👆. 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.

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

3. Relabel **Decision**, **Condition**, and **Default** to identify the business tasks for this node more clearly:

   a. In the node's **Name** box, place **Decision**, with **Sales Tax Calculation Needed?**, then press **Enter**.

      **Note:** If the **Name** box is not open, double-click **Decision** to open it.

   b. To enter a label to replace **Condition** and identify the *true* path, double-click **Condition** and enter **Yes**, then press **Enter**.

   c. To enter a label to replace **Default** and identify the *false* path, double-click **Default** and enter **No**, then press **Enter**.

   The Decision node in your business process should now appear in the **Design** view as shown in the following figure.
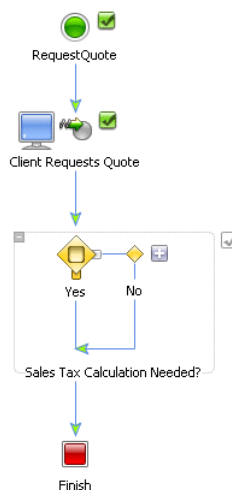
### To Define a Condition in This Decision Node

1. Double-click the condition node ⬦ to invoke the decision builder. It provides a task-driven user interface that helps you design the decision logic.
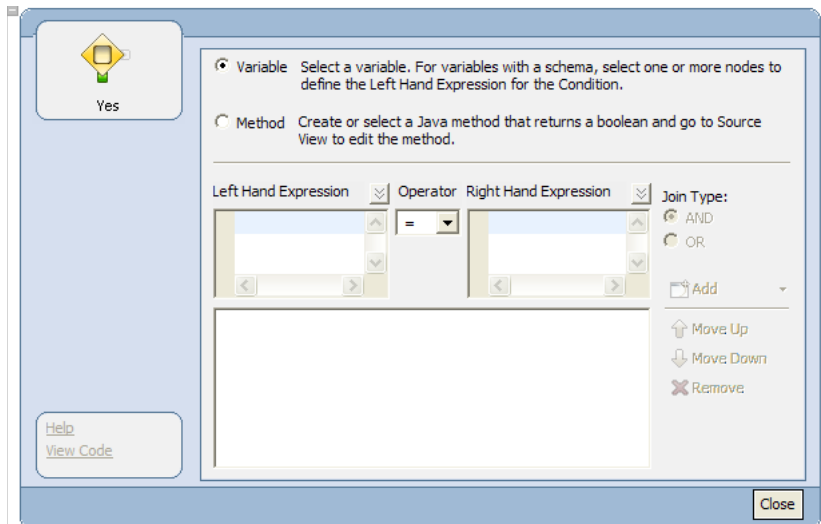


In the decision builder, **Variable** is selected by default. Do not change the selection because, in this case, you design the decision based on the value of an element in an XML document, which is valid against an XML Schema.

2. Select an XML element on which the decision is made. To do so, complete the following steps:

   a. In the decision builder, select a variable by clicking the ⋁ for the **Left Hand Expression**.

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

   A representation of the XML schema for the **QuoteRequest** is displayed in the **Select Expression Node** pane:

The elements and attributes of an XML document assigned to this variable are represented as nodes in a hierarchical representation, as shown in the preceding figure. The schema in our example (QuoteRequest.xsd) specifies a root element (quoteRequest), and child

elements: customerName, shipAddress, and a repeating element (identified by ⊕⁺): widgetRequest. The shipAddress element contains the following attributes: street, city, state, zip.

b. In the **Select Expression Node** panel, click the **state** attribute.

This selects the node in the XML document that represents the element for which you want to define the condition.

The **Selected Expression** field is populated with the following expression:

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

c. Click **Select**.

The **Left Hand Expression** field is populated with expression.

d. If necessary, select the **=** operator from the **Operator** list.

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

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

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

This completes the design of the first condition on this node.

g. Select the expression in the condition list pane, as shown in the following figure:
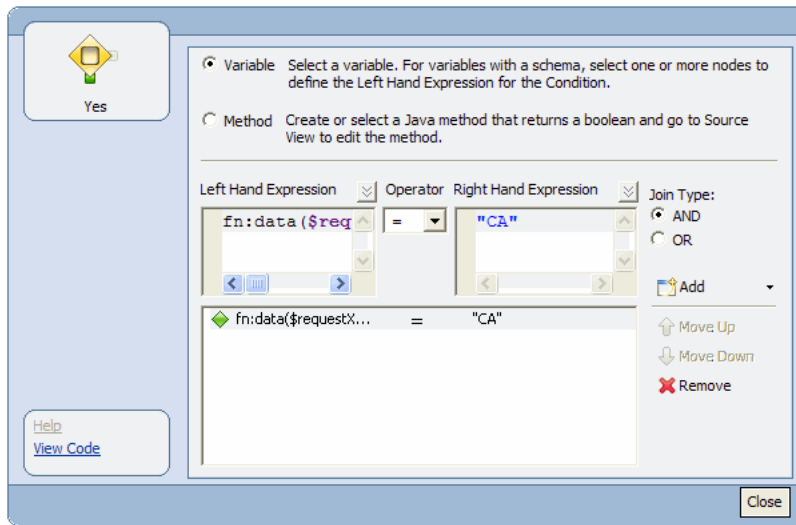
h. Change the **Join Type** option to **OR**.

i. In the **Right Hand Expression** field, select "CA", then change the entry to "**California"**.

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

k. Select the arrow beside the **Update** button, then select **Add** from the menu.

l. Repeat the process of selecting the expression and then adding it to the condition list changing the entry in the **Right Hand Expression** field to **NJ** and **New Jersey** consecutively.

The conditions you specify are listed in the condition list pane, as shown in the following figure.
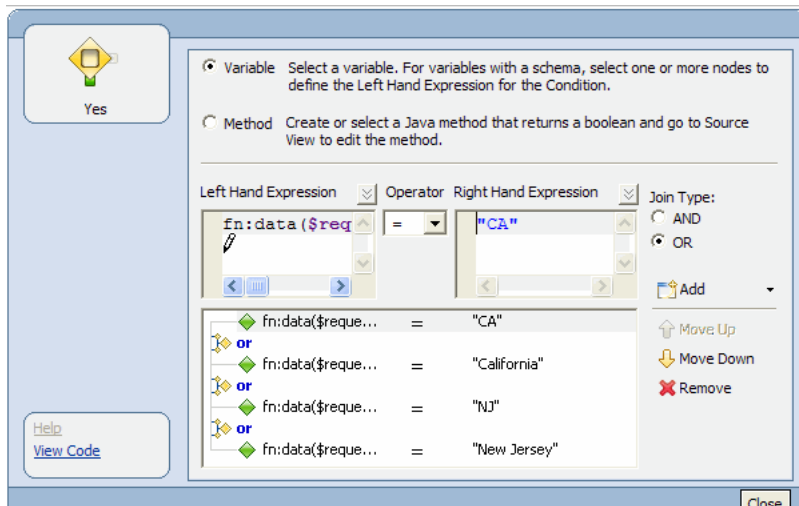
3. Click **Close** to close the decision builder.

   The icon for the **Condition** node in the **Design** view has changed from ⬦ to ⬦. It is a visual reminder that the condition you defined on this node is based on the evaluation of XML.

   This step completes the design of the condition that is evaluated when the flow transitions to the Decision node at run time. Your condition logic is written in source code as an XQuery expression—see the following section: XML Conditions in the Source Code.

   You are ready to define the actions on the subsequent paths in the flow—proceed to Step 4: Invoke a Web Service.

### XML Conditions in the Source Code

As you define your XML conditions in the decision builder, BEA Workshop writes an XQuery expression to the JPD file. Specifically, XQuery expressions are written in the `Process Language` region of the JPD file.

To view the XQuery expression written in keeping with your work in the preceding section, click the **Source** tab.

The condition defined by following the example in steps 2 through 9 in the preceding section creates the following XQuery expression in the source code:

```
@com.bea.wli.common.XQuery(prolog=
        "declare namespace ns0=\"http://www.example.org/request\";" +
```

```
  "declare function cond_requestXML_1($requestXML as element()) as
   xs:boolean  {" +

  " (((data($requestXML/ns0:shipAddress/@state) = \"CA\") or
      (data($requestXML/ns0:shipAddress/@state) = \"California\"))
       or (data($requestXML/ns0:shipAddress/@state) = \"NJ\")) or
 (data($requestXML/ns0:shipAddress/@state) = \"New Jersey\")" + "};" +
  "declare function get_requestXML1($requestXML as element())
       as element()* { " +

       "$requestXML/ns0:widgetRequest" +

"};",
```

## Related Topics

[Defining Conditions for Branching](#)

# Step 4: Invoke a Web Service

By default, a **Decision** node consists of one condition; a path below the condition node, which represents the path of execution followed when the condition, or set of conditions that evaluate to *true*; and a path to the right of the condition, which represents the path of execution followed when the condition evaluates to *false* (the default path).

**Note:** You can add additional condition nodes and paths to a **Decision** node, but in this scenario, we need only one set of conditions, and two paths.

In this step, you learn how to add logic to one path of execution for your **Decision** node (**Sales Tax Calculation Needed?**). Specifically, you learn how to design your business process to interact with resources via controls. Your business process invokes a Web service and handles the data returned from the Web service. This step describes the following topics:

- What is the Tax Calculation Control?
- Design the Interaction Between Your Business Process and a Web Service

## What is the Tax Calculation Control?

Java Controls are server-side components managed by the Workshop framework. They encapsulate external resources and business logic for use in Workshop applications. In other words, controls 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 add an instance of a control to your project and then invoke its methods.

In this scenario, the business process calls a Web service, which calculates and returns a sales tax rate. Business Processes invoke Web services via Web Service controls. The Web service control (`TaxCalcControl.java`) is created for you and included in your application's project (specifically in the `Tutorial_Process_Application_WEB\requestquote\services` folder, where *myapplications* represents the location in which you created your tutorial application).

A complete description of how to create the `TaxCalc.java` Web service and its associated control (`TaxCalcControl.java`) is beyond the scope of this tutorial. The goal of Step 4 in this tutorial is to describe how to create the appropriate nodes in your business process, and design their communication with this Web Service control.

To learn about creating Web services, and creating a control from your Web service, see Tutorial: Web Services and Controls and Transactions.

## Related Topics

Tutorial: Web Services

Buffering Methods and Callbacks

Transaction Boundaries

# Design the Interaction Between Your Business Process and a Web Service

This section describes how to create the activities that are performed when the condition defined in your Decision node evaluates to true. The condition evaluates to true if the value of `shipAddress/state` in the XML document received from a client, equals any one of the following: CA, California, NJ, or New Jersey.

In this section, you learn how to invoke a Web service from your business process, and create a callback handler to receive the data returned by the Web service. It includes the following tasks:

- To Create an Instance of the Web Service Control in Your Project

- To Call the Tax Calculation Web Service From Your Business Process

- To Receive the Tax Rate Calculation From the Web Service

### To Create an Instance of the Web Service Control in Your Project

1. If **Design** view is not visible, click the **Design** tab.

2. From the BEA Workshop menu choose **Windows→Show View→Data Palette**.

3. Click ▽ on the **Data Palette Controls** tab. A drop-down list of controls that represent the resources with which your business process can interact is displayed.

4. Select **Local** control, select **TaxCalcControl - requestQuote.services**.

5. Enter the Field Name in the **Insert Control:TaxCalcControl - requestQuote.services** dialog box.

Note:    You can also drag and drop the `TaxCalcControl.java` from the **Package Explore**r pane to the **Data Palette**. The file is available at
`Tutorial_Process_application_WEB/src/requestquote.services/TaxCalcContro`
`l.java`.

### To Call the Tax Calculation Web Service From Your Business Process

In this step, you create the logic to call the tax calculation control from your business process.

1. In the **Data Palette**, click the + beside the **taxCalcControl**. The list of methods available on the **taxCalcControl** is displayed.

2. From the list of **taxCalcControl** methods, click the following method:

   `void requestTaxRate(String stateID_arg)`

3. Drag and drop the method onto the business process, placing it on the **Sales Tax Calculation Needed?** node immediately below the condition (Yes) node
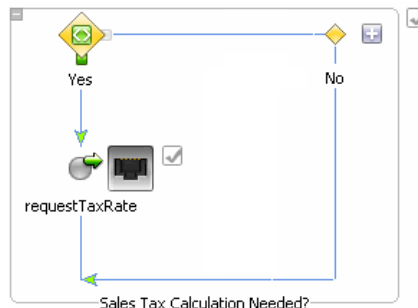


A Control Send node is created representing the asynchronous call to your **taxCalculation** Web Service control. The node is named according to the name of the method you dragged onto the business process—in this case: **requestTaxRate**.

