



BEA WebLogic Integration™

Guide to Building Business Processes

Copyright

Copyright © 2004-2005 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, BEA WebLogic Server, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic JRockit, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server Process Edition, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

Guide to Building Business Processes

Creating a Business Process Application

Components of Your Application	2-1
Designing Your Application	2-3
Creating a Business Process Application	2-3
Setting the Business Process Properties	2-4

Starting Your Business Process

Designing Start Nodes	3-1
Client Request Start (Asynchronous)	3-4
Client Request with Return Start (Synchronous)	3-5
Adding Nodes to Your Client Request with Return Node Group	3-12
Naming the Methods on Client Request with Return Nodes	3-13
Subscription Start (Asynchronous)	3-13
Subscription Start (Synchronous)	3-18
Event Choice Start	3-23
Exception Handlers on Start Nodes	3-26

Interacting With Clients

Receiving Messages From Clients	4-2
Create a Client Request Node in Your Business Process	4-2
Design Your Client Request Node	4-3

Naming the Methods on Client Request Nodes	4-6
Sending Messages to Clients	4-6
Create a Client Response Node in Your Business Process	4-7
Design Your Client Response Node	4-8
Adding Dynamic Callback Properties	4-10
Buffering Client Messages	4-11

Interacting With Resources Using Controls

Designing Interactions Between Business Processes and Resources	5-1
Create Control Nodes in Your Business Process	5-2
Designing Your Control Nodes	5-4
Adding Instances of Controls to Your Business Process Project	5-4
Configuring Control Nodes	5-11
Setting Control Properties	5-15

Receiving Multiple Events

Create an Event Choice Node in Your Business Process	6-2
Design Your Event Choice Group	6-4

Creating Parallel Paths of Execution

Understanding Parallel Execution in Your Business Process	7-1
Create a Parallel Node in Your Business Process	7-2
Design Your Parallel Node	7-3

Defining Conditions For Branching

Creating a Decision Node in Your Business Process	8-2
Designing Your Decision Node	8-3

Creating Case Statements

Comparing Decision Nodes and Switch Nodes	9-1
---	-----

Creating a Switch Node	9-2
Designing a Switch Node	9-3

Writing Custom Java Code in Perform Nodes

Creating Looping Logic

Understanding While Node Groups.	11-1
Creating While Node Groups in Your Business Process.	11-2
Designing While Node Groups	11-3

Looping Through Items in a List

Creating For Each Nodes in Your Business Process.	12-1
Designing For Each Nodes	12-2

Specifying Endpoints in Your Business Process

Grouping Nodes in Your Business Process

Handling Exceptions

Types of Exception Handlers.	15-1
Creating Exception Handler Paths.	15-2
Deleting Exception Handler Paths.	15-5
Order of Execution of Exception Handlers	15-5
Handling Exceptions in Transaction Blocks	15-6
Using Exception Handlers for Compensation	15-7
Compensation Example	15-7
Unhandled Exceptions.	15-9

Adding Message Paths

Creating a Message Path	16-1
Deleting Message Paths.	16-4

Adding Timeout Paths

Creating a Timeout Path	17-1
Deleting Timeout Paths	17-3

Running and Testing Your Business Process

Using the Test Browser	18-1
Testing the Public Methods of Your Business Process	18-3
Testing a Message Broker Channel	18-4
Viewing the Process Graph	18-5
Understanding the Service URL	18-6

Business Process Variables and Data Types

Creating Variables	19-1
Deleting Variables	19-6
Working with Data Types	19-6
Assigning MFL Data to XML Variables and XML Data to MFL Variables	19-9

Versioning Business Processes

Creating a New Version of a Business Process	20-2
Configuring the New Versions of Your Business Process	20-4
Editing Versions of Business Processes	20-4
Deleting Versions of a Business Process	20-5
Using Versioning with Long-Running Business Processes	20-5
Importing Versioned Business Processes	20-6

Validating Schemas

Validating a Typed XML Variable	21-1
Typing and Validating an Untyped XML Type	21-2

Building Stateless and Stateful Business Processes

Working with Variables in Stateless Processes	22-2
---	------

Building Synchronous and Asynchronous Business Processes

Working with Subprocesses	23-2
Synchronous Subprocesses	23-2
Asynchronous Subprocesses	23-3
Synchronous Clients for Asynchronous Business Processes	23-4
Limitations	23-9
Synchronous-Asynchronous Security	23-11

Transaction Boundaries

Implicit Transaction Boundary Rules	24-2
An Implicit Transaction Boundary Example	24-3
Explicit Transaction Boundaries	24-6
Creating Explicit Transaction Boundaries	24-7
Setting the Explicit Transaction Properties	24-8
Handling Exceptions in Transaction Blocks	24-8

Business Process Source Code

Overview	25-2
Business Process Language	25-2
Variables	25-3
Control Declarations	25-3
Client Operations and Control Communication Methods	25-4
Can You Edit Code in Protected Sections?	25-4
Perform Methods	25-6
XQuery Statements	25-6

Building ebXML Participant Business Processes

About the ebXML Participant Business Process File	26-2
Creating an ebXML Participant Business Process	26-3
Customizing an ebXML Participant Business Process	26-3
Configuring Business Process Properties (Required)	26-4
Customizing Names and Argument Types (Optional)	26-5
Retrieving the ebXML Message Envelope (Optional)	26-6

Building RosettaNet Participant Business Processes

About the RosettaNet Participant Business Process File	27-2
Creating a RosettaNet Participant Business Process	27-5
Customizing a RosettaNet Participant Business Process	27-6
Configuring Business Process Properties (Required)	27-7
Customizing Argument Types (Optional)	27-7
Configuring Data Transformation (Required)	27-8
Integrating with the Private Participant Process (Required)	27-9
Setting Up the Notification of Failure (Required)	27-9

Guide to Building Business Processes

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. Business Processes allow you to orchestrate the execution of business logic and the exchange of business documents among back-end systems, users and trading partners (systems and users) in a loosely coupled fashion.

This guide introduces the tools in WebLogic Workshop that allow you to create Business Processes graphically, allowing you to focus on the application logic rather than on implementation details as you develop.

The first step in the design of your business process is to build a graphical representation of the business process that meets the business requirements for your project. You create a graph of component nodes in your business process by dragging components from the **Business Process Palette** and dropping them onto the **Design View** pane. Program control is represented visually by these nodes (or shapes) and the connections between them. Effectively, you create a graphical representation of your business process and its interactions with clients and resources, such as databases, JMS queues, file systems, and other components.

Topics Included in This Section

Chapter 2, “Creating a Business Process Application”

Describes how to start WebLogic Workshop and WebLogic Server and provides step-by-step instructions for creating a business process project in WebLogic Workshop.
Describes how some of the high-level components you create as you build your business

process application (specifically, the names you choose for these components) surface in the finished application.

Chapter 3, “Starting Your Business Process”

Describes how to design the trigger that starts your business process. You can design your business process to start as the result of receiving a request from a client, as the result of receiving a message from a message broker channel to which the business process is subscribed, or as the result of receiving any one of the former types of messages, via an Event Choice node.

Chapter 4, “Interacting With Clients”

Provides step-by-step instructions for creating nodes in your business process that handle interactions with client applications. A business process must be able to receive messages from clients and send messages to clients.

Chapter 5, “Interacting With Resources Using Controls”

Describes how to create nodes in your business process that manage the interactions with external resources, such as databases, EJBs, Web services, and so on. WebLogic Workshop Controls represent the interface between a business process and these external resources.

Chapter 6, “Receiving Multiple Events”

Describes how to create nodes at which your business process waits to receive multiple events, from clients or controls. Event Choice nodes handle the receipt of multiple events. Event Choice nodes, in turn, contain Client Response or Control Receive, or both.

Chapter 7, “Creating Parallel Paths of Execution”

Describes how you can design your business process to execute tasks in parallel.

Chapter 8, “Defining Conditions For Branching”

Describes how to design a Decision node and its associated conditions in your business process. A Decision node is used to select exactly one path of execution based on the evaluation of one or more conditions.

Chapter 9, “Creating Case Statements”

Describes how to design Java-like case statements through using Switch nodes. A Switch node is used to select one path of execution based on the evaluation of an expression specified on a condition node. A Switch node contains one condition node, one or more case paths, and one default path.

Chapter 10, “Writing Custom Java Code in Perform Nodes”

Describes the Perform nodes, which you can customize with Java code.

Chapter 11, “Creating Looping Logic”

Describes how you can design logic in your business process in which the activities enclosed in a loop are performed repeatedly while a specific condition is true.

Chapter 12, “Looping Through Items in a List”

Describes how to design For Each nodes in a business process, that is, how to create the logic that allows your business process to perform a set of activities repeatedly, once for each item in a list.

Chapter 13, “Specifying Endpoints in Your Business Process”

Describes how to design the final node in your business process.

Chapter 14, “Grouping Nodes in Your Business Process”

Describes how to combine business process nodes into a group, for which you can specify properties, such as exception, message, and timeout paths.

Chapter 15, “Handling Exceptions”

Describes exception handlers: global exception handlers, exception handlers on a block or group of nodes, exception handlers for individual nodes, and unhandled exceptions.

Chapter 16, “Adding Message Paths”

Describes how to use Message Paths to execute process nodes in a parallel path to a node or group of nodes after a certain message is received from a client or a resource (via a control). Message paths can be associated with individual nodes, a group of nodes, or with the process (global).

Chapter 17, “Adding Timeout Paths”

Describes how to use timeout paths to execute process nodes in a parallel path to a node or group of nodes after a certain amount of time has lapsed. Timeout paths can be associated with individual nodes, a group of nodes, or with the process (global).

Chapter 18, “Running and Testing Your Business Process”

Describes how you can compile and test a business process using the Test Browser tool.

Chapter 19, “Business Process Variables and Data Types”

Describes the data types supported in your business process application and how to create business process variables.

Chapter 20, “Versioning Business Processes”

Describes how you can make changes to your business process without interrupting any instances of the process that are currently running by using the WebLogic Workshop versioning feature.

Chapter 21, “Validating Schemas”

Describes the different methods you can use to validate your schemas.

Chapter 22, “Building Stateless and Stateful Business Processes”

Describes the differences between building Stateless and Stateful business processes.

Chapter 23, “Building Synchronous and Asynchronous Business Processes”

Describes the differences between building Synchronous and Asynchronous business processes. Also includes information on enabling synchronous clients to interact with business processes that have asynchronous interactions with resources.

Chapter 24, “Transaction Boundaries”

Describes the rules for implicit and explicit transaction boundaries and how to create explicit transaction boundaries.

Chapter 25, “Business Process Source Code”

Describes the source code WebLogic Workshop writes to a business process file (a JPD file), in keeping with your business process design in the graphical design environment.

Chapter 26, “Building ebXML Participant Business Processes”

Describes the template that you can use to build an ebXML participant business process in WebLogic Workshop.

Chapter 27, “Building RosettaNet Participant Business Processes”

Describes how to build public participant business processes for RosettaNet conversations using the RosettaNet participant business process file in WebLogic Workshop.

Related Topics

[How Do I: Use the Design View?](#)

[Calling Business Processes](#)

Creating a Business Process Application

WebLogic Integration extends the WebLogic Workshop graphical design environment to allow the building of integrated enterprise applications. An application in turn contains projects and files. A project can contain several components including, business processes, Web services, and XML files.

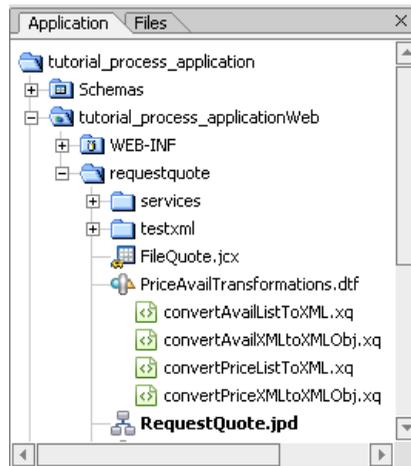
This section describes the components of an application, the steps you follow to create an application in WebLogic Workshop, and how to incorporate a business process in your application. It includes the following topics:

- [Components of Your Application](#)
- [Designing Your Application](#)

Components of Your Application

This section outlines some of the high-level components you create as you build your business process application and how they appear in the deployed application based on the names that you choose for these components. Note the following components:

Application—The components of the application you are creating are represented in a hierarchical tree structure on the **Application** pane in your WebLogic Workshop environment. If the **Application** pane is not visible in WebLogic Workshop, choose **View—Application** from the menu bar. An example **Application** pane is shown in the following figure:



J2EE applications and their components are deployed on the WebLogic Server as Enterprise Application Archive (EAR) files. The name you specify for the application becomes the name of the EAR file that you use to deploy your application.

Projects—Projects contained in your application represent WebLogic Server Web applications. That is, when you create a project, you are creating a Web application. The name of your project will be included in the URL your clients use to access your application. For example, the preceding figure represents an application named `tutorial_process_application`. It contains a project named **tutorial_process_applicationWeb**, which in turn contains a business process named **RequestQuote.jspd**. Clients can access your business process via the following URL:

`http://host:port/tutorial_process_applicationWeb/requestquote/RequestQuote.jspd`

In the preceding URL, *host* and *port* represent the name of your host server and the listening port.

Note: When you create a Process Application, a Web application (process project folder) is created in the application, by default. The default process project folder (and therefore, the Web application) is named using the name you specified for your application with the word **Web** appended to it (that is, `process_application_nameWeb`). If you create additional process projects (Web applications) in your Process Application, you can specify any name you want for them; the additional process projects will not include the **Web** suffix.

Schemas—To make the XML Schemas, MFL files, and Channel files in your application available in your business process, you must place them in a **Schemas** folder. When you create your process application or project using a template, the **Schemas** folders are created as child

folders of your business process application folder, as shown in the preceding figure. When you add XML Schemas and MFL files to the **Schemas** folder in your business process project, they are compiled to generate XML Beans. In this way, WebLogic Workshop generates a set of interfaces that represent 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.

To learn more, see [Importing Schemas](#).

Designing Your Application

You build your application in WebLogic Workshop by adding *projects* to an *application*. A project contains components of your application such as business processes, Web services, control files, and XML files.

Creating a Business Process Application

To quickly get started designing business processes, you can create an application that contains a basic business process file, which you can customize with your business process logic. To do so, complete the following procedure:

To Create a New Application

1. Choose **File**→**New**→**Application** from the WebLogic Workshop menu to display a **New Application** dialog box.
2. To create a business process application, select **Process** in the left pane in the dialog box. In the right pane, select **Process Application**.

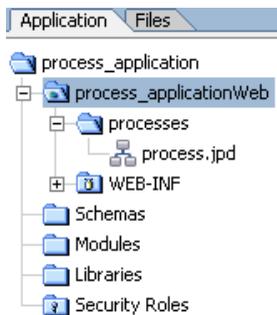
This creates an application that contains a basic business process project, which includes a business process file that contains only a Start and Finish node (`process.jpdl`).

Note: If you select **Tutorial: Process Application** instead of **Process Application**, WebLogic Workshop creates an application containing components for the Business Process and Data Transformation tutorials. To learn about taking the tutorial, see [Tutorial: Building Your First Business Process](#), and [Tutorial: Building Your First Data Transformation](#). You can also build an ebXML or RosettaNet participant business process in WebLogic Workshop by using specially created templates. For more information about how to create these participant processes, see [Building ebXML Participant Business Processes](#) and [Building RosettaNet Participant Business Processes](#).

3. Specify the directory in which to create the Application folder.

4. Specify a name for your new application.
5. In the **Server** field, select the sample integration domain or any other WebLogic Integration domain in which your application runs. Click **Browse** to browse the file system to find a WebLogic Server configuration file.
6. Click **Create**.

Your application is displayed on the **Application** tab in WebLogic Workshop, as shown in the following figure:



The [How Do I: Use the Design View?](#) topic briefly describes the components and tools you use to design your business process in the WebLogic Workshop graphical design environment.

Subsequent topics in this guide describe in detail how to design specific business process patterns, including tasks such as:

- Adding methods and callbacks to client nodes in your business process to create the interface between your business process and its clients.
- Adding controls to represent the interfaces with resources such as Web services, databases, and EJBs.
- Mapping disparate data types in your business process, using XML Schemas, and constructing sequences of XML elements over which your business process can iterate to perform specified activities.
- Viewing and editing the JPD file in the **Source View**.

To learn about these tasks and others, see [Topics Included in This Section](#).

Setting the Business Process Properties

There are several properties which you can view and configure for your business process in the **Property Editor** of your business process start node.

To Set the Business Process Properties

1. Select the **Start** node of the business process for which you want to configure the properties.
2. If the **Property Editor** is not visible in WebLogic Workshop, choose **View**→**Property Editor** from the menu bar.

In the **Property Editor**, the following properties are displayed: [general](#), [process](#), and [version](#).

general

- **name**—This is the name of your business process, which is displayed throughout the WebLogic Workshop application, including the WebLogic Integration Administration Console. You can change the name to anything you would like by clicking this property and entering a new name.
- **notes**—Enter any notes that you want associated with your business process by clicking this property and then clicking  to open the **Property Text Editor**. Notes entered in the editor will be also be displayed in the WebLogic Integration Administration Console.

process

- **freeze on failure**—When a business process fails and there is no exception handler configured to handle the exception thrown, the business process is placed into an aborted state and no recovery is possible. However, if the business process is configured to freeze on failure, the business process rolls back to the last commit point and the state is persisted if it fails. The process can then be restarted from the WebLogic Integration Administration Console. To configure a business process to freeze on failure: select **true** from the **freeze on failure** drop-down menu.

For more information about business process exception handlers, see [Handling Exceptions](#). For more information about how to unfreeze business processes in the WebLogic Integration Administration Console, see [Process Instance Monitoring](#) in *Managing WebLogic Integration Solutions* at the following URL:

<http://edocs.bea.com/wli/docs81/manage/processmonitoring.html>

- **persistence**—This property sets how a stateful business process is persisted. More specifically, it determines whether a conversation is maintained in memory or stored in a database repository. Normally, stateful processes are persisted to a database. However, you may want to use non-persistent stateful processes for the following:
 - When the native communication mechanism requires it.
 - When multiple send-receive operations need to be done in parallel.

- When the performance of a stateful process using a database does not meet performance goals.

To set the type of persistence, from the **persistent** drop-down menu:

- Select **always** when you want your process conversations saved in the database repository. These conversations can be recovered in the event of an abnormal shutdown or crash. This setting is the WebLogic Integration default.
- Select **never** when you do *not* want your process conversations saved in the database repository. These conversations *cannot* be recovered in the event of an abnormal shutdown or crash.
- Select **on overflow** when you want your process conversations saved in the database repository after reaching a certain number. Until this number is reached, conversations are non-persistent. To set the overflow, set the **Max Beans in Cache** deployment descriptor. To learn more about configuring deployment descriptors, see [EJB->Configuration->Descriptors](#) in the *WebLogic Server Administration Console Online Help* at the following URL:

```
http://edocs.bea.com/wls/docs81/ConsoleHelp/domain_ejbcomponent_conf  
ig_descriptors.htm
```

- **on sync failure**—This property only applies to your process if it is configured to be a synchronous subprocess, it is ignored for any other business processes. If a synchronous subprocess fails, the default behavior is to mark it as **rollback**, which causes both the subprocess and the parent process to rollback. However, if the **on sync failure** property is set to **rethrow**, only the subprocess is rolled back. To learn more about synchronous subprocesses and the **on sync failure** property, see [Working with Subprocesses](#).
- **retry count**—Specify how many times, after the first attempt, the process engine should try to execute the business process.

If your business process contains an asynchronous Client Request node or multiple Client Request nodes, any one of which is asynchronous, then you can set the **retry count** for the business process. You cannot set the **retry count** property for business processes that contain *only* synchronous Client Request nodes (that is, **Client Request with Return** nodes).

- **retry delay**—Specify the amount of time (in seconds) you want to pass before a retry is attempted.

If your business process contains an asynchronous Client Request node or multiple Client Request nodes, any one of which is asynchronous, then you can set the **retry delay** for the business process. You cannot set the **retry delay** property for business processes that

contain *only* synchronous Client Request nodes (that is, **Client Request with Return** nodes).

- **stateless**—This property is for viewing only, it cannot be edited. It displays whether your business process is stateless (property displays **true**) or stateful (property displays **false**). To learn more about stateless and stateful business processes, see [Building Stateless and Stateful Business Processes](#).
- **binding**—This property specifies whether the business process uses the Web service, ebXML, or RosettaNet protocol. The default value is **webservice**. If your business process is an **ebXML** or a **RosettaNet** process, select **ebxml** or **rosettanet**. In keeping with your selection in the **Property Editor**, an attribute is written to the `@jpd: process` annotation in the source code. For example:

```
@jpd:process binding="rosettanet" process::
```

To learn about ebXML and RosettaNet business processes, see [Building ebXML Participant Business Processes](#) and [Building RosettaNet Participant Business Processes](#).

version

- **strategy**—This describes how to invoke sub processes when different versions of the parent process exists. From the strategy drop-down menu:
 - Select **loosely-coupled** if you want the subprocess version to be set at the time that the sub process is invoked.
 - Select **tightly-coupled** if you want the subprocess version to be set at the time the parent process is invoked.

For more detailed information about the version strategy property, see [Configuring the New Versions of Your Business Process](#).

ebxml

For information about ebXML properties, see [@jpd:ebxml Annotation](#).

rosettanet

For information about RosettaNet properties, see [@jpd:rosettanet Annotation](#).

Related Topics

[How Do I: Start WebLogic Workshop?](#)

[How Do I: Start and Stop WebLogic Server?](#)

Creating a Business Process Application

[How Do I: Create a New Application?](#)

[How Do I: Create a New Project?](#)

[How Do I: Create a New Business Process File?](#)

[How Do I: Open an Existing Business Process?](#)

[How Do I: Use the Design View?](#)

[Handling Exceptions](#)

[Process Instance Monitoring at](#)

<http://edocs.bea.com/wli/docs81/manage/processmonitoring.html>

[Building Synchronous and Asynchronous Business Processes](#)

[Building Stateless and Stateful Business Processes](#)

[Versioning Business Processes](#)

Starting Your Business Process

This section describes how to design the first node in your business process (the **Start** node) to represent the starting point of a business process.

A business process can be started as a result of receiving a request from a client, as the result of receiving a message from a Message Broker channel to which the business process is subscribed (a business process can subscribe to channels to receive events from for example: File event generators, JMS event generators, and Timer Event Generators), and by a choice of one of several events. This section includes the following topics:

- [Designing Start Nodes](#)
- [Exception Handlers on Start Nodes](#)

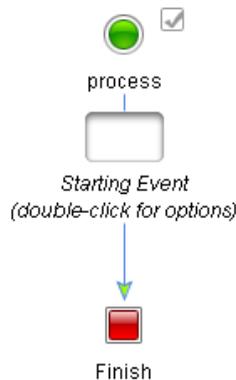
Related Topics

[How Do I: Call a Business Process?](#)

Designing Start Nodes

A **Start** node represents the starting point of a business process. Depending on the method by which your business process starts, the **Starting Event** of your process can contain any combination of **Client Request**, **Client Request with Return**, or **Subscription** nodes. You design the **Starting Event** of your process by double-clicking the *Starting Event* place holder placed just below your **Start** node.

To create a new business process, complete the tasks described in [Creating a Business Process Application](#). When you create a new business process, it initially contains an empty **Start** node, a *Starting Event* place holder, and a **Finish** node, as shown in the following figure:



The first action in the business process is specified at the **Start** node. That is, you specify how the business process is started at run time by defining a **Starting Event**. The empty node attached to the **Start** node, as well as the gray check box , shown in the preceding figure, indicate that the start method for this business process is not defined.

While you are building your business process by adding process nodes to it, you can go back to the start node to check the stateless status of your process. If your process at any time becomes stateful, the stateless property in the Start node property editor displays false. To learn more about stateless and stateful business processes, see [Building Stateless and Stateful Business Processes](#).

The Start Node also indicates any business-process-wide problems, such as when a control declaration has an error or when an incorrect variable type is used for a variable. Any such problems are indicated by an  appearing next to the Start Node. If you place your cursor over this icon, WebLogic Workshop will display a message about the problems.

To Define the Start Method for Your Business Process

You can design the start node properties by invoking the starting event node builder. Node builders provide a task-driven interface that allow you to specify the logic required at nodes in your business process.

1. Double-click the *Starting Event* placeholder on the **Start** node in the **Design View** to display the Start node builder.
2. In the node builder, select the method by which you want your business process to start:

– **Invoked via a Client Request**

Select this option if you want your business process to start as the result of receiving a message from a client.

– **Invoked synchronously via a Client Request with Return**

Select this option if you want your business process to start as the result of receiving a synchronous request from a client. Any nodes added between the receive and send nodes inside the Client Request with Return group will be executed within the scope of the synchronous operation.

– **Subscribe to a Message Broker channel and start via an Event (Time, Email, File, Adapter, etc.)**

Select this option if you want your business process to start as a result of receiving an asynchronous message from a Message Broker channel. You create a static subscription to a Message Broker channel on this node. This option also allows you to start your business process via an event through File, JMS, Email, or Timer controls, which facilitate publishing events to Message Broker channels.

Note: In WebLogic Integration, subscriptions to Message Broker channels defined at a Start node are referred to as static subscriptions, and subscriptions defined using a Message Broker Subscription control are referred to as dynamic subscriptions. See “Note about Static and Dynamic Subscriptions” in [@jpd:mb-static-subscription Annotation](#).

– **Subscribe synchronously to a Message Broker channel and start via an Event**

Select this option if you want your business process to start as a result of receiving a synchronous message from a Message Broker channel. You create a static subscription to a Message Broker channel on this node. This option also allows you to start your business process via an event through File, JMS, Email, or Timer controls, which facilitate publishing events to Message Broker channels.

– **Invoked via one of several Client Requests or Subscriptions (Event Choice)**

Select this option if you want your business process to start as a result of receiving one of a number of possible events. When an **Event Choice** node is used at the start of a business process, you can configure it to contain **Client Request**, **Client Request with Return**, or **Message Broker Subscription** nodes.

3. To close the node builder, click the **X** in the top right-hand corner.

The drop target on the **Start** node is populated with an icon representing the method by which the business process starts.

To learn more about specifying the appropriate start node for your business process, see:

- [Client Request Start \(Asynchronous\)](#)
- [Client Request with Return Start \(Synchronous\)](#)
- [Subscription Start \(Asynchronous\)](#)
- [Subscription Start \(Synchronous\)](#)
- [Event Choice Start](#)

Related Topics

[How Do I: Call a Business Process?](#)

Client Request Start (Asynchronous)

If you specified that your business process starts when it receives a message from a client, that is using the **Invoked via a Client Request** option, your **Start** node is displayed as shown in the following figure:



To Complete the Design of Your Client Request Node

1. Double-click the **Client Request** node to invoke the node builder for the **Client Request** node.
2. To complete the specification of events for your **Client Request** node, see [Design Your Client Request Node](#).

Related Topics

[Sending Messages to Clients](#)

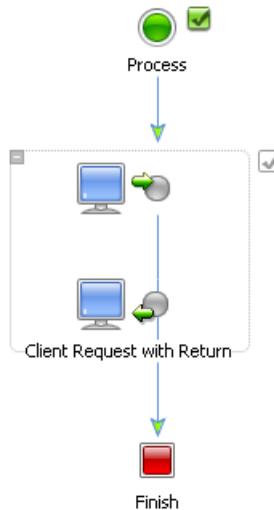
[Handling Exceptions](#)

[Client Operations and Control Communication Methods](#)

[How Do I: Call a Business Process?](#)

Client Request with Return Start (Synchronous)

If you specified that your business process starts when it receives a message from a client and a synchronous response is sent back to the client, that is using the **Invoked synchronously via a Client Request with Return** option, your **Start** node is displayed as shown in the following figure:



Note the following properties for the **Client Request with Return** group node:

- indicates that the design of this node is incomplete. To complete the design, see [To Complete the Design of Your Client Request with Return Node Group](#).
- By default the name for the node is **Client Request with Return**. You can change the name in the following ways:

- Right-click the node name in the **Design View** and select **Rename** from the drop-down menu. Then enter a new name to replace **Client Request with Return**.
- Double-click the node name in the **Design View**, then enter a new name to replace **Client Request with Return**.
- Double-click either of the **Client Request with Return** icons in your business process to display one of the node builders. Click the name beneath the node builder icon and enter a new name to replace **Client Request with Return**.

After you add any node to your business process, you can design its properties and behavior by invoking the node builder and completing the tasks appropriate for that node. You can also add optional nodes between the Request and Return part of the Client Request with Return node. This allows you to process data or perform tasks after the message from the client is received and before the return is sent back to the client. For more information on how to add optional nodes to your Client Request with Return node, see [Adding Nodes to Your Client Request with Return Node Group](#).