Note:    This interaction is designed to be asynchronous, meaning that the business process can send a request to the **taxCalcControl** from this node, and does not block waiting for a response from the control. In other words, the business process can continue processing and receive a response from the **taxCalcControl** service when that service completes the request.

4. 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)**, respectively.

5. Click the **Send Data**.

   By default, the **Send Data** opens on the **Variable Assignment** pane. The **Control Expects** field is populated with the data type expected by the requestTaxRate() method exposed by the **taxCalcControl** Web services: **String stateID**.

   **Note:**  As you learned in a previous step, **Send Data** tabs have two modes:

   – **Variable Assignment**—Use this mode when you want to assign the data received from the client to a variable of the same data type.

   – **Transformation**—Use this mode when you want to create a transformation between the data assigned to a variable and that expected by the method parameter.

   In this case, you must switch to the **Transformation** mode because the data type required as input to the **taxCalcControl** control is a Java String type, and the variable in which the Request for Quote message (which includes the value of shipAddress/state) is stored, is of type XML (that is, QuoteRequestDocument, which is valid against an XML Schema).

   BEA Products provides a data mapping tool to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation files. You can think of transformation files as another resource with which your business process interacts via controls. That is, when transformation files containing your data transformations are built, they are built as controls. The controls expose transformation methods, which business processes invoke to map disparate data types.

6. Click **Transformation**. A pane that allows you to define a transformation between your variable and the expected data type of the parameter on the control method.

7. Click **Select Variable** to display the variables in your project, then choose requestXML (QuoteRequestDocument)—that is, the variable you created for the **Client Request** node at the start of your business process. Click **Next**.
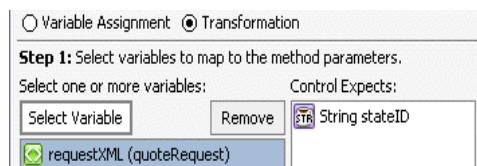


8. In the next dialog box, click **Create Transformation**. The Transformation tool opens, which displays a representation of the QuoteRequest XML document in the **Source** pane, and a **String** in the **Target** pane.

9. Click **state** in the **Source** pane and drag your mouse pointer over to **String** in the **Target** pane. A line is drawn between the **state** and **String** elements in the **XML Map** pane. It represents the transformation between the two data types.
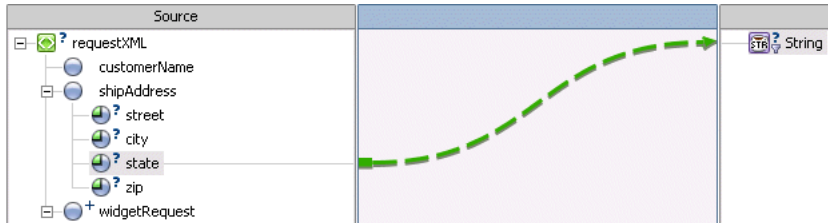


10. Save the .xq file before switching to the process, and in the **Package Explorer** pane, and click the **RequestQuote.java** tab.

> **Note:** Creating the transformation in the preceding steps creates a Transformation control in your project: A Java file, named `RequestQuoteTransformation.java` is created. An XQ file, which contains the query (written in the XQuery language) for the transformation method is also created. Both the Java and XQ files are displayed in the **Package Explorer** tab. Also, an instance of the Transformation control was created and is represented as   **transformations** in the **Data Palette** (**Controls** folder).

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

    This step completes the design of the **Request Tax Rate** node.

### To Receive the Tax Rate Calculation From the 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 (via a control). To add the logic in your business process that receives the tax rate returned by the tax calculation control, complete the following steps:

1. In the **Data Palette**, if needed, click the + beside the **taxCalcControl** to expand the list of methods available on the **taxCalcControl**.

2. From the list of **taxCalcControl** methods, click the following method:

   ```
   void returnTaxRate(float taxRate)
   ```

3. Drag and drop the method onto the business process placing it on the **Sales Tax Calculation Needed?** node immediately below the **requestTaxRate** node:

A Control Receive node is created representing the asynchronous response from your Web Service control.
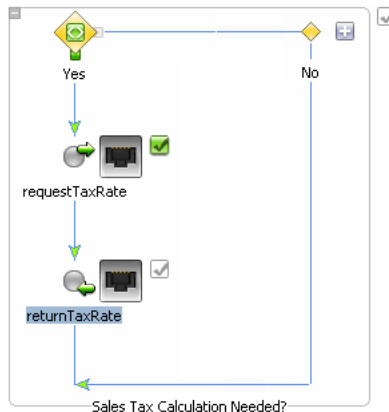


The node is named according to the name of the method you dragged onto the business process—in this case: **returnTaxRate**.

4. Double-click the **returnTaxRate** node. The node builder opens on the **General Settings** tab. The Control instance and target methods are already selected: **taxCalculation** and **returnTaxRate(float taxRate)**, respectively.

5. Click the **Receive Data** tab. The tab opens with the **Variable Assignment** pane selected.

   The **Control Expects** field is populated with the data type and name of the parameter returned by the `returnTaxRate()` method on the **taxCalculation** control: `float taxRate`.

6. In the **Variable Assignment** pane, click the arrow in the field under **Select variables to assign**, then select **Create new variable...**. The **Create Variable** dialog box is displayed.

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

8. From the **Simple** tab, expand **Primitive**, and then select **float**.

9. In the **Default value** field, enter **0**. This initializes the value of **taxRate** to zero.
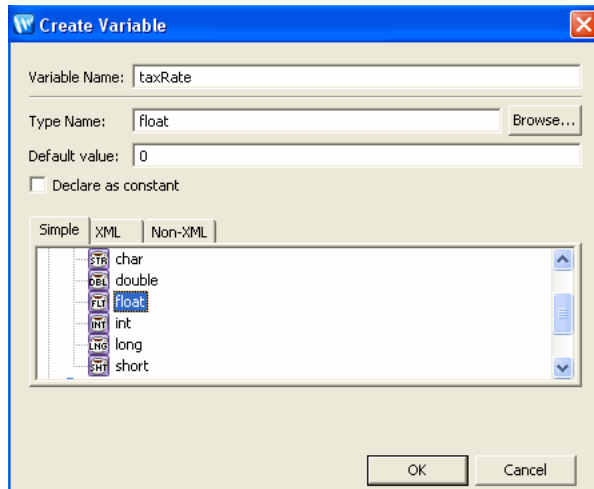
10. Click **OK**. Your new variable, to which the sales tax rate is assigned at run time, is created and is listed as a **Java** variable in the **Variables** tab.

11. Click **Close** in the node builder. The Control Receive node builder closes.

This step completes the design of your **returnTaxRate** node and the design of the activities performed by your business process when the condition on the **Decision** node evaluates to true. In the **Design** view, your business process resembles that shown in the following figure:
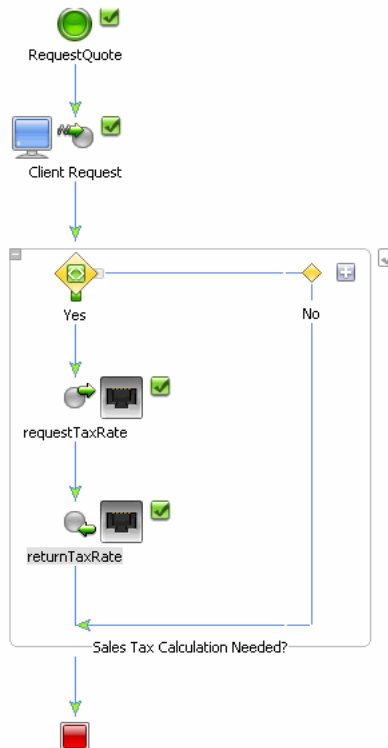
Note that the Start node icon changed from ⚫ to ⚫ after you added the asynchronous call to the Web Service control. The former icon indicates that your business process is stateless, and the latter indicates that it is under state.

The icons reflect the specification for the **stateless** property for your business process. To see whether the **stateless** property is defined as **true** or **false**, click the Start node icon ⚫ and view the **Property Editor**. To learn about stateful and stateless business processes, see Building Stateless and Stateful Business Processes. To understand why the property changed from stateless to stateful, see Transaction Boundaries.

**Note:** If the **Property Editor** is not displayed in BEA Workshop, choose **Windows→ShowView→Property Editor** from menu bar.

12. From the Workshop menu, select **File→Save All**.

**Note:** No further design is required for this Decision node. If the condition evaluates to true, the path of execution proceeds via the **Yes** path and the tax rate for the order is calculated. If the

condition evaluates to false—no sales tax is required—the path of execution proceeds via the **No** path and a value of zero is assigned to the variable **taxRate**. Remember, you specified that **taxRate** is initialized to zero when you designed the **taxRate** variable in the preceding section.

# Related Topics

Interacting With Resources Using Controls

Creating and Testing Maps

*Guide to Data Transformation*

Building Stateless and Stateful Business Processes

# Step 5: Run Your Business Process

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

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,** select and right-click on **RequestQuote.java**, click **Run As**, and click **Run On Server**.

3.  In the Define a New Server dialog box, select either a **Choose an existing server** option or **Manually define a server** (if there is no server defined), and click **Next**.

4.  In the **BEA WebLogic v9.2 Server** dialog box, to manually define a server, click **Browse**, and select the samples integration domain directory from the product installation directory available at `BEA_HOME\weblogic92\samples\domains\integration`, where BEA_HOME represents the directory in which you installed WebLogic Platform. Click **Finish**.

    The samples domain integration server is started, and the RequestQuote application is deployed on it. When WebLogic Server is running, the following indicator is visible in the **Servers** view:

    Status
    ⊞ Started

5.  After the application is deployed, the Test Browser is displayed.

6.  Click the **Test Form** tab. Open the file browser by clicking **Browse** beside the **xml requestXML (file value)** field.

7.  Select **RequestQuote.xml** from the **requestquote\testxml** folder in your project.

The **QuoteRequest.xml** file is available at the following location in your file system:

```
Tutorial_Process_Application_WEB\src\testxml\QuoteRequest.xml
```

8. Click the **Test Form** tab. In the **Test Form** page, click the button labeled with the method name on your business process (**quoteRequest**) to start the business process.



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



9. Click **Refresh** on the Message Log to refresh the entries in the log until this instance of the business process completes running. Entries in the Message Log correspond to the methods on your business process:

   – The **quoteRequest** method that starts the business process.

   – A call from your business process to the **taxCalculation** Web service:
   **taxCalculation.requestTaxRate**

   – A response from the service to your business process: **taxCalculation.returnTaxRate**

   – The **Instance ID**—When the business process finishes, a message similar to the following is displayed in the Message Log:

   ```
   Instance instanceID is Completed.
   ```

   where *instanceID* represents the ID 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 **taxCalculation.returnTaxRate**, you can view the response from the **taxCalculation** service—in this case, the tax rate was calculated, based on the input value (**NJ)** for the **state** element in the test XML.



In the sample XML message you used, `state="NJ"`. That is, the state to which the order is shipped is `NJ`. This XML message is designed to cause the flow of execution through the **Yes** branch on your **Sales Tax Calculation Needed?** node. The preceding figure 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, via a control.

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

### Note About Additional Functionality in the 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. The interactive process graph requires Adobe SVG Viewer Version 3.0. The first time you open the **Process Graph** tab, you will be asked if you would like to download the Viewer from the Adobe Web site. You can also download the viewer directly from the Adobe Web site at
`http://www.adobe.com/svg/viewer/install/main.html`.

**Note:** This viewer is not available for some configurations that WebLogic Platform 9.2 supports. For details, please see "Browser Requirements for the Interactive Graph" in Process Instance Monitoring at `http://edocs.bea.com/wli/docs92/adminhelp/processmonitoring.html` in *WebLogic Integration Admin Console Online Help*. For detailed information about the operating systems and browsers WebLogic Platform supports, see WebLogic Platform Supported Configurations at `http://edocs.bea.com/platform/suppconfigs/index.html`.

As previously mentioned, the Process Graph is a graphical representation of your business process and its execution path. The Process Graph highlights the node currently being executed. When the instance of the business process completes, the path of execution followed in your test is highlighted. In this scenario, the **Yes** path is executed—the **No** path is gray on the Process Graph to indicate that this path was not taken during the execution of this instance of the business process.

**Note:** Press **Alt** and drag the mouse pointer over the Process Graph to move and position it on the Test Browser page. To zoom in, press **Ctrl+click**; to zoom out, press **Ctrl+Shift+click**. Alternately, right-click on the Process Graph and select the **Zoom In** or **Zoom Out** command from the drop-down menu.

You will review your running business process in the **Process Graph** in a later step in the tutorial.

**Note:** Use the back and forward arrows ← → to navigate between the pages in the BEA Workshop Test Browser.

**Monitor**

Click **Monitor** to open the BEA Products Administration Console in a Web Browser. Login using username = `weblogic` and password = `weblogic`. The BEA Products Administration Console opens to the **Process Instance Details** page. The WebLogic Integration Administration Console allows you to administer and manage your BEA Products applications. For example, if you click **View Statistics** on the Process Instances navigation pane, you access a **Process Instance Statistics** page. This page displays a summary of business process instances grouped by the process type. To view the instances of a process type that ran or are running on your server, click the process name. Processes instances are identified by their *instanceID*. Note that the instanceID displayed for your **RequestQuote** business process matches the instanceID displayed on the Message Log pane (see the preceding figures in this topic).

**Monitor all RequestQuote.jpd processes**

Click **Monitor all RequestQuote.jpd processes** at the top of the **Test Form** to open the BEA Products Administration Console. Login using username = `weblogic` and password =

`weblogic`. When you use this link to open the Administration Console, it opens on the **Process Instance Summary** page, which displays a summary of all the instances of business processes that ran or are running.

## Related Topics

*Managing WebLogic Integration Solutions* at
`http://edocs.bea.com/wli/docs92/manage/index.html`

# Part II   Call a Business Process Using a Process Control

Part II of the tutorial demonstrates how simple it is to interchange calls to different external resources in your business process.

You learn how to design a call to another business process from your Request for Quote process. Specifically, you create a new Process control and change the asynchronous call to the Web Service you designed in Part I, making it instead an asynchronous call to another business process, via the new Process control.

Proceed to Chapter 8, "Step 6: Invoke a Business Process Using a Process Control," to complete Part II of the tutorial.

# Step 6: Invoke a Business Process Using a Process Control

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.

For this part of the tutorial, we are going to change the design of the business process you created in Part I to take advantage of a tax calculation service provided by a business process instead of using the tax calculation Web service you initially used. You can do so by first creating a Process control from the tax calculation business process. Then you simply change the Control nodes you designed in Part I in such a way that, instead of communicating with the tax calculation *Web service* via the Web Service control, they communicate with the tax calculation *business process*, via the new Process control.

The tasks in this step include:

- To Create a Process Control

- To Change the Control Send Node in the Request Quote Business Process to Interact With the Process Control

- To Change the Control Receive Node in the Request Quote Business Process to Interact With the Process Control

- To Test the Request Quote Process and its Call to the Tax Calculation Process

### To Create a Process Control

The tutorial application provides you with a simple business process (`TaxCalcProcess.java`) that calculates the sales tax for a Request for Quote. (See
`\Tutorial_Process_ApplicationWeb\requestquote\services\TaxCalcProcess.java` in the

**Package Explorer**.) In this step, you learn how to create a Process control for the
`TaxCalcProcess.java` business process.

**Note:** If the **Data Palette** is not visible in WebLogic Workshop, choose
**Windows→Show View→Data Palette** from the menu bar. Instances of controls already
available in your project are displayed in the **Data Palette** tab in the **Controls** folder.

1. 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.

2. Point to **Integration Controls**, then select **Process** to view the **Insert Control:Process** dialog
   box.

3. Enter **taxCalcProcess** as the name for the instance of the Process control you are about to create
   in the **Field Name** field, and click **Next**.

4. In the **Create Control** dialog box, enter **TaxCalcProcess** in the **Name** field, and click **Next**.

5. In the **Insert Control - Process** dialog box, click **Browse** beside the **Process** field, then choose
   **TaxCalcProcess.java** from the
   `/Tutorial_Process_Application_WEB/src/requestquote/services` folder.

   The **Start Method** field is populated with **requestTaxRate**, which is the start method for
   **TaxCalcProcess.java**.
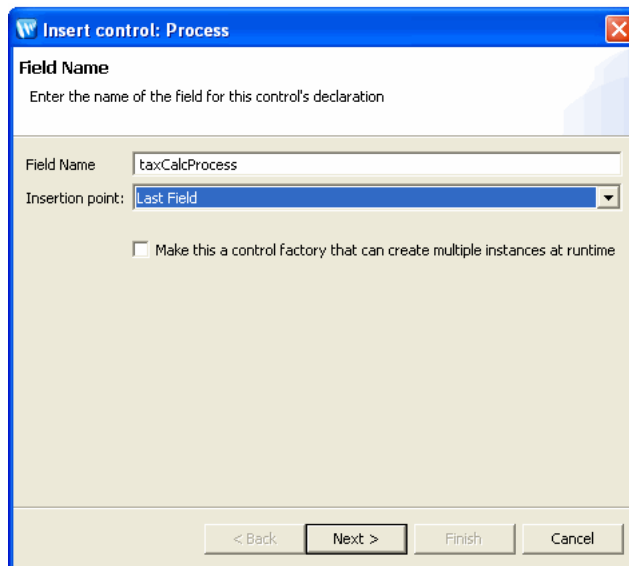
6. Click **Finish**. The Process control (**TaxCalcProcess.java**) is created and displayed in the **Applications** tab. Also, an instance of the control (**taxCalcProcess**) is added to the **Data Palette**. The **Controls** tab on the **Data Palette** should now resemble the following figure:
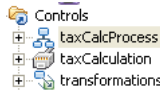


**To Change the Control Send Node in the Request Quote Business Process to Interact With the Process Control**

1. In the **Data Palette**, click + beside **taxCalcProcess** under the **Controls** folder to expand the list of methods on the control.

2. In **Design** view, select the following method:

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

3. Drag and drop the method onto the **requestTaxRate** node in your `RequestQuote.java`. The following message is displayed:

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

4. Click **OK**. The **requestTaxRate** node changes to reflect the change in the type of control with which it is associated. The node representation changes from:



5. Double-click the **requestTaxRate** node to open its node builder on the **General Settings** pane.

6. Confirm that **taxCalcProcess** is selected in the **Control** field and that the following method is selected in the **Method** field:

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

7. Click the **Send Data** tab to open the second pane in the node builder. The **Variable Assignment** option is selected by default, and the **Control Expects** field is populated with **QuoteRequestDocument** to indicate the format and type of the message expected by the tax calculation process.

   **Note:** The tax calculation process expects to receive 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.

8. Click the arrow in the **Select variables to assign** field, and select **requestXML(QuoteRequest)**.
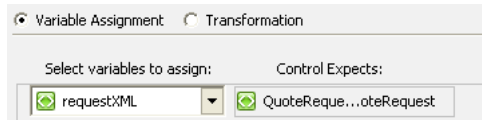


9. Click **Close** to close the node builder.

This step completes the procedure to remove the call from your Request for Quote business process to a tax calculation Web Service—changing it to a call to a tax calculation business process (via the Process control you created).

### To Change the Control Receive Node in the Request Quote Business Process to Interact With the Process Control

1. In **Design** view, from the **Data Palette**, select the following method on the **taxCalcProcess**:

```
void returnTaxRate(float salesTaxRate)
```

2. Drag and drop the method onto the **returnTaxRate** node in your RequestQuote.java.

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 change in the type of control with which it is associated. The node representation changes from.



4. Double-click the **returnTaxRate** node to open its node builder on the **General Settings** pane.

5. 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. Click the **Receive Data** tab to open the second panel in the node builder. The **Variable Assignment** option is selected by default, and the **Control Returns** field is populated with **float salesTaxRate** to indicate the type and name of the parameter expected to be returned by the tax calculation process.

7. Click the arrow in the **Select variables to assign** field, and select **taxRate (float)**.
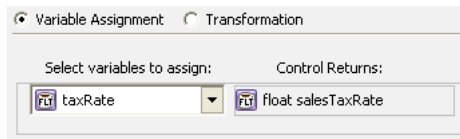


8. Click **Close** to close the node builder.

   This step completes the procedure to remove the callback handler that receives a message from a tax calculation Web Service—changing it to a callback handler that receives a message from a tax calculation business process (via the Process control you created).

9. From the Workshop menu, select **File→Save All**.

### To Test the Request Quote Process and its Call to the Tax Calculation Process

You can run and test the business process, which now contains an asynchronous call to another business process (via the Process control) in the same way as you tested the business process you created in Part I. To do so, follow steps 1 through 7, as described in Chapter , "Step 5: Run Your 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.
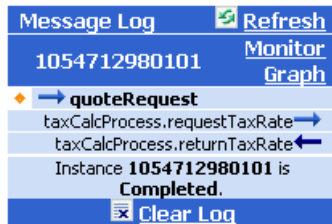


Entries in the Message Log correspond to the methods on your business process:

- The **quoteRequest** method that starts the business process.

- A call from your **RequestQuote** business process to the **taxCalcProcess** business process: **taxCalcProcess.requestTaxRate**. Note that in this case, the entire *Request for Quote* XML document (contained in the **requestXML** variable) is passed to the subprocess. This is different to the case in which your business process called the tax calculation Web service (Part I)—in that case, only the state field from the *Request for Quote* XML document 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 in the business process you created in Part I, the Process control returns the raw float value (0.08).

- The **Instance ID**—When the business process finishes, a message similar to the following is displayed in the Message Log:

```
Instance instanceID is Completed.
```

where *instanceID* represents the ID generated when the **quoteRequest** method in your business process was called.

### Note About Additional Functionality in the 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. The interactive process graph requires Adobe SVG Viewer Version 3.0. The first time you open the **Process Graph** tab, you will be asked if you would like to download the Viewer from the Adobe Web site. You can also download the viewer directly from the Adobe Web site at `http://www.adobe.com/svg/viewer/install/main.html`.

**Note:** This viewer is not available for some configurations that WebLogic Platform 9.2 supports. For details, please see "Requirements for the Interactive Graph" in Process Instance Monitoring at

`http://edocs.bea.com/wli/docs92/adminhelp/processmonitoring.html` in *WebLogic Integration Admin Console Online Help*. For detailed information about the operating systems and browsers WebLogic Platform supports, see BEA WebLogic Platform Supported Configurations at

`http://e-docs.bea.com/platform/suppconfigs/index.html`.

As previously mentioned, the Process Graph is a graphical representation of your business process and its execution path. The Process Graph highlights the node currently being executed. When the instance of the business process completes, the path of execution followed in your test is highlighted. In this scenario, the **Yes** path is executed—the **No** path is gray on the Process Graph to indicate that this path was not taken during the execution of this instance of the business process.

**Note:** Press **Alt** and drag the mouse pointer over the Process Graph to move and position it on the Test Browser page. To zoom in, press **Ctrl+click**; to zoom out, press **Ctrl+Shift+click**. Alternately, right-click on the Process Graph and select the **Zoom In** or **Zoom Out** command from the drop-down menu.

You will review your running business process in the **Process Graph** in a later step in the tutorial.

**Note:** Use the back and forward arrows ← → to navigate between the pages in the BEA Workshop Test Browser.

**Monitor**

Click **Monitor** to open the BEA Products Administration Console in a Web Browser. Login using username = weblogic and password = weblogic. The BEA Products Administration Console opens to the **Process Instance Details** page. The WebLogic Integration Administration Console allows you to administer and manage your BEA Products applications. For example, if you click **View Statistics** on the Process Instances navigation pane, you access a **Process Instance Statistics** page. This page displays a summary of business process instances grouped by the process type. To view the instances of a process type that ran or are running on your server, click the process name. Processes instances are identified by their *instanceID*. Note that the instanceID displayed for your **RequestQuote** business process matches the instanceID displayed on the Message Log pane (see the preceding figures in this topic).

**Monitor all RequestQuote.java processes**

Click **Monitor all RequestQuote.java processes** at the top of the **Test Form** to open the BEA Products Administration Console. (Login using username = weblogic and password = weblogic.) When you use this link to open the Administration Console, it opens on the **Process Instance Summary** page, which displays a summary of all the instances of business processes that ran or are running.

To stop the **Test Browser**, you can simply close it, or return to BEA Workshop and click ![icon] on the tool bar.

This step completes Part II of the tutorial.

## Related Topics

Process Control

*Managing WebLogic Integration Solutions* at
`http://edocs.bea.com/wli/docs81/manage/index.html`

# Part III   Adding Looping Logic, Parallel Paths . . .

Part III is comprised of Steps 7 through 12. You add more complex business logic to the business process you created in Part I, "Build and Run a Simple Business Process" and Part II, "Call a Business Process Using a Process Control". You learn how to 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, and use a Client Response node to return the quote to the client that invoked the business process. The final step in Part III is to run and test the business process you built.