The following sections describe how to complete the design of your **Client Request with Return** nodes:

To Complete the Design of Your Client Request with Return Node Group

To complete the design of your **Client Request with Return** node, you need to complete the following sections:

- [Specify General Settings for the Request Part of Your Node Group](#)
- [Specify Receive Data Settings for the Request Part of Your Node Group](#)
- [Specify General Settings for the Return Part of Your Node Group](#)
- [Specify Send Data Settings for the Return Part of Your Node Group](#)

Specify General Settings for the Request Part of Your Node Group

1. Double-click the  icon (upper icon) in the **Client Request with Return** node group in your business process.

The request part of the node builder is displayed. It contains two tabs: **General Settings** and **Receive Data**.

2. In the **General Settings** tab, enter a name in the **Method Name** field to specify the name of the method on this **Client Request with Return** node.

The name you assign to the method is the name of the method that is exposed via the Web Services Description Language (WSDL) when you make your business process available as a Web service. To learn more about how the methods in your project are exposed to clients, see [Components of Your Application](#).

3. In the **General Settings** tab, click **Add** and select the type and format of the data your **Client Request with Return** node expects to receive from clients (that is, the data type for the method parameter). You can also specify a name for the method parameter.

4. Select the type and format of your data. The options available are:

- **XML Types**

Lists the XML Schemas that are available in your business process project and the untyped XMLObject and XMLObjectList data types. To learn how to import a Schema into your project, see [Importing Files into the Schemas Project](#).

- **Non-XML Types**

Lists the Message Format Language (MFL) files available in your business process project and the untyped RawData data type. WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe the structure of non-XML data. Every MFL file available in your project is listed in **Non-XML Types**. Note that an XML Schema representation of each MFL file is built by WebLogic Workshop and is also available in the XML Types listing.

- **Java Types**

Lists Java primitive data types.

For more detailed descriptions of the data types, see [Working with Data Types](#).

5. After you select the data type, click **OK**. The parameter type field is populated with that parameter type.

Note: If you selected typed XML or typed non-XML data type in the previous step, you can select the **Validate** box to have the incoming message validated against your specified schema before the message is received by the node. For more information about schemas, see [Validating Schemas](#) and [Importing Files into the Schemas Project](#).

6. In the **General Settings** tab, continue clicking **Add** and select the type and format of your data until you have added as many parameters as you want to use.
7. To remove a variable from the node builder pane, select the variable in the list and then click **Remove**.

Note: This action removes the variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

Specify Receive Data Settings for the Request Part of Your Node Group

1. Click the **Receive Data** tab.

This tab allows you to define one or more variables to hold the data that your business process receives from clients.

2. If the data types of your method parameters and the data type of the variables you are going to use match, you can map your variables to the corresponding methods directly.

- a. If not already selected, select the **Variable Assignment** option.

The **Client Sends** field is populated with the parameter(s) you specified on the **General Settings** tab.

- b. If you want to assign a variable that you have already created in your project to the method parameters, under **Select Variables to assign**, click the arrow in the drop-down list and select it from the menu. The variable you select is added to the node builder pane.
 - c. If you want to create a new variable and assign it to the method parameter, click the arrow in the drop-down list, select **Create new variable...**, then follow the instructions in the [To Create a New Variable in the Node Builder](#) section.
 - d. If the data types of your method parameters and your variables match, close the node builder by clicking the **X** in the top right-hand corner.
3. If the data types of your method parameters and your variables are different, you can use the data mapping tool included in WebLogic Integration to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Files (DTF). When DTFs 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.
 - a. To create a transformation map, select the **Transformation** option in the node builder. The node builder transformation window displays the data types expected by your method in the **Client Sends** pane.
 - b. In **Step 1** of the **Transformation** option window, click **Select Variable** to select one or more variables to be used.

Note: To remove a variable from the node builder pane, select the variable in the list and then click **Remove**. This action removes the variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

When designing a business process, you use a Transformation to create maps between disparate data types. Your project must contain an instance of a Transformation control (defined by a DTF file) for you to create the map.

- c. If an appropriate instance of a Transformation control is not available in your project, you can create a new one by clicking **Create Transformation** to invoke the **Transformation Mapping** tool window. This automatically applies changes to the builder and opens a transformation editor in a new window.

The mapping tool displays a representation of the source schema and target schema in **Source** and **Target** panes. You can create a map between the data type of the method parameter and the data type of the variable, or variables, to which you assign the data. To learn how to create and test a map using the mapping tool, see the [Guide to Data Transformation](#).

Note: To return to node builder, in the **Application** pane, double-click the JPD file.

- d. If the appropriate instance of a Transformation control is available in your project, click **Advanced...** in the node builder. The **Advanced Option** window opens. In this window select the **Control** and **Method**. If the method arguments and return type matches those as selected in the **Transformation** pane, click **OK**.
- e. To close the node builder, click the **X** in the top right-hand corner.

About Editing Node Configurations

You can edit the configuration at any node by opening the node builder and changing the existing specifications. If you add or remove variables in a node builder that already contains a configured transformation, you must edit or recreate the transformation. To do so, add or remove the variables, then click **Edit Transformation** or **Create Transformation**.

Note: When selecting a variable in a node builder's Transformation pane, and then clicking **Remove**, removes the selected variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

Specify General Settings for the Return Part of Your Node Group

1. Double-click the  icon (lower icon) in the **Client Request with Return** node in your business process.

The request part of the node builder is displayed. It contains two tabs: **General Settings** and **Send Data**.

2. In the **General Settings** tab, enter a name in the **Method Name** field to specify the name of the method on this **Client Receive with Return** node.

The name you assign to the method is the name of the method that is exposed via the Web Services Description Language (WSDL) when you make your business process available as a Web service. To learn more about how the methods in your project are exposed to clients, see [“Components of Your Application” on page 2-1](#).

3. In the **General Settings** tab, click **Select** and select the type and format of the data your **Client Request with Return** node expects to send to clients (that is, the data type for the return value).
4. Select the type and format of your data. The options available are:

- **XML Types**

Lists the XML Schemas that are available in your business process project and the untyped XMLObject data type. To learn how to import a Schema into your project, see [Importing Files into the Schemas Project](#).

- **Non-XML Types**

Lists the Message Format Language (MFL) files available in your business process project and the untyped RawData data type. WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe the structure of non-XML data. Every MFL file available in your project is listed in **Non-XML Types**. Note that an XML Schema representation of each MFL file is built by WebLogic Workshop and is also available in the XML Types listing.

- **Java Types**

Lists Java primitive data types.

For more detailed descriptions of the data types, see [Working with Data Types](#).

5. After you select the data type, click **OK**. The return type field is populated with the parameter types you added in the preceding steps.

Specify Send Data Settings for the Return Part of Your Node Group

1. Click the **Send Data** tab.

This tab allows you to define one or more variables to hold the data your business process send to clients.

2. If the data types of your return value and the data type of the variables you are going to use match, you can map your variables to the corresponding return value directly.

- a. If not already selected, select the **Variable Assignment** option.

The **Client Expects** field is populated with the return type you specified on the **General Settings** tab.

- b. If you want to assign a variable that you already created in your project to the return value, select it from the drop-down menu.
 - c. If you want to create a new variable and assign it to the method parameter, select **Create new variable...**, then follow the instructions in the [To Create a New Variable in the Node Builder](#) section.
 - d. If the data types of your return value and your variables match, close the node builder by clicking the **X** in the top right-hand corner.

3. If the data types of your return value and your variables are different, you can use the data mapping tool included in WebLogic Workshop to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files. When DTF 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.

- a. To create a transformation map, select the **Transformation** option.

The node builder transformation window displays the data types expected by your method displayed in the **Client Expects** pane.

- b. In **Step1** of the **Transformation** option window, click **Select Variable** to select one or more variables to be used.

Note: To remove a variable from the node builder pane, select the variable in the list and then click **Remove**. This action removes the variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

When designing a business process, you use a Transformation to create maps between disparate data types. Your project must contain an instance of a Transformation control (defined by a DTF file) for you to create the map.

- c. If an appropriate instance of a Transformation control is not available in your project, you can create a new one by clicking **Create Transformation** to invoke the **Transformation Mapping** tool window. This automatically applies changes to the builder and opens a transformation editor in a new window.

The mapping tool displays a representation of the source schema and target schema in **Source** and **Target** panes. You can create a map between the data type of the method parameter and the data type of the variable, or variables, to which you assign the data. To learn how to create and test a map using the mapping tool, see [Guide to Data Transformation](#).

Note: To return to node builder, in the **Application** pane, double-click the JPD file.

- d. If the appropriate instance of a Transformation control is available in your project, click **Advanced...**. The Advanced Option window opens. In this window select the **Control** and **Method**. If the method arguments and return type matches those as selected in the **Transformation** pane, click **OK**.
- e. To close the **Mapping tool** window, click the **X** in the top right-hand corner.

Note: To learn about changing the configuration you design in the Transformation pane of a node builder, see [About Editing Node Configurations](#).

4. To close the node builder, click the **X** in the top right-hand corner.

In the **Design View**, the  icon indicates that you completed the configuration and design of this node.

5. To save your work, select **File—Save**.

Adding Nodes to Your Client Request with Return Node Group

The **Client Request with Return** node functions as a combination of a **Client Request** node and a **Client Receive** node within a synchronous interaction. As such, you can add additional nodes in between the request and the return part of your **Client Request Node** but you cannot add any nodes that *wait* or *block*. To add a node to your **Client Request with Return** node, select the node you want to add in the Palette and drag and drop it into your **Client Request with Return** node.

The following nodes can be added:

- Client Response (See [Create a Client Response Node in Your Business Process](#).)

- Control Send (See [Create Control Nodes in Your Business Process.](#))
- Control Send with Return (See [Create Control Nodes in Your Business Process.](#))
- Perform (See [To Create a Perform Node in Your Business Process.](#))
- Decision (See [To Create a Decision Node in Your Business Process.](#))
- Switch (See [Creating Case Statements.](#))
- While Do (See [To Add A While group to Your Business Process.](#))
- Do While (See [To Add A While group to Your Business Process.](#))
- For Each (See [To Add A For Each Node to Your Business Process.](#))

Naming the Methods on Client Request with Return Nodes

The names that you assign to methods on your **Client Request with Return** nodes correspond to the names of the methods that are exposed via the Web Services Description Language (WSDL) when you make your business process available as a Web service. The name must be a valid Java class name.

Related Topics

[Sending Messages to Clients](#)

[XQuery Statements](#)

[Handling Exceptions](#)

[Client Operations and Control Communication Methods](#)

[How Do I: Call a Business Process?](#)

Subscription Start (Asynchronous)

If you specified that your business process is started via the **Subscribe to a Message Broker channel and start via an Event (Time, Email, File, Adapter, etc.)** option (see [To Define the Start Method for Your Business Process](#)), your **Start** node is displayed as shown in the following figure:



A static subscription to a Message Broker channel is defined on the **Subscription** node. Your business process is started as the result of receiving a message from a Message Broker channel.

Note: In WebLogic Integration, subscriptions to Message Broker channels defined at a Start node are referred to as static subscriptions, and subscriptions defined using a Message Broker Subscription control are referred to as dynamic subscriptions. See “Note about Static and Dynamic Subscriptions” in [@jpd:mb-static-subscription Annotation](#).

The following sequence concisely describes the message flow at run time:

1. A service publishes a message to a Message Broker channel, using a MB (Message Broker) Publish control, a File event generator, Timer event generator, or a JMS event generator. To learn more about how events are published to Message Broker channels, see [Message Broker Publish Control](#) and [Using Event Generators to Publish to Message Broker Channels](#).
2. A business process instance subscribes to, and receives messages from the Message Broker channel via the **Subscription** node. To ensure scalability of your application, the inbound messages by default are buffered on the queue for the current business process. To learn about buffering, see [Buffering Client Messages](#).

Note: An asynchronous subscription start causes the subscribed business process to run in a different transaction from the publisher’s transaction. In general, this is the recommended design pattern to use when you want to design your business process to start when it receives a message from a Message Broker channel. To learn about the scenarios for which a synchronous subscription start is recommended, see “[About Choosing Synchronous or Asynchronous Subscription Start Nodes](#)” on page 3-19.

To Complete the Design of Your Subscription Start Node

1. Double-click the **Subscription** node associated with the **Start** node in your business process to invoke the **Subscription** node builder.

Tabs on the node builder include:

- **General Settings**
- **Specify Filter**
- **Receive Data**

The following steps describe the tasks available on these tabs.

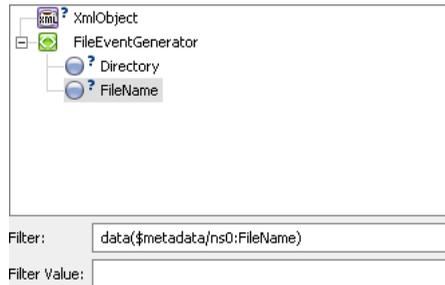
2. Complete the following tasks on the **General Settings** tab:
 - a. In the **Method Name** field, enter a name for the subscription request method.

The data type and format of the data your subscription request method (that is, the data type for the method parameter) is specified automatically, based on the configuration of your channel file.
 - b. Select a channel name from the drop-down list of Message Broker channels associated with the **Channel Name** field.

Note: If no appropriate channels are available for you to select, you must create a file that specifies the Message Broker channels for your application. To learn how to create this file, see [How Do I Create Message Broker Channels?](#)
3. Click the **Specify Filter** tab.

Specifying a filter is optional. Filters can be applied to the data type the business process receives from the channel, or when you have specified a qualified metadata type in your channel configuration.

The field in the **Specify Filter** tab is populated with the data type for the subscription method parameter you specified on the preceding tab. If you specified your channel to be able to receive qualified metadata, the **Qualified Metadata** attribute is also listed and you can filter on that parameter instead.



To specify a filter:

- a. Select the input type or schema element on which you want to filter.
- b. An XQuery expression is generated, and the **Filter** field is populated with the XQuery expression based on your selection in the preceding step.

Note: If you want to filter on an XMLObject parameter, you have to enter the XQuery statement in the **Filter** field or edit your source code directly.

- c. In the **Filter Value** field, enter a value against which you want to match the filter.

4. Click the **Receive Data** tab.

This tab allows you to define one or more variables to hold the data that your business process receives from the channel.

5. If the data types of your method parameters and the data type of the variables you are going to use are the same, you can map your variables to the corresponding methods directly.

- a. If it is not already selected, select the **Variable Assignment** option.

The **Client Sends** field is populated with the parameter(s) you specified on the **General Settings** tab, in other words, the parameter type of the channel.

- b. If you want to assign a variable that you already created in your project to the method parameters, select it from the drop-down menu.
- c. If you want to create a new variable and assign it to the method parameter, select **Create new variable...**, then follow the instructions in the [To Create a New Variable in the Node Builder](#) section.
- d. If the data types of your method parameters and your variables match, click the **X** in the top right-hand corner to close the node builder.

6. If the data types of your method parameters and your variables are different, you can use the data mapping tool included in WebLogic Integration to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files. When DTF 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.

- a. To create a transformation map, select the **Transformation** option.

The node builder transformation screen is displayed; the data types expected by your method are displayed in the **Client Sends** pane.

- b. In **Step 1** on the **Transformation** option window, click **Select Variable** to select one or more variables to be used.

Note: To remove a variable from the node builder pane, select the variable in the list and then click **Remove**. This action removes the variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

When designing a business process, you use a Transformation to create maps between disparate data types. Your project must contain an instance of a Transformation control (defined by a DTF file) for you to create the map.

- c. If an appropriate instance of a Transformation control is not available in your project, you can create a new one by clicking **Create Transformation** to invoke the transformation tool. This automatically applies changes to the builder and opens the transformation tool in a new window.

The transformation tool displays a representation of the source schema and target schema in **Source** and **Target** panes. You can create a map between the data type of the method parameter and the data type of the variable, or variables, to which you want to assign the data. To learn how to create and test a map using the mapping tool, see [Guide to Data Transformation](#).

Note: To return to node builder, in the **Application** pane, double-click the JPD file.

- d. If the appropriate instance of a Transformation control is available in your project, click **Advanced...** The Advanced Option window opens. In this window select the **Control** and **Method**. If the method arguments and return type matches those as selected in the **Transformation** pane, click **OK**.
- e. Close the Transformation tool by clicking the **X** in the top right-hand corner.

Note: To learn about changing the configuration you design in the Transformation pane of a node builder, see [About Editing Node Configurations](#).

7. To close the node builder, click the **X** in the top right-hand corner.

In the **Design View**, the  icon indicates that you completed the configuration and design of this node. To learn about buffering on your subscription node, see [Buffering Client Messages](#).

8. To save your work, select **File→Save**.

Related Topics

[Subscription Start \(Synchronous\)](#)

[Sending Messages to Clients](#)

[Handling Exceptions](#)

[Client Operations and Control Communication Methods](#)

[How Do I Create Message Broker Channels?](#)

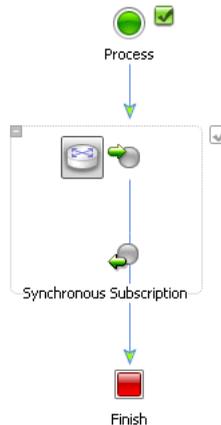
[How Do I: Call a Business Process?](#)

[Message Broker Publish Control](#)

[Using Event Generators to Publish to Message Broker Channels](#)

Subscription Start (Synchronous)

If you specified that your business process is started via the **Subscribe synchronously to a Message Broker channel and start via an event** option (see [To Define the Start Method for Your Business Process](#)), your **Start** node is displayed as shown in the following figure:



A synchronous static subscription to a Message Broker channel is defined on the **Synchronous Subscription** node. Your business process is started as the result of receiving a synchronous message from a Message Broker channel.

Note: In WebLogic Integration, subscriptions to Message Broker channels defined at a Start node are referred to as static subscriptions, and subscriptions defined using a Message Broker Subscription control are referred to as dynamic subscriptions. See “Note about Static and Dynamic Subscriptions” in [@jpd:mb-static-subscription Annotation](#).

The following sequence summarizes the message flow at run time for the scenario in which you design a **Synchronous Subscription** node at the start of your business process:

1. A service publishes a message to a Message Broker channel, using a MB (Message Broker) Publish control, a File event generator, Timer event generator, or a JMS event generator. To learn more about how events are published to Message Broker channels, see [Message Broker Publish Control](#) and [Using Event Generators to Publish to Message Broker Channels](#).
2. A business process instance subscribes to, and receives messages from, the Message Broker channel via the **Synchronous Subscription** node.

About Choosing Synchronous or Asynchronous Subscription Start Nodes

In general, an asynchronous subscription start pattern is recommended because it causes the subscribed business process to run in a different transaction from the publisher’s transaction. In contrast, a synchronous subscription start causes the subscribed business process to run in the same transaction as the publisher. This type of subscription decreases loose coupling and can associate the results of a transaction rollback of one subscriber with an otherwise independent

subscriber. However, there are two scenarios in which the synchronous subscription start pattern is recommended:

- JMS event generators publish to Message Broker channel which has one subscriber.

When a JMS event generator publishes to a channel that is known to have one subscriber, it generally improves performance to use the synchronous subscription start method on the subscriber. Note that in this case, the subscriber is doing work on the event generator thread, so you should adjust the event generator thread count accordingly.

- JMS event generators and subscribers use the **suppressible** attribute.

Setting **suppressible** to **true** specifies that the static subscription is suppressed in favor of dynamic subscriptions. In other words, you use **suppressible=true** to prevent specific messages on a Message Broker channel from starting a new business process; instead the messages can be received, using a dynamic subscription, by a business process that is already running.

To learn about this scenario, see [Using the Suppressible Attribute for a Static Subscription](#).

To Complete the Design of Your Synchronous Subscription Start Node

1. Double-click the **Subscription** node associated with the **Start** node in your business process to invoke the **Subscription** node builder.

Note: You can configure only the node that represents the message received by the business process. That is, you can only invoke a node builder for the first of the icons in the pair that represents the Synchronous Subscription Start node.

Tabs on the node builder include:

- **General Settings**
- **Specify Filter**
- **Receive Data**

The following steps describe the tasks available on these tabs.

2. Complete the following tasks on the **General Settings** tab:
 - a. Select a channel name from the drop-down list of Message Broker channels associated with the **Channel Name** field.

Note: If no appropriate channels are available for you to select, you must create a file that specifies the Message Broker channels for your application. To learn how to create this file, see [How Do I Create Message Broker Channels?](#)

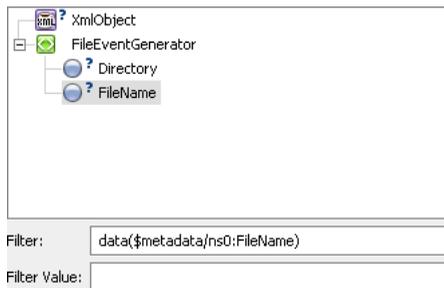
- b. In the **Method Name** field, enter a name for the subscription request method.

The data type and format of the data your subscription request method (that is, the data type for the method parameter) is specified automatically, based on the configuration of your channel file.

3. Click the **Specify Filter** tab.

Specifying a filter is optional. Filters can be applied to the data type the business process receives from the channel, or when you have specified a qualified metadata type in your channel configuration.

The field in the **Specify Filter** tab is populated with the data type for the subscription method parameter you specified on the preceding tab. If you specified your channel to be able to receive qualified metadata, the **Qualified Metadata** attribute is also listed and you can filter on that parameter instead.



To specify a filter:

- a. Select the data type on which you want to filter.
- b. An XQuery expression is generated, and the **Filter** field is populated with the XQuery expression based on your selection in the preceding step.

Note: If you want to filter on an XMLObject parameter, you will have to enter the XQuery statement in the **Filter** field or edit your source code directly.

- c. In the **Filter Value** field, create a value against which you want to match the filter.
 4. Click the **Receive Data** tab.
- This tab allows you to define one or more variables to hold the data that your business process receives from the channel.
5. If the data types of your method parameters and the data type of the variables you are going to use are the same, you can map your variables to the corresponding methods directly.

- a. If it is not already selected, select the **Variable Assignment** option.

The **Client Sends** field is populated with the parameter(s) you specified on the **General Settings** tab, in other words, the parameter type of the channel.

- b. If you want to assign a variable that you already created in your project to the method parameters, click **Select Variable** and select it from the drop-down menu. The variable you select is added to the node builder pane.
 - c. If you want to create a new variable and assign it to the method parameter, select **Create new variable...**, then follow the instructions in the [To Create a New Variable in the Node Builder](#) section.
 - d. If the data types of your method parameters and your variables match, close the node builder by clicking **X** in the top right-hand corner.
6. If the data types of your method parameters and your variables are different, you can use the data mapping tool included in WebLogic Integration to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files. When DTF 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.
 - a. To create a transformation map, select the **Transformation** option.

The node builder transformation pane is displayed; the data types expected by your method are displayed in the **Client Sends** pane.
 - b. In **Step 1** on the **Transformation** option pane, click **Select Variable** to select one or more variables to be used.

Note: To remove a variable from the node builder pane, select the variable in the list and then click **Remove**. This action removes the variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

When designing a business process, you use a Transformation to create maps between disparate data types. Your project must contain an instance of a Transformation control (defined by a DTF file) for you to create the map.

- c. If an appropriate instance of a Transformation control is not available in your project, you can create a new one by clicking **Create Transformation** to invoke the **Transformation Mapping** tool. This automatically applies changes to the builder and opens the transformation tool in a new window.

The transformation tool displays a representation of the source schema and target schema in **Source** and **Target** panes. You can create a map between the data type of the method parameter and the data type of the variable, or variables, to which you want to assign the data. To learn how to create and test a map using the mapping tool, see [Guide to Data Transformation](#).

Note: To return to node builder, in the **Application** pane, double-click the JPD file.

- d. If the appropriate instance of a Transformation control is available in your project, click **Advanced...**. The Advanced Option window opens. In this window select the **Control** and **Method**. If the method arguments and return type matches those as selected in the **Transformation** pane, click **OK**.

- e. To close the node builder, click the **X** in the top right-hand corner.

Note: To learn about changing the configuration you design in the Transformation pane of a node builder, see [About Editing Node Configurations](#).

In the **Design View**, the check box icon indicates that you completed the configuration and design of this node.

7. To save your work, select **File—Save**.

Related Topics

[Subscription Start \(Asynchronous\)](#)

[Sending Messages to Clients](#)

[Handling Exceptions](#)

[Client Operations and Control Communication Methods](#)

[How Do I Create Message Broker Channels?](#)

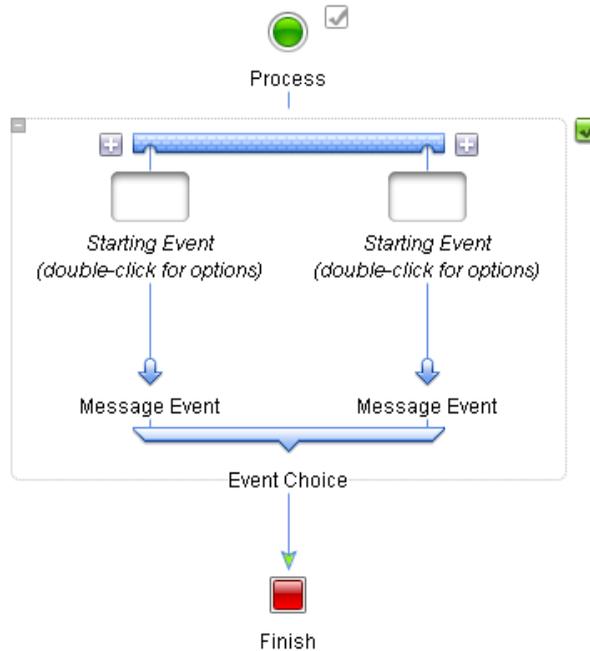
[How Do I: Call a Business Process?](#)

[Message Broker Publish Control](#)

[Using Event Generators to Publish to Message Broker Channels](#)

Event Choice Start

If you specified that your business process is **Invoked via one of several Client Requests or Subscriptions (Event Choice)**, (see [To Define the Start Method for Your Business Process](#)), your **Start** node is displayed as shown in the following figure:



By default, **Event Choice** nodes are created with two branches. Click **+** to create additional branches. A new branch is added on the left or right of the existing branches.

You can add additional nodes to the paths in your **Event Choice** node to specify the events executed at run time after the business process starts. The *Starting Event* targets at the start of each branch indicate that only certain nodes are allowed at these locations: specifically, when you use an **Event Choice** node at the start node in your business process, it can contain only **Client Request**, **Client Request with Return** or **Subscription** nodes.

Note: When you create an **Event Choice** node at locations other than the **Start** node in your business process, it can contain **Client Request** nodes and **Control Receive** nodes. To learn more about designing **Event Choice** nodes, see [Receiving Multiple Events](#).

To Complete the Design of Your Event Choice Start Node

To specify the events to be executed on each branch of your **Event Choice Start** node, complete the following tasks for each branch of the node:

1. Double-click the *Starting Event* placeholder to invoke the node builder.



2. From the node builder, select the event for which this branch waits:
 - **A Client Request**
 - **A Client Request with Return**
 - **A Message Broker Subscription**
 - **A Synchronous Message Broker Subscription**
3. Close the node builder by clicking the **X** in the top right-hand corner.
The drop target on your **Event Choice** branch is changed to reflect the event you specified.
4. To complete the specification of events, double-click the event nodes on the **Event Choice** branches to invoke the associated node builder:
 - To learn how to use the node builder to complete the **Client Request** node, see [Design Your Client Request Node](#).
 - To learn how to use the node builder to complete the **Client Request with Return** node, see [To Complete the Design of Your Client Request with Return Node Group](#).
 - To learn how to use the node builder to complete the **Message Broker Subscription** node, see [To Complete the Design of Your Subscription Start Node](#).
 - To learn how to use the node builder to complete the **Synchronous Message Broker Subscription** node, see [To Complete the Design of Your Synchronous Subscription Start Node](#).
5. To save your work, select **File–Save**.

Related Topics

[How Do I: Create a New Business Process File?](#)

[How Do I: Open an Existing Business Process?](#)

[Message Broker Controls](#)

[File Control](#)

[JMS Control](#)

[Business Process Source Code](#)

[Handling Exceptions](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

[How Do I: Call a Business Process?](#)

Exception Handlers on Start Nodes

You can create a global exception handler for your business process by creating an *exception path* for the **Start** node. You create the logic for the exception handler path to define the flow of execution in the case when an exception is thrown by your business process. A global exception handler responds to exceptions that are otherwise not handled in the business process.

To learn how to create exception handler paths on **Start** nodes, see [Handling Exceptions](#).

Interacting With Clients

Clients invoke business processes to perform one or more operations. Business Processes expose their functionality through methods.

Client Request nodes represent the points in a business process at which a client invokes a method on the business process and possibly sends input to the business process. The names you assign to methods on **Client Request** nodes correspond to the names of the methods that are exposed via the Web Services Description Language (WSDL) when you make your business process available as a Web service.

Note: The nodes in a business process are always communicating asynchronously with clients, except for when you invoke a Start node of a business process by using the **Client Request with Return** option or configure the starting event on a Message path to wait for a **Client Request with Return**. To learn more about using the Client Request with Return node, see [Client Request with Return Start \(Synchronous\)](#).