The steps in Part III include:

**Chapter 9, "Step 7: Looping Through Items in a List"**
Describes how to create the logic to extract a list of items from the Request for Quote document received from a client and performs a set of activities repeatedly, once for each item in the list.

**Chapter 10, "Step 8: Design Parallel Paths of Execution"**
Describes how to design your business process to execute tasks in parallel. This step also includes instructions about how to design your business process to interact with resources via controls and transform the data exchanged with those controls, as required.

**Chapter 11, "Step 9: Create Quote Document"**
Describes how to transform the price and availability data from untyped XML data to typed XML, and then combine the price and availability data, which is returned to the Request Quote business process by a number of external services, to produce a single Quote document.

**Chapter 12, "Step 10: Write Quote to File System"**
Describes how to write business process data to a log using a File control.

**Chapter 13, "Step 11: Send Quote From Business Process to Client"**

Describes how to send the final *quote* message from the business process to a client.

**Chapter 14, "Step 12: Run the Request Quote Business Process"**

Describes how to compile and test the business process you created by following the steps in Part III.

# Step 7: Looping Through Items in a List

In this step, you create the logic to extract a list of items from the Request for Quote document received from a client, and begin the work of 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 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). An *iteration* variable holds the current element being processed in the **For Each** loop, for the life of the loop.

This section includes the following topics:

- Understanding XML Schemas and For Each Nodes
- Design a For Each Loop in Your Business Process

## Understanding XML Schemas and For Each Nodes

The business process you build in this tutorial is designed to start when it receives a Request for Quote XML document from a client. The Request for Quote document must contain valid XML, that is, XML valid against an XML Schema, specifically `QuoteRequest.xsd`. The `QuoteRequest.xsd` Schema is located in your application at the following location:
`Tutorial_Process_Application_WEB\Schemas`.

In the preceding line, *myapplications* represents the location of your tutorial application.

**Note:** To make the Schemas in your project available in your business process, you must place them in a **Schemas** folder in Utility project which is currently `Tutorial_Process_Application_UTILITY`. A project is one of the types of folders project that Workshop applications can contain. The projects added to your BEA Workshop applications are represented in the BEA Workshop file hierarchy as child folders of your application folder. To learn about creating and populating projects in your BEA Products applications, see Related Topics.

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 provides Java counterparts for all built-in Schema types, and generates Java counterparts for any derived types in your Schema.

In Chapter , "Step 2: Specify How the Process is Started," you created a variable (`requestXML`) to which the Request for Quote document (which 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 Request for Quote document from clients is valid:
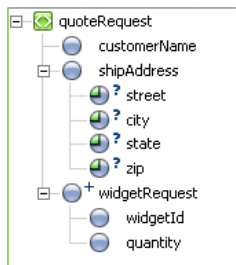


Note the following characteristics of the `QuoteRequest.xsd` Schema:

– The elements and attributes of the XML schema are represented as nodes. Note that `quoteRequest` is a root element.

– The `quoteRequest` element 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 Request for Quote documents from clients is processed. This processing begins with a **For Each** node—each iteration through the **For Each** loop processes one of a set of `widgetRequest` items.

In this section, you design the **For Each** node to first extract a list of items (the `widgetRequest` items) from the `requestXML` variable, and then to perform an activity (or set of activities) repeatedly, once for each item in the list.

## Related Topics

How Do I: Create a Schemas Folder?

How Do I: Import Files into a Schemas Folder?

# Design a For Each Loop in Your Business Process

Complete the following steps to create the logic that causes your business process to iterate over the sequence of nodes in the Request for Quote XML document:

- To Add a For Each Node to Your Business Process

- To Select a Repeating XML Element Over Which to Iterate

- To Design the Activities in Your For Each Loop

### To Add a For Each Node to Your Business Process

1. Click ![icon] **For Each** in the **Node Palette**.

2. In **Design** view, drag and drop the **For Each** node onto the RequestQuote business process placing it immediately after the **Sales Tax Calculation Needed?** (**Decision**) node.

3. Press **Enter** to name the node **For Each**.

   The **Design** view is updated to contain the **For Each** node:
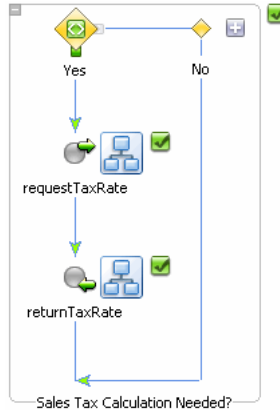
## To Select a Repeating XML Element Over Which to Iterate

1. In the **Design** view, double-click the **For Each** node to invoke its node builder.

2. In the node builder, click **Select Variable**. A drop-down list of variables (of typed XML) in your project is displayed.

3. Select **requestXML (QuoteRequestDocument)**. The **requestXML** variable contains the repeating XML element over which you want to design the iteration logic. A representation of the XML in the `requestXML` variable is displayed in the **Select Node** pane. The repeating element is identified by `⊙+`.
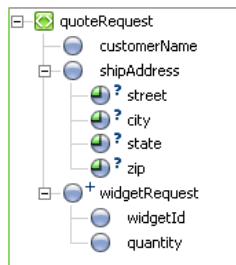


4. In the **Select Node** pane, if not already selected, click +**widgetRequest**.

   The **Repeating Element** and **Iteration Variable** fields are populated with the following data:

   – **Repeating element**—Contains the following XPath expression, which when applied against 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**. This variable is of XML type **WidgetRequestDocument**.**WidgetRequest**.

To learn how the iteration variable is used in the **For Each** loop, see To Design the Create PriceList Node.

This step completes the design of the iteration logic for your **For Each** node. Note that in the **Design** view, the node is updated graphically to reflect the work you did to define the condition:



☑ indicates that the design of the task on the node is complete, as compared to ☑, which indicates that the design is not complete. ◉ indicates that an XML query is defined on the node.

### To Design the Activities in Your For Each Loop

After you create the iteration logic in your **For Each** node, you must define the activity or set of activities performed during each iteration over the items in the list you created.

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

# 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 preceding step, you created a **For Each** loop to iterate over a set of repeating elements in a Request for Quote document. In this step, you design the activities within the **For Each** loop. That is, you design the activities that are performed for each iteration your business process makes through the loop.

When your business process interacts with multiple different systems, as is the case during the price and availability processing in this scenario, you can increase throughput in the business process by executing tasks in parallel. You add **Parallel** nodes to your business process when you want to create two or more such *parallel* branches of execution.

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 for the price calculation; one for the 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. Your business process interacts with the resources via controls. The tutorial uses two Web services: one returns the price for each `widgetID` specified in the client's request document; a second service returns availability information, based on the `widgetID` and the `quantity` specified in the request document. The controls with which your `RequestQuote` business process interacts are provided for you in your project folder: `\Tutorial_Process_ApplicationWeb\requestquote\services`. The controls are `PriceProcessorControl.java` and `AvailProcessorControl.java`.

## Related Topics

Understanding Parallel Execution in Your Business Process

# Create a Parallel Node

### To Add A Parallel Node to Your Business Process

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

2. In **Design** view, select  **Parallel** in the **Node Palette**, then drag and drop the **Parallel** node onto the business process, placing it inside the **For Each** loop.

3. Press **Enter** to name the node **Parallel**.

   The **Design** view is updated to contain a representation of the **Parallel** node as shown in the following figure:
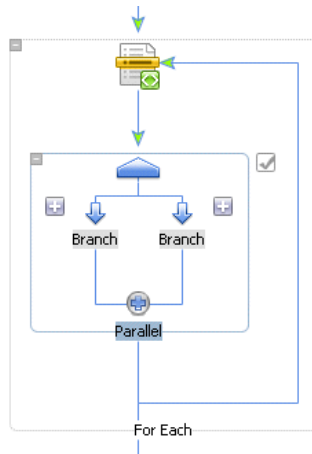


4. Change the names of the branches contained within the **Parallel** node to identify the activities that your business process executes in parallel:

   – Double-click the label on the left **Branch** and enter **Get Price**, then press **Enter**.

   – Double-click the label on the right **Branch** and enter **Get Availability**, then press **Enter**.

### Note About Join Conditions

By default, **Parallel** nodes specify an **AND** *join condition*, represented by  on the Parallel branch lines. In this case, the activities on all branches must complete before the flow of execution proceeds to the node following the parallel node.

In the case of your RequestQuote business process, because you want both branches of the **Parallel** node to complete, do not change the default **AND** join condition.

If an **OR** join condition is specified, when the activities on one branch complete the execution of activities on all other branches terminates, and the flow of execution proceeds to the node following the **Parallel** node. (The **OR** join condition is represented as ⊞ in the **Design View**.)

In **Design View**, you can view and edit the **join condition** property in the **Property Editor**. Click ⊕ or ☁ (at the top of the **Parallel** node) to display the properties of the **Parallel** node in the **Property Editor**. The **Property Editor** for your **Parallel** node should resemble that shown in the following figure:

| Property | Value |
|---|---|
| ⊟ general | |
|     name | Parallel |
|     notes | |
| ⊟ parallel | |
|     continue transaction | false |
|     join condition | AND |

# Create Logic to Assemble Price and Availability Data

In this section, you learn how to:

- Invoke the price and availability services (via controls) from the parallel branches you created.

- Design callbacks on your branches to wait for and handle the responses from the controls.

- Construct an XML document, to which the response data from controls is appended for each iteration through the **For Each** loop. (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:

- To Create Instances of the PriceProcessor and AvailProcessor Controls in Your Project

- To Add Control Nodes to Your Business Process

- To Design the Activities on the Get Price Branch

- To Design the Activities on the Get Availability Branch

### To Create Instances of the PriceProcessor and AvailProcessor Controls in Your Project

The `PriceProcessorControl.java` and `AvailProcessorControl.java` are provided for you in the **Data Palette**. The controls are available in the `Tutorial_Process_Application_WEB\src\requestquote.services` folder in the Package Explorer. The goal of this section is to describe how to create the appropriate controls in your application, and then design the communication between your business process and these controls.

1. Click  on the **Data Palette Controls** tab to display a list of controls that represent the resources with which your business process can interact.

2. Choose **Local Controls**, and select **priceProcessorControl - requestquote.Services**. The **Insert Control** dialog box is displayed.

3. Enter **priceProcessorControl** in the **Field Name** field, and click **Finish**.

4. Similarly, choose **Local Controls**, and select **availProcessorControl - requestquote.Services**.The **Insert Control** dialog box is displayed.

5. Enter **availProcessorControl** in the **Field Name** field, and click **Finish**.

   The **priceProcessorControl** and **availProcessorControl** Web Service control instances are added to the **Data Palette**:
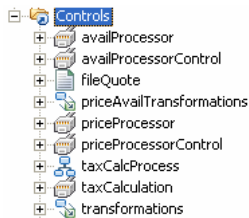


### To Add Control Nodes to Your Business Process

You learned in Parts I and II that you can create Control nodes in your business process by dragging the methods from the appropriate control on the **Data Palette** onto the business process in the **Design** view. You can also create Control nodes by selecting **Control Send**, **Control Receive**, or **Control Send with Return** from the **Node Palette** and dragging them onto the business process. You subsequently bind the appropriate methods to the control node you created. In this section you will use the latter approach.

Add the following nodes from the **Node Palette** to each branch on your **Parallel** node:

- **Control Send**

- ◾ **Control Receive**

- ◾ **Control Send with Return**

In **Design** view, select each of the listed nodes, then drag and drop the node onto the business process, placing the nodes on the **Parallel** branches until you create a **Parallel** group as shown in the following figure:
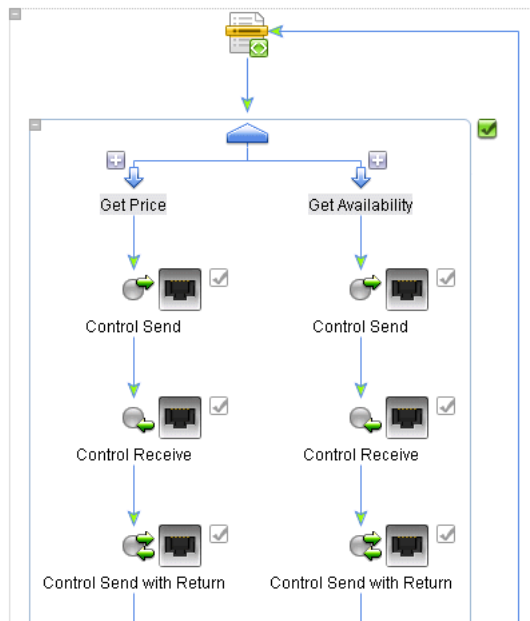


In this way, each branch is designed for the following flow of execution:

1. Call a resource (via 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.

### To Design the Activities on the Get Price Branch

1. Rename the nodes on the **Get Price** Branch (in the order in which they are executed) as follows: **Request Price**, **Receive Price**, **Create PriceList**.

2. Complete the following tasks:

- To Design the Request Price Node

- To Design the Receive Price Node

- To Design the Create PriceList Node

### To Design the Request Price Node

1. Double-click the **GetPrice** node to open its node builder. The node builder opens on the **General Settings** tab.

2. Click the arrow beside the **Control** field to display a drop down list of the instances of the controls in your project and select **priceProcessorControl**.

   The **Method** panel is populated with a list of the *asynchronous send methods* you can invoke on the **priceProcessorControl**.

3. Select the following method: `void getPrice(int itemID)`

4. Click **Send Data** to open the second tab in the node builder.

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

   **Note:** The **priceProcessor** service takes the ID of the item requested as input, and returns the price of the widget.

5. Select **Transformation** to switch modes in the **Send Data** tab.
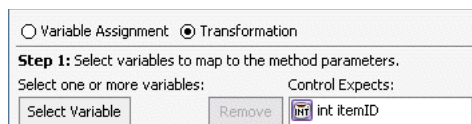


   **Note:** In this case, you must switch modes because the data type required as input to the **priceProcessorControl** is `int`, and the **iter_forEach1** variable, which holds the value of `widgetId` in the **For Each** loop, is of type XML (`WidgetRequestDocument` is valid against an XML Schema).

   The iteration variable was created for you when you specified the repeating element over which the **For Each** loop iterates. At run time, it holds the current `widgetRequest` element—that is, the one currently being processed in the **For Each** loop. (See Design a For Each Loop in Your Business Process.)

6. Click **Select Variable** to display the variables in your project, then choose **iter_forEach1 (widgetRequest)**.
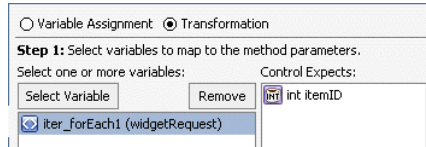
7. Click **Create Transformation**.

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

8. Click **widgetID** in the **Source** pane and drag your mouse over to **int** in the **Target** pane. A line is
   drawn between the **widgetID** and **int** elements in the map pane. It represents the transformation
   between the two data types.
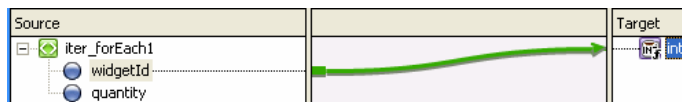


   As you draw the line in the map pane, BEA Workshop will display the following warning:

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

   **Note:** Creating this transformation creates a new method under the
   `RequestQuoteTransformation.java` already created in your project and prebuilt for
   you in the tutorial application. It is available in the
   `Tutorial_Process_Application_WEB\requestquote` folder. A new XQ file called
   `RequestQuotepriceProcessorControlgetPrice.xq`, which contains the query for this
   transformation method, is also created. See Note About Transformations.

9. In the **Navigation** pane, click **RequestQuote.java** to return to your process.

10. To close the **Request Price** node builder, click **Close**.

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

## To Design the 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. Click the arrow beside the **Control** field to display a list of the instances of controls in your
   project and select **priceProcessor**.

   The **Method** panel is populated with a list of the *asynchronous receive methods* on the
   **priceProcessor** control.

3. Select the following method from the list:

```
void returnPrice(int itemID_arg,float price_arg)
```

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

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



   The **PriceProcessor** service takes the itemID (an `int`) as input and returns an `int` and a `float`—containing values for the itemID and the price, respectively.

   In this case, you must switch from the **Variable Assignment** mode displayed in the preceding figure to the **Transformation** mode because you want to assign the data returned by the **priceProcessor** service to a variable of type XML. To do so, your business process must transform the Java data types returned from the priceProcessor service to typed XML.

5. Click **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. In the **Select variable Type** pane, ensure that **XML** is selected.

9. Click the + beside `priceQuote.xsd` in **XML Types** to expand the list from the **Typed** folder, then select **priceRequest** from the list. The **Variable Type** field is populated with `org.example.price.PriceRequestDocument`.

```
PriceQuote.xsd
    price
    priceRequests
    widgetId
    shipAddress
    priceRequest
        widgetId
        price
    priceQuote
    customerName
```
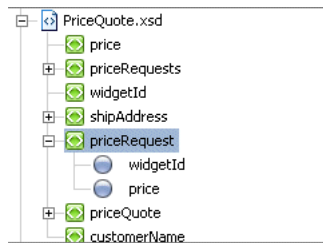
10. Click **OK**. The **Create Variable** dialog box closes and the new variable is displayed in the **Receive Data** tab. It is also listed as an **XML Type** variable in the **Data Palette**.

11. On the **Receive Data** tab, click **Create Transformation**. The Transformation tool opens and displays a representation of the **int (itemID)** and **float (price)** in the **Source** pane, and the **price** variable in the **Target** pane.

12. Map the elements in the **Source** pane to the elements in the **Target** pane, as shown in the following figure:

   **itemID** to **widgetId**
   **price** to **price**



| Source | | Target |
|---|---|---|
| itemID | | priceRequest |
| price | | widgetId |
| | | price |

**Note:** Creating this transformation creates a new method under the `RequestQuoteTransformation.java` already 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, which contains the query for this transformation method, is also created.

13. To return to your business process, click **RequestQuote.java** in the **Package Explorer**.

14. To close the **Receive Price** node builder, click **Close**.

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

### To Design the 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, as input from your business process, *typed* XML data and returns *untyped* XML (XmlObject). 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**. (See Note About Using the XmlObjectList Data Type.)

> **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. To learn more about Transformations, see Note About Transformations.

### To Create an Instance of the PriceAvailTransformations Control in Your Project

1.  If the **Data Palette** pane is not visible in BEA Workshop, choose **Windows→Show View→Data Palette** from the menu bar.

2.  On the **Package Explorer** pane, click the **priceAvailTransformations.java** file.

3.  Drag the **PriceAvailTransformations.java** file from the **Package Explorer** pane onto the **Controls** pane of the **Data Palette**. The instance of your control (**priceAvailTransformations**) is created and displayed in the **Data Palette** as shown in the following figure:



### To Design the Interaction of the Create PriceList Node With the Transformation

1.  In the **Data Palette**, expand the **priceAvailTransformations** instance, as shown in the preceding figure, then click the following method:

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

2. 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:



3. Double-click the **Create PriceList** node to open its node builder. The node builder opens on the **General Settings** tab.

4. Confirm that the method you dragged onto the node is selected:

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

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

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

6. Click the arrow on the field under **Select variable to assign** to display a list of variables, then select **price (PriceRequestDocument)**.

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

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

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

   An `XmlObject` is a Java data type that specifies data in untyped XML format. In other words, this data type represents XML data that is *not* valid against an XML Schema.

8. Click the arrow on the field under **Select variable to assign** and select **Create new variable ...**. The **Create Variable** dialog box is displayed.

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

10. If necessary, in the **Select Variable Type** pane, select **XML** 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).

11. Select **XmlObjectList** and click **OK**.

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

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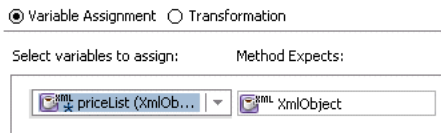


13. To close the **Create PriceList** node builder, click **Close**.

This step completes the design of the **Get Price** branch on the **Parallel** node. 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.

14. From the Workshop menu, select **File→Save All**.

### To Design the Activities on the Get Availability Branch

1. Rename the nodes on the **Get Availability** Branch (in the order in which they are executed) as follows: **Request Availability**, **Receive Availability**, **Create AvailList**.

2. Complete the following tasks:

- To Design the Request Availability Node
- To Design the Receive Availability Node
- To Design the Create AvailList Node

### To Design the Request Availability Node

1. Double-click the **Request Availability** node. The node builder opens on the **General Settings** tab.

2. Click the arrow beside the **Control** field to display a list of the instances of controls available in your project and select **availProcessor**.

The **Method** panel is populated with a list of the *asynchronous send methods* you can invoke on the **availProcessor** control.

3. Select the following method from the list:

```
void getAvail(int itemID_arg,int quantity_arg)
```

4. Click **Send Data** to open the second tab in the node builder.

By default, the **Send Data** tab opens on the **Variable Assignment** pane. The **Control Expects** field is populated with 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** service takes, as input, the itemID (int) and the quantity (int) requested by the client. It returns the itemID (int), the quantity available (int), a boolean to indicate whether the widgets are in stock, and a ship date (String).

5. Select **Transformation** to switch modes in the **Send Data** tab.
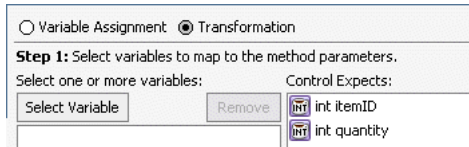


**Note:** In this case, you must switch modes because you must transform the data you input to **availProcessor**. The **availProcessor** control requires its input as `int` data types, and the **iter_forEach1** variable, which holds the value of `widgetId` and `quantity` in the **For Each** loop, is of type XML (`WidgetRequestDocument` valid against an XML Schema).

6. In **Step 1**, click **Select Variable** to display the variables in your project, then choose **iter_forEach1 (WidgetRequest)**.
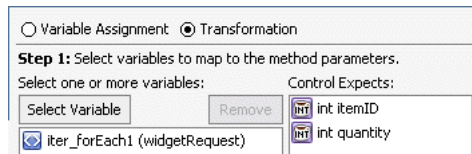


7. In **Step 2**, click **Create Transformation**.

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 shown in the following figure:

**widgetID** to **itemID_arg**
**quantity** to **quantity_arg**
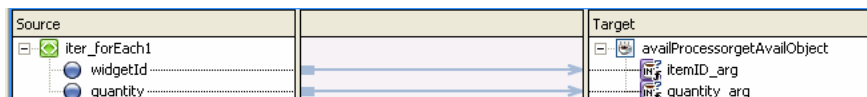
A line is drawn between the elements in the map pane. It represents the transformation between the data types.

**Note:**   Creating this transformation creates a new method under the `RequestQuoteTransformation.java` already 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, which contains the query for this transformation method, is also created.

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

10. To close the **Request Price** node builder, click **Close**.

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

### To Design the Receive Availability Node

1. Double-click the **Receive Availability** node. The node builder opens on the **General Settings** tab.

2. Click the arrow beside the **Control** field to display a list of the instances of controls available in your project and select **availProcessor**.

    The **Method** panel is populated with a list of the *asynchronous receive methods* on the **availProcessor** control.

3. Select the following method from the list:

    ```
    void avail(int itemID, int qty, boolean avail, String date)
    ```

4. 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(int itemID_arg, int qty_arg, boolean avail_arg, String date_arg)` 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 you want to assign the data returned by the **availProcessor** service to a variable of type XML. To do so, your process must transform the Java data types returned to typed-XML.

5. Click **Transformation**. The **Receive Data** tab is displayed as shown in the following figure:
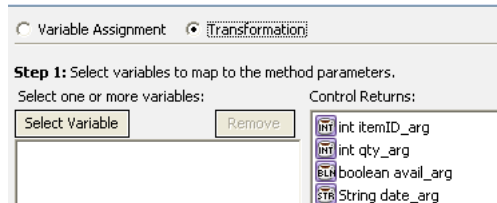
6. Click **Select Variable**, then **Create new variable...**. The **Create Variable** dialog box is displayed.

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

   • In the **Type** Name pane, ensure that **XML** is selected.