Client Response nodes represent the points in a business process at which business processes send messages to clients.

This section describes how to add nodes to your business process and design the interactions of business processes with clients. It includes the following topics:

- [Receiving Messages From Clients](#)—Design nodes in your business process to receive asynchronous messages from clients. A business process can be started as a result of receiving a message from a client.
- [Sending Messages to Clients](#)—Design nodes in your business process to send asynchronous messages to clients.

- [Buffering Client Messages](#)—Design your business process in such a way that the messages sent to clients from Client Response nodes are buffered.

Receiving Messages From Clients

Client Request nodes provide a way for a client to make a request to a business process.

The tasks you must complete to design a Client Request node include:

- [Create a Client Request Node in Your Business Process](#)
- [Design Your Client Request Node](#)
- [Naming the Methods on Client Request Nodes](#)

Create a Client Request Node in Your Business Process

1. On the **Application** tab, click the business process (JPD file) you want to design.
Your business process is displayed in the **Design View**.
2. If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the WebLogic Workshop menu.
3. Drag and drop  **Client Request** from the **Palette** onto the business process in the **Design View**, placing it on the business process at the point at which you want to design the client interaction.

Note: As you drag your selection onto the **Design View**, targets  appear on your business process. Each target represents a location in the flow where you can place the node. As you drag the node near a 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. If the location you chose is not a valid one, an  will appear next to your node. If you place your cursor over this icon, WebLogic Workshop will display a message about the violation.

The Client Request node is displayed in your business process in the **Design View**.

Note the following properties for the Client Request node:

- indicates that the design of this node is incomplete. To complete the design, see [Design Your Client Request Node](#).
- By default the name for the node is Client Request. You can change the name in the following ways:
 - Double-click the node name in the **Design View** and enter a new name to replace **Client Request**.
 - Right-click the node name in the **Design View** and select **Rename** from the drop-down menu. Then enter a new name to replace **Client Request**.
 - Double-click the **Client Request** node in your business process to display the node builder. Click the name beneath the node builder icon and enter a new name to replace **Client Request**.

Design Your Client Request Node

After you add any node to your business process, you can design its properties and behavior by invoking the node builder and completing the tasks appropriate for that node. The following sections describe how to complete the design of interactions with clients in your **Client Request** nodes:

- [To Specify General Settings](#)
- [To Specify Receive Data](#)

To Specify General Settings

1. Double-click the **Client Request** node in your business process.

The node builder is displayed. It contains two tabs: **General Settings** and **Receive Data**.

2. In the **General Settings** tab, enter a name in the **Method Name** field to specify the name of the method on this **Client Receive** node.

The name you assign to the method is the name of the method that is exposed via the Web Services Description Language (WSDL) when you make your business process available as a Web service. To learn more about how the methods in your project are exposed to clients, see [Components of Your Application](#).

3. In the **General Settings** tab, click **Add** to select the type and format of the data your **Client Request** node expects to receive from clients (that is, the data type for the method parameter). The node builder displays the following types of data:
 - [XML Types](#)

Lists the XML Schemas that are available in your business process project and the untyped XMLObject and XMLObjectList data types. To learn how to import a Schema into your project, see [Importing Files into the Schemas Project](#).

– **Non-XML Types**

Lists the Message Format Language (MFL) files available in your business process project and the untyped RawData data type. WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe the structure of non-XML data. Every MFL file available in your project is listed in **Non-XML Types**. Note that an XML Schema representation of each MFL file is built by WebLogic Workshop and is also available in the XML Types listing.

– **Java Types**

Lists Java primitive data types.

For more detailed descriptions of the data types, see [Working with Data Types](#).

4. Click **OK**.

After you select a data type, the field is populated with the parameter types you added in the preceding steps.

Note: If you selected a typed XML or typed non-XML data type in the previous steps, you can select the **Validate** box to have the incoming message validated against your specified schema before the message is received by the node. For more information about schemas, see [Validating Schemas](#) and [Importing Files into the Schemas Project](#).

To Specify Receive Data

1. Click the **Receive Data** tab.

This tab allows you to define one or more variables to hold the data your business process receives from clients.

2. If the data types of your method parameters and the data type of the variables you are going to use match, you can map your variables to the corresponding methods directly.

a. If not already selected, select the **Variable Assignment** option.

The **Client Sends** field is populated with the parameter(s) you specified on the **General Settings** tab.

b. If you want to assign a variable that you already created in your project to the method parameters, select it from the drop-down menu.

- c. If you want to create a new variable and assign it to the method parameter, select **Create new variable...**, then follow the instructions in [To Create a New Variable in the Node Builder](#).
 - d. If the data types of your method parameters and your variables match, click the **X** in the top right-hand corner to close the node builder.
3. If the data types of your method parameters and your variables are different, you can use the Transformation tool included in WebLogic Workshop to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files. When DTF 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.

- a. To create a transformation map, select the **Transformation** option.

The node builder transformation screen is displayed with the data types expected by your method displayed in the **Client Sends** pane.

- b. In **Step 1** of the **Transformation** option window, click **Select Variable** to select one or more variables to be used.

Note: To remove a variable from the node builder pane, select the variable in the list and then click **Remove**. This action removes the variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

When designing a business process, you use a Transformation to create maps between disparate data types. Your project must contain an instance of a Transformation control (defined by a DTF file) for you to create the map.

- c. If an appropriate instance of a Transformation control is not available in your project, you can create a new one by clicking **Create Transformation** to invoke the **Transformation Mapping** tool window. This automatically applies changes to the builder and opens a transformation editor in a new window.

The mapping tool displays a representation of the source schema and target schema in **Source** and **Target** panes. You can create a map between the data type of the method parameter and the data type of the variable, or variables, to which you assign the data. To learn how to create and test a map using the mapping tool, see [Guide to Data Transformation](#).

Note: To return to node builder, in the **Application** pane, double-click the JPD file.

- d. If the appropriate instance of a Transformation control is available in your project, click **Advanced...** The Advanced Option window opens. In this window, select the **Control** and **Method**. If the method arguments and return type matches those as selected in the **Transformation** pane, click **OK**.
 - e. Close the Transformation tool by clicking the **X** in the top right-hand corner.
4. To close the node builder, click the **X** in the top right-hand corner.

In the **Design View**, the icon indicates that you completed the configuration and design of this node.

Note: To learn about changing the configuration you design in the Transformation pane of a node builder, see [About Editing Node Configurations](#).

5. To save your work, select **File—Save**.

Naming the Methods on Client Request Nodes

The names that you assign to methods on your **Client Request** nodes correspond to the names of the methods that are exposed via the Web Services Description Language (WSDL) when you make your business process available as a Web service. The name must be a valid Java class name.

Related Topics

[Sending Messages to Clients](#)

[Buffering Client Messages](#)

[XQuery Statements](#)

[Handling Exceptions](#)

[Client Operations and Control Communication Methods](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

Sending Messages to Clients

Client Response nodes provide a way for a business process to send messages to clients. The tasks you must complete to design a **Client Response** node include:

- [Create a Client Response Node in Your Business Process](#)
- [Design Your Client Response Node](#)

Create a Client Response Node in Your Business Process

1. On the **Application** tab, click the JPD file you want to design.
Your business process is displayed in the **Design View**.
2. If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the WebLogic Workshop menu.
3. Click  **Client Response** in the **Palette**.
4. Drag and drop the **Client Response** node onto the business process in the **Design View**, placing it on the business process at the point in your business process at which you want to send a message to a client.

Note: As you drag your selection onto the **Design View**, targets  appear on your business process. Each target represents a location in the flow where you can place the node. As you drag the node near a 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. If the location you chose is not a valid one, an  will appear next to your node. If you place your cursor over this icon, WebLogic Workshop will display a message about the violation.

The **Client Response** node is displayed in your business process in the **Design View**.

Note the following properties for the **Client Response** node:

- indicates that the design of this node is incomplete. To complete the design, see [Design Your Client Response Node](#).
- By default the name for the node is **Client Response**. You can change the name in the following ways:
 - Double-click the node name in the **Design View** and enter a new name to replace **Client Response**.

- Right-click the node name in the **Design View** and select **Rename** from the drop-down menu. Then enter a new name to replace **Client Response**.
- Double-click the **Client Response** node in your business process to display the node builder. Click the name beneath the node builder icon and enter a new name to replace **Client Response**.

Design Your Client Response Node

The following sections describe how to complete the design of interactions with clients in your **Client Response** nodes:

- [To Specify General Settings](#)
- [To Specify Send Data](#)

To Specify General Settings

1. Double-click the **Client Response** node in your business process.

The node builder is displayed. It contains two tabs: **General Settings** and **Send Data**.

2. In the **General Settings** tab, enter a name in the **Method Name** field to specify the name of the method on this **Client Response** node.
3. In the **General Settings** tab, click **Add** to specify the type and format of the data your business process sends to clients via the **Client Response** node (that is, the data type for the method parameter). The node builder displays the following types of data:

- **XML Types**

Lists the XML Schemas that are available in your business process project. To learn how to import a Schema into your project, see [Importing Files into the Schemas Project](#).

- **Non-XML Types**

Lists the Message Format Language (MFL) files available in your business process project. WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe the structure of non-XML data. Every MFL file available in your project is listed in **Non-XML Types**. Note that an XML Schema representation of each MFL file is built by WebLogic Workshop and is also available in the XML Types listing.

- **Java Types**

Lists Java primitive data types, and XMLObject, XMLObjectList, and RawData types.

To learn more about data types, see [Working with Data Types](#).

4. Click **OK**.

After you select a data type from the list of supported types, the field is populated.

To Specify Send Data

1. Click the **Send Data** tab.

This tab allows you to define one or more variables to hold the data your business process sends to clients.

2. If the data types of your method parameters and the data type of the variables you are going to use match, you can map your variables to the corresponding methods directly.

- a. If not already selected, select the **Variable Assignment** option.

The **Client Expects** field is populated with the parameter(s) you specified on the **General Settings** tab.

- b. If you want to assign a variable that you already created in your project to the method parameters, select it from the drop-down menu.
 - c. If you want to create a new variable and assign it to the method parameter, select **Create new variable...**, then follow the instructions in [To Create a New Variable in the Node Builder](#).
 - d. If the data types of your method parameters and your variables match, close the node builder by clicking the **X** in the top right-hand corner

3. If the data types of your method parameters and your variables are different, you can use the data mapping tool included in WebLogic Workshop to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files. When DTF 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.

- a. To create a transformation map, select the **Transformation** option.

The node builder transformation screen is displayed with the data types expected by your method displayed in the **Client Expects** pane.

- b. In **Step 1** in the **Transformation** tab, click **Select Variable** to select one or more variables to be used.

Note: To remove a variable from the node builder pane, select the variable in the list and then click **Remove**. This action removes the variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

When designing a business process, you use a Transformation to create maps between disparate data types. Your project must contain an instance of a Transformation control (defined by a DTF file) for you to create the map.

- c. If an appropriate instance of a Transformation control is not available in your project, you can create a new one by clicking **Create Transformation** to invoke the **Transformation Mapping** tool window.

The mapping tool displays a representation of the source schema and target schema in **Source** and **Target** panes. You can create a map between the data type of the method parameter and the data type of the variable, or variables, to which you assign the data. To learn how to create and test a map using the mapping tool, see [Guide to Data Transformation](#).

Note: To return to node builder, in the **Application** pane, double-click the JPD file.

- d. If the appropriate instance of a Transformation control is available in your project, click **Advanced Option**. The Advanced Option window opens. In this window select the **Control** and **Method**. If the method arguments and return type matches those as selected in the **Transformation** pane, click **OK**.
- e. To close the node builder, click the **X** in the top right-hand corner.

In the **Design View**, the  icon indicates that you completed the configuration and design of this node.

Note: To learn about changing the configuration you design in the Transformation pane of a node builder, see [About Editing Node Configurations](#).

4. To save your work, select **File—Save**.

Adding Dynamic Callback Properties

You can set dynamic callback properties for your Client Response node by using the **XQuery Dynamic Selector**. The Dynamic Selector allows you to configure a lookup property based on a LookupControl or TPM function. You can then configure your business process in the WebLogic Integration Administration Console such that, at run time, the security of the callback to the client is handled differently, based on the value of the lookup property that you specified in the Dynamic Selector.

To Set the Dynamic Callback Property

1. Select the Client Response node for which you want to set a Dynamic Callback property.
2. In the **Property Editor**, in the **xquery** field under the **selector** section, click .

The Dynamic Selector window opens with the method schema that is configured for the Client Response node displayed.
3. Select the **LookupControl** or the **TPM** option.
4. Select the element which you want to use as the base for your lookup in the method schema.

An xquery function is created for you and displayed in the **XQuery** pane of the window.
5. If you want to test your xquery, click the **Test** tab. The **Test** tab displays the **Source XML** in the left pane.
6. Click **Test** to display the **Result XML**. Your xquery execution status is displayed in the **XQuery Execution Messages** pane of the window.
7. Click **OK** to close the **Dynamic Selector** window.

The xquery you created is displayed in the **Property Editor** of the Client Response node.

For information about how to configure the security information associated with your dynamic callback property, see “Adding or Changing Dynamic Client Callback Selectors” in [Process Configuration](#) in *Managing WebLogic Integration Solutions* at the following URL:

<http://edocs.bea.com/wli/docs81/manage/processconfig.html>

8. To save your work, select **File—Save**.

Related Topics

[Receiving Messages From Clients](#)

[Buffering Client Messages](#)

[Handling Exceptions](#)

[Client Operations and Control Communication Methods](#)

Buffering Client Messages

To ensure the scalability of your business process applications, incoming messages from clients are buffered by default on the queue for the Web application.

Outgoing messages to clients are not buffered by default, but they can be configured to be buffered on the same Web application queue.

To Buffer an Outgoing Client Message

1. Select the Client Response node that is configured with the callback method you want to buffer.
2. In the **Property Editor**, in the **message buffer** property:
 - a. From the **enable** attribute drop-down menu, select **true**.
 - b. Select the **retry-count** attribute, then enter a value for the callback method. This specifies how many times the process engine should try to send your message to the queue.
 - Select the **retry-delay** attribute, then enter a value for the callback method. This specifies the amount of time (in seconds) you want to pass before a retry is attempted.

This completes the configuration of the callback method on the Client Response node; the callback message is configured to be buffered.

Note: The business process considers a buffered operation completed when the message is successfully enqueued, *not* when the message is delivered to the client.

3. To save your work, select **File**→**Save**.

Related Topics

[Receiving Messages From Clients](#)

[Sending Messages to Clients](#)

[Handling Exceptions](#)

[Client Operations and Control Communication Methods](#)

Interacting With Resources Using Controls

WebLogic Workshop controls make it easy to access enterprise resources, such as databases, Enterprise Java Beans (EJBs), and Web services, from within your application.

When you access a resource through a control, your interaction with the resource is greatly simplified; the underlying control implementation takes care of most of the details for you. You add an instance of a control to your business process project and then invoke its methods. Controls expose Java interfaces that can be invoked from your business process.

You can use controls generated from other services built with WebLogic Workshop or generate controls from WSDL files available from other services (regardless of the programming language in which those services were implemented).

Designing Interactions Between Business Processes and Resources

Control Send nodes represent points in business processes at which processes send asynchronous messages to resources (via controls). **Control Receive** nodes represent points in business processes at which processes receive asynchronous messages from resources (via controls). A business process waits at a **Control Receive** node until it receives a message from the specified control. **Control Send with Return** nodes handle synchronous exchange of messages between business processes and resources (via controls). These three types of controls are *mutable*. In other words, you can change them into another type of control by dragging and dropping a control method of a different type.

This section describes how to add nodes to your business process that represent the interactions of your business process with resources. It includes the following topics:

- [Create Control Nodes in Your Business Process](#)
- [Designing Your Control Nodes](#)
- [Adding Instances of Controls to Your Business Process Project](#)
- [Configuring Control Nodes](#)
- [Setting Control Properties](#)

Related Topics

[Using Integration Controls](#)

Create Control Nodes in Your Business Process

To Create a Control Node in Your Business Process

In the **Design View**, an interaction between a business process and an external resource is represented by one of three **Control** nodes: **Control Send**, **Control Receive**, or **Control Send with Return**. The following steps describe how to add a **Control** node to your business process:

1. On the **Application** tab, click the business process (JPD file) you want to design.
2. Click the **Design View** tab to view your business process.
3. Add a control node to your business process using one of the following methods:
 - [Drag and Drop a Method from a Control in the Data Palette onto the Design View](#)
 - [Create a Control in the Design View First, Then Assign the Appropriate Method](#)

Drag and Drop a Method from a Control in the Data Palette onto the Design View

- a. If the Data Palette is not visible in WebLogic Workshop, choose **View**→**Windows**→**Data Palette** from the WebLogic Workshop menu.
- b. If you have already added an instance of your control to your business project ([Adding Instances of Controls to Your Business Process Project](#)), select the relevant method on that control by clicking the method in the **Data Palette**.
- c. Drag and drop the method onto the business process in the **Design View** at the location at which you want to define the interaction.

As you drag your selection onto the **Design View**, targets  appear on your business process. Each target represents a location in the flow where you can place the node. As

you drag the node near a 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. If the location you chose is not a valid one, an  will appear next to your node. If you place your cursor over this icon, WebLogic Workshop will display a message about the violation.

The **Control** node is created in your business process in the **Design View**; it is named according to the method you dragged and dropped from the **Data Palette**.

Create a Control in the Design View First, Then Assign the Appropriate Method

- a. If the Palette is not visible in WebLogic Workshop, choose **View**→**Windows**→**Palette** from the WebLogic Workshop menu.
- b. If you have not yet created a control, click the control on the **Palette** that fits the action you want to create:

 **Control Send**—Select the **Control Send** if you want to create an asynchronous call from your business process to a control.

 **Control Send with Return**—Choose the **Control Send with Return** node if you want to create a synchronous call from your business process to a control.

 **Control Receive**—Choose the **Control Receive** if you want to create a handler for a callback from a control to your business process.

- c. Drag and drop the **Control** node onto the business process in the **Design View** at the location at which you want to define the interaction.

The **Control** node is created in your business process in the **Design View**; it is named **Control Send**, **Control Send with Return**, or **Control Receive**, depending on which control you dragged onto the **Design View** from the **Palette**.

The node in the **Design View** indicates only the type of interaction (asynchronous send, asynchronous receive, or synchronous send/receive) between your business process and a resource; it does not identify the resource.  is a placeholder for a type of control. That is, it represents a location in your business process where you must specify the type of resource (control) with which you want your business process to interact.

- d. Specify the control for this placeholder node in one of the following ways:
 - Drag a control method from an instance of a control in the **Data Palette** and drop it onto the placeholder control in the **Design View**. (To learn how to add instances of

controls to your project, see [Adding Instances of Controls to Your Business Process Project](#).)

– Double click the placeholder control  in the **Design View** to open the node builder for this control and complete the specifications in the node builder.

Note the following properties for the **Control** nodes:

- indicates that the design of this node is incomplete. To complete the design, see [Configuring Control Nodes](#).
 - Each node is labeled with a default name. You can change the name by clicking the name of the node or right-clicking the node in the **Design View** and selecting **Rename** from the drop-down menu. Then enter a new name to replace **Control Send**.
4. To save your work, select **File—Save**.

Related Topics

[Designing Your Control Nodes](#)

[Adding Instances of Controls to Your Business Process Project](#)

[Configuring Control Nodes](#)

[Setting Control Properties](#)

Designing Your Control Nodes

Designing the **Control** nodes includes adding an instance of the control with which you want your business process to interact, then specifying the methods on the control, and the variables to which the messages exchanged between your business process and the control are assigned. This section describes how to design **Control** nodes. It includes the following topics:

- [Adding Instances of Controls to Your Business Process Project](#)
- [Configuring Control Nodes](#)

Adding Instances of Controls to Your Business Process Project

- [To Add an Instance of a Control to Your Business Process Project](#)
- [To Edit or Delete an Instance of a Control](#)

To Add an Instance of a Control to Your Business Process Project

Before you can specify the resource with which your business process interacts at this node, you must add an instance of the associated control to your project.

To add an instance of a control to your project:

1. In the **Data Palette**, click **Add** on the **Controls** tab. A drop-down list of controls is displayed. These controls represent the resources with which your business process can interact.

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

2. Select a control from the main controls list or the **Integrations Controls** drop-down menu.

Note: This table contains information about the standard controls used in WebLogic Integration. Other custom and plug-in controls may be available.

Type of Control	Description
Application View	<p>The Application View control allows WebLogic Workshop Web services and business processes to interact with enterprise applications using simple Java APIs. They allow access to an enterprise application even if they do not know any of the details of the application's implementation. The Application View control provides a means to invoke application view services both synchronously and asynchronously, and start a new business process when an EIS (Enterprise Information System) event occurs. In both the service and event cases, you use XML and mapping tools to interact with the Application View control. It is not necessary to understand the particular protocol or client API for the enterprise application or EIS. Events are delivered using the Message Broker Subscription control. Message Broker integration is provided by publishing all application view events to the Message Broker through its API.</p> <p>Note: The Application View control uses application views defined using the Application Integration Design console provided with WebLogic Integration. The Application View control is available in WebLogic Workshop only if you are licensed to use WebLogic Integration.</p> <p>To learn about Application View controls, application views, and their relationship to enterprise applications, see Application View Control.</p>

Database	<p>Database controls provide simplified access to a relational database, allowing your business process to call Java methods and operate on Java objects that are appropriate to the operations being performed. The Database control automatically performs the translation from Java objects to database queries and vice versa.</p> <p>Each Database control is customized to access a particular database and perform specified operations on that database.</p> <p>A Database control can operate on any database for which an appropriate JDBC (Java Database Connectivity) driver is available and for which a data source is configured in WebLogic Server. To find more information about Database controls, see Database Control.</p>
Dynamic Transformation	<p>The Dynamic Transformation control provides a business process with the ability to dynamically select and invoke a query during run time. Specifically, this control allows a business process to dynamically select a particular XQuery, XSLT, or MFL file at run time. You use this control after creating and testing your XQuery files with the Transformation control during design. To get information about Dynamic Transformation controls, see Dynamic Transformation Control.</p>
ebXML	<p>The ebXML control enables WebLogic Workshop business processes to exchange business messages and data among trading partners via ebXML (Electronic Business using eXtensible Markup Language). ebXML is a business protocol that enables enterprises to conduct business over the Internet. The ebXML control supports both the ebXML 1.0 and ebXML 2.0 messaging services.</p> <p>Note: The ebXML control is available in WebLogic Workshop only if you are licensed to use WebLogic Integration.</p> <p>For more information about ebXML controls, see ebXML Control.</p>
EJB Control	<p>EJB controls provide an interface to an existing EJB. It is a simplified way for your business process to act as a client of an existing EJB.</p> <p>After you create an EJB control, your business process can use it to access the EJB's <i>business methods</i>. The EJB control simplifies the work you need to do to use an EJB; the control manages communication with the EJB, including all JNDI lookup, interface discovery, and EJB instance creation and management. To find more about EJB controls, see EJB Control.</p>
Email	<p>The Email control enables WebLogic Workshop Web services and business processes to send e-mail to a specific destination. The body of the e-mail message can be text (plain, HTML, or XML) or can be an XML object. The control is customizable, allowing you to specify e-mail transmission properties in an annotation or to use dynamic properties passed as an XML variable. To learn more about Email controls, see Email Control.</p>

File	File controls can be used to read and write XML and binary files to a local file system. In addition, through the use of a callback mechanism, the files in a specified directory can be read as they are created in a directory. For more information about File controls, see File Control .
Http	This control enables WebLogic Workshop and business processes to work with HTTP requests and to send responses to a specific URL. It supports two modes for data transfer: GET and POST. By using the GET mode, you can send your business data along with the URL. By using the POST mode, you can send binary, XML, and string documents. You can specify HTTP properties in an annotation, or pass dynamic properties via an XML variable. For more information, see HTTP Control .
JMS	<p>Use this control to send and receive messages via a Java Message Service (JMS) queue or topic that is not involved in exchanging information with enterprise systems through an integration application. Using JMS controls, your business process can interact with any messaging system that provides a JMS implementation. To learn more about JMS controls, see JMS Control.</p> <p>To exchange JMS messages as part of an integration application, you use the WLI JMS. This control is available as part of the integration controls if you have a current WLI license. For more information, see WLI JMS.</p>
Liquid Data	<p>You can use the Liquid Data control in WebLogic Workshop to develop applications that use data from Liquid Data queries. For example, data from Liquid Data queries can be used as an input to business processes. To learn about BEA Liquid Data, see the Liquid Data documentation at http://edocs.bea.com/liquiddata/docs81/index.html. Specifically, to learn about the Liquid Data control, see Using Liquid Data Controls to Develop Workshop Applications.</p>
MB Publish	<p>Message Broker (MB) Publish controls allow your business process to publish messages to Message Broker channels.</p> <p>Publish and subscribe messaging to Message Broker channels is accomplished in similar fashion to publish and subscribe messaging to JMS topics, but a Message Broker channel is optimized for use with BPM (business process management) services. The Message Broker provides typed channels to which messages can be published and to which services can subscribe to receive messages. Message Broker also supports a message filtering capability. For more information about Message Broker Publish controls, see Message Broker Publish Control.</p>

MB Subscription	<p>Message Broker (MB) Subscription controls allow your business process to dynamically register for and receive messages from a Message Broker topic.</p> <p>In WebLogic Integration, subscriptions to Message Broker channels defined at a Start node are referred to as static subscriptions, and subscriptions defined using a Message Broker Subscription control are referred to as dynamic subscriptions. See “Note about Static and Dynamic Subscriptions” in @jpd.mb-static-subscription Annotation.</p> <p>Publish and subscribe messaging to Message Broker channels is accomplished in similar fashion to publish and subscribe messaging to JMS topics, but a Message Broker channel is optimized for use with BPM (business process management) services. The Message Broker provides typed channels to which messages can be published and to which services can subscribe to receive messages. Message Broker also supports a message filtering capability. To learn more about Message Broker Subscription controls, see Message Broker Subscription Control.</p>
MQSeries	<p>The MQSeries control enables WebLogic Workshop business processes to work with MQSeries for sending and receiving messages, to and from MQSeries queues. MQSeries is a middleware product from IBM that runs on multiple platforms and enables applications to send messages to other applications. For more information about the MQSeries control, see MQSeries Control.</p>
Process	<p>A Process control provides an interface to another business process in your project. Using a process control, your business process can invoke the methods and handle the callbacks on another business process.</p> <p>To create a Process control, on the Application tab in the Design View, right-click a business process (JPD) file to display a drop-down menu. Select Generate Process Control File from the drop-down menu. WebLogic Workshop creates a Business Process control file (JCX file) in your project. For more information about Process controls, see Process Control.</p>
RosettaNet	<p>The RosettaNet control enables WebLogic Workshop business processes to exchange business messages and data among trading partners via RosettaNet. RosettaNet is a business protocol that enables enterprises to conduct business over the Internet.</p> <p>Note: The RosettaNet control is available in WebLogic Workshop only if you are licensed to use WebLogic Integration.</p> <p>To learn more about the RosettaNet controls, see RosettaNet Control.</p>

Service Broker	The Service Broker control allows a business process to send requests to and receive callbacks from another business process, a Web service, or a remote Web service defined in a WSDL file. The Service Broker control lets you dynamically set control attributes. This allows you to reconfigure control attributes without having to redeploy the application. To learn more about Service Broker controls, see Service Broker Control .
TPM	The TPM (Trading Partner Management) control provides read-only access to trading partner information stored in the TPM repository. For more information about how to use TPM controls, see TPM Control .
Task	The Task control creates a single Task instance, manages its state and data, and provides callback methods to report status. A Task control identifies intimately with a single Task instance; their relationship is one to one. You generally use a Task control in a JPD file like most other IDE controls. For more information about Task controls, see Worklist Controls .
Task Worker	The Task Worker control assumes ownership of Tasks, works on them, completes them, and provides administrative privileges—starting, stopping, deleting, assigning, and other functionality. Task Worker controls allow operations on several Task instances; the relationship between a Task Worker control and Task instance can be one to many. You generally use a Task Worker control with JSP user interfaces. For more information about Task Worker controls, see Worklist Controls .
Timer	A Timer control notifies your business process when a specified period of time has elapsed or when a specified absolute time has been reached. To learn more about how to work with Timer control, see Timer Control .
Transformation	Use a Transformation to achieve data transformations for data in your business processes. A Transformation is defined in a DTF file, which can be created and edited from within communication nodes in a business process and via the File—New—Transformation File option on the WebLogic Workshop menu. To learn more, see Note About Transformations .
Tuxedo	You can include one or multiple Tuxedo services in your business process logic by using a Tuxedo control. By creating connections to Tuxedo services, you can invoke the Tuxedo services and retrieve the responses, and convert your application's Java data types to and from Tuxedo buffer types. To learn about the Tuxedo control, see BEA Tuxedo Control .