8. In **XML Types**, click the + beside availQuote.xsd to expand the list, then select **availRequest** from the list. The **Variable Type** field is populated with **org.example.avail.AvailRequestDocument**.
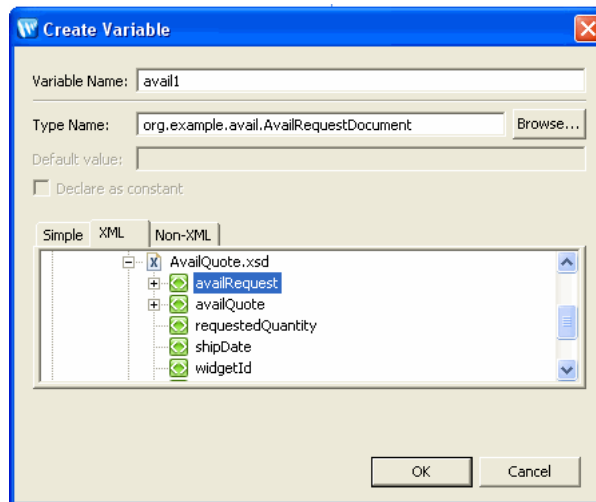


9. Click **OK**. The **Create Variable** dialog box is closed and your new variable is created and is listed as an **XML Type** variable in the **Data Palette**.

10. In **Step 2**, click **Create Transformation** to open the Transformation tool, which displays a representation of the data types returned by the **availProcessor** control in the **Source** pane, and the **avail** variable in the **Target** pane.

11. Map the **Source** values to the **Target** elements as shown in the following:

**itemID** to **widgetId**
**qty** to **requestedQuantity**
**avail** to **quantityAvail**
**date** to **shipDate**)
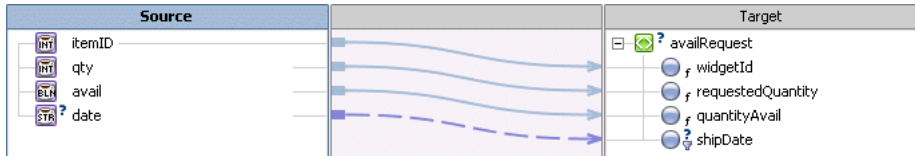
| Source | | Target |
|---|---|---|
| itemID | | ☐ ⊟ ◉ ? availRequest |
| qty | | ◯ _f_ widgetId |
| avail | | ◯ _f_ requestedQuantity |
| date | | ◯ _f_ quantityAvail |
| | | ◯ ? shipDate |

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

12. Click **File →Save All**.

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

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

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

### To Design the Create AvailList Node

In the same way as you designed the business process to append the price data to a single variable when
you designed the **Get Price** branch of the **Parallel** node, in this step, you call a method on the
**priceAvailTransformations** control to append the availability data returned to a single variable, of
type **XmlObjectList**. (See Note About Using the XmlObjectList Data Type.)

1. Expand the **priceAvailTransformations** control instance in the **Data Palette**, then click the
   following method:

```
XmlObject convertAvailXMLtoXMLObj(org.example.avail.AvailRequestDocument
_availRequestDoc)
```

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, as shown in the
   following figure:
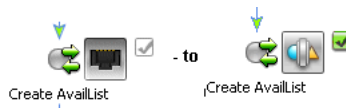
3. Double-click the **Create AvailList** node. The node builder opens on the **General Settings** tab.

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:

```
XmlObject convertAvailXMLtoXMLObj(org.example.AvailRequestDocument
_availRequestDoc)
```

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

   The **Control Expects** field is populated with `AvailRequestDocument`, which is the data type expected by the `convertAvailXMLtoXMLObj(org.example.avail.AvailRequestDocument _availRequestDoc)` method on the **priceAvailTransformations** control.

6. Click the arrow on the field under **Select variable to assign** to display a list of variables. Select **avail (AvailRequestDocument)**.

   In this case, note that the data type of your **avail** variable (**AvailRequest**) matches that of the data expected by the **priceAvailTransformations** control.

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

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

   An `XmlObject` is a Java data type that specifies data in untyped XML format. In other words, this data type represents XML data that is not valid against an XML Schema.

8. Click the arrow on the field under **Select variable to assign** and select **Create new variable ...**. The **Create Variable** dialog box is displayed.

9.  In the **Variable Name** field, enter **availList**.
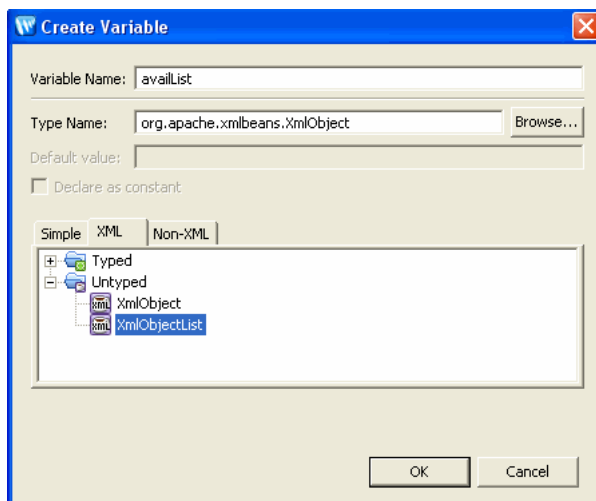
10. In the **Select Variable Type** pane, if necessary, select **XML** to display a representation of the XML data types in your application.

11. Select **XmlObjectList**, under **Untyped**, then click **OK**.

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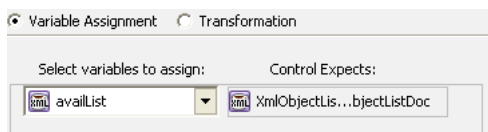


12. To close the **Create AvailList** node builder, click **Close**.

    This step completes the design of the **Get Availability** branch on the **Parallel** node. At run time, by executing this branch, your business process appends the 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 Workshop menu, select **File→Save All**.

### Note About Using the XmlObjectList Data Type

On each iteration through the **For Each** loop, the priceProcessor service returns price data, which is assigned to the `price` variable; and the availProcessor 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.

An `XmlObjectList` is a Java data type that specifies a sequence of untyped XML format data. In other words, 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 your **Parallel** node, your business process assigns the data from the `price` variable to the `priceList` variable (of type `XmlObjectList`). In this way, a single variable holds the price data for each of the widgets in the Request for Quote over which the **For Each** loop iterates. In the same way, a single variable holds the availability data for each widget.

To learn how the **XmlObjectList** variable is used, see To Design the Create PriceList Node and To Design the Create AvailList Node.

# Related Topics

Note About Transformations

Creating Maps

Testing Maps in the Test View

*Guide to Data Transformation*

Understanding Parallel Execution in Your Business Process

# 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 quote data are assigned to the priceList variable, and the availability quote data are assigned to the availList variable. Both the priceList and the availList variables are of data type XmlObjectList (an 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 (that is, to XML that is valid against the XML Schemas provided in your project). Subsequently, you combine the XML-typed price and availability data to produce a single *quote* document, which comprises the response your business process sends to the client that invoked it.

## Note About Transformations

BEA Products allows you to create Transformations in the following ways:

- Using the node builders in your 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 Request for Quote message to the input of the taxCalculation control. (To review, see "To 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 you designed.

- By choosing **File**→**New**→**Transformation File** from the BEA Workshop menu. Transformation files you create in this way can be called from your business process via Control nodes.

The following Transformation files were created using this method, and are provided for you in the tutorial application: `PriceAvailTransformations.java` and `TutorialJoin.java`. You used `PriceAvailTransformations.java` in the previous step (Step 8: Design Parallel Paths of Execution) and you will use it again in this step, as well as `TutorialJoin.java` in this step.

In this step, you design the logic in your business process that creates a single quote document from the price and availability data already calculated. This involves designing **Control** nodes that call the `PriceAvailTransformations.java` and `TutorialJoin.java` Transformation files.

**Note:** A description of how to create these Transformation files is outside the scope of this tutorial. However, to learn how to create `TutorialJoin.java`, see *Tutorial: Building Your First Data Transformation*.

In this step, in which you create a single quote document for a client, you must complete the following tasks:

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

# Convert Price List to XML Quote Document

Complete the following steps to design a node to transform the price list (created as a result of iteration through the **For Each** loop) to a variable whose data type is typed-XML. To do so, you use methods on the **priceAvailTransformations** control.

### To Design the Interaction With the Transformation Control

1. With the **priceAvailTransformations** control instance expanded in the **Data Palette**, click the following method:

   ```
   PriceQuoteDocument convertPriceListToXML
   (QuoteRequestDocument _quoteRequestDoc, XmlObjectList _XmlObjectListDoc)
   ```
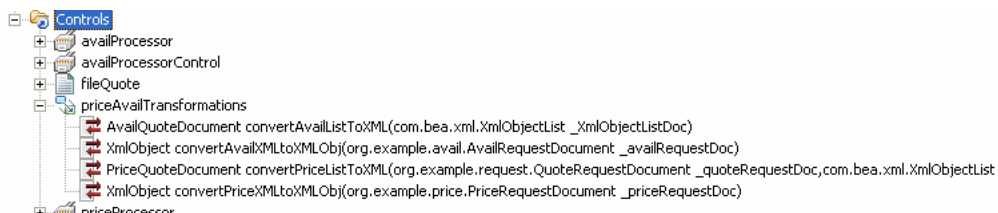
   

2. Drag the method from the **Data Palette** and drop it on your **RequestQuote** business process in the **Design** view, placing it immediately after, and outside, the **For Each** block.
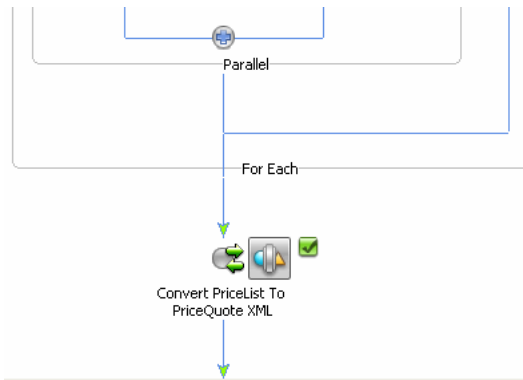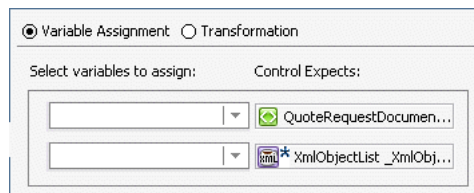
3. Rename the node from **convertPriceListToXML** to **Convert PriceList To PriceQuote XML**.

4. Double-click the **Convert PriceList To PriceQuote XML** node to open its node builder.

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

```
PriceQuoteDocument convertPriceListToXML
(org.example.request.QuoteRequestDocument _quoteRequestDoc,
com.bae.xml.XmlObjectList _XmlObjectListDoc)
```
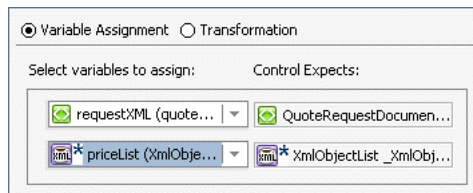
6. Click **Send Data** to open the second tab in the node builder.

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



   **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, the shipping address, and the price quote data (the price list) 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`. To learn more about the **priceAvailTransformations** control, see Note About the Transformation on This Node.

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

   The **Control Returns** field is populated with `PriceQuoteDocument`, which is the data type returned by the `convertPriceListToXML()` method on the **priceAvailTransformations** control.

9. Click the arrow associated with the **Select variables to assign** field, and click **Create new variable ...**. The **Create Variable** dialog box is displayed.
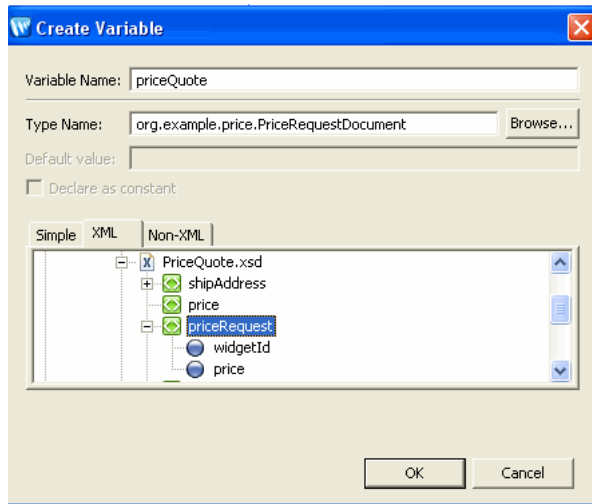


10. In the **Variable Name** field, enter **priceQuote**.

11. In the **Select Variable Type** field, select **priceQuote** in the **XML Types** list. The **Variable Type** field is populated with `org.example.price.PriceQuoteDocument`.

12. Click **OK** to close the **Create Variable** dialog box.

13. To close the node builder, click **Close**.

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

### Note About the Transformation on This Node

The `convertPriceListToXML()` method on the **priceAvailTransformations** control does the work of creating the price quote XML data in the preceding step.

In brief, 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 complete.

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

It is left as an exercise to the reader to view this and other transformation methods on the **priceAvailTransformations** control. For example, you can double-click **PriceAvailTransformations.java** in the **Package Explorer** to display the Transformation control in the **Source** view. Right-click on the **convertAvailXMLtoXMLObj** method, and select **Goto XQuery Document** to open the Transformation tool. Use the **Design** view and **Source** view tabs in the transformation tool to see the data map that represents the transformation and the corresponding XQuery. Use the **Test View** tab to test the XQuery. For example the following figure shows the map for the `convertAvailXMLtoXMLObj()` method:
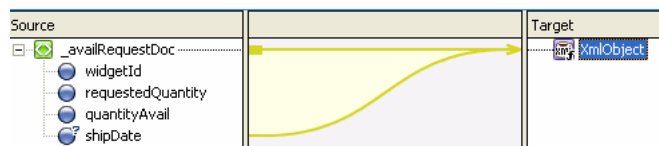


# Related Topics

*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 variable whose data type is typed-XML. To do so, you use methods on the **priceAvailTransformations** control.

### To Design the Interaction With the Transformation Control

1. Expand the **priceAvailTransformations** control instance in the **Data Palette**, then click the following method:

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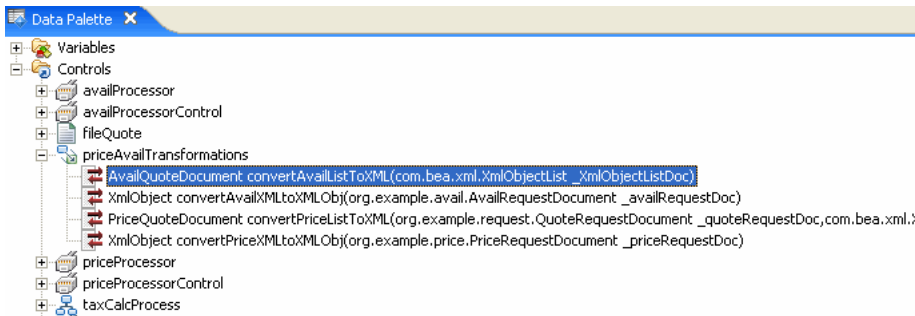


2. Drag the method from the **Data Palette** and drop it on your **RequestQuote** business process in the **Design** view, placing it immediately after the **Convert PriceList to PriceQuote XML** node.



3. Rename the node from **convertAvailListToXML** to **Convert AvailList to AvailQuote XML**.

4. Double-click the **Convert AvailList to AvailQuote XML** 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. Click **Send Data** to open the second tab in the node builder.

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

7. On the **Send Data** tab, under **Select variables to assign**, click the arrow in the variable assignment field, and select **availList (XmlObjectList)**.

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

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

9. Click the arrow associated with the **Select variables to assign** field, and click **Create new variable ...**. The **Create Variable** dialog box is displayed.

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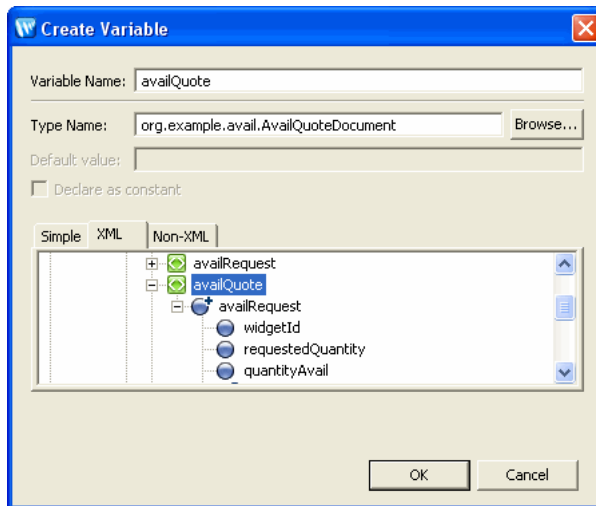


11. In the **Select Variable Type** field, click the + beside **AvailQuote.xsd** in the **XML Types** list, then select **availQuote** from the list. The **Variable Type** field is populated with `org.example.avail.AvailQuoteDocument`.

12. Click **OK** to close the **Create Variable** dialog box.

13. To close the node builder, click **Close**.

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

### Note About the Transformation on This Node

The `convertAvailListToXML()` method on the **priceAvailTransformations** control does the work of creating the availability quote XML data. The input to `convertAvailListToXML()` is the availability data returned by the **availProcessor** control after the iterations in the **For Each** node complete.

You can double-click **PriceAvailTransformations.java** in the **Package Explorer** to display the Transformation control in **Source** view. Right click on `convertAvailListToXML` method, and select **Go to XQuery Document** from the drop-down menu to open the Transformation tool. The following figure shows the map for the `convertAvailListToXML()` method:
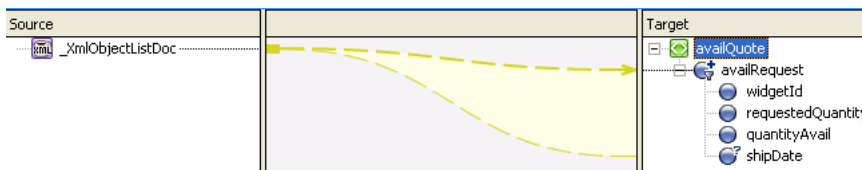


The preceding figure shows the transformation of the data in a variable of type `XmlObjectList`, which contains a repeating set of untyped XML data, to the repeating element in an XML-typed variable. Note that to achieve this transformation, the repeating element in the target schema *must* be the single 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
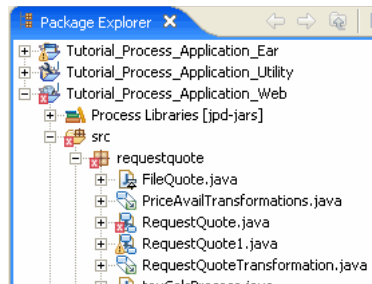
Complete the following tasks:

- To Create an Instance of the TutorialJoin Control in Your Project
- To Design the Process Interaction With the TutorialJoin Control

### To Create an Instance of the TutorialJoin Control in Your Project

The `TutorialJoin.java` control is provided in your tutorial application. It is available in the **requestquote** folder in your **Tutorial_Process_ApplicationWeb** project folder. To learn how to build the `TutorialJoin.java` control, see *Tutorial: Building Your First Data Tansformation*.

Complete the following steps to add an instance of this control to your business process.

1. If the **Data Palette** is not visible in BEA Workshop, choose **Windows**→**Show View**→**Data Palette** from the menu bar.

2. On the **Package Explorer** pane, click the **TutorialJoin.java** file.



3. Drag the **TutorialJoin.java** file from the **Package Explorer** pane onto the **Data Palette**. The instance of your control (**tutorialJoin**) is created and displayed in the **Data Palette** as shown in the following figure:
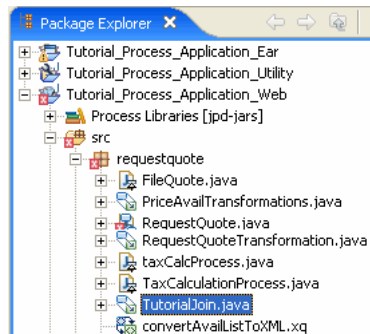


## To Design the Process Interaction With the TutorialJoin Control

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 `join` method does the work of combining the data returned to your business process from different systems and creating a single XML response document (quote), which is subsequently returned to the business process' client.

1. Expand the **tutorialJoin** control instance in the **Data Palette**, then click 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 placing it immediately after the **Convert AvailList to AvailQuote XML** node.

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



Convert PriceList To
PriceQuote XML

Convert AvailList to AvailQuote
XML

Combine Price and Avail

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, which you dragged onto the node from the **Data Palette**, is selected in the **Method** field:

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

6. Click **Send Data** to open the second tab in the node builder.

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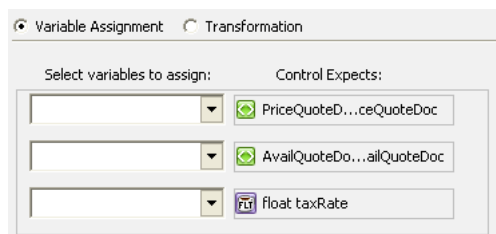


7. Under **Select variables to assign**, select the variables such that their data types match the data type expected (Control Expects) in the input parameters to the `join()` method, as follows:

   – For **PriceQuoteDocument** select **priceQuote (PriceQuote)**.

**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 (AvailQuote)**.

**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.
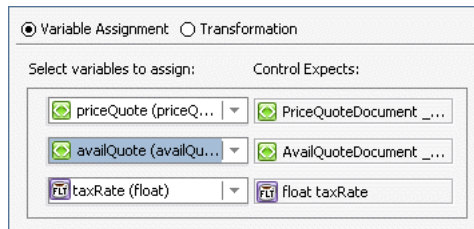


8. Click **Receive Data** to open the third tab in the node builder.

On the **Receive Data** tab, the **Control Returns** field is populated with `QuoteDocument`, which is the data type returned by the `join()` method.

9. Click the arrow in **Select variable to assign**, then choose **Create new variable...**. The **Create Variable** dialog box is displayed.
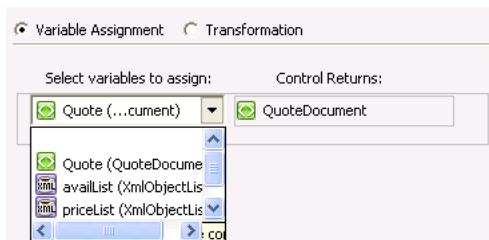


10. In the **Variable Name** field, enter **Quote**.

11. In the **Select Variable Type** field, select **quote** from the list of **XML** types, as shown in the following figure:
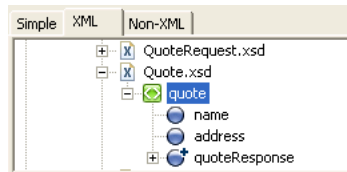
The **Variable Type** field is populated with `org.example.quote.QuoteDocument`.

12. Click **OK** to create the new variable. The **Quote** variable is displayed in the **Receive Data** tab, and also in the **XML** list in the **Data Palette**.

13. To close the node builder, click **Close**.

    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.

14. From the Workshop menu, select **File→Save All**.

    To complete Part III of the tutorial, it only remains to write the quote to your file system (an optional step), and to create the **Client Response** node in your 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

    Step 11: Send Quote From Business Process to Client

## Related Topics

To learn how to create Transformation controls, and specifically to learn how to design the **TutorialJoin.java** control used in this section, see *Tutorial: Building Your First Data Tansformation*.

# Step 10: Write Quote to File System

Complete this step to create a node, at which your business process writes the quote created in the preceding step to your file system. A File control makes it easy to read, write, or append to a file in a file system.

Complete the following tasks to design your business process to write the combined price and availability quote to your file system:

- To Create an Instance of a File Control in Your Project

- To Design a Control Send Node in Your Business Process to Interact With Your File Control

The following tasks are optional. They are provided to deepen your understanding of File controls but are not required for the completion of the tutorial.

- To Assign File Control Properties to a Variable in Your Business Process

- To Use the File Control Properties in Your Business Process

### To Create an Instance of a File Control in Your Project

In this scenario, you add one instance of the File control to your business process.

1. Click ▽ on the **Data Palette Controls** tab to display a list of controls that represent the resources with which your business process can interact.

2. Select **Integration Controls→File**. The **Insert Control** dialog box is displayed.

3. In the **Insert Control** dialog box:

   a. Enter **myFileQuote** as the variable name for this control, and click **Next**.

b. In the **Create Control** dialog box, select **Create a new File control to use**, then enter **MyFileQuote** in the **Name** field.

c. In the **Insert Control - File** dialog box, enter values in the following fields:

**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.

d. Click **Finish** to close the **Insert Control** dialog box.

An instance of a File control, named **myFileQuote**, is created in your project and displayed under the **Controls** folder.

4. From the Workshop menu, select **File→Save**.

**Note:**   In the simple case, each instance of the File control allows you to manipulate a separate file. To learn about how your File control can operate on multiple files, see File Control.

### To Design a Control Send Node in Your Business Process to Interact With Your File Control

1. Expand the **myFileQuote** control instance in the **Data Palette**, then click the following method:

```
FileControlPropertiesDocument write(XmlObject someData)
```

2. From the **Data Palette**, drag the method and drop it on your **RequestQuote** business process, placing it immediately after the **Combine Price and Avail Quotes** node (and immediately before the **Finish** node). The node is named **write** by default.