Web Service	<p>A Web Service control provides an interface to a Web service, allowing your business process to invoke the methods and handle the callbacks of the Web service. The Web service can be developed with WebLogic Workshop or any Web service for which a WSDL file is available.</p> <p>A Service control is defined in a CTRL file. A specific Service control CTRL file provides a way to communicate with a specific Web service. The name and location of the Web service are specified in the CTRL file. For more information about how to work with and configure Web Service controls, see Web Service Control and Controls and Transactions.</p>
WLI JMS	<p>The WLI JMS Control is an extension for the Workshop JMS control, it is used to exchange JMS messages as part of an integration application. Once a WLI JMS control is defined, Web services and business processes may use it like any other WebLogic Workshop control. For more information, see JMS Control.</p> <p>Note: The WLI JMS control is available in WebLogic Workshop only if you are licensed to use WebLogic Integration. If you do not have a current license, use the JMS control to send and receive messages via JMS. For more information, see JMS.</p>
XML MetaData Cache	<p>The XML MetaData Cache control only allows you to retrieve XML metadata from the XML cache. The XML cache is managed using the WebLogic Integration Administration Console. For more information, see XML MetaData Cache Control.</p>

3. After you select a type of control from the drop-down list in the **Controls** tab, an **Insert Control** dialog box, which contains tasks specific for the control you selected, is displayed.
4. In the **Insert Control** dialog box, enter the information specific for the control you want to create and click **Create**. To learn about creating and configuring specific controls, see [Using Integration Controls](#).

This step completes the creation of an instance of a specific control in your application. The controls you create are displayed in the **Controls** tab.

The methods available on the control are shown in the **Controls** tab.

Note About Transformations

Transformations handle mapping heterogeneous data types in your application. WebLogic Workshop provides a data mapping tool to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files. The DTF files can hold multiple transformations and are designed to enable packaging, sharing and reuse of transformation formats. When DTF files containing your data transformations are

built, they are built as controls. The controls expose transformation methods, which business processes invoke to map the disparate data types.

In addition to creating Transformations from the **Controls** tab in the **Design View**, as described in this section, you can create them in the following ways:

- By choosing **File—New—Transformation File** from the WebLogic Workshop menu.
- By choosing **File—New—Other File Types—Processes—Transformation File** from the WebLogic Workshop menu.
- In the node builders for any **Control** or **Client** node in your business process. During the design of a node that sends or receives data in your business process, you can create a new Transformation, or write new methods to an existing instance of a Transformation control in your project. In this way you can create new Transformations or Transformation methods on an existing control from within a business process node. To learn about node builders, see *Node Builders* in [How Do I: Use the Design View?](#)

To learn about data transformations in business processes, see [Guide to Data Transformation](#).

To Edit or Delete an Instance of a Control

In the **Controls** tab, right-click a control to display a drop-down menu. Select **Delete** or **Edit** from the menu. When you select **Edit**, the control, including its methods and callbacks, is displayed in the **Design View**. Click the **Source View** tab to view and edit the source code for your control.

Related Topics

[Create Control Nodes in Your Business Process](#)

[Designing Your Control Nodes](#)

[Configuring Control Nodes](#)

[Setting Control Properties](#)

[Using Integration Controls](#)

[Using Built-In Java Controls](#)

Configuring Control Nodes

This section describes how to finalize the design of **Control** nodes in your business process.

After you add a **Control** node specific for the type of interaction you want to design—**Control Send**, **Control Receive**, or **Control Send with Return**—the **Control** node you selected is displayed in your business process in the **Design View**:



As with other nodes in your business process, you can design the properties and behavior of **Control** nodes by invoking their node builders. This section describes how to complete the design of the interaction with resources via your **Control** nodes.

To Invoke the Control Node Builders

Double-click the appropriate **Control** node in your business process to invoke its node builder.

Each **Control** node builder provides a task-driven interface through which you can design the communication between the **Control** node and a control. The tasks are displayed on tabs on **Control** node builders: **General Settings**, **Send Data**, and **Receive Data**.

The following sections describe how to specify your control settings on the tabs in the node builders:

- [General Settings \(Select a Control Instance and a Target Method\)](#)
- [Send Data/Receive Data \(Map Variables to the Control Send \(or Control Callback\) Method Parameters\)](#)

General Settings (Select a Control Instance and a Target Method)

1. In the node builder, click the arrow beside the **Control** field to display a drop-down list of the instances of controls that are available in your project. (See [Adding Instances of Controls to Your Business Process Project](#).)

2. Select a control from the list.

3. The **Method** panel is populated with the methods available on the control you selected.

Note: Asynchronous send and return methods, as well as synchronous send and receive methods can be defined for a given control. Only the methods appropriate for the kind of control node you are designing (**Control Send**, **Control Receive**, or **Control Send with Return**) are displayed in the list.

4. Select the method you want to specify at this point in your business process.

5. To close the node builder, click the **X** in the top right-hand corner.

Send Data/Receive Data (Map Variables to the Control Send (or Control Callback) Method Parameters)

If your **Control** node is expecting data or sending data, in other words it is a **Control Send**, a **Control Receive**, or a **Control Send with Return**, the node builders display either **Send Data** or **Receive Data** tabs in addition to the **General Settings** tab. Tasks on these tabs allow you to define one or more variables to map to method parameters. At run time, input data sent by your business process to controls, or data returned by controls is assigned to these variables.

1. Click the **Send Data** or **Receive Data** tab (depending on the type of Control node you are designing).

This tab allows you to define one or more variables to hold the data that your business process receives from clients.

2. If the data types of your method parameters and the data type of the variables you are going to use match, you can map your variables to the corresponding methods directly.

- a. If not already selected, select the **Variable Assignment** option.

The **Control Expects** field is populated with the parameter(s) you specified on the **General Settings** tab.

- b. If you want to assign a variable that you already created in your project to the method parameters, select it from the drop-down menu.
 - c. If you want to create a new variable and assign it to the method parameter, select **Create new variable...**, then follow the instructions in the [To Create a New Variable in the Node Builder](#) section.
 - d. If the data types of your method parameters and your variables match, click **X** to close the node builder.
3. If the data types of your method parameters and your variables are different, you can use the data mapping tool included in WebLogic Workshop to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files. When DTF 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.
 - a. To create a transformation map, select the **Transformation** option.

The node builder transformation screen is displayed with the data types expected by your method displayed in the **Control Expects** pane.

- b. In **Step 1** of the **Transformation** option window, click **Select Variable** to select one or more variables to be used.

Note: To remove a variable from the node builder pane, select the variable in the list and then click **Remove**. This action removes the variable from the node builder, not from your business process. The variable is still included in your business process; it is visible in the **Variables** pane in the **Data Palette**.

When designing a business process, you use a Transformation to create maps between disparate data types. Your project must contain an instance of a Transformation control (defined by a DTF file) for you to create the map.

- c. If an appropriate instance of a Transformation control is not available in your project, you can create a new one by clicking **Create Transformation** to invoke the **Transformation Mapping** tool window. This automatically applies changes to the builder and opens a transformation editor in a new window.

The mapping tool displays a representation of the source schema and target schema in **Source** and **Target** panes. You can create a map between the data type of the method parameter and the data type of the variable, or variables, to which you assign the data. To learn how to create and test a map using the mapping tool, see [Guide to Data Transformation](#).

Note: To return to node builder, in the **Application** pane, double-click the JPD file.

- d. If the appropriate instance of a Transformation control is available in your project, click **Advanced...** The Advanced Option window opens. In this window, select the **Control** and **Method**. If the method arguments and return type matches those as selected in the **Transformation** pane, click **OK**.
4. To close the node builder, click the **X** in the top right-hand corner.

In the **Design View**, the  icon indicates that you completed the configuration and design of this node and  is replaced with an icon that represents the resource with which this node communicates. That is, a new control-specific icon replaces the former placeholder icon.

5. To save your work, select **File—Save**.

Related Topics

[Create Control Nodes in Your Business Process](#)

[Designing Your Control Nodes](#)

[Adding Instances of Controls to Your Business Process Project](#)

[Setting Control Properties](#)

[Client Operations and Control Communication Methods](#)

[Guide to Data Transformation](#)

[Grouping Nodes in Your Business Process](#)

[Handling Exceptions](#)

[About Editing Node Configurations](#)

Setting Control Properties

Controls you create in your application are represented as JCX files in the **Application** pane in WebLogic Workshop. (Transformations are represented as DTF files—see [Note About Transformations](#).) *Instances* of controls that you create in your business process are represented in the **Data Palette**. You can view and edit the properties of control instances and their parent types in the **Property Editor**.

[To View and Edit Properties for Control Types](#)

[To View and Edit Properties for Control Instances](#)

To View and Edit Properties for Control Types

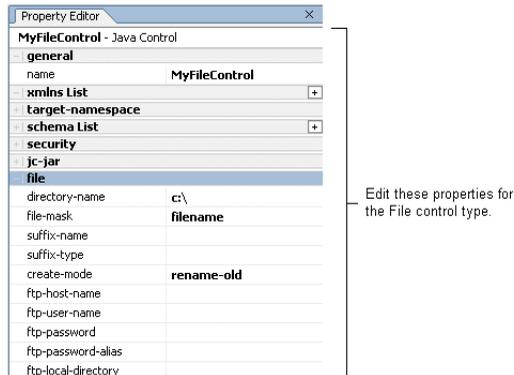
Double-click the control type (JCX or DTF file) on the **Application** tab.

The file is displayed in the **Design View** and the **Source View**, and its properties are displayed in the **Property Editor**. The properties you see and edit in the **Property Editor** depend on the control you are using.

Values you specify for the properties in the **Property Editor** are written to the file. In other words, the **Source View** is updated in keeping with the work you do in the **Property Editor**. Properties you specify for the control are inherited by any instances of the control you create based on this type.

For example if you create a File control named **MyFileControl**, the control type (**MyFileControl.jcx**) is displayed in the **Application** pane in WebLogic Workshop. The

following figure displays the **Property Editor** for the example File control (**MyFileControl.jcx**). It displays the properties you can edit for the control type.

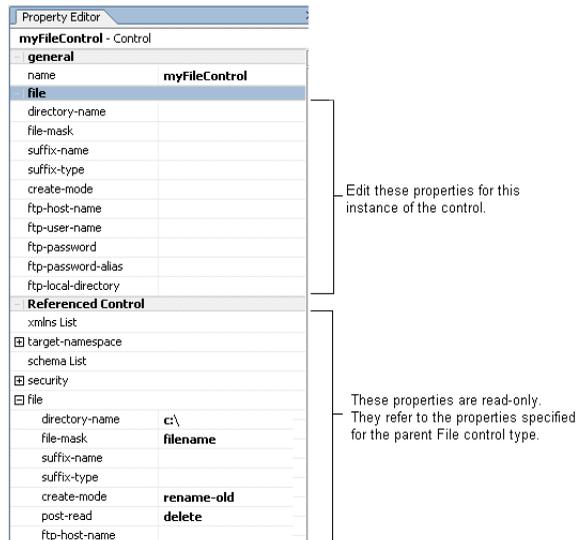


To View and Edit Properties for Control Instances

Double-click the control instance in the **Data Palette** to display its properties in the **Property Editor**. The properties you can see and edit depend on the control you are using. Note that when you open the **Property Editor** for an instance of a control, the properties for that instance, are listed at the top of the **Property Editor** and the properties specified for the parent control (that is, the control on which the current instance is based) are listed at the bottom—in the **Referenced Control** section. The properties displayed in the **Referenced Control** section are read-only. You can edit the referenced control properties by opening the JCX file as described in [To View and Edit Properties for Control Types](#).

Building on the example we used in the preceding section, if you create an instance of a File control in your business process (based on the **MyFileControl.jcx** type), and name it **myFileControl**, the instance is displayed in the **Data Palette**. Click **myFileControl** in the **Data Palette** to display its properties in the **Property Editor**.

The following figure shows the **Property Editor** for our example **myFileControl** instance:



Note the properties displayed in the **Referenced Control** section are those that were specified for the parent control **MyFileControl.jcx**. You can edit them by opening the parent control (in this case **MyFileControl.jcx**) as described in [To View and Edit Properties for Control Types](#).

Related Topics

[Using Integration Controls](#)

[Adding Instances of Controls to Your Business Process Project](#)

[Configuring Control Nodes](#)

[Client Operations and Control Communication Methods](#)

[Guide to Data Transformation](#)

[Handling Exceptions](#)

Interacting With Resources Using Controls

Receiving Multiple Events

An **Event Choice** node group represents a point in a business process at which the business process waits to receive one of a possible number of events. Once it receives one of the possible events, the flow of the business process continues. You design other nodes within an **Event Choice** node group to handle the incoming events. The first node on each branch of an **Event Choice** node group handles the receipt of one event. The flow of execution proceeds along one branch in an **Event Choice** node; the branch containing the event that happens first.

If an **Event Choice** node is used to start a business process, it can contain **Client Request**, **Client Request with Return**, and **Subscription** nodes. An **Event Choice** node at a point other than the **Start** node in a business process can contain **Client Request** nodes and **Control Receive** nodes. You can also add a **Timer Branch** to your **Event Choice** node to start that branch after a specified amount of time has passed.

To learn about designing an **Event Choice** node at the Start of your business process, see [Designing Start Nodes](#).

Note: The Timer branch of an **Event Choice** node is not available when the node group is used as the **Starting Event** of a business process. To do timed starts of a process, you have to use a Message Broker subscription in tandem with a Timer event generator. For more information about Message Broker subscriptions and Timer event generators, see [Using Integration Controls](#).

This section describes how to design **Event Choice** nodes at points in your business process other than the **Start** node. It contains the following topics:

- [Create an Event Choice Node in Your Business Process](#)

- [Design Your Event Choice Group](#)

Create an Event Choice Node in Your Business Process

Create an **Event Choice** node at a point in a business process at which the business process should wait to receive multiple events. The events can include:

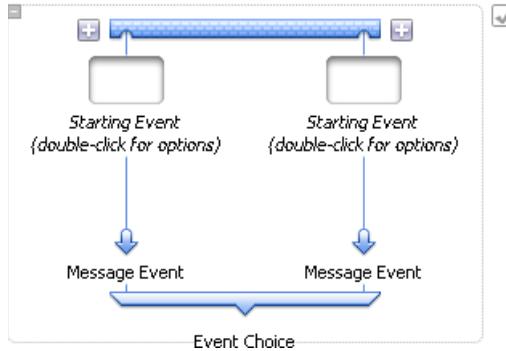
- Receiving messages from clients.
- Receiving messages from resources, such as a database, a JMS queue, an EJB, and so on. (A business process interacts with resources using controls.)
- A Timer event. The timer starts when the execution of the business process reaches the **Event Choice** node and pauses to wait for an event.

To support these types of events, the first node on a branch can be a **Client Request**, a **Control Receive**, or a **Timer** node. The flow of execution proceeds along one branch in an **Event Choice** node; the branch containing the event that happens first.

To create an **Event Choice** node:

1. On the **Application** tab, click the business process (JPD file) you want to design.
Your business process is displayed in the **Design View**.
2. If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the WebLogic Workshop menu.
3. Click  **Event Choice** in the **Palette**. Then drag and drop it onto the business process in the **Design View**, placing it on the business process at the point in your business process where you want to handle the receipt of multiple events.

The **Design View** is updated to contain a representation of the **Event Choice** node as shown in the following figure:



Note the following characteristics of the **Event Choice** node:

- An **Event Choice** node is, in effect, a group of nodes. You can view and edit the properties of your **Event Choice** node by clicking the outline or label (name) of the group to select it, then viewing the group properties in the **Property Editor**. To learn about groups, see [Grouping Nodes in Your Business Process](#).
 - By default, **Event Choice** nodes are created with two branches. Click  to create additional branches. A new branch is added on the left or right of the existing branches.
 - You can add additional nodes to the paths in your **Event Choice** group to specify the events executed at run time. The empty nodes (labeled *Starting Event*) at the start of each branch indicate that only certain nodes are allowed at these locations: specifically, you can add only **Client Request** or **Control Receive** nodes at the start of the branches.
 - A **Timer** branch is not included by default. You can add one **Timer** branch to your **Event Choice** group. To do so, right-click the **Event Choice** group and select **Add Timer Branch**—a **Timer** branch is added as the right-most branch in an **Event Choice** group. You can only add one **Timer** branch per **Event Choice** group.
- Note:** The **Timer** branch of an **Event Choice** node is not available when the node group is used as the **Starting Event** of a business process. To do timed starts of a process, you have to use a **Message Broker** subscription in tandem with a **Timer** event generator. For more information about **Message Broker** subscriptions and **Timer** event generators, see [Using Integration Controls](#).
- By default, the group is named **Event Choice**, and each branch is labeled **Message Event**, **Add Branch**, or **Timer Event** depending on the type of branch. You can change the names by double-clicking them and entering a new name.

- indicates that the design of this node is incomplete. When you complete the design of the node, is replaced by . An **Event Choice** node is complete when all starting events have been specified.
4. To save your work, select **File—Save**.

Related Topics

[Design Your Event Choice Group](#)

[Comparing Parallel Nodes and Event Choice Nodes](#)

[Handling Exceptions](#)

[Adding Timeout Paths](#)

[Adding Message Paths](#)

Design Your Event Choice Group

Designing your **Event Choice** node includes specifying the type of events handled on each branch of the node, and then adding the activities you want executed on each branch when the associated event occurs.

The following sections describe how to complete the tasks necessary to design an **Event Choice** node:

- [To Receive Events From Clients or Resources](#)
- [To Receive Timer Events](#)

To Receive Events From Clients or Resources

To design a branch in an **Event Choice** node to receive messages from clients or resources, you must create **Client Request** or **Control Receive** nodes on the branch:

1. Double-click the empty node (**Starting Event**) on a branch. The options you can use to design the starting event for the branch are displayed.
2. Select the event for which this branch waits during execution of your business process:
 - **A Client Request**
 - **A Control Receive**
3. Click **X** in the top right-hand corner of the node. The drop target on your **Event Choice** branch is changed to reflect the event you specified.

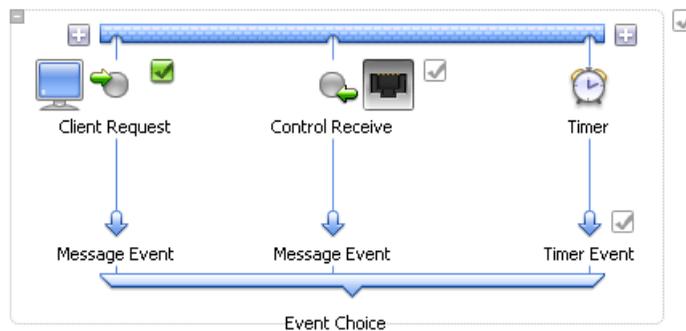
4. To complete the specification of events, double-click the starting event node (**Client Request** or **Control Receive**) on the **Event Choice** branches to invoke the associated node builder:
 - To learn how to use the node builder to complete the **Client Request** node, see [Design Your Client Request Node](#).
 - To learn how to use the node builder to complete the **Control Receive** node, see [Designing Your Control Nodes](#).

To Receive Timer Events

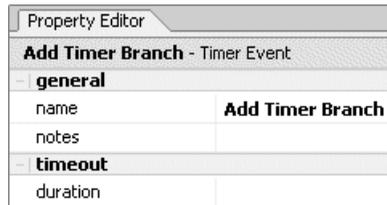
A Timer event in a **Event Choice** node is executed if one of the events on another branch (**Control Receive** or **Client Request**) does not execute before a specified time. To create a Timer branch, and specify the *timer value*, in your **Event Choice** node, complete the following tasks:

1. Right-click the **Event Choice** node and select **Add Timer Branch** from the drop-down menu.

A Timer branch, similar to the one shown in the following figure, is added to the **Event Choice** node:



2. You can set the properties for the Timer branch (and other properties for this group of nodes) in the **Property Editor**.
 - a. If the **Property Editor** is not visible in the **Design View**, choose **View—Property Editor** from the WebLogic Workshop menu.
 - b. Select the Timer branch. The **Property Editor** for the **Event Choice** node shown in the preceding figure appears as shown in the following figure:



- c. In the **timeout** property, select the **duration** attribute, then specify the number of seconds before the timer path is triggered. (The expected format is Xs, for example 7s.)

Note that you can change the name of the node, or any of its branches in the **Property Editor**.

- 3. To save your work, select **File—Save**.

Related Topics

[Create an Event Choice Node in Your Business Process](#)

[Grouping Nodes in Your Business Process](#)

[Handling Exceptions](#)

[Business Process Source Code](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

Creating Parallel Paths of Execution

A **Parallel** node represents a point in a business process at which a number of activities are executed in parallel.

By default, parallel nodes contain an **AND** join condition. In this case, the activities on all branches must complete before the flow of execution proceeds to the node following the parallel node. You can change the join condition to **OR**. In this case, 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.

This section describes how to create and define Parallel nodes. It includes the following topics:

- [Understanding Parallel Execution in Your Business Process](#)
- [Create a Parallel Node in Your Business Process](#)
- [Design Your Parallel Node](#)

Understanding Parallel Execution in Your Business Process

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

Parallel branches are synchronized only at their termination points. A *join condition* is defined at the termination of multiple branches. It specifies how the termination of branches terminates the overall parallel activity.

Valid join conditions are AND and OR:

- When the join condition is AND, the parallel activity terminates when all of its branch activities have terminated. When the activities on all branches complete, the flow of execution proceeds to the node that follows the parallel node.
- When the join condition is OR, the parallel activity terminates when one of its branch activities has terminated—activities associated with other branch activities are terminated prematurely. In other words, when the activities on *one* branch complete, the flow of execution proceeds to the node that follows the parallel node.

Comparing Parallel Nodes and Event Choice Nodes

How does a **Parallel** node, which specifies an OR join condition, differ from an **Event Choice** node?

For a scenario in which an OR join condition is specified for a **Parallel** node, the business process executes activities on *all* branches in parallel. 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. In other words, the activities on all parallel branches are initiated and proceed until the first one finishes, at which point the activities on all other branches are terminated.

In the case of an **Event Choice** node, the business process waits to receive multiple events. The first node on each branch within an **Event Choice** node handles the receipt of one event. The flow of execution proceeds along the branch containing the event that happens first. In other words, the activities on one, and only one branch in an **Event Choice** node are executed.

Related Topics

[Create a Parallel Node in Your Business Process](#)

[Design Your Parallel Node](#)

Create a Parallel Node in Your Business Process

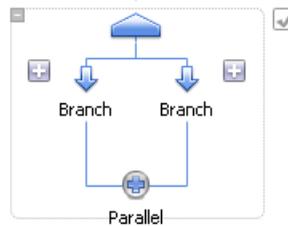
To Add A Parallel Node to Your Business Process

1. On the **Application** tab, click the business process (JPD file) you want to design.

Your business process is displayed in the **Design View**.

2. If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the WebLogic Workshop menu.
3. Click  **Parallel** in the **Palette**. Then drag and drop the **Parallel** node onto the business process in the **Design View**, placing it on the business process at the point in your business process at which you want to create parallel paths of execution.

The **Design View** is updated to contain a **Parallel** node, as shown in the following figure:



Note the following characteristics of the **Parallel** node:

- By default, a **Parallel** node consists of two branches; you can click  to add branches.
 - By default, the node is named **Parallel**, and each branch is labeled **Branch** or **Add Branch**. You can change the names by double-clicking them and entering a new name.
 - indicates that the design of this node is incomplete. When you complete the design of the node, is replaced by . A parallel node is completed when each branch contains at least one node.
4. To save your work, select **File—Save**.

Related Topics

[Design Your Parallel Node](#)

Design Your Parallel Node

Designing a Parallel node includes the following tasks:

- [To Define a Join Condition](#)
- [To Add Logic to the Branches in Your Decision Node](#)

To Define a Join Condition

A **Parallel** node is, in effect, a group of nodes. You can set the properties for a group of nodes using the **Property Editor**.

1. View the properties of your **Parallel** node by clicking the outline of the group to select it, then view the group properties in the **Property Editor**.

Note: If the **Property Editor** is not visible in the **Design View**, choose **View—Property Editor** from the WebLogic Workshop menu.

2. To change the value of the Join Condition from **AND** (the default) to **OR**, in the **Property Editor**, select **OR** from the drop-down menu associated with **Join Condition**. The node in your business process will be updated with a  to indicate the **OR** condition.

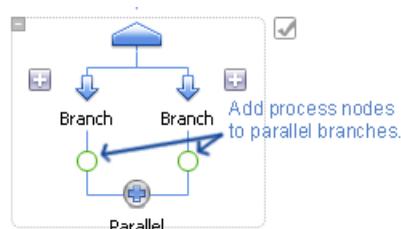
To learn how the Join Condition affects the flow of execution in a **Parallel** node, see [Understanding Parallel Execution in Your Business Process](#).

3. To change the name of the node or any of its branches, in the **Property Editor**, click the **name** attribute in the node or branch names, then enter the new name.

To Add Logic to the Branches in Your Decision Node

For each branch in your **Parallel** node:

1. In the **Palette**, click a **Process Node** that represents the type of logic you want to add to the business process.
2. Drag and drop the node from the **Palette** onto the appropriate branch.



3. Complete the design of the nodes added on each branch. In this way, you create the activities appropriate for the business logic defined by your business process.

Note: You can create nested **Parallel** nodes in your business process by dragging a  **Parallel** node from the **Palette** on to one of the branches in a **Parallel** node already created in the **Design View**.

4. To save your work, select **File—Save**.

Related Topics

[Create a Parallel Node in Your Business Process](#)

[Grouping Nodes in Your Business Process](#)

[Handling Exceptions](#)

[Business Process Source Code](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

Creating Parallel Paths of Execution

Defining Conditions For Branching

A common design pattern in business processes is one which selects one path of execution based on the evaluation of one or more conditions. You can create this pattern by designing a **Decision** node in your business process.

By default, a **Decision** node consists of one condition, a path below the condition, which represents the path of execution followed when the decision evaluates 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). A **Decision** node can contain additional conditions, in which case if the first condition evaluates to false, the second condition is evaluated. If the second condition evaluates to false, the next condition is evaluated, and so on. The default path is executed if no conditions are met.

Note: To create case statements, WebLogic Integration provides a customized node, called a Switch node. To learn about using Switch nodes and how they differ from Decision nodes, see [Comparing Decision Nodes and Switch Nodes](#) in [Creating Case Statements](#).

This section describes how to add a **Decision** node to your business process, define conditions, and define activities for the alternative paths of execution in the **Decision** node. It contains the following topics:

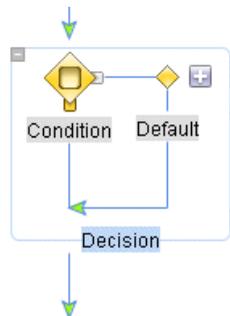
- [Creating a Decision Node in Your Business Process](#)
- [Designing Your Decision Node](#)

Creating a Decision Node in Your Business Process

To Create a Decision Node in Your Business Process

1. On the **Application** tab, click the business process (JPD file) you want to design.
Your business process is displayed in the **Design View**.
2. If the **Palette** is not visible in WebLogic Integration, choose **View→Windows→Palette** from the WebLogic Integration menu.
3. Click  **Decision** on the **Palette**.
4. Drag and drop the **Decision** node onto the business process in the **Design View**, placing it on the business process at the point in your business process that requires branching to one of several possible paths of execution, based on the evaluation of one or more conditions.

The **Design View** is updated to contain a **Decision** node, as shown in the following figure:



Note the following characteristics of the **Decision** node:

- Adding a **Decision** node to a business process adds, by default, a single *Condition* node, and a representation for the two paths of execution after the *Condition* node.
- You can add additional condition nodes. To do so, right-click on the **Decision** node and select **Add Condition** from the drop-down list, or click .

At run time, when more than one condition is defined, if the first condition evaluates to false, the second condition is evaluated. If the second condition evaluates to false, the next condition is evaluated, and so on. The default path is executed if no conditions are met.

- You can change the name of the **Decision** node, the **Default** branch, and each **Condition** in a **Decision** node. To do so, click the name assigned to the **Condition**, **Add Condition** branch, **Default** branch, or **Decision** node and enter a new name.

- indicates that the design of this node is incomplete. When you complete the design of the node, is replaced by (see [Example Decision Node](#)). A **Decision** node is completed when all conditions have been configured.
- A **Decision** node is, in effect, a group of nodes. You can view and edit the properties of your **Decision** node by clicking the outline of the group to select it, then viewing the group properties in the **Property Editor**. To learn about groups, see [Grouping Nodes in Your Business Process](#).

Related Topics

[Designing Your Decision Node](#)

[Creating Case Statements](#)

Designing Your Decision Node

To create logic for your **Decision** node, you must complete the following steps:

- [To Design the Condition Logic](#)
- [To Add Activities to the Paths in Your Decision Node](#)

To Design the Condition Logic

1. Double-click the **Condition** node to invoke the decision builder.
2. Select one of the options:
 - **Variable**—Select this option if, at run time, you want the business process to make a decision, based on the value of an element in an XML or non-XML variable.
 - **Method**—Select this option if, at run time, you want the business process to make a decision, based on a boolean result returned from Java code that you create.

The node builder displays different options depending on whether you select **Variable** or **Method**.

3. Complete the selections in the node builder appropriate for the selection you made in the preceding step: [Variable \(Schema\)](#) or [Method](#).

Variable (Schema)

The following steps describe how to select a business process variable that is associated with an XML or MFL schema.

Note: To learn about creating business process variables and importing schemas to your project, see [Business Process Variables and Data Types](#) and [Importing Files into the Schemas Project](#).

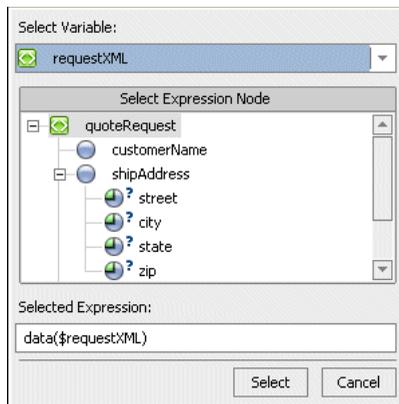
1. In the condition builder, select a business process variable by clicking .

A drop-down list of business process variables in your project is displayed.

For example, if you imported an XML Schema (`QuoteRequest.xsd`) into your project, and created a business process variable (`requestXML`) of type `quoteRequest` (based on the `QuoteRequest.xsd` schema), the `requestXML` variable is available in the drop-down list of business process variables.

2. Click the arrow in the **Select Variable** drop-down list, then select a variable that contains the XML or typed non-XML on which you want to build the condition.

A representation of the XML Schema associated with that variable is displayed in the **Select Expression Node** field.



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. Note that the schema in the example (`QuoteRequest.xsd`) specifies a root element (`quoteRequest`), and child elements: `customerName`, `shipAddress`, and `widgetQuoteRequests`. The `widgetQuoteRequests` element, in turn, specifies a repeating element: `widgetQuoteRequest`. (A repeating XML element is represented by  in the GUI representation of the Schema.)

3. In the **Select Expression Node** field, select the node in the XML Schema for which you want to define the condition.

To continue with the example, supposed you selected **customerName** from the XML variable represented in the preceding figure. The **Selected Expression** field is populated with the following expression:

```
data($requestXML/ns0:customerName)
```

4. Click **Select**. Your new variable is displayed in the **Left Hand Expression** field.
5. Select an operator from the **Operator** drop-down list.
For example, =
6. In the **Right Hand Expression** field, enter a value or choose a variable and expression with which to create the decision logic.
For example, enter BEA.
7. Click **Add**. The condition you created is added to the condition list.
For example, `data($requestXML/ns0:customerName = "BEA"`
8. Select a join option of either **AND** or **OR** to qualify your conditions.
9. To add a condition based on an existing value in the **Left Hand Expression** field:
 - a. In the condition list pane, select a condition. The **Left Hand Expression**, **Operator**, and **Right Hand Expression** fields are populated with the appropriate values.
 - b. In the **Right Hand Expression** field, select the value.
For example, BEA.
 - c. Change the entry you selected.
For example, Avitek.
 - d. Select the arrow beside the **Update** button, then select **Add** from the menu.
The new condition is added to the bottom of the condition list.
10. To edit the conditions after you create them:
 - a. In the condition list pane, click the condition that you want to change. The **Left Hand Expression**, **Operator**, and **Right Hand Expression** fields are populated with the appropriate values.
 - b. Change the value in any of the fields.
 - c. Click **Update**.

Alternatively, you can edit conditions directly in the code. To do so, in the **Condition** builder, click **View Code** in the lower left-hand corner. The XQuery function that was written to the file from the design work in the condition builder is displayed at the line of code in your JPD file; it is indicated by the `@jpd:xquery prologue` annotation.

11. To edit Join Options after you create them:

- a. In the condition list pane, click the **Join Option** that you want to change.
- b. Select the appropriate join option.
- c. Click **Update**.

12. In the **Design View**, click **X** in the top right-hand corner of the condition builder.

In the **Design View**, note that the **Condition** in your **Decision** node displays the following icons:

-  is a visual reminder that the condition you defined on this node is based on the evaluation of an XML document.
-  is a visual reminder that the condition you defined on this node is based on the evaluation of a MFL file.

Defining an XML or MFL condition produces an XQuery function that is written to your JPD file, which you can see in the **Source View**. The condition defined by following the preceding example (in steps 1 through 7) creates the following XQuery function in the JPD file:

```
/* Process Language */
* @jpd:xquery prologue::
*   define function cond_requestXML_1(element $requestXML)
*   returns xs:boolean {
*       data($requestXML/customerName) = "BEA"
*   }
*   ::
```

13. To save your work, select **File**—**Save**.

Method

The following steps describe how to select a business process variable that is associated with an XML or MFL schema.

1. In the **Java Method Name** field, enter a name for the Java method, or, to choose an existing method, click .
2. Click **View Code** in the lower left-hand corner of the node builder.

The **Source View** is displayed at the line of code in your JPD file at which the Java method is written.

3. Edit your Java method.
4. To return to the **Design View**, click the **Design View** tab.
5. Close the condition builder by clicking **X** in the top right-hand corner.

In the **Design View**, note that the **Condition** in your **Decision** node displays the following icon: . It is a representation of the condition you defined in source code that specifies the Java method on which to base the decision. To make any further changes to the condition represented on this node, you must edit the source code in the **Source View**.

6. To save your work, select **File—Save**.

To Add Activities to the Paths in Your Decision Node

After you define the condition that is evaluated when the flow transitions to the **Decision** node at run time, you are ready to define the actions on the paths that represent the paths of execution in the flow.

1. Add a node (or nodes) to each path in the **Decision** node to define the activity that is executed when the conditions you defined on the **Condition** node at the beginning of the path evaluates to true.

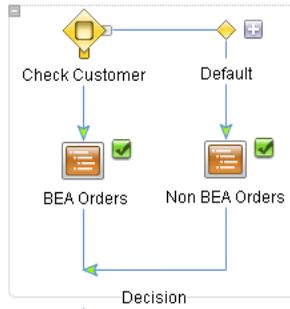
This can be any node that performs an activity appropriate for your business process business logic. For example you can use a control to interact with an external resource, such as a database, a JMS queue, or an EJB.

2. Add a node (or nodes) to the default path that defines which activities are executed when no condition evaluates to true at run time. The nodes on the default path can be any that define activities appropriate for your business process business logic.

When you complete the addition of activities on the paths of your **Decision** node, your decision logic is represented as a series of conditions and actions in your business process.

Example Decision Node

The following figure shows an example **Decision** node in the **Design View**.



Building on the QuoteRequest example used in building the [Variable \(Schema\)](#) condition, two **Perform** nodes are added to the paths on the **Decision** node. At run time, the following sequence represents the flow of control in this decision node:

1. The condition defined on the **Check Customer** condition node is evaluated:

```
data($requestXML/ns0:customerName = "BEA")
```

Note: The XML evaluated by the condition node is assigned to the `requestXML` business process variable.

2. If the **Check Customer** condition evaluates to *true* at run time, the activities defined on the **BEA Orders** node are performed, then the flow exits the **Decision** node.
3. If the **Check Customer** condition evaluates to *false* at run time, the path of execution is the **Default** path. The activities defined on the **Non BEA Orders** node are performed, then the flow of control exits the **Decision** node.

Related Topics

[Creating a Decision Node in Your Business Process](#)

[Grouping Nodes in Your Business Process](#)

[Handling Exceptions](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

[Business Process Source Code](#)

[Interacting With Resources Using Controls](#)

Creating Case Statements

A Switch node is used to select one path of execution based on the evaluation of an expression specified on a condition node. A Switch node contains one condition node, one or more case paths, and one default path. At run time, the expression on the condition node is executed, and the resulting value is compared to the values associated with each case path. Execution continues with activities inside the first case path that contains a matching value (case paths are evaluated left-to-right in the Switch node). When no conditions are met, activities defined on the default path are executed.

This section describes how to add a Switch node to your business process, define conditions, and define activities for the alternative paths of execution in the Switch node. It contains the following topics:

- [Comparing Decision Nodes and Switch Nodes](#)
- [Creating a Switch Node](#)
- [Designing a Switch Node](#)

Comparing Decision Nodes and Switch Nodes

How does a **Decision** node differ from a **Switch** node?

A **Decision** node can include one or more conditions to be evaluated at run time. For a scenario in which a **Decision** node is defined, the business process evaluates the conditions (one on each path) sequentially, and executes the path for the first condition that evaluates as *true*. (Conditions are evaluated left-to-right in the **Decision** node.) In other words, if the first condition evaluates to false, the second condition is evaluated. If the second condition evaluates to false, the next

condition is evaluated, and so on. The activities defined on the default path are executed if no conditions are met.

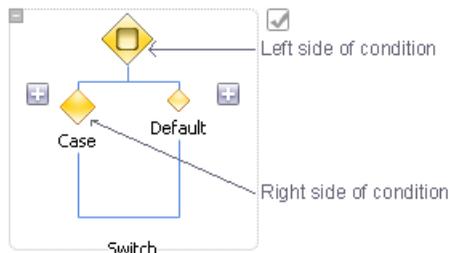
A **Switch** node includes a single condition. For a scenario in which a **Switch** node is defined, the business process evaluates an expression specified on a single condition node and selects one path of execution based on the evaluation of that expression. The possible paths of execution in a **Switch** node include one or more case paths, and one default path. Execution continues with activities inside the first case path that contains a matching value. (Case paths are evaluated left-to-right in the **Switch** node.) If the value resulting from the evaluation of the condition expression does not match any of the case paths, then the activities defined on the default path are executed.

Creating a Switch Node

To Create a Switch Node in Your Business Process

1. On the **Application** tab, click the business process (JPD file) you want to design.
Your business process is displayed in the **Design View**.
2. If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the WebLogic Workshop menu.
3. Click  **Switch** in the **Palette**.
4. Drag and drop the **Switch** node onto the business process in the **Design View**, placing it on the business process at the point in your business process that requires branching to one of several possible paths of execution, based on the evaluation of one or more conditions.

The **Design View** is updated to contain a **Switch** node, as shown in the following figure:



Note the following characteristics of the **Switch** node:

- Adding a **Switch** node to a business process adds, by default, a single *Switch* node, a *Case* node, and a *Default* node.

- You can add one or more additional Case nodes. To do so, right-click on the **Switch** node and select **Add Case** from the drop-down list, or click the  on either side of the Case tree.

At run time, the case branch which matches the received data on the node is executed. If no matching case is found, the default path is executed.

- You can change the name of the **Switch** node and each **Case** in a **Switch** node. To do so, double-click the name assigned to the **Case** or **Switch** node and enter a new name.
- indicates that the design of this node is incomplete. When you complete the design of the node, is replaced by . A Switch node is completed when the condition and all cases are fully configured.
- A **Switch** node is, in effect, a group of nodes. You can view and edit the properties of your **Switch** node by clicking the outline of the group to select it, then viewing the group properties in the **Property Editor**. To learn about groups, see [Grouping Nodes in Your Business Process](#).

Related Topics

[Designing a Switch Node](#)

Designing a Switch Node

To create logic for your **Switch** node, you must complete the following steps:

- [To Design the Switch Logic](#)
- [To Specify the Case Statement](#)
- [To Add Activities to the Paths in Your Switch Node](#)

To Design the Switch Logic

1. Double-click the **Switch** node to invoke the condition builder.
2. Select the option which you want the left side of your condition to be based on:
 - **Variable**—Select this option if, at run time, you want the business process to evaluate a match based on the value of an element in an XML document or a MFL file.
 - **Method**—Select this option if, at run time, you want the business process to evaluate a match, based on a result returned from Java code that you create.

The node builder displays options depending on whether you selected **Variable** or **Method**.

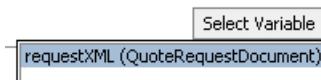
3. Complete the selections in the node builder appropriate for the selection you made in the preceding step: Schema or Method.

Variable (Schema)

The following steps describe how to select a business process variable, which is associated with an XML or MFL schema. Select one or more nodes in the schema on which to define a switch or case node.

1. In the decision builder, select a business process variable by clicking **Select Variable**.
A drop-down list of business process variables in your project is displayed.
2. Select a variable that you have already created in your project, or create a new variable to use in your switch node:
 - a. If you want to use a variable that is already created, select the variable that contains the XML or typed non-XML on which you want to build the condition.

For example, if we import an XML Schema (`QuoteRequest.xsd`) into our project, and create a business process variable (`requestXML`) of type `quoteRequest` (based on the `QuoteRequest.xsd` schema), the `requestXML` variable is available in the drop-down list of business process variables:



Note: (To learn about creating business process variables and importing schemas to your project, see [Business Process Variables and Data Types](#) and [Importing Files into the Schemas Project](#).)

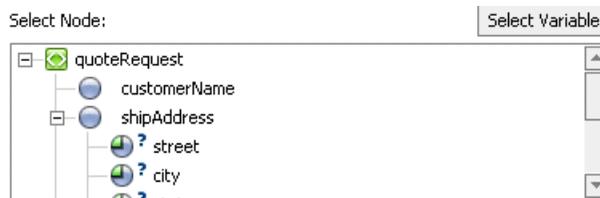
When you select a variable, a representation of the XML Schema associated with that variable is displayed in the **Select Node** pane. Go to [step 3](#).

- b. If you want to create a new variable, select **Create new variable...** from the drop-down list.
The **Create Variable** dialog box opens.
- c. Enter a name for your new variable in the **Variable Name** field.

- d. Select the **XML** or **nonXML** option, depending on whether your variable is based on an XML document or MFL file and select the appropriate variable type in the displayed list of type options.
- e. Click **OK**.

The **Create Variable** dialog box closes and your new variable is displayed in the **Select Node** pane.

3. Building on our `requestXML` variable example, the following figure shows the XML Schema represented when the `requestXML` variable is selected:



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. Note that the schema in our example (`QuoteRequest.xsd`) specifies a root element (`quoteRequest`), and child elements: `customerName`, `shipAddress`, and `widgetQuoteRequests`. The `widgetQuoteRequests` element, in turn, specifies a repeating element:

`widgetQuoteRequest`. (A repeating XML element is represented by  in the GUI representation of the Schema.)

4. In the **Select Node** panel, select the node in the XML Schema for which you want to define the switch.
5. The node which you selected is displayed in the **Selected Element** field. For example, if you selected the element `street` in the preceding example, the `data($requestXML/ns0:shipAddress/@street)` is displayed in the **Selected Element** field.
6. Click the **X** in the top right-hand corner to return to the **Design View**.

In the **Design View**, note that the **Condition** in your **Decision** node displays the following icons:

-  is a visual reminder that the condition you defined on this node is based on the evaluation of an XML document.
-  is a visual reminder that the condition you defined on this node is based on the evaluation of a MFL file.

7. To save your work, select **File—Save**.

Method

1. Enter a name for the Java method in the **Java Method Name** field.

Note: To select an existing method, click  on the right side of the **Java Method Name** field.

2. Click **View Code** in the lower right-hand corner of the Switch builder.

The **Source View** is displayed at the line of code in your JPD file at which the Java method is written.

3. Edit your Java method and click the **Design View** tab to return to the **Design View**.
4. Click the **X** in the top right-hand corner to close the decision builder.

In the **Design View**, note that the **Condition** in your **Switch** node displays the following icon: . It is a representation of the condition you defined in source code to specify the Java method, on which to base the decision. To make any further changes to the condition represented on this node, you must edit the source code in the **Source View**.

To Specify the Case Statement

1. Double-click the **Case** node to invoke the case builder.
2. Select the option which you want the right side of your condition to be based on:
 - **Schema**—Select this option if, at run time, you want the business process to evaluate a match based on the value of an element in an XML document or an MFL file.
 - **Method**—Select this option if, at run time, you want the business process to evaluate a match based on a result returned from Java code that you create.
 - **Constant or Variable**—Select this option if, at run time, you want the business process to evaluate a match based on a constant that you specify.

The node builder displays options depending on whether you selected **Schema**, **Method**, or **Constant or Variable**.

3. Complete the selections in the node builder appropriate for the selection you made in the preceding step: Schema, Method, or Constant or Variable.

Schema

For information about how to complete the Case builder when using the **Schema** option, see [Variable \(Schema\)](#) in the preceding section.

Method

For information about how to complete the Case builder when using the **Method** option, see [Method](#) in the preceding section.

Constant or Variable

1. In the **Value** field, enter the constant value or variable that you want to match the case statement to.

Note: You can select an existing variable or create a new one by clicking  on the right side of the **Constant Value** field.

2. To close the node builder, click the **X** in the top right-hand corner.

To Add Activities to the Paths in Your Switch Node

After you define the condition that is evaluated when the flow transitions to the **Switch** node at run time, you are ready to define the actions on the paths that represent the paths of execution in the flow.

1. Add a node (or nodes) to each path in the **Switch** node to define the activity that is executed when the conditions you defined on the **Case** nodes at the beginning of the path match at run time.

This can be any node that performs the activity appropriate for your business process business logic. For example you can use a control to interact with an external resource, such as a database, a JMS queue, or an EJB.

2. Add a node (or nodes) to the default path, to define activities that are executed when none of the case statements match at run time. The nodes on the default path can be any that define activities appropriate for your business process business logic.

When you complete the addition of activities on the paths of your **Switch** node, your decision logic is represented as a series of conditions and actions in your business process.

3. To save your work, select **File—Save**

Related Topics

[Grouping Nodes in Your Business Process](#)

Creating Case Statements

[Creating a Switch Node](#)

[Handling Exceptions](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

Writing Custom Java Code in Perform Nodes

Although users are free to modify and write custom Java code almost anywhere, **Perform** nodes provide a means for visually representing custom code within the process diagram. When you add a **Perform** node to your business process, a method is created in the JPD file. You subsequently customize the method signature in the **Source View**.

This section describes how to create and customize a **Perform** node for your business process.

To Create a Perform Node in Your Business Process

1. On the **Application** tab, click the business process (JPD file) you want to add the **Perform** node to.

Your business process is displayed in the **Design View**.

2. If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the WebLogic Workshop menu.
3. Click  **Perform** in the **Palette**. Then drag and drop the **Perform** node onto the business process in the **Design View**, placing it on the business process at the point in your business process at which you want to create custom Java code.

The **Design View** is updated to contain the **Perform** node.

4. Double-click the **Perform** node in the **Design View** to open the node builder.

This node builder allows you to name the node and the associated Java method.

Note: You can select an existing method by clicking  on the right side of the **Java Method Name** field.

5. Click **View Code** in the lower right-hand corner of the **Perform** builder.

The **Source View** is displayed at the line of code in your JPD file at which the Java method is written.

For example, if you created a method named `checkInventory`, the following code is written to the source file.

```
public void checkInventory() throws Exception {  
}
```

6. Customize this method with your Java code.
7. Click the **Design View** tab to return to the **Design View**.
8. Click the **X** in the top right-hand corner to close the node builder.

Your JPD file is updated to reflect the changes you made in the node builder.

9. To save your work, select **File—Save**.

Related Topics

[Perform Methods](#)

[Handling Exceptions](#)

Creating Looping Logic

Frequently, your business logic requires that you create looping logic in your business processes. That is, you need to design logic in your business process in which the activities enclosed in a loop are performed repeatedly while a specific condition is true.

Do While, **While Do**, and **For Each** node groups represent such points in your business processes. This section describes how to design looping logic in your business processes. It includes the following topics:

- [Understanding While Node Groups](#)
- [Creating While Node Groups in Your Business Process](#)
- [Designing While Node Groups](#)
- [Looping Through Items in a List](#)
- [Creating For Each Nodes in Your Business Process](#)
- [Designing For Each Nodes](#)

Understanding While Node Groups

Both **Do While** and **While Do** node groups support looping logic. Both types of groups represent a point in a business process at which the activities enclosed by the group are performed repeatedly while a specific condition is true. However, **Do While** and **While Do** groups represent different execution logic, as described in the following sections:

- [While Do Node Groups](#)

- [Do While Node Groups](#)

While Do Node Groups

At run time, the condition on a **While Do** group is evaluated before the activities in the loop are performed. Therefore, the activities inside **While Do** groups are performed zero or many times, depending on the results of the evaluation of the condition.

Do While Node Groups

In the case of **Do While** groups, business process activities are added before the condition in the loop. At run time, the activities defined in a **Do While** loop are performed; then the condition is evaluated. Therefore, the activities inside a **Do While** group are performed one or many times, depending on the results of the evaluation of the condition.

Related Topics

[Creating While Node Groups in Your Business Process](#)

[Designing While Node Groups](#)

Creating While Node Groups in Your Business Process

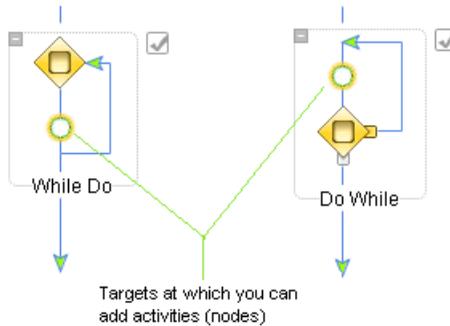
To Add A While group to Your Business Process

1. On the **Application** tab, click the business process (JPD file) you want to add the While group node to.

Your business process is displayed in the **Design View**.

2. If the **Palette** is not visible in WebLogic Workshop, choose **View**—**Windows**—**Palette** from the menu bar.
3. Determine whether you need *While Do* or *Do While* looping logic, and click  **While Do** or  **Do While** in the **Palette**.
4. Drag and drop the node group you selected onto the business process, placing it on the business process at the point in your business process at which you want to design a looping logic.

The **Design View** is updated to contain a representation of the group you selected.



Note the following characteristics of the **While** groups:

- Each **While** group represents a placeholder for one or more additional nodes, a looping mechanism, and a condition builder . The condition builder allows you to build the condition, which your business process evaluates for each iteration through the loop at run time.
- In the case of **While Do** groups, the design specifies that the condition on the group is evaluated before the activities in the loop are performed. In contrast, in the case of the **Do While** groups, the design specifies that the activities defined in the loop are executed first at run time, then the condition is evaluated.
- You can change the name of the **While** groups by double-clicking the name of the group in the **Design View**, and entering a new name.
- indicates that the design of a group is incomplete. When you complete the design of the group, is replaced by . **While** groups are completed when the condition group is properly configured.

Related Topics

[Designing While Node Groups](#)

[Understanding While Node Groups](#)

Designing While Node Groups

To create logic for your **While** group, you must complete the following steps:

- [Design the Condition Logic](#)

- [Add Activities to the Paths in Your While group](#)

Design the Condition Logic

Double-click  in the **While** group you want to design the condition logic for.

The node builder is displayed. It allows you to create the condition or conditions that are evaluated at run time by the business process.

Note: The node builder in which you create the conditions for looping is the same as that in which you create conditions on **Decision** groups. To learn how to design the condition logic for the **While** group, see [To Design the Condition Logic in Defining Conditions For Branching](#).

Add Activities to the Paths in Your While group

After you define the condition that is evaluated in your **While** loop at run time, you are ready to define the actions on the loop. To do so, add a node (or nodes) to the path in the **While** loop. You can add any nodes that perform the activities appropriate for the business logic that you require at this point in your business process.

When you complete the addition of activities on the group, your looping logic is represented by a condition or conditions and a series of business process nodes in the **While** loop.

You can view and edit the properties of your **While** group by clicking the outline of the group to select it, then viewing the group properties in the **Property Editor**.

For any *group* of nodes in your business process, including **While** groups, you can collapse the group to save space on the **Design View** canvas. Collapsed groups appear in the **Design View** as shown in the following figure:



To expand the group, click .

Related Topics

[Defining Conditions For Branching](#)

[Looping Through Items in a List](#)

[Grouping Nodes in Your Business Process](#)

[Handling Exceptions](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

Creating Looping Logic

Looping Through Items in a List

A frequently designed pattern in your business processes is one that specifies the performance of a set of activities once for every iteration of the flow over a sequence of XML elements, retrieved from an XML document.

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 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 XML document (or a section of an XML document) is passed into the **For Each** loop in a business process variable. An iteration variable holds the current element being processed in the **For Each** loop, for the life of the loop.

This section describes how to add this looping logic to your business process. It includes the following topics:

- [Creating For Each Nodes in Your Business Process](#)
- [Designing For Each Nodes](#)

Creating For Each Nodes in Your Business Process

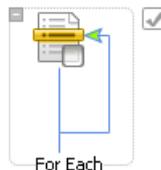
To Add A For Each Node to Your Business Process

1. On the **Application** tab, click the business process (JPD file) you want to add the **For Each** node to.

Your business process is displayed in the **Design View**.

2. If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the WebLogic Workshop menu.
3. Click  **For Each** in the **Palette**. Then drag and drop the **For Each** node onto the business process, placing it on the business process at the point in your business process at which you want to design a pattern in which a set of activities is performed repeatedly, once for each item in a list.

The **Design View** is updated to contain a representation of the **For Each** node as shown in the following figure:



Note the following characteristics of the **For Each** node:

- The **For Each** node represents a placeholder for one or more additional nodes, and a looping mechanism. The node builder, which is described in [Designing For Each Nodes](#), allows you to easily select a sequence of nodes from a business process variable.
- By default, the node is named **For Each**. You can change the name by double-clicking the name and entering the new name.
- indicates that the design of this node is incomplete. When you complete the design of the node, is replaced by . A **For Each** node is completed when the iterator has been specified in the node builder.

Related Topics

[Designing For Each Nodes](#)

Designing For Each Nodes

Before you can add the logic that causes the iteration over a sequence of XML nodes in your business process, your project must contain an XML Schema or MFL file that defines the repeating XML or MFL element over which you want your business process to iterate. To learn how to import an XML Schema or MFL file into your project, see [Importing Files into the Schemas Project](#).

After importing an XML Schema or MFL file into your project, you can complete the design of the **For Each** node. It includes the following tasks:

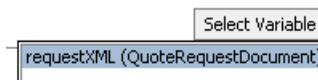
- [To Select a Repeating XML or MFL Element Over Which to Iterate](#)
- [To Add Activities to the For Each Node](#)

To Select a Repeating XML or MFL Element Over Which to Iterate

The **For Each** node only iterates over repeating elements. The node builder allows you to select a repeating node from the variable you created in the preceding section.

1. In the **Design View**, double-click the **For Each** node to invoke its node builder.
2. Click **Select Variable** to select a variable that you have already created in your project or create a new variable to use in your decision node:
 - a. If you want to use a variable that is already created, select the variable that contains the XML or typed non-XML on which you want to build the condition.

For example, if we import an XML Schema (`QuoteRequest.xsd`) into our project, and create a business process variable (`requestXML`) of type `quoteRequest` (based on the `QuoteRequest.xsd` schema), the `requestXML` variable is available in the drop-down list of business process variables:



Note: To learn about creating business process variables and importing schemas to your project, see [Business Process Variables and Data Types](#) and [Importing Files into the Schemas Project](#).

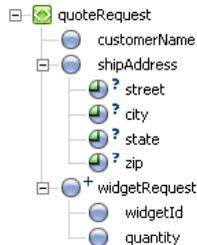
Go to [step 3](#).

- b. If you want to create a new variable, select **Create new variable...** from the drop-down list.

The **Create Variable** dialog box opens.
- c. Enter a name for your new variable in the **Variable Name** field.
- d. Select the **XML** or **nonXML** option, depending on if your variable is based on a XML document or MFL file and select the appropriate variable type in the displayed list of type options.
- e. Click **OK**.

The **Create Variable** dialog box closes and your new variable is displayed in the **Select Node** pane.

- Continuing with our example, a representation of the XML in the `requestXML` variable is displayed in the **Select Node** panel.



Note the following characteristics of the `QuoteRequest.xsd` schema as displayed in the preceding figure:

- The elements and attributes of the XML Schema are represented as nodes. Note that the Schema in the example (`QuoteRequest.xsd`) specifies a root element named `quoteRequest`.
- Child elements include: `customerName`, `shipAddress`, and `widgetRequest`.
- The `shipAddress` element specifies the following attributes: `street`, `city`, `state`, and `zip`.
- The `widgetRequest` element is a repeating element.

There can be one or more occurrences of the `widgetRequest` element in an associated XML document; this is represented by \oplus in the GUI representation of the schema. The `widgetRequest` element, in turn, contains two elements: `widgetId` and `quantity`.

Note: In the example in the preceding figure, the repeating XML element (*one or more* occurrences) is represented by \oplus in the GUI representation of the schema. A repeating XML element that specifies *zero or more* occurrences is represented by \oplus^* .

- Select a repeating element in the **Select Node** field.

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

- Repeating Element**—Contains an XPath expression, which when applied against the XML document associated with the XML variable, returns the set of repeating XML elements. Building on our example, if you select the repeating element **widgetRequest** in the **Select Node** panel, the **Repeating Element** field is populated with the following

expression: `$requestXML/ns0:widgetRequest`. This expression returns all the `widgetRequest` elements in an XML document.