3. Rename the node from **write** to **Write Quote to File**.

4. Double-click the **Write Quote to File** node. Its node builder opens on the **General Settings** tab.

5. 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. Click **Send Data** to open the second tab in the node builder. The **Control Expects** field is populated with XmlObject someData, which is the data type expected by the write() method.

7. In the **Select variables to assign** field, click the arrow to display the list of variables in your project, then choose **Quote (quote)**. (Recall that you created the Quote variable to hold the quote in Step 9: Create Quote Document.)

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, proceed to Note About File Control Properties.

8.  To continue with the tutorial without specifying a variable on the **Receive Data** tab, click Close to close the node builder.

9.  From the Workshop menu, select **File→Save**.

    This step completes the design of your File control node. At run time, the quote document you created in Step 9: Create Quote Document is written to your file system in the location specified by you.

10. Proceed to Step 10: Write Quote to File System.

# Note About File Control Properties

This optional section provides additional steps you can use to further define the **Write Quote to File** node you created in the preceding section. You are *not* required to complete the steps in this section to complete the tutorial. The steps are provided to help you understand and use the *File Control Properties* returned to your business process by the File control's `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 document of type XML: `FileControlPropertiesDocument`. The `FileControlPropertiesDocument` is valid against an XML Schema: `DynamicProperties.xsd`. The Schema is provided for you in the **project** in your tutorial application. (See the **project** in the **Package Explorer** tab.)
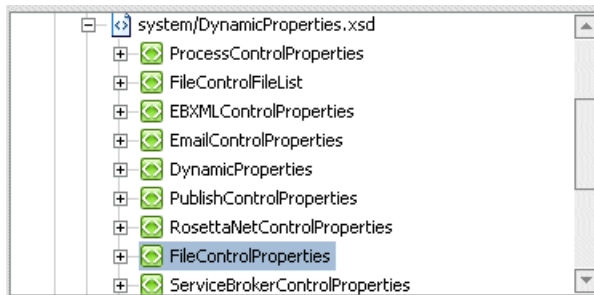
### To Assign File Control Properties to a Variable in Your Business Process

The following steps describe 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 as described in To Design a Control Send Node in Your Business Process to Interact With Your File Control.

1.  If the **Write Quote to File** node builder is not open, double-click the node.

2. Click **Receive Data** to open the third tab in the node builder. The **Control Returns** field is populated with `FileControlPropertiesDocument`, which is the data type returned by the `write()` method.

3. In the **Select variables to assign** field, click the arrow to display the list of variables in your project, then choose **create new variable...**. The **Create Variable** dialog box is displayed.



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

5. The Variable type field populated with `com.bea.wli.control.dynamicProperties.FileControlPropertiesDocument`. In the **Select Variable Type** pane, expand **system/DynamicProperties.xsd**, then select **FileControlProperties**.

   **Note:** By default the `com.bea.wli.control.dynamicProperties.FileControlPropertiesDocument` is selected in the Variable type field.

6. Click **OK**. The new variable is displayed in the node builder.
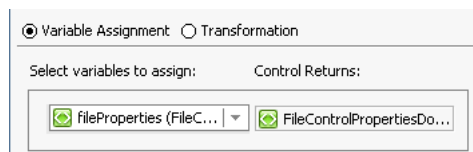


7. To close the node builder, click **Close**.

8. From the Workshop menu, select **File→Save**.

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

Note:    The `Dynamic Properties.xsd` XML Schema must be available in a **project** in your
application before you can create a variable to hold the file control properties that are
returned to your business process from the File control. `Dynamic Properties.xsd` is one
of the system schemas available to you when you create BEA Products applications in BEA
Workshop.

## To Use the File Control Properties in Your Business Process

In the preceding steps, you assigned the data returned from the File control to a variable named
**fileProperties**. You can derive information about the file you wrote from **fileProperties**.

Click the **Source** view tab to view your **RequestQuote.java** file in **Source** view. By completing the
steps described in the preceding section, the following code is written in your JAVA file in keeping with
the work you did in the **Design** view.

The **fileProperties** variable declaration is shown in the following listing:

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

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

```
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 of code) to write code that derives
information from the **fileProperties** variable. For example, the following line of code returns the
FileMask:

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

To illustrate this example further, edit the `public void fileQuoteWrite()` method in **Source View**
to include the line of code shown in bold in the following listing:

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

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

```
}
```

Note that you must add the code *after* the PROTECTED SECTION comment. Code completion in the **Source View** helps you write the code. When you switch back to the **Design** view, note that the **Write Quote to File** node changes to include the following icon:  . 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 FileMask) is printed to the console.

## Related Topics

File Control

*Using Integration Controls*

How Do I: Create a project Folder?

How Do I: Import Files into a project Folder?

# 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 Chapter , "Step 2: Specify How the Process is Started." This section describes how to add operations that send messages from your business process to a client. That is, in this section you learn how to design **Client Response** nodes.

This step describes the following tasks:

- To Add a Client Response Node to Your Business Process
- To Design Your Send Quote Node

### To Add a Client Response Node to Your Business Process

1. On the **Package Explorer** pane, double-click **RequestQuote.java** to ensure that your business process is displayed in **Design** view.

2. In the **Node Palette,** select 🖥️ **Client Response**, then drag and drop the node onto the business process immediately before the **Finish** node. The **Design** view is updated to contain the **Client Response** node.

3. Change the name of the node from **Client Response** to **Send Quote**.
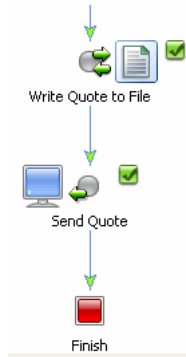
### To Design Your Send Quote Node

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 this node.

1. Double-click the **Send Quote** node in your business process. The node builder is displayed.

2. In the **General Settings** tab, change the name in the **Method Name** field from **clientResponse** to **quoteResponse**.

3. Click **Add** to display the panel of data types.
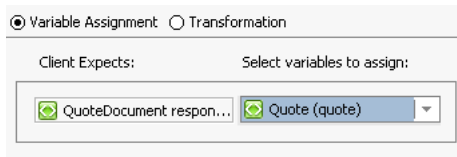
   **Note:** In the **Combine Price and Avail Quotes** node, you created an XML variable to hold the quote. This data assigned to this variable is valid against the `Quote.xsd` Schema. Therefore we are concerned with **XML Types** at this node.

4. If it is not already selected, select **XML**.

   a. If necessary, click the + beside **XML Types** to see a list of XML Schemas in your project.

   b. Expand **Quote.xsd**, then click the **quote** node. The **Type** field is populated with `org.example.quote.QuoteDocument`.

   c. In the **Name** field, replace **x0** with **responseXML**. In this way, you name the parameter that returns the **QuoteDocument**.

5. 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: ☑

6. Click the **Send Data** tab. A tab that allows you to define one or more variables to hold the data your business process sends to clients is displayed.

   The **Client Expects** field is populated with the data type and the name of the parameter you specified on the **General Settings** tab: **QuoteDocument responseXML**

7. Under **Select variables to assign**, select the **Quote (quote)** variable.



8. To close the **Client Response** node builder, click **Close**.

9. From the Workshop menu, select **File→Save**.

   This step completes the design of your RequestQuote business process. To run it, proceed to Step 12: Run the Request Quote Business Process.

# Step 12: Run the Request Quote Business Process

You can run and test the functionality of the business process you created using WebLogic Workshop's browser-based interface. Using the Workshop Test Browser, you play the role of the client, invoking the methods on the business process and viewing the responses.

## To Launch the Test Browser

To run and test the business process that you have created, complete the following steps:
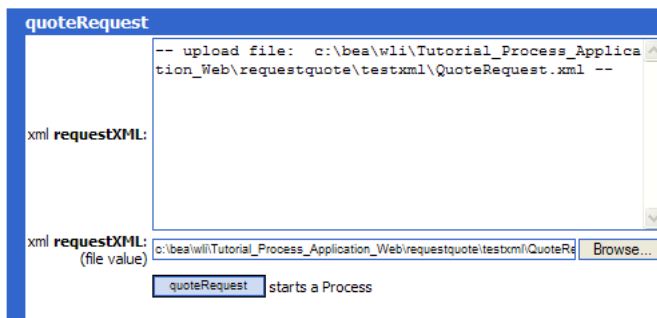
1.  If WebLogic Server is not already running, from the BEA Workshop menu, choose **Window→ Show View →Other →Server→ Servers**, and click **OK**. A **Server** view is displayed in which the Server and its state are shown.

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

3.  In the Define a New Server dialog box, select either a **Choose an existing server** option or **Manually define a server** (if there is no server defined), and click **Next**.

4.  In the **BEA WebLogic v9.2 Server** dialog box, to manually define a server, click **Browse**, and select the samples integration domain directory from the product installation directory available at `BEA_HOME\weblogic92\samples\domains\integration`, where BEA_HOME represents the directory in which you installed WebLogic Platform. Click **Finish**.

    The samples domain integration server is started, and the RequestQuote application is deployed on it. When WebLogic Server is running, the following indicator is visible in the **Servers** view.

    Status
    Started

5.  After the application is deployed, the Test Browser is displayed.

6. Click the **Test Form** tab. Open the file browser by clicking **Browse** beside the **xml requestXML (file value)** field.

7. Select **RequestQuote.xml** from the **requestquote\testxml** folder in your project.

   The **QuoteRequest.xml** file is available at the following location in your file system:
   `Tutorial_Process_Application_WEB\src\testxml\QuoteRequest.xml`

8. Click the **Test Form** tab. In the **Test Form** page, click the button labeled with the method name on your business process (**quoteRequest**) to start the business process.



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

9. Click **Refresh** on the Message Log to refresh the entries in the log until this instance of the business process completes running. Entries in the Message Log correspond to the methods on your business process:

   – The **quoteRequest** method that starts the business process.

   – A call from your business process to the **taxCalculation** Web service:
     **taxCalculation.requestTaxRate**

   – A response from the service to your business process: **taxCalculation.returnTaxRate**

   – The **Instance ID**—When the business process finishes, a message similar to the following is displayed in the Message Log:

     `Instance instanceID is Completed.`

   where `instanceID` represents the ID 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 **taxCalculation.returnTaxRate**, you can view the response from the **taxCalculation** service—in this case, the tax rate was calculated, based on the input value (**NJ)** for the **state** element in the test XML.



In the sample XML message you used, `state="NJ"`. That is, the state to which the order is shipped is `NJ`. This XML message is designed to cause the flow of execution through the **Yes** branch on your **Sales Tax Calculation Needed?** node. The preceding figure 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, via a control.

### To Monitor Instances of Your Business Process

You can use the BEA Products Administration Console to monitor running processes or view statistics for processes that already ran.

- Click **Monitor** to open the BEA Products Administration Console in a Web Browser. Login using username = weblogic and password = weblogic. The BEA Products Administration Console opens to the **Process Instance Details** page. The WebLogic Integration Administration

Console allows you to administer and manage your BEA Products applications. For example, if you click **View Statistics** on the Process Instances navigation pane, you access a **Process Instance Statistics** page. This page displays a summary of business process instances grouped by the process type. To view the instances of a process type that ran or are running on your server, click the process name. Processes instances are identified by their *instanceID*. Note that the instanceID displayed for your **RequestQuote** business process matches the instanceID displayed on the Message Log pane (see the preceding figures in this topic).

- Click **Monitor all RequestQuote.java processes** at the top of the **Test Form** to open the BEA Products Administration Console. Login using the default username: `weblogic` and password: `weblogic`. When you use this link to open the Administration Console, it opens on the **Process Instance Summary** page, which displays a summary of all the instances of business processes that ran or are running. It allows you to:

  - View process instance statistics, including the number of instances in each state (running, suspended, aborted, and completed).

  - View the summary or detailed status for selected instances.

  - Suspend, resume, or terminate, selected instances.

- Other ways to invoke the BEA Products Administration Console include the following:

  - From the BEA Workshop **Tools** menu, select **Tools→WebLogic Integration→WebLogic Integration Administration Console**

  - Entering the following URL in a Web browser: `http://localhost:7001/wliconsole`

    The default username is `weblogic` and password is `weblogic` for the sample integration server.

To learn about using the BEA Products Administration Console, see the console's online help at `http://edocs.bea.com/wli/docs92/adminhelp/index.html`.

# Related Topics

*WebLogic Integration Admin console Online Help* at

`http://edocs.bea.com/wli/docs92/adminhelp/index.html`

Understanding the Service URL

Testing Your Application with Test View