- **Iteration Variable**—Contains the name of an iteration variable. An iteration variable is generated to hold the current element being processed in the **For Each** loop at run time. In our example, the iteration variable is named `iter_forEach1`, by default. You can change the name by entering a new name in the **Iteration Variable** field.
5. Click the **X** in the top right-hand corner to close the node builder and return to the **Design View**.

In the **Design View**, note that the icon in your **For Each** node displays the following graphics:

-  is a visual reminder that the iteration variable you defined on this node is based on an XML element.
 -  is a visual reminder that the iteration variable you defined on this node is based on a MFL or typed non-XML element.
6. To save your work, select **File**→**Save**.

To Add Activities to the For Each Node

You must define the activity or set of activities that are performed for each item in the list you created in the preceding step ([To Select a Repeating XML or MFL Element Over Which to Iterate](#)). Each iteration of the **For Each** loop executes the activity or activities you specify in a node (or nodes) in the loop.

1. In the **Palette**, click a node that represents the logic you want to add to the business process.
2. Drag and drop the node from the **Palette** onto the business process in the **Design View**, placing it on the business process within the **For Each** loop.

As you drag a node onto the **For Each** loop, a target  appears on the loop, representing a valid location in the **For Each** loop where you can place the node. As you drag the node near the valid location, the target is activated  and the cursor changes to an arrow . You can release the mouse button and the node snaps to the **For Each** loop.

3. On the node (or nodes) you add to the **For Each** loop, create the activities appropriate for your business process's business logic.

To learn about designing **For Each** loops and how data is assigned to variables within the loops, see [Looping Through Items in a List](#) in *Tutorial: Building Your First Business Process*.

Related Topics

[Creating Looping Logic](#)

[Grouping Nodes in Your Business Process](#)

[Handling Exceptions](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

Specifying Endpoints in Your Business Process

When you create a business process, it contains by default a **Start** and a **Finish** node. You can specify additional (optional) endpoints of your business process by adding **Finish** nodes to those locations where you want the business process to cease execution. A **Finish** node is always the last node in a business process. You can place a **Finish** node at the end of the main flow or on any branch of a business process.

To Create a Finish Node in Your Business Process

1. On the **Application** tab, click the business process (JPD file) you want to design.
Your business process is displayed in the **Design View**.
2. If the **Palette** is not visible in WebLogic Workshop, choose **View—Windows—Palette** from the WebLogic Workshop menu.
3. Click  **Finish** in the **Palette**.
4. Drag and drop the **Finish** node onto the business process, placing it on a branch in the business process at the point where you want to specify the termination of your business process.
The **Design View** is updated to contain your **Finish** node.
5. To save your work, select **File—Save**.

Specifying Endpoints in Your Business Process

Grouping Nodes in Your Business Process

You can create a *group* from one or more nodes or other groups. You can simplify the display of your business process in the **Design View**, by collapsing a group of nodes into a single node. A group can provide an extra level of exception handling logic—exception handlers that you specify for a group catch exceptions that are not handled by exception handlers defined for nodes inside the group.

You can specify groups of nodes in a business process. Also, some business process nodes are implicit groups—Client Request with Return, Decision, For Each, Do While, While Do, Parallel, Switch and Event Choice. Groups that represent these nodes and groups defined by you have the same characteristics.

This section describes how to work with groups in your business processes. It includes the following topics:

- [To Create Groups—Alternative 1](#)
- [To Create Groups—Alternative 2](#)
- [To Delete, Ungroup, Collapse, or Expand Groups](#)
- [To Activate Exception, Message, and Timeout Paths for Groups](#)

To Create Groups—Alternative 1

1. In the **Design View**, shift-click on the nodes you want to group to select them or click and drag your mouse around the nodes you want to group.
2. Right-click one of the selected nodes and select **Group Selected** from the drop-down menu.

The specified nodes are grouped inside a collapsible box in the **Design View**.

You can name the group by double-clicking the default name (**Group**) and entering the name you want to assign.

Note: If you select a set of nodes and right-click to display the drop-down menu, the **Group Select** command is unavailable if the grouping of these nodes would create an invalid business process.

To Create Groups—Alternative 2

1. In the Design View, drag and drop  **Group** from the **Palette** onto the business process, placing it on the business process at the point at which you want to create a group.

An empty **Group** is created in the business process.

2. Drag and drop nodes from the **Palette** that you want to add to the group onto the business process, placing them within the group.

To Delete, Ungroup, Collapse, or Expand Groups

1. Click the outline or label (name) of a group to select it.
2. Right-click to display the drop-down menu.
 - To collapse the group into a single entity, select **Collapse**. The group is collapsed.



Group

- To expand the representation of the group again, right-click the collapsed group, select **Expand**.
- To undo the grouping of the node, select **Ungroup**.

Note: You must first expand a collapsed group to ungroup it.

Collapsing simplifies the view of your business process in the **Design View**.

You can also toggle between collapsed and expanded groups by clicking  or  in the upper left-hand corner of the group.

- Select **Delete** to delete the group.

Warning: When you delete a group, you delete all the contents of that group.

To Activate Exception, Message, and Timeout Paths for Groups

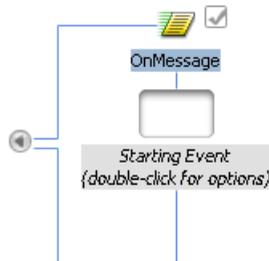
Activate an exception, a message, or a timeout path for a group of nodes in the following way:

1. In the **Design View**, click the outline of the group to select it.
2. Right-click and select **Add Exception Path, Add Message Path, or Add Timeout Path** from the drop-down menu.
 - The following graphic associated with a group indicates that an exception handling path is activated for the group:



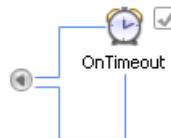
For more information about exception paths and how to configure them, see [Handling Exceptions](#). To learn about the settings in the **Property Editor**, see [Creating Exception Handler Paths](#).

- The following graphic associated with a group indicates that a message path is activated for the group:



For information about how to configure your message path, see [Adding Message Paths](#).

- The following graphic associated with a group indicates that an Timeout path is activated for the group:



For information about how to configure your timeout path, see [Adding Timeout Paths](#).

You can add business process nodes to the paths shown in the preceding figure, as required to define the exception handling logic.

Related Topics

[Handling Exceptions](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

Each of the nodes described in the following topics is implicitly a group of nodes:

[Defining Conditions For Branching](#) (**Decision** nodes)

[Creating Case Statements](#) (**Switch** nodes)

[Receiving Multiple Events](#) (**Event Choice** nodes)

[Creating Looping Logic](#) (**While Do** and **Do While** nodes)

[Looping Through Items in a List](#) (**For Each** nodes)

Handling Exceptions

Business Process exceptions are Java exceptions that are not caught by the Java handler methods. This section describes the ways in which you can handle exceptions in your business processes. It includes the following topics:

- [Types of Exception Handlers](#)
- [Creating Exception Handler Paths](#)
- [Deleting Exception Handler Paths](#)
- [Order of Execution of Exception Handlers](#)
- [Handling Exceptions in Transaction Blocks](#)
- [Using Exception Handlers for Compensation](#)
- [Unhandled Exceptions](#)

Types of Exception Handlers

You can use the **Design View** to create exception paths on business process nodes, collapsible groups of nodes, and the Start node. Specifically, using the Design View, you can create the following types of exception handlers in your business process:

- Global exception handler

You can create a global exception handler for your business process by creating an *exception path* for the **Start** node. You create logic for the exception handler path to define

the flow of execution in case of an exception. A global exception handler responds to exceptions that are otherwise not handled in the business process.

- Exception handler for a group of nodes

You can associate an *exception path* with a group of nodes and create logic for the exception path that defines the flow of execution in case of an exception.

- Exception handler for an individual node

You can associate an *exception path* with an individual node and create logic for the exception path that defines the flow of execution in case of an exception.

In general, exceptions propagate upwards from a node exception path, to a group exception path, to a global exception path until they are handled. In other words, the exception path associated with a node executes first, then the path associated with a group executes, and then the path associated with a start node (global path) executes. The exception is only handled once, unless the exception path throws an exception, then the exception propagates upward again in the same order. You can take advantage of this behavior and create exception path logic that satisfies the particular exception handling necessary for your business process. For more information, see [Order of Execution of Exception Handlers](#).

Note: The graphical design environment does not support or represent the design of exception paths on the following nodes in the process language: `<if>` and `<default>` blocks inside Decision nodes, `<branch>` blocks inside Parallel nodes, `<finish>` nodes, `<messageEvent>`, `<timeoutEvent>`, and the `<onException>` path itself.

Creating Exception Handler Paths

You can associate an exception handler path with individual nodes in your business process and with groups of nodes. An exception path that you associate with a **Start** node is a special case. That is, the exception path associated with a **Start** node is the global exception handler for the business process. To learn more about **Start** nodes, see [Starting Your Business Process](#). This section contains the following steps and procedures:

- [To Create an Exception Path](#)
- [To Configure an Exception Path](#)

To Create an Exception Path

1. Select the node or groups of nodes for which you want to create an exception path. (For information on how to group nodes, see [Grouping Nodes in Your Business Process](#))

2. Right-click the node or group of nodes and select **Add Exception Path** from the drop-down menu.

An exception path is added to your node or group of nodes and is displayed as follows.



You can rename the exception path to anything you like by double-clicking **OnException** and entering the new name. You can also change the name in the **name** field of the **Property Editor**.

To Configure an Exception Path

1. Select the exception path which you want to configure.

The related properties are displayed in the **Property Editor**. If the **Property Editor** is not visible in WebLogic Workshop, choose **View—Property Editor** from the menu bar.

2. In the **Property Editor**, configure the following properties:

- **name**—Enter the name you want displayed in the graphical design environment for this exception path.
- **notes**—Enter any notes you want associated with this exception path. These notes can then be accessed through the WebLogic Integration Administration Console.
- **after execute**—Select the action you want to take place after an exception path is executed and all retries have been exhausted. Choose from:

skip—Skip the node or group with which the exception path is associated. That is, resume execution of the process at the node following the node or group for which the exception path is defined.

resume—Execution resumes after the closest transaction block, exception block, or failing node on the stack. In other words, the execution of the process resumes at the node following the one that threw the exception (this could be within a group, or if the node that threw the exception is the last in a group, the node following a group of nodes).

rethrow—Nodes on the exception path are executed and then the same exception is rethrown and handled by the exception handler at the next level up.

- **retry count**—Specify how many times, after the first attempt, the process engine tries to execute the node or group of nodes contained in the path, before the `afterExecute` path is taken. The counter is evaluated and incremented at the end of handler execution.
- **execute on rollback**—Set this to true if you want this exception path executed when the associated transaction is rolled back.

The execute on rollback parameter enables exception handlers to be used for compensation. By grouping nodes and adding an exception path to that group with the **execute on rollback** property set to true, you can specify that the exception handler should be run before transaction rollback, thereby providing an opportunity to *clean-up* non-transactional resources that would otherwise not be effected by the rollback. For more information, see [Using Exception Handlers for Compensation](#).

Note: Be aware of the following behavior when you specify a **retry count** in combination with setting the **after execute** parameter to **resume**: Specifying a **retry count** on an exception handler that is attached to a group causes the retry to start at the beginning of the group—not at the offending node. However, if you also specify the **resume** option for the **after execute** property, after all the retries are exhausted, the execution can continue from a point *within* the group, following the offending node.

3. Add any business process nodes to the exception path, as required to define the exception handling logic. If you want your process to stop after catching an exception, place a Finish node on the exception path. For information about how to create a Finish node, see [To Create a Finish Node in Your Business Process](#).

Viewing Exception Handlers in the Design View

When you create an exception handler path, the following icon appears beside a node (or group of nodes) in the **Design View**, which indicates that an exception path is activated for the specified node:



This icon represents the exception path in your business process. In this case, the path appears empty, indicating that the logic to handle an exception is not defined yet.

To define the exception handling logic, add business process nodes by dragging the nodes from the Business Process **Palette** and dropping them on the exception path.

To collapse the view of any exception handler (or message or timeout path), click the grey arrow of the exception path icon. The following figure shows the icon associated with your node to indicate a collapsed path.



You can toggle between collapsed and expanded views of paths in the **Design View** by clicking the exception path icon.

Deleting Exception Handler Paths

To Delete an Exception Path:

1. If the exception path that you want to delete is collapsed, expand it.
2. Right-click the exception path, then select **Delete** from the drop-down menu.

The exception path is deleted and removed from the **Design View**.

Warning: Deleting an exception path deletes any business process nodes you defined on that path. When you attempt to delete an exception path, a dialog box displays a warning message that you must acknowledge before proceeding with the deletion.

Order of Execution of Exception Handlers

If an exception occurs, the normal flow of execution stops. The business process executes the activities inside the exception handler path defined closest to the point of the exception.

You typically define a number of exception handlers in your business process. The following sequence defines the order of their execution when an exception is thrown:

1. The business process engine first executes the exception handler at the node on which the exception occurs.
2. If the exception handler path completes execution normally, the business process resumes execution at the node following the node associated with the executed exception handler, based on the post-execute parameter setting.
3. If the exception handler throws an exception while it is executing, the exception is propagated upwards to be handled either by an exception handler on the group of nodes in which the node is contained, or by the global exception handler defined on the Start node.

Note: If you have the **after execute** property set to rethrow, the exception itself will also propagate upwards.

4. When a business process fails and there is no exception handler configured to handle the exception thrown, the business process is placed into an aborted state and no recovery is possible. However, if the business process is configured to **freeze on failure**, the business

process rolls-back to the last commit point and the state is persisted if it fails. The process can then be restarted from the WebLogic Integration Administration Console. To learn more about the **freeze on failure** property see, [Setting the Business Process Properties](#).

Note: If an exception occurs within a transaction block, the transaction is rolled back and the exception handler is called at a later time. However, if the business process is marked **freeze on failure**, instead of calling the exception handler later, the process freezes and the exception handler is never called. In this case, when **freeze on failure** is configured inside a transaction block, you should either not use a transaction block or include global transaction logic within your transaction blocks.

Handling Exceptions in Transaction Blocks

If a node or group within a transaction throws an exception, the transaction will only *see* the exception if the exception is not handled or if an exception handler throws an exception. The following algorithm is used to handle the exception:

- If the transaction is not marked for **rollback only**, the exception is dispatched to the exception handlers defined within the transaction block, if any.
- If an exception handler within the transaction block does not throw or rethrow an exception, the transaction is not rolled back, and execution resumes after the block that enclosed the exception handler.
- If the transaction is marked for **rollback only**, or if there is no exception handler within the transaction block, or if all the exception handlers throw exceptions, the transaction is rolled back. After exhausting a specified number of retries, a business process exception is thrown from the transaction block in a new transaction. The business process exception is a generic exception, because there is no way to retain the root cause exception on a rollback.

Note: Whether a transaction is marked for **rollback only** depends on the types of transactional resources you use in your business process.

For transactional resources that force a transaction to roll back immediately in the case of an error, an exception handler on a node or group of nodes does not run before the transaction rolls back. However, you can use exception handlers with the **execute on rollback** property for compensation and to clean up the non-transactional resources. For more information, see [Using Exception Handlers for Compensation](#).

Using Exception Handlers for Compensation

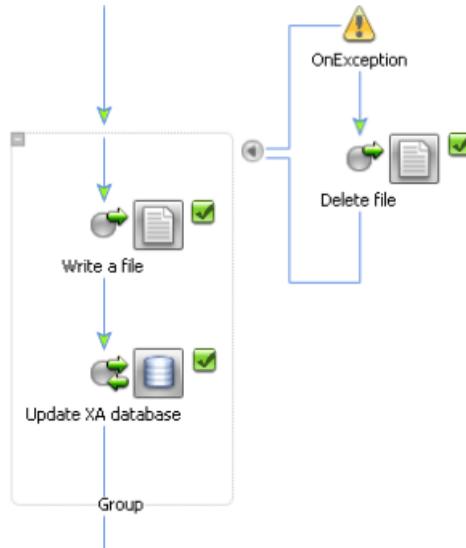
Transactional resources are any resources that communicate with your business process through a Control Request node or any of the transactional controls: Database, JMS, Application View (if JCA adapter is transactional), Worklist, Timer, EJB, Message Broker, and Transformation.

For transactional resources that force a transaction to roll back immediately in the case of an error, an exception handler on a node or group of nodes does not run by default. However, you can force the exception handler to run before the rollback occurs by placing an exception path inside an explicit transaction block in your business process, and setting the **execute on rollback** property of that path to **true**. In this way, the exception path has access to the current state (the process variables, and so on) and the logic added to the exception path can be used for compensation and to clean up the non-transactional resources, as described in the [Compensation Example](#).

To learn about the Control Request node, and about controls and transactions see [Create a Client Request Node in Your Business Process](#) and [Controls and Transactions](#).

Compensation Example

The following figure shows an example of how to use an exception path for compensation.



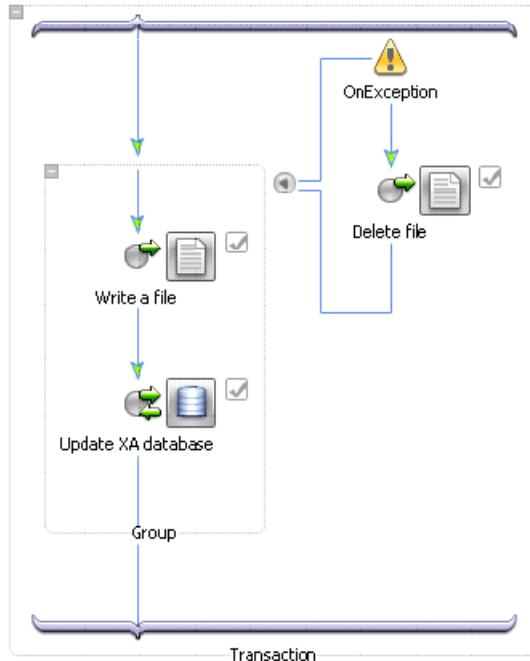
In the example, two nodes are running in the same transaction. The first node writes to a file (non-transactional) and the second node updates a database (transactional). Both nodes are within

a group that has an associated exception handler. The exception handler **execute on rollback** property is set to true to force the exception handler to run before any rollback occurs.

In this example, if the database operation fails and marks the transaction for rollback only, the following sequence of events occurs:

1. The exception handler runs before the rollback occurs.
2. The transaction handler path is executed and the node on the path deletes the file that was written earlier.
3. The transaction rolls back.

Note: The above example is for an implicit transaction. You can use the same technique for compensation with explicit transactions. However, be sure that you put the exception handler path inside the explicit transaction. Putting it on the explicit transaction itself does not result in the desired behavior. The following figure shows an example of an explicit transaction with an exception handler path with compensation logic.



Unhandled Exceptions

If you do not handle exceptions in a business process, the exception will be wrapped in one of the following:

- `com.bea.wli.bpm.runtime.UnhandledProcessException`
- `ProcessControlException`
- `ServiceControlException`
- `JpdProxyException`

If you need to obtain the original exception, you can call `getCause()` on the unhandled process exception. To learn more about this method, see `getCause()` in the *Java 2 Platform, Standard Edition, v 1.4.2 API Specification*, which is available at the following URL:

[http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Throwable.html#getCause\(\)](http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Throwable.html#getCause())

Related Topics

[Grouping Nodes in Your Business Process](#)

[Writing Custom Java Code in Perform Nodes](#)

[Adding Message Paths](#)

[Adding Timeout Paths](#)

[Transaction Boundaries](#)

Handling Exceptions

Adding Message Paths

A Message Path is used to interrupt an executing process on delivery of a message from either a client or a control. This allows the process to halt the current stream of execution and take alternate actions. You can have as many message paths as you like in your business process.

Message paths can be associated with individual nodes, a group of nodes, or with the process (global).

Note: Message paths are not supported on the following individual nodes: Perform, Client Response, Control Send, and Control Send with Return.

A Message Path can contain a Client Request or Control Receive node at which it receives the message. For the case in which an On Message path is specified for the process (that is, specified at the Start node) the first node on the path can be a Client Request with Return node.

This section contains the following topics:

- [Creating a Message Path](#)
- [Deleting Message Paths](#)

Creating a Message Path

You can associate a message path with individual nodes in your business process, with groups of nodes, or with the whole process (global). You create a global message path by adding a message path to the start node of your process.

This section contains the following topics:

- [To Create a Message Path](#)

- [To Configure a Message Path](#)
- [To Delete a Message Path](#)

To Create a Message Path

1. Select the node or groups of nodes for which you want to create a message path. (To learn about grouping nodes, see [Grouping Nodes in Your Business Process](#))
2. Right-click the node or group of nodes and select **Add Message Path** from the drop-down menu.

A message path is added to your node or group of nodes and is displayed as follows.



You can rename the path anything you like by double-clicking **OnMessage** and entering the new name. You can also change the name in the **name** field of the **Property Editor**.

To Configure a Message Path

1. Double-click **Starting Event** to invoke the starting event node builder.
2. Select the event which you want your message branch to wait for. Choose one of the following options:
 - **A Client Request**—Select this option if you want your message path to wait for a message from a client.
 - **A Client Request with Return**—Select this option if you want your message path to wait for a message from a client and then send a synchronous response back to the client. You can add optional nodes between the receive and send nodes inside the Client Request with Return node.

Note: This option is only available when a Message Path is added to a Start node of a business process.
 - **A Control Receive**—Select this option if you want your message path to wait for a message from a specified control.

3. Click the **X** in the top-right corner to close the node builder.

The node you selected is added as the starting event to your message path. To configure your starting node see, [step 6](#).

4. Select the message path which you want to configure.

The related properties are displayed in the **Property Editor**. If the **Property Editor** is not visible in WebLogic Workshop, choose **View—Property Editor** from the menu bar.

5. In the **Property Editor**, configure the following properties:

- **name**—Enter the name you want displayed in the WebLogic Workshop for this path.
- **notes**—Enter any notes you want associated with this message path. These notes can then be accessed through the WebLogic Integration Administration Console.
- **after execute**—Select the action you want to take place after a message path is executed. Choose from:

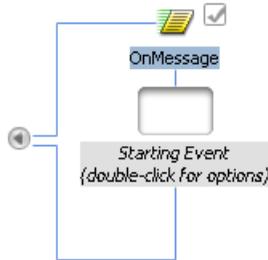
skip—Skip the node or group with which the message path is associated. That is, resume execution of the process at the node following the node or group for which the path is defined.

resume—Resume execution of the process at the node that was executing when the message was received. The process state returns to what it was before the message path executed and the message port is still active.

- **retry count**—Specify how many times, after the first attempt, the process engine tries to execute the node or group of nodes contained in the path, before the `afterExecute` path is taken.
6. Configure your starting event by double-clicking the node you chose as the starting event. The node builder is invoked. For information on how to configure:
 - Client Request with Return nodes, see [To Complete the Design of Your Client Request Node](#).
 - Client Request nodes, see [Design Your Client Request Node](#).
 - Control Receive nodes, see [Designing Your Control Nodes](#).
 7. Add any business process nodes to the exception path, as required to define the message path logic.

Viewing Message Paths in the Design View

When you create message path, the following icon appears beside a node (or group of nodes) in the **Design View** to indicate that an exception path is activated for the specified node:



This icon represents the message path in your business process. In this case, the path appears empty, indicating that the logic to execute when a message is received is not defined yet.

To define the exception handling logic, add business process nodes by dragging the nodes from the Business Process **Palette** and dropping them on the message path.

You can collapse the view of any message path (or exception handler or timeout path) by clicking the grey arrow of the message path icon. The following figure shows the icon associated with your node to indicate a collapsed path.



You can toggle between collapsed and expanded views of paths in the **Design View** by clicking the message path icon.

Deleting Message Paths

To Delete a Message Path

1. Right-click the path which you want to delete.
2. Select **Delete** from the drop-down menu.

The path is deleted and removed from the **Design View**.

Warning: Deleting a path deletes any business process nodes you defined on that path. When you attempt to delete a path, a dialog box displays a warning message that you must acknowledge before proceeding with the deletion.

Related Topics

[Grouping Nodes in Your Business Process](#)

[Writing Custom Java Code in Perform Nodes](#)

[Handling Exceptions](#)

[Adding Timeout Paths](#)

[Transaction Boundaries](#)

Adding Timeout Paths

A *timeout path* is used to interrupt an executing process after a certain amount of time has lapsed. Timeout paths can be associated with individual nodes, a group of nodes, or with the process (global). If you add a Timeout path to a start node, the timer starts when the process begins. If you add a Timeout path to any other node, or group of nodes, the timer starts when the process reaches that point of execution.

Note: Perform, Client Response, and any of the Control nodes do not support timeout paths on individual nodes.

This section contains the following topics:

- [Creating a Timeout Path](#)
- [Deleting Timeout Paths](#)

Creating a Timeout Path

You can associate a timeout path with individual nodes in your business process, with groups of nodes, or with the whole process (global). You create a global timeout path by adding a timeout path to the start node of your process. If you add a Timeout path to a start node, the timer starts when the process begins. If you add a Timeout path to any other node, or group of nodes, the timer starts when the process reaches that point of execution. This section contains the following topics:

- [To Create a Timeout Path](#)
- [To Configure a Timeout Path](#)
- [To Delete a Timeout Path](#)

To Create a Timeout Path

1. Select the node or groups of nodes for which you want to create a timeout path. (For information on how to group nodes, see [Grouping Nodes in Your Business Process](#))
2. Right-click on the node or group of nodes and select **Add Timeout Path** from the drop-down menu.

A timeout path is added to your node or group of nodes and is displayed as follows.



You can rename the path anything you like by double-clicking **OnTimeout** and entering the new name. You can also change the name in the **name** field of the **Property Editor**.

To Configure a Timeout Path

1. Select the Timeout path which you want to configure.

The related properties are displayed in the **Property Editor**. If the **Property Editor** is not visible in WebLogic Workshop, choose **View—Property Editor** from the menu bar.

2. In the **Property Editor**, configure the following properties:
 - **name**—Enter the name you want displayed in the WebLogic Workshop for this path.
 - **notes**—Enter any notes you want associated with this timeout path. These notes can then be accessed through the WebLogic Integration Administration Console.
 - **duration**—Specify the number of seconds that should laps before the path is triggered. (Expected format is Xs, for example 5s.)
 - **after execute**—Select the action you want to take place after a timeout path is executed. Choose from:
 - skip**—Skip the node or group with which the Timeout path is associated. That is, resume execution of the process at the node following the node or group for which the Timeout path is defined.
 - resume**—Resume execution of the process at the node that was executing when the timeout was triggered. The process state returns to that before the On Timeout path executed, and the On Timeout path resets (that is, timeout begins again).

- **retry count**—Specify how many times, after the first attempt, the process engine tries to execute the node or group of nodes contained in the path, before the `afterExecute` path is taken.
3. Add any business process nodes to the exception path, as required to define the timeout path logic.

Viewing Timeout Paths in the Design View

When you create a timeout path, the following icon appears beside a node (or group of nodes) in the **Design View** to indicate that an exception path is activated for the specified node:



This icon represents the timeout path in your business process. In this case, the path appears empty, indicating that the logic to execute when a timeout is received is not defined yet.

To define the exception handling logic, add business process nodes by dragging the nodes from the Business Process **Palette** and dropping them on the timeout path.

You can collapse the view of any timeout path (or exception handler or message path) by clicking the grey arrow of the exception path icon. The following figure shows the icon associated with your node to indicate a collapsed path.



You can toggle between collapsed and expanded views of paths in the **Design View** by clicking the path icon.

Deleting Timeout Paths

To Delete a Timeout Path

1. Right-click the path that you want to delete.
2. Select **Delete** from the drop-down menu.

The path is deleted and removed from the **Design View**.

Warning: Deleting a timeout path deletes any business process nodes you defined on that path. When you attempt to delete a timeout path, a dialog box displays a warning message that you must acknowledge before proceeding with the deletion.

Related Topics

[Grouping Nodes in Your Business Process](#)

Adding Timeout Paths

[Writing Custom Java Code in Perform Nodes](#)

[Handling Exceptions](#)

[Adding Message Paths](#)

[Transaction Boundaries](#)

Running and Testing Your Business Process

WebLogic Workshop provides a browser-based interface with which you can test the functionality of your business process. Using this Test View interface, you play the role of the client, invoking the business process's methods and viewing the responses.

This step describes how to test a business process you have created in WebLogic Workshop using the Test Browser tool. It includes the following topics:

- [Using the Test Browser](#)
- [Understanding the Service URL](#)

Using the Test Browser

To Launch the Test Browser

1. On the **Application** tab, click the business process (JPD file) you want to test.
2. If it not already selected, select the **Design View** tab.
The business process you selected in the Application is displayed in the Design View.
3. If WebLogic Server is not already running, from the WebLogic Workshop menu, choose **Tools—WebLogic Server—Start WebLogic Server**.

If WebLogic Server is running, the following indicator is visible in the status bar at the bottom of the WebLogic Workshop visual development environment:



4. After the Server is running, from the WebLogic Workshop menu, click **Build—Build Application**. WebLogic Workshop builds your project.
5. If necessary, correct any errors in the project.
6. Click  on the WebLogic Workshop menu bar.

A Web browser is launched to display the Workshop Test Browser, through which you can test your business process using sample input values.

The Workshop Test Browser contains the following tabs:

- **Overview**—This tab displays public information about your business process. Code in this area is generated automatically and 2-way editing is fully supported in the Process Language. Changes you make here will appear in the Design View.
- **Console**—This tab displays private information about your business process, such as how services are implemented on the back end, and with what version of WebLogic Workshop it was created. It also displays information about log settings, such as how many log messages to keep and the number of characters after which log entries are truncated.
- **Test Form**—This tab provides a simple test environment for the public methods of your business process. You can provide parameters for a method and examine its return value. You can also track and test the different parts of a conversation.
- **Test SOAP**—This tab shows the XML data that is being sent to your business process when you test its XML methods. You can use this page to examine and modify the XML data that is passed to a method of your business process.
- **Message Broker**—This tab provide a space for you to publish messages to channels available in channel files in your application. It allows you to test your process interactions with asynchronous events and simulate Timer, Email, File, JMS, and other event generators.
- **Process Graph**—This tab allows you to view an interactive or printable graph of the deployed process type. The graphical view represents your business process and its interactions with clients and resources, such as databases, JMS queues, file systems. It shows the path taken thus far by the business process and provides additional information about the state of each node in the process. If your browser is not already configured with the SVG plug-in when you click this tab, WebLogic Workshop will offer to download and install it for you.

For more information about the different tabs in the Workshop Test Browser, see [Workshop Test Browser](#). For specific information about how to use the Test Form, Message Broker, and Process

Graph tabs, see [To Test the Public Methods of Your Business Process](#), [To Test a Message Broker Channel](#), and [To View a Process Graph](#).

To stop the **Test Browser**, return to WebLogic Workshop and click  on the tool bar.

Warning: As you use the Test Browser, take care to not run very large or data intensive business processes. Doing so may cause the Test Browser to fail.

Testing the Public Methods of Your Business Process

To Test the Public Methods of Your Business Process

1. Launch the Workshop Test Browser. (To learn more, see [To Launch the Test Browser](#).)
2. If necessary, click **Test Form**.
 You can enter data that your business process can receive as part of a client request directly on the **Test Form** page. Alternatively, you can browse your file system and upload a file which contains your test data.
3. If your client operation accepts input, enter the required information in to the field or fields.
Note: To upload a file to test data, click **Browse** beside the **xml myfile: (file value)** field to open the file browser, then select the file that contains the test data you want to use. You can also enter the test data by entering (copy/paste) the content of a file into the field.
4. Click the button labeled with your business process's method name to invoke the method with the values you entered. The Test Form page refreshes to display a summary of your request parameters and your business process's response:
 - Under **Service Request**, a summary of the data that was sent by the client (you) when the method was called, including the values of method parameters, is displayed.
 - For business processes that involve multiple communications with clients, or communications with resources such as other Web services, the Message Log on the left side of the Test Form page displays an entry for each call to a method or callback so that you can view the data for each. Click any log entry to see the details of that interaction.
 - Business Processes participate in conversations with clients. The Test Browser displays the instance ID in the Message Log. Select the instance ID or  **Refresh** to access continue and finish methods in that conversation.

5. When the business process finishes, a message similar to the following is displayed in the Message Log:

`Instance instanceID is Completed.`

In the preceding line, *instanceID* represents the ID generated when the first method in your business process was called.

6. Click the **Test SOAP** tab.

The Test SOAP tab displays the XML data that is being sent to your business process when you test its methods in the soap body field. You can use this page to examine and modify the XML data that is passed to a method of your business process.

7. Click the button with the name of your method to start a new conversation. The Test Form page refreshes to display a summary of your request parameters and your business process's response.

When the business process finishes, a message similar to the following is displayed in the Message Log:

`Instance instanceID is Completed.`

In the preceding line, *instanceID* represents the ID generated when the first method in your business process was called

To stop **Test View**, return to WebLogic Workshop and click  on the tool bar.

Testing a Message Broker Channel

To Test a Message Broker Channel

1. Launch the Workshop Test Browser. (To learn more, see [To Launch the Test Browser.](#))
2. Click **Test Form** and enter test data that can be used for to test the public methods that are published on your channel. To learn more, see [To Test the Public Methods of Your Business Process.](#)
3. Click the **Message Broker** tab.

The Message Broker test tab is displayed with details of the conversation routed through your channel. The conversation id is displayed in the message log. Click any of the methods displayed in your message log to view details on the right side of the window about the external services communications (callbacks and responses).

Viewing the Process Graph

The Process Graph tab of the Workshop Test Browser provides a SVG graph of your process as it is running. The graph represents your business process and its interactions with clients and resources, such as databases, JMS queues, and file systems.

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 the following URL:

`http://www.adobe.com/svg/viewer/install/main.html`

This viewer is not available for some configurations that the WebLogic Platform 8.1 supports. For details, please see “Browser Requirements for the Interactive Graph” in [Process Monitoring](#) in *Managing WebLogic Integration Solutions* at the following URL:

`http://edocs.bea.com/wli/docs81/manage/processmonitoring.html`

For detailed information about the operating systems and browsers WebLogic Platform supports, see [BEA WebLogic Platform Supported Configurations](#) at the following URL:

`http://edocs.bea.com/platform/suppconfigs/index.html`

To View a Process Graph

1. Launch the Workshop Test Browser. (To learn more, see [To Launch the Test Browser](#).)
2. Click the **Process Graph** tab.

The Adobe SVG Viewer displays the interactive view. The Process Graph Visual cues are provided to indicate node status as described in the following table:

If the node . . .	And the tracking level is . . .	The node appears . . .
Has been visited	Full or Node	Normal
	Minimum	Normal
Is currently executing	Full or Node	Highlighted
	Minimum	Highlighted

If the node . . .	And the tracking level is . . .	The node appears . . .
Has not been visited	Full or Node	Dimmed
	Minimum	Normal

To learn about business process tracking levels, see “Viewing and Changing Process Details” in [Process Configuration](#) in *Managing WebLogic Integration Solutions* at the following URL:

<http://edocs.bea.com/wli/docs81/manage/processconfig.html>

The top panel of the Process Graph tab displays selected process properties. To learn more about the properties displayed, see “Viewing Process Instance Details” in [Process Instance Monitoring](#) in *Managing WebLogic Integration Solutions* at the following URL:

<http://edocs.bea.com/wli/docs81/manage/processmonitoring.html>

3. Do any of the following:

- To display node status, click the node. The node name, type, and description are displayed in the **Node Info** panel. If the tracking level is set to Full, the start time, elapsed time, finish time, and completed visits are also displayed. If the tracking level is set to Node or Minimum, this additional information is not available.
- To scroll the view, press and hold down the **Alt** key. The cursor changes to a hand  tool. Click and drag to scroll the process graph vertically or horizontally.
- To zoom in, press and hold down the **Ctrl** key. The cursor changes to a zoom in  tool. Click to zoom in.
- To zoom out, press and hold down the **Ctrl+Shift** keys. The cursor changes to a zoom out  tool. Click to zoom out.
- To change to a printable view, click **Print View**. The process graph is displayed as a PDF.

To stop the **Test Browser**, return to WebLogic Workshop and click  on the tool bar.

Understanding the Service URL

In the Test browser, a URL is displayed in the upper-right corner of the Test Form tab. The URL you see when you launch Test View for your business process should be similar to the following URL:

```
http://localhost:7001/samples/myBusinessProcess.jspd
```

In the preceding line:

- `http://localhost:7001/`—Represents the machine name and its default listening port (7001). Specifically, this means that the request (the call to the business process) from your Test browser is intercepted by WebLogic Server on port number 7001 of your local machine.
- `samplesWeb/`— This refers to the Web application of which the service is a part. When you create a project in WebLogic Workshop, you are also creating a WebLogic Server *Web application*. The project name becomes part of the URL for all Web services in that project. Keep that in mind when naming new projects so that the resulting Web service URLs are meaningful and appropriate.
- `myBusinessProcess.jspd`—The filename of the business process JPD file. WebLogic Server is configured to recognize the JPD extension and respond appropriately by serving the request as a Web service—rather than, say, an HTML page or a JSP.

Related Topics

[How Do I: Test A Web Service Using WebLogic Workshop's Test View?](#)

Running and Testing Your Business Process

Business Process Variables and Data Types

In the Design View, the **Variables** tab displays the variables associated with the Java class that constitutes your business process. All business process variables are *global* to the business process instance.

This section describes business process variables and their data types. It includes the following topics:

- [Creating Variables](#)
- [Deleting Variables](#)
- [Working with Data Types](#)
- [Assigning MFL Data to XML Variables and XML Data to MFL Variables](#)

Creating Variables

There are two ways of creating variables for your business process. Variables can be created in the **Data Palette** by selecting **Add–Variable** or they can be created in the node builder when you are configuring the **Send Data** or **Receive Data** section of a **Client Request**, **Client Request with Return** (Start Node only), **Client Response**, **Subscription** (Start Node only), **Control Send**, **Control Send with Return**, or **Control Receive** node for your business process nodes. Whichever method you choose, you can always access your variables from the **Data Palette** after they are created. When you select a variable in the list on the **Data Palette**, its properties are displayed in the **Property Editor**.

Before you can create an XML variable of a particular XML Schema type, you must first import the XSD file that contains the XML Schema into the WebLogic Integration schemas project. For instructions on importing an XSD files, see [Importing Files into the Schemas Project](#).

Before you can create a non-XML variable of a particular non-XML type, you must first import the MFL file that contains the schema for the non-XML type into the WebLogic Integration schemas project. For instructions on importing an MFL files, see [Importing Files into the Schemas Project](#).

Before you can create a variable of a Java class type, the Java class file must first be available in the WebLogic Workshop project. To learn more about including a Java class in your project, see [Using Existing Applications](#).

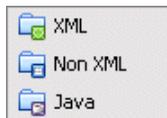
To Create a New Variable in the Data Palette

You can create variables using the **Add** menu on the **Variables** tab. To learn how to create variables in the node builder, see [To Create a New Variable in the Node Builder](#).

1. If the **Variables** tab is not visible in WebLogic Workshop, choose **View**—**Windows**—**Data Palette** from the menu bar.

The **Data Palette**, which contains a **Variables** and a **Controls** tab, is displayed in the **Design View**.

2. In the **Variables** tab, click **Add** to display the **Select Variable Type** menu.



A description of the possible variable types options are as follows:

- **XML Types**

Lists the XML Schemas that are available in your business process project and the untyped XMLObject and XMLObjectList data types.

- **Non-XML Types**

Lists the Message Format Language (MFL) files available in your business process project and the untyped RawData data type. WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe the structure of non-XML data. Every MFL file available in your project is listed in **Non-XML Types**. Note that an XML Schema representation of each MFL file is built by WebLogic Workshop and is also available in the XML Types listing.

– **Java Types**

Lists Java primitive data types.

Variables can be created from the Java classes in the current project but the list of available Java classes are *not* however, listed in the **Select Variable** pane. To create variables from Java classes, you must explicitly specify the Java class in the **Variable type** field as described below.

For more detailed descriptions of the data types, see [Working with Data Types](#).

3. From the drop-down menu, select the appropriate variable types from the selected option: **XML**, **NonXML**, or **Java**.

The **Create Variable** dialog box is displayed with the selected variable type.

4. In the **Variable Name** field, enter a name for your variable.
5. In the **Select Variable Type** field, select or in the **Variable type** field, enter a variable type:
 - For a Java class type variable, enter the full package name of the class in the **Variable type** field. For example, for a class named `Book` in the package named `library`, enter: `library.Book`. (For Java simple types, see step 7.)
 - For all other all variable types, select a variable type from the available list.

The **Variable type** field is populated with the variable type you selected.

6. If you want to assign a default value to your variable, enter it in the **Default value** field.
7. If your variable is a Java simple type and you want it to be a constant that cannot be updated, select **Declare as constant**, then enter the constant value in the **Default value** field. This will create the variable as **static** and **final**.

Note: If you select the **Declare as constant** and leave the **Default value** field blank, the run time default value used is:

- non-primitive: null
- numbers: 0
- boolean: false
- char: ‘\0’

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

The **Variables** tab in the **Data Palette** is populated with the variable you created; the name and type of the variable is displayed. When you select a variable in the list, the variable properties are displayed in the **Property Editor**.

To Create a New Variable in the Node Builder

You can create variables directly in the node builder while you are configuring a node's **Receive** or **Send Data** tab. To learn how to create variables outside of the node builder, see [To Create a New Variable in the Data Palette](#).

1. In the **Receive** or **Send Data** tab, select the **Variables Assignment** option if necessary.
2. From the **Select variables to assign:** drop-down menu, select **Create new variable...**. The **Create Variable** dialog box is displayed.

Note: The other fields in the dialog box are already populated with the variable types expected by the method you specified on the **General Settings** tab.

3. In the **Variable Name** field, enter a name for your variable.
4. If you would like to use a variable of a type other than what is expected by the method you specified on the **General Setting** tab:

a. Select a variable type option for your variable:

– **XML Types**

Lists the XML Schemas that are available in your business process project and the untyped XMLObject and XMLObjectList data types.

– **Non-XML Types**

Lists the Message Format Language (MFL) files available in your business process project and the untyped RawData data type. WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe the structure of non-XML data. Every MFL file available in your project is listed in **Non-XML Types**. Note that an XML Schema representation of each MFL file is built by WebLogic Workshop and is also available in the XML Types listing.

– **Java Types**

Lists Java primitive data types.

For more detailed descriptions of the data types, see [Working with Data Types](#).

The **Select Variable Type** field is populated with the variable type you selected.

5. Select or enter a variable type:
 - For a Java class type variable in the **Variable type** field enter the full package name of the class in the **Variable type** field. For example, for a class named `Book` in the package named `library`, enter: `library.Book`. (For Java simple types, see step 6.)

- For all other all variable types, select a variable type from the available list.

The **Variable type** field is populated with the variable type you selected.

6. If your variable is a Java simple type and you want it to be a constant that cannot be updated, select **Declare as constant**, then enter the constant value in the **Default value** field. This will create the variable as **static** and **final**.

Note: If you select the **Declare as constant** and leave the **Default value** field blank, the run time default value used is:

- non-primitive: null
- numbers: 0
- boolean: false
- char: ‘\0’

7. If you want to assign a default value to your variable, enter it in the **Default value** field.
8. Click **OK** to create the new variable.

The new variable you created is displayed in the **Select variables to assign:** drop-down menu in the node builder and is added to the **Variables** tab in the **Data Palette**; the name and type of the variable is displayed. When you select a variable in the list, the variable properties are displayed in the **Property Editor**.

To Convert Application Variable Data Types

The node builders support assigning typed data to untyped process variables, and untyped data to typed process variables in the following ways:

- **Typed to Untyped**

- You can assign strongly typed XML data (XML Bean) to untyped XML variables (XMLObject).
- You can assign typed non-XML data (MFLObject) to untyped non-XML variables (RawData).

- **Untyped to Typed**

- You can assign untyped XML data (XMLObject) to strongly typed XML variables (XML Bean).
- You can assign untyped non-XML data (RawData) to typed non-XML variables (MFLObject).

Related Topics

[Deleting Variables](#)

[Working with Data Types](#)

[Assigning MFL Data to XML Variables and XML Data to MFL Variables](#)

Deleting Variables

To Delete a Variable

On the **Data Palette**, in the **Variables** tab, right-click the name of a variable and choose **Delete** from the drop-down menu. The variable is deleted from the **Variables** tab and from the source code for your application. To learn about variables in your source code, see [Variables](#) in [Business Process Source Code](#).

Working with Data Types

The data types supported for your business process applications include:

- [XML Types](#)
- [Non-XML Types](#)
- [Java Types](#)

XML Types

XML Schemas are an XML vocabulary that describe the rules that your business data must follow. XML Schemas specify the structure of documents, and the data type of each element and attribute contained in the document. XML Schema files have an XSD file suffix. You can create new schemas or import schemas into your schemas folder, see [Importing Files into the Schemas Project](#).

Note: To make the Schemas in your project available in your business process, you must place them in the **Schemas** folder. The Schemas folder is a child folder of your business process application folder in WebLogic Workshop. To learn about the Application and project folders in the **Design View**, see [Components of Your Application](#) and [How Do I: Create a New Application](#).

When you add XML Schemas to the Schemas folder in your business process project, they are compiled to generate XML Beans. In this way, WebLogic Workshop generates a set of interfaces

that represent aspects of your Schema. 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.

When you load an XML file that conforms to a particular XML Schema into an XML Bean generated from the Schema, you can access the XML as instances of the XML Bean types. To learn more about XML Beans, see [Getting Started with XML Beans](#).

The XML typed data also includes:

XMLObject—This XML data type specifies untyped XML format data. In other words, this data type represents XML data that is not valid against an XML Schema.

XMLObjectList—This XML data type specifies a sequence of untyped XML format data. In other words, this data type represents a set of repeating elements of XML elements that are not valid against an XML Schema.

Tip for XML Object

When you assign an XMLObject variable or typed XML variable to an XMLObjectList, the XML document is added to the list (instead of directly assigning the variable).

Tip for Creating XML Schemas

When you create XML Schema definitions, which contain declarations for attributes, we recommend that you make these declarations inside, or local to, the element declarations. If you declare attributes at the top level of the XML Schema document (that is, immediately under the **xsd:schema** root), they must be qualified by a target namespace, if one exists. Consequently, for an XML instance document to be valid against such a Schema, the attributes within the XML document must be qualified with a namespace prefix associated with the target namespace. If you do not specify this prefix in an XML instance document, transformations or validations against the Schema fails.

Non-XML Types

WebLogic Integration uses a metadata language called Message Format Language (MFL), based on XML, to describe the structure of (typed) non-XML data. The Format Builder tool creates and maintains metadata as a data file, called an MFL document.

Note: When you create MFL files for use in your business process project, to make them available in your application, you must add the files to your **Schema** folder. The Schema folder is a child folder of your business process application folder in WebLogic

Workshop. To learn about the Application and project folders in the **Design View**, see [Components of Your Application](#).

Every MFL file available in your project is listed in **Non-XML Types** in the **Create Variable** dialog box. However, an XML Schema representation of each MFL file is built by WebLogic Workshop. This XML Schema representation of your MFL data is available in the [XML Types](#) listing. In other words, you can work with every MFL file in your project in its non-XML data representation (in non-XML MFL format) and in its XML Schema representation (XML typed data). For example if you add an MFL file named `mydata.mfl` to your business process project, **mydata.mfl** is listed in **Non XML Types**, and the corresponding XML Schema representation, **mydata.mfl.xsd**, is listed in [XML Types](#). Although you are provided with a typed XML version of your typed non-XML format, both types are not automatically populated. That is, if you receive data in a typed non-XML format and then assign it to a typed non-XML variable and you create a variable for the corresponding typed XML version, it will not automatically contain the data that is in the typed non-XML variable. You must use a transformation map to accomplish this. For more information about MFL files and Non-XML data, see [Transforming Non-XML Data](#).

Note: Non-XML variables are equivalent to *Binary* variables in prior versions of WebLogic Integration.

The non-XML type data also includes the **RawData** type that specifies non-XML data for which no MFL file exists and therefore no known schema.

Note: Although both `XMLObject` and `RawData` are both *untyped* data types in WebLogic Integration, the `XMLObject` data type is still XML and therefore has a structure that can be parsed. `RawData` is just a stream of data that has no known structure. Therefore, you cannot do things like use a `RawData` parameter in a XQuery expression or in a transformation method.

Java Types

Contains the following Java data types:

Java Primitive Data Types—boolean, byte, double, float, int, long, short, and String.

Java Classes—Variables can be created from the Java classes in the current project. However, the Java classes available in the project are *not* listed in the **Select Variable** pane in the node builders. You must explicitly specify the Java class in the **Variable type** field as described in the following sections: [To Create a New Variable in the Data Palette](#) and [To Create a New Variable in the Node Builder](#). To learn more about including Java classes in your project, see [Using Existing Applications](#).

Java class variables can be used in business processes without any conversion. When you use Java classes in data transformations, WebLogic Integration converts the Java class into an internal XML Schema representation of the Java class file. The fields of Java class that cannot be converted to an XML Schema type are ignored. To learn more about the conversion of Java classes into this internal XML Schema representation, see [Java Class Conversion](#). To learn about the Java classes that are created when you import schemas into your application, see [Java Classes Created From Importing Schemas](#).

Tip for Java Collection

When you assign a variable to a collection, it is added to the collection (instead of directly assigning the variable).

Related Topics

[Validating Schemas](#)

[Variables in Business Process Source Code](#)

[Assigning MFL Data to XML Variables and XML Data to MFL Variables](#)

Assigning MFL Data to XML Variables and XML Data to MFL Variables

As described in [Non-XML Types](#), an XML Schema representation of each MFL file in your application is built by WebLogic Workshop. You can work with every MFL file in your project in its non-XML data representation (in non-XML MFL format) and in its XML Schema representation (XML typed data).

The variable assignment panes in the WebLogic Integration node builders treat MFL and their corresponding XML variables interchangeably, such that you can assign MFL data directly to the corresponding variables of type XML and XML data directly to the corresponding variables of type MFL; no data transformation is required.

In other words, the WebLogic Workshop graphical design environment allows you to assign MFL data (that is passed into a business process from a client or a control) to strongly typed-XML variables directly, and to assign typed-XML data (sent from a business process to a client or a control) directly to MFL variables.

The node builders for the following nodes support the direct assignment of MFL data to XML variables and XML data to MFL variables: **Client Request**, **Client Response**, **Control Send**,

Control Return, Control Send with Return. The example described in the following section describes a **Client Request** and a **Client Response** node; the steps are similar for any node.

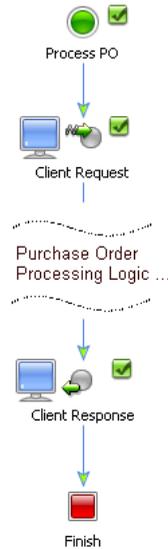
Example Scenario—Requires Assignment of MFL Data to an XML Variable and Assignment of XML Data to an MFL Variable

Consider the following example:

Your business process is started by a request from a client. The request contains a purchase order document in MFL format, which is represented by an MFL file in a Schemas folder in your application. To learn about importing XSD and MFL schemas into your application, see [Importing Files into the Schemas Project](#).

To process the purchase order, your business process must first assign the MFL data to an XML variable (at a **Client Request** node). This XML variable is used in the processing of the purchase order at subsequent nodes. When the processing is complete, the business process sends a response document (from a **Client Response** node) to the client. The processed data (a price quote) is in XML format in your business process; because the client expects MFL data, a **Client Response** node assigns the XML data to a variable of type MFL before sending the response.

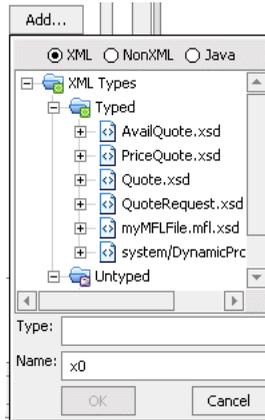
The business process in this scenario includes a **Client Request** node, a **Client Response** node, and the nodes between them (not described in this example) at which the processing logic is designed:



The following steps describe how to design the **Client Request** and **Client Response** nodes to do the MFL-to-XML and XML-to-MFL assignments required for the scenario described in this example.

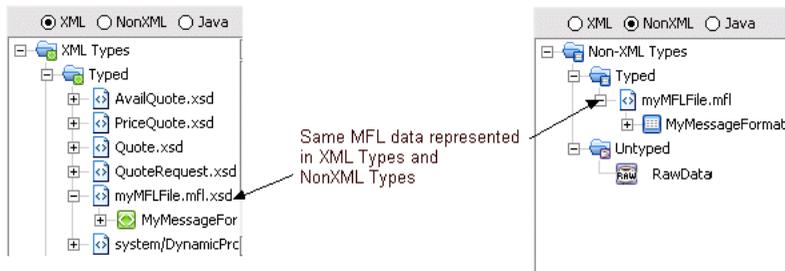
To Design the Client Request Node

1. In the **Design View**, double-click the **Client Request** node to invoke its node builder.
2. In the **General Settings** tab, enter a name in the **Method Name** field to specify the name of the method on this **Client Receive** node (by default, the method is named **clientRequest**).
3. In the **General Settings** tab, click **Add** to select the type and format of the data your **Client Request** node expects to receive from clients (that is, the data type for the method parameter). As shown in the following figure, the **XML** option is selected by default and **XML Types** are displayed. However, **Non-XML Types** and **Java Types** are also available. To display the Non-XML and Java data types, select the **NonXML** or **Java** options on the panel.

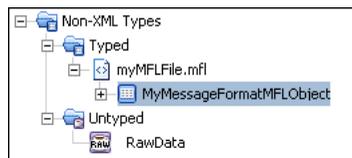


Note: Every MFL file type in your project is listed in the **Non-XML Types** pane and every XML file type in your project is listed in the **XML Types** pane. In addition, because an XML Schema representation of each MFL file is built by WebLogic Workshop, an XML Schema representation of your MFL data is also available in the **XML Types** list.

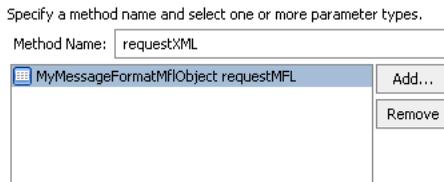
The following figure shows an example of the **XML Types** listing and the **NonXML Types** listing for an application which includes an MFL file named **MyMFLFile.mfl**:



4. Select **NonXML** to display the Non-XML Types (Typed and Untyped) in your application:



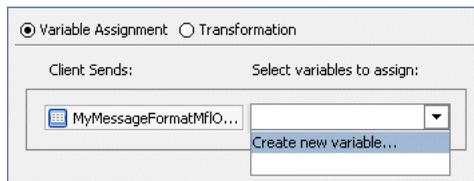
- Click the + associated with the name of the MFL file that represents the type of the request the client makes to the business process. In this example, we click the + beside **myMFLFile.mfl** to display the root element **MyMessageFormatMflObject**.
- Click the root element—in this case, **MyMessageFormatMflObject**. The data type is displayed in the **Type** field.
- Enter a name for the method parameter in the **Name** field (in this example, we entered **requestMFL**), and click **OK**. The parameter type is displayed in the node builder:



- Click the **Receive Data** tab.

The **Receive Data** tab allows you to define the variable to hold the data your business process receives from clients.

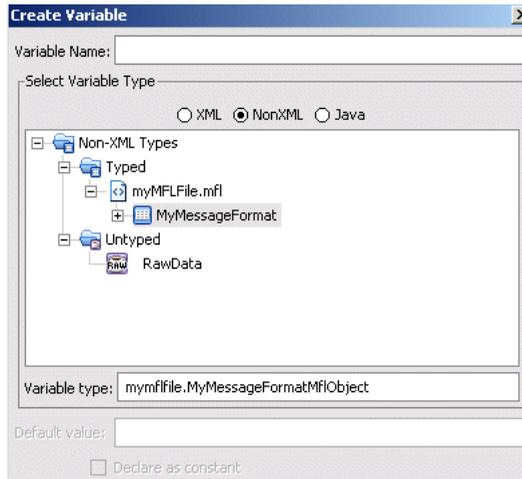
The **Client Sends** field is populated with the parameter (or parameters) you specified on the **General Settings** tab. In this example, the data is of type **MyMessageFormatMflObject**:



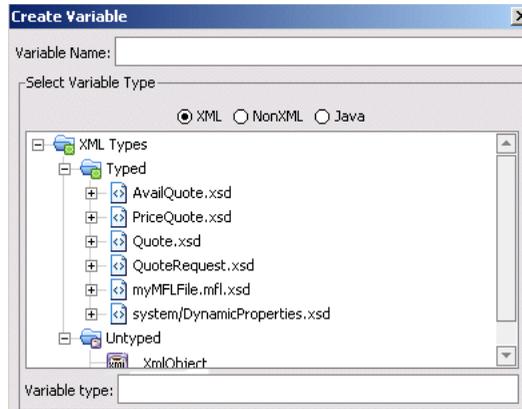
- Create a new variable to which the data supplied in the method parameter will be assigned.

To create a new variable, from the **Select variables to assign** drop-list, select **Create new variable...**

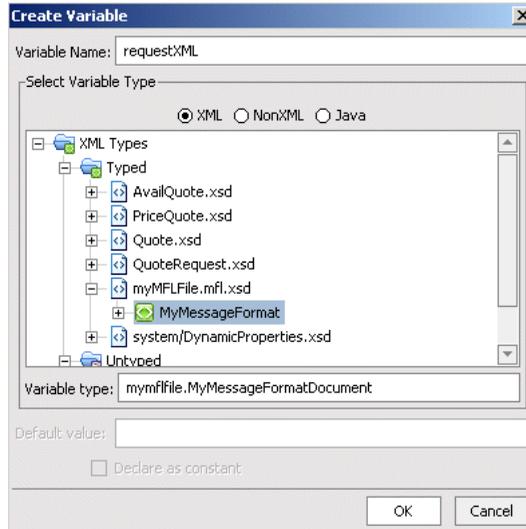
The **Create Variable** dialog box is displayed with the fields already populated with the variable types expected by the method you specified on the **General Settings** tab.



10. In the **Select Variable Type** pane, select the **XML** option to switch the display to the XML types in your application.

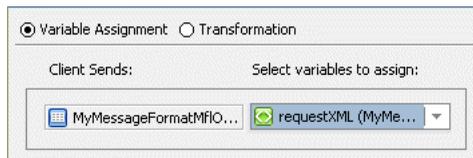


11. Click the XML Schema that corresponds to the MFL data your business process expects to receive at this node. In this example, click the + beside **myMFLFile.mfl.xsd** to expand its structure.
12. Click the root node of the schema. In this case, click **MyMessageFormat**. The **Variable Type** field is populated with the data type: **myMFLFile.MyMessageFormatDocument**, as shown in the following figure:



13. In the **Variable Name** field, enter a name. In this example, we entered **requestXML**.

14. Click **OK**. The node builder displays the assignment.



15. Click the **X** in the top right-hand corner to close the node builder.

16. To save your work, select **File—Save**.

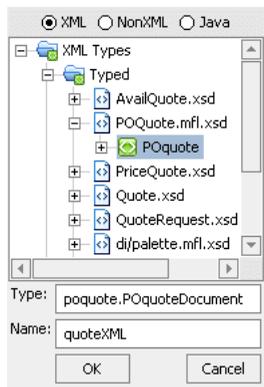
The preceding steps described how to design a **Client Request** node to do a direct assignment of MFL data to an XML variable using the graphical design environment; no data transformation is required.

To Design the Client Response Node

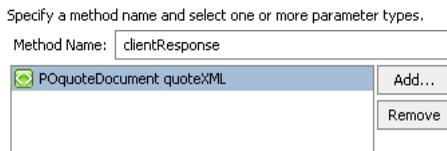
For this example scenario, assume that some processing is done by the business process to process the purchase order request. As a result of the processing, the business process creates a typed-XML price quote document. The client expects a quote in MFL format (the quote message must be valid against an MFL schema named `POQuote.mfl`). Therefore, the business process stores the price quote data in an XML variable that is valid against the XML schema associated with this MFL file (in this case named `POQuote.mfl.xsd`)

Before sending the response to the client, the typed-XML price quote must be assigned to a typed-non-XML (MFL) variable at the Client Response node. The following steps describe how to design the Client Response node for this XML-to-MFL scenario:

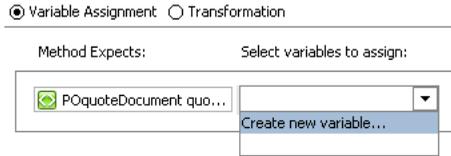
1. In the **Design View**, double-click the **Client Response** node to invoke its node builder.
2. In the **General Settings** tab, enter a name in the **Method Name** field to specify the name of the method (by default, the method is named **clientResponse**).
3. In the **General Settings** tab, click **Add** to select the data type for the method parameter for your **Client Request** node.
4. Click the + associated with the name of the XSD file that represents the type of the price quote created by the business process. In the example scenario, we click the + beside **POQuote.mfl.xsd** to display the root element **POQuote**.
5. Click the root element: in this case, **POQuote**. The data type is displayed in the **Type** field (**poquote.POQuoteDocument**):



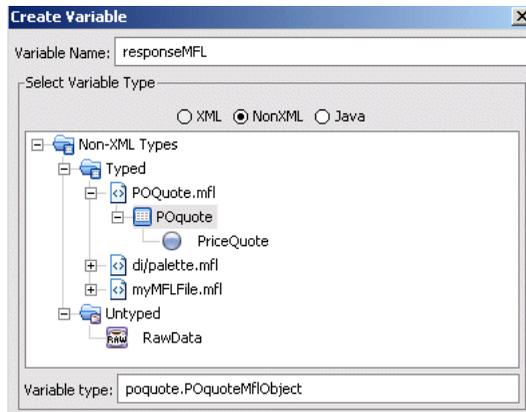
6. In the **Name** field, enter a name for the parameter (in this example, we entered **quoteXML**), and then click **OK**. The parameter type is displayed in the node builder:



7. Click the **Send Data** tab. The **Client Expects** field is populated with the parameter you specified on the **General Settings** tab. In the example, the method expects data of type **poquote.POQuoteDocument**.



8. Create a new variable to which the data supplied in the method parameter will be assigned.
To create a new variable, from the **Select variables to assign** drop-list, select **Create new variable...** The **Create Variable** dialog box is displayed and the fields are populated with the variable types expected by the method you specified on the **General Settings** tab.
9. Select **NonXML** in the **Select Variable Type** pane to display the Non-XML types in your application.
10. Click the MFL Schema that specifies to the MFL data your client expects to receive from the business process. In this example, click the + beside **POQuote.mfl** to expand its structure.



11. Click the root node of the schema. In this case, click **POquote**. The **Variable type** field is populated with the data type: **poquote.POquoteMflObject**, as shown in the preceding figure.
12. Enter a name in the **Variable Name** field. In this example, enter **responseMFL**.
13. Click **OK**. The node builder displays the assignment.



14. Click the **X** in the top right-hand corner to close the node builder.

15. To save your work, click **File—Save**.

The preceding steps described how to design a **Client Response** node to do a direct assignment of XML data to its corresponding MFL variable using the graphical design environment; no data transformation is required.

Versioning Business Processes

By using the WebLogic Integration versioning feature in the WebLogic Workshop graphical design environment, you can make changes to your business process without interrupting any instances of the process that are currently running. Versioning provides the ability for any new process instances to use the newly-activated version, while process instances that are already in progress run to completion using the version that was active when they started.

Note: Before using versioning with long-running business processes, please read [Using Versioning with Long-Running Business Processes](#).

When you version a business process, you create a child version of a business process that shares the same public URI (interface) as its parent. At run time, the version of the process that is marked as active is the process that will be accessed by external clients through the public URI.

Caution: You can version business processes, but not the individual controls associated with that process or other business process related components, such as schemas and transformations. When you version a business process, you must also version the subprocesses of that process; they are not versioned automatically when their parent process is versioned. This means that any changes to you make to these components also impact any instances of prior process versions that are currently running. To avoid this problem, as you make the necessary changes, create duplicates of these components and utilize the duplicates instead of the originals.

This section contains the following topics:

- [Creating a New Version of a Business Process](#)
- [Configuring the New Versions of Your Business Process](#)

- [Editing Versions of Business Processes](#)
- [Deleting Versions of a Business Process](#)
- [Using Versioning with Long-Running Business Processes](#)
- [Importing Versioned Business Processes](#)

Creating a New Version of a Business Process

The first time you create a new version of a business process, the content of your original process is copied into the new version and the old process is no longer editable. If you ever want to return to the original state of your business process, it is recommended that you leave the first version of the process intact and only make any edits or updates to the second version of your process. To create a new version of your business process, complete the procedures in the following sections:

- [To Create the First Version of a Business Process](#)
- [To Create a New Version of Your Process](#)

To Create the First Version of a Business Process

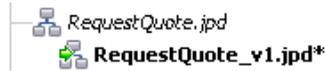
1. On the **Application** tab, right-click the business process (JPD file) for which you want to create a new version and select **Version Process...**

The **Create Version** window opens.

Note: If your Application pane is not visible in the WebLogic Workshop, select **View**→**Application**.

2. In the **Create Version** window, enter the following properties:
 - **Public URI**—This is the URI (instance) by which external clients access the most active version of your business process. The default value is the public instance by which clients accessed the original version of the business process.
 - **Version URI**—This is the name of the versioned file and also the URI by which this version of the business process can be accessed in the WebLogic Workshop.
3. Click **OK**.

The **Create Version** window closes and the new version of your business process is added to the Application pane.



The  indicates that this version of the business process is the active version of the process. By default, the first version of a process becomes the new version since and the original version becomes a virtual URI which points to the active version of the process. All currently running instances of the process will run to completion using the original process, but the next time an instance of the business process is invoked through the public URI, the version you just created will be used for processing.

Note: When you are creating process or service broker controls by right-clicking the virtual URI, the control will be created based on the active version of the business process with that URI. If you create the control by right-clicking the public URI of a version of a process, the control will be created based on the version of the business process that you selected.

To Create a New Version of Your Process

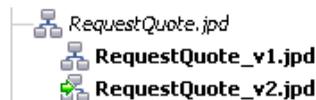
1. On the **Application** tab, right-click the business process (JPD file) which you want to create a new version for and select **Create New Version...**

The **Create New Version File** window opens.

Note: If your Application pane is not visible in the WebLogic Workshop, select **View**→**Application**.

2. In the **Create Version** window, enter a value for the **Version URI**, that is the URI by which this version of the business process can be accessed in the WebLogic Workshop.
3. If you want this version of the business process to be the active version of the process, select the **Active Version** check box. You can change a version of a business process to be active at any time, see [To Make a Version of a Business Process the Active Version](#).
4. Click **OK**.

The Create Version window closes and the new version of your business process is added to the Application pane.



The  indicates that this version of the business process is the active version of the process. All currently running instances of the process will finish processing, but the next

time an instance of the business process is invoked through the public URI, the version you just created will be used for processing.

Configuring the New Versions of Your Business Process

To Make a Version of a Business Process the Active Version

You can change which version of your business process you want to be the active version at any time. To do so:

1. On the **Application** tab, right-click the business process (JPD file) that you want to set to active and select **Make Active Version**.

Note: If your Application pane is not visible in WebLogic Workshop, select **View**→**Application**.

The version of the business process that you selected is updated in the Application pane to be the new active version, by changing its icon to . All currently running instances of the process will finish processing, but the next time an instance of the business process is invoked through the public URI, the version you just marked as active will be used for processing.

Editing Versions of Business Processes

You edit any version of a process the same way that you edit any original business process. However, there are a some things you should keep in mind:

- If you add or change a client operation in a version of a business process, all other versions of that business process will be out of synchronization with the public URI. This is indicated on the **Application** tab by changing the icon of the JPD file(s) to .
- If you edit or add any internal resources, such as variables, they will only be available in the process in which they were edited or created. Other versions of the process will not be able to access them.
- If you edit any external resources, such as transformations (DTF files) in a process, these changes will affect older versions of the business process, possibly breaking them. Additionally, any calls to that external resource from older version of the process may no longer be valid. Therefore, it is recommended that rather than editing an external resource, you create a copy of that resource and give it a new name and edit any calls to that resource within your version of the process to reflect the new name. The easiest way to do this is to use the search and replace function in the **Source View**.

Deleting Versions of a Business Process

To Delete a Version of a Business Process

1. On the **Application** tab, right-click the business process (JPD file) that you want to delete and select **Delete Version**.

Note: If your **Application** pane is not visible in the WebLogic Workshop, select **View**→**Application**.

The version of the business process you selected is deleted and removed from the Application pane.

Notes: If you delete the active version of a business process, the newest version of that business process automatically become the active version. If you delete the last remaining version of a business process, the content of that process is copied into the original JPD file and the **Application** tab is updated accordingly.

Using Versioning with Long-Running Business Processes

Some business processes are by nature *long-running*—meaning that they have a prolonged life span during which an ongoing business task is being automated or managed. By default, WebLogic Integration’s versioning capability allows only process modifications to be applied to new process instances, not to those already in progress at the time of the change. However, in the case of long-running processes, it is sometimes desirable to make changes to the process definition to reflect changing business conditions and to have these changes applied to process instances already in progress. To accomplish this goal, BEA recommends the following design practices:

- Split long-running business processes into multiple subprocesses.
- Specify the version strategy as **loose-coupling** (between business processes and subprocesses). This allows uptake of new subprocess versions when appropriate. See [To Specify the Version Strategy of a Business Process](#).
- Whenever possible, use generalized or untyped interfaces between processes and subprocesses. This further reduces the impact of changes made to subprocesses. For example, Client Requests should take XmlObjects, not a specific schema type. This ensures that when the schema is changed, the control method signature does not also have to change.

Note: You can enable, disable, or set the activation time for versions using the WebLogic Integration Administration Console. To learn more, see “Managing Process Versions” in *Managing WebLogic Integration Solutions*, which is available at the following URL:

<http://e-docs.bea.com/wli/docs81/manage/processconfig.html>

To Specify the Version Strategy of a Business Process

1. Select the **Start** node of the business process which version strategy parameter you want to change.
2. In the **Property Editor**, select the **version strategy** method that you want to use for the subprocesses process logic. From the list box, select one of the following:
 - **loosely-coupled**—select this option if you want the subprocess version to be set at the time that the subprocess is invoked. In other words, if an instance of your business process is currently running but has not yet reached the state of invoking the subprocess that you have created a new version for, the *new* version of your subprocess will be used when the process invokes the subprocess.
 - **tightly-coupled**—select this method if you want the subprocess version to be set at the time the parent process is invoked. In other words, if an instance of your business process is currently running but has not yet reached the state of invoking the subprocess that you have created a new version for, the *old* version of your subprocess will be used when the process invokes the sub process. The next time the main process is invoked, it will use the new version of the sub process when it invokes the subprocess.

Importing Versioned Business Processes

When you import multiple versions of a business process (JPDs) from another application, the versioning relationship is not preserved in the imported JPD. For example, if the original versions looks like the following:

```
JPD.jpdc
  JPD_v1.jpdc
  JPD_v2.jpdc
```

the imported business processes will look like:

```
JPD_v1.jpdc
JPD_v2.jpdc
```

This means that you need to manually edit the `wlw-config.xml` configuration file, which is located in the `\Web\WEB-INF` folder of your application. To learn about this configuration file, see [wlw-config.xml Configuration File](#).

Related Topics

For more information about versioning, see the information published regarding the `VersionException` class in the `com.bea.wli.bpm.runtime.versioning` package and the `VersioningConfigurationMBean` interface in the `com.bea.wli.management.configuration` package in the [WLI Javadoc](#) at the following URL:

`http://edocs.bea.com/wli/docs81/javadoc/index.html`

[wlw-config.xml Configuration File](#)

Versioning Business Processes

Validating Schemas

In addition to the validation check box on the node builders, you can validate XML Schemas in the source code by using the code examples illustrated in this section.

If you check the **Validate** check box in the business process nodes, and at run time a validation error occurs, a SOAP fault is thrown and your request is stopped before it gets to the business process or any exception handler defined for your business process. To design the validation of schemas and handling of their related exceptions within a business process, you can use the `validate()` method as described in the examples in the following section.

The examples in this section describe only validation procedures to validate against XML Schemas. You can accomplish similar tasks for MFL data by using the WebLogic Integration `MflObject` interface (see [Using the MflObject Interface to Transform Non-XML Data Programmatically](#)).

The following validation scenarios are described:

- [Validating a Typed XML Variable](#)
- [Typing and Validating an Untyped XML Type](#)

Validating a Typed XML Variable

In this example, we assume that the type of XML received by the business process is known at design time. That is, a document received by the business process is known to be strongly-typed XML. The following code describes how to deliver the XML to the business process and validate the XML data against the corresponding XML schema:

Validating Schemas

```
public void receiveTypedXML(POORDERDocument lineItem) {
    if (lineItem.validate()) {
        //
    } else {
        // Handle error
    }
}
```

In the preceding example, `POORDERDocument` is the name of the XML schema against which you want to validate the XML data received by your business process.

Typing and Validating an Untyped XML Type

In this example, we take an untyped XML data received by your business process, convert it to a strongly typed XML data, and subsequently validate it against an XML schema associated with the type.

This can be accomplished by writing the following code:

```
public void receiveUntypedXML(XmlObject xml) {
    if (xml instanceof POSUBLINEDocument) {
        POSUBLINEDocument sublineItem = (POSUBLINEDocument) xml;
        if (sublineItem.validate()) {
            // item is valid
        } else {
            // item is not valid
        }
    } else {
        // Handle error - the XmlObject is not a POSUBLINEDocument
    }
}
```

In the preceding example, `POSUBLINEDocument` is the name of the XML schema against which you want to validate the XML data.

Related Topics

[Using the `MfiObject` Interface to Transform Non-XML Data Programmatically](#)

[Working with Data Types](#)

[Class XmlUtils](#) at

<http://edocs.bea.com/wli/docs81/javadoc/com/bea/xml/XmlUtils.html>

[Importing Schemas](#)

[Schemas Project Folder](#)

[How do I: Import Files into the Schemas Project Folder](#)

Validating Schemas

Building Stateless and Stateful Business Processes

Business Processes are either Stateless or Stateful, depending on how many transactions are contained in the process.

- **Stateless**—A business process which is compiled into a stateless session bean and runs within one JTA transaction.
- **Stateful**—A business process which is compiled into an entity bean and runs within the scope of one or more JTA transactions.

To learn more about stateless session and entity beans, see [Overview: Enterprise Java Beans \(EJBs\)](#). To learn more about JTA transactions, see [Programming WebLogic JTA](#) at the following URL:

```
http://e-docs.bea.com/wls/docs81/jta/index.html
```

Stateless processes are intended to support business scenarios that involve short-running logic and have high-performance requirements. Because a stateless process does not persist its state to a database, it is optimized for lower-latency, higher-performance execution. An example of a stateless process is one that receives a message asynchronously from a client, transforms the message, and then sends it asynchronously to a resource using a control. Another example is a process that starts with a message broker subscription, transforms a message, and publishes it to another message broker channel. Such a process is analogous to the kinds of *routing rules* used by traditional message brokering or message routing system.

Stateful processes are intended to support business scenarios that involve complex, long-running logic and therefore have specific reliability and recovery requirements. A process is made stateful by the addition of stateful nodes or logic that forces transaction boundaries (see, [Transaction](#)

[Boundaries](#)). For example, a process that receives a message, transforms it, sends it to a business partner, and then waits for an asynchronous response is stateful because the act of *waiting* forces a transaction boundary. This is necessary to ensure that:

- The process can recover and continue execution without loss of data in the event of a system outage during this waiting period.
- System resources are used efficiently during this waiting period.

By default, a business process is **Stateless** until you add any blocking construct to the data flow, that is, add any process that affects a transaction boundary. For more information about transaction boundaries, see [Transaction Boundaries](#).

To View Whether Your Business Process is Stateless or Stateful

The Start node **Property Editor** indicates whether a business process is Stateless or Stateful in two different ways:

- The **Stateless** property in the **Property Editor** of your **Start** node.
- The icon of the **Start** node in the **Design View**.

The following table summarizes the ways in which WebLogic Workshop indicates if your Business Process is Stateless or Stateful.

	Stateless	Stateful
Property Editor	stateless = true	stateless = false
Start Node Icon		

You can use the **persistence** property in the Property Editor to set how a stateful business process is persisted. For more information about the Start node Property Editor, see [Setting the Business Process Properties](#).

Working with Variables in Stateless Processes

Because stateless processes are compiled into stateless session beans and because these stateless session beans are reused at run-time to provide the performance advantage enjoyed by stateless processes, some care is required when working with variables. If a default value is specified for a variable in a stateless process, that variable will only be initialized the first time the process is

run. Subsequent process instances will reuse the same stateless session bean instance, and therefore will inherit the last known value of the variable in question.

When building stateless processes that require variables with default values, you should place a **Perform** node at the start of the process (immediately following the Start node) and manually initialize the variables at that location. This way, you can be assured that the variables will be initialized with each run of the process, because the Perform node will be explicitly executed each time. For more information about how to create Perform nodes, see [Writing Custom Java Code in Perform Nodes](#).

If your goal is merely to have a variable with a constant value, this does not pose an issue in the case of stateless processes. When creating the variable, select the **Declare as Constant** check box in the **Create Variable** dialog box. This creates the variable as **static** and **final** and ensures that the constant behaves as expected during each run of the stateless process. For more information about how to create variables, see [Creating Variables](#).

Related Topics

[Transaction Boundaries](#)

[Starting Your Business Process](#)

[Writing Custom Java Code in Perform Nodes](#)

[Creating Variables](#)

[Overview: Enterprise Java Beans \(EJBs\)](#)

[Programming WebLogic JTA](#) at <http://edocs.bea.com/wls/docs81/jta/index.html>

[Setting the Business Process Properties](#)

Building Stateless and Stateful Business Processes

Building Synchronous and Asynchronous Business Processes

Business Processes are synchronous or asynchronous, depending on which method you choose to invoke your business process. However, both methods can contain both types of activity.

- **Synchronous**—A business process that is invoked by a synchronous method. In other words, the Starting Event is represented by a Client Request with Return node or a Synchronous Subscription node.

A synchronous business process can contain asynchronous operations, but they must be added after the starting event in the process flow. That is, at run time, the processes are executed after the synchronous starting event is complete. You cannot put stateful logic inside a synchronous operation. To learn more about stateful and stateless business processes, see [Building Stateless and Stateful Business Processes](#).

Note: If your synchronous process requires asynchronous operations within the Client Request with Return node, see [Synchronous Clients for Asynchronous Business Processes](#).

- **Asynchronous**—A business process that is invoked by an asynchronous method. In other words, the Starting Event is represented by an asynchronous node. This includes business processes that are invoked via a Client Request node, an Asynchronous Subscription node, or one of several Client Request or Subscription nodes (that is, an Event Choice node). A asynchronous process can call a synchronous or asynchronous methods without additional configuration.
- **Synchronous to Asynchronous**—Enables synchronous clients to interact with business processes that have asynchronous interactions with resources. To learn more, see [Synchronous Clients for Asynchronous Business Processes](#).

Working with Subprocesses

A subprocess is any process that is called to from your business process through a process control or a service broker control. They can be called synchronously or asynchronously.

Synchronous Subprocesses

The Process control allows a business process (also a WebLogic Workshop Web service or pageflow) to send requests to (and receive callbacks from) another business process. Process control invocations are Java Remote Method Invocation (RMI) calls. The target business process must be hosted on the same WebLogic Server domain as the caller. Transaction contexts are propagated from the parent processes to the subprocesses over the Process control calls.

The Service Broker control allows a business process (or a Web service) to invoke and receive callbacks from another service using one of several protocols; the most commonly used protocol is SOAP over HTTP. (To learn about the protocols, see [Using Dynamic Binding](#).) The target service must expose a WSDL interface. Because the transport used is HTTP or JMS, the transaction contexts are not propagated over the Service Broker control calls.

A synchronous subprocess called through a Process control runs in the same transaction as its caller (parent) process. Synchronous subprocesses behave differently than asynchronous subprocesses, particularly when it comes to un-handled exceptions.

An un-handled exception in a subprocess causes the shared transaction to be marked as rollback only, which causes both the subprocess and the caller (parent) process to roll back. This behavior is the default because it prevents a scenario in which one of the processes is rolled back, leaving the other process in an inconsistent or uncompensated state.

You can override the default behavior by setting the **on sync failure** property for the subprocess to **rethrow**. You do so in the **Property Editor** in the WebLogic Workshop graphical design environment.

To Configure the On Sync Failure Property

1. In the **Design View**, select the **Start** node of the subprocess for which you want to configure the **on sync failure** property.

Note: If the **Property Editor** is not visible, from the menu bar, choose **View** → **Property Editor**.

2. In the **Property Editor**, from the **on sync failure** drop-down menu, select **rethrow**.

Your subprocess is now configured to throw exceptions in the case of failure.

Note: Setting the **on sync failure** property does not force a rollback, it only causes the subprocess to throw an exception.

Asynchronous Subprocesses

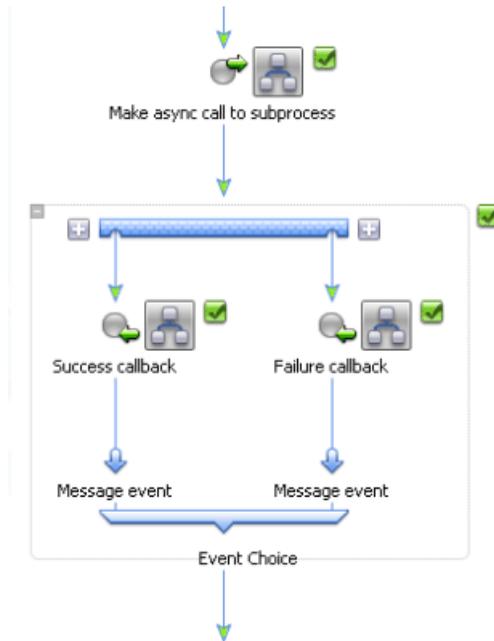
For asynchronous operations, the transaction is never propagated to the subprocess. In other words, a subprocess runs in its own transaction. Messages sent to subprocesses are buffered on the process queue of the subprocess. The caller process considers message delivery successful if the message is properly delivered to the queue. Consequently, failure of the subprocess is not communicated to the caller. For example, an unhandled exception causes the subprocess to fail, but the caller process is not notified.

Note: The business process (JPD) generated session beans have a default time-out value of 300 seconds. If this value is insufficient and leads to the timing out of long-running processes, you can alter this value. Information about this value, is located in the WebLogic Server documentation; see “trans-timeout-seconds” in [Programming WebLogic Enterprise JavaBeans](#) in the weblogic-ejb-jar.xml Deployment Descriptor Reference, which is located at the following URL:

<http://e-docs.bea.com/wls/docs81/ejb/DDreference-ejb-jar.html>

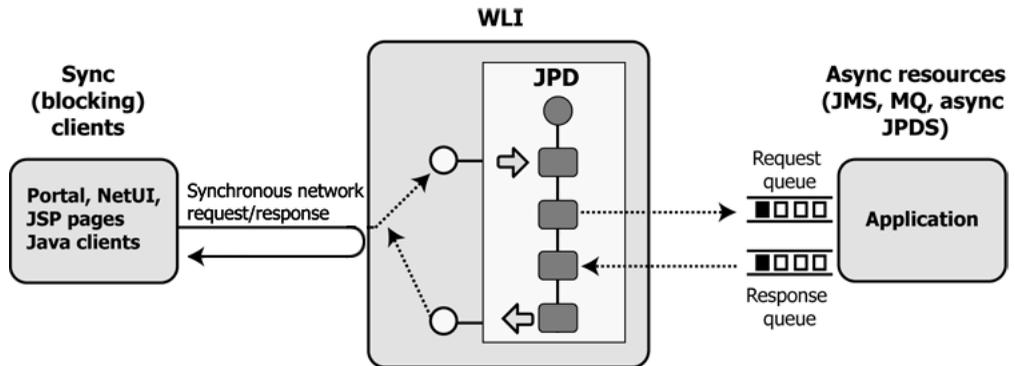
Since asynchronous failure is not automatically visible to the caller business process, you should consider the following design pattern for your process to subprocess communication:

- Design your subprocess such that it contains multiple callbacks for communicating success or failure. For example, use an exception handler path to catch any thrown exceptions and add a node on the path that makes a callback to the caller process that communicates that there was a failure.
- Use an **Event Choice** node in the caller business process to block and wait for either type of callback (success or failure) and take action as appropriate, as illustrated in the following figure.

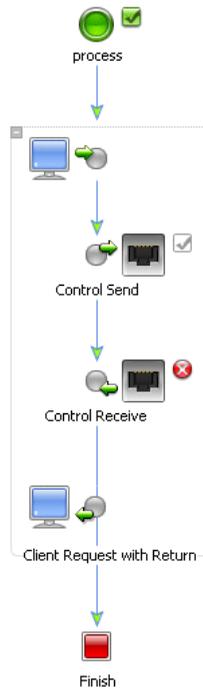


Synchronous Clients for Asynchronous Business Processes

You can enable synchronous clients to interact with business processes that have asynchronous interactions with resources. For example, a synchronous WebLogic Workshop client, such as a JSP or Portal page that uses a Java control, may need to invoke a business process and then block. While the client is blocking, the business process may perform asynchronous activities, such as enqueueing a JMS message and waiting for a JMS receive, and then return the response to the client, after which the client unblocks. The following figure demonstrates this scenario.



While it is not possible to create a synchronous process that requires asynchronous operations within a **Client Request with Return** node, as shown in the following figure (and indicated by  on the **Control Receive** node), you can create an asynchronous process with **Client Request** and **Client Response** nodes that accomplishes this task. This business process will appear to be synchronous to clients that use its web service interface. Additionally, this scenario will work for Java clients created with the WebLogic Server `clientgen` utility or with a WebLogic Workshop entity that uses the Service Broker control.



To enable synchronous clients to interact with business processes that have asynchronous interactions with resources, you can create a business process with a **Client Request** node with an attribute property called **sync/async callback name**. This **Client Request** node property holds the name of the callback method used by the associated **Client Response** node. The **Client Request** and **Client Response** nodes delineate the activities (including asynchronous activities) that occur while the client is blocking. After setting this property, you need to generate the sync-to-async WSDL. The synchronous WSDL generation process replaces the SOAP address of the service with a modified SOAP address. The modified address causes the synchronous servlet to process the client request and subsequent return action. The generated service entry looks like the following:

Normal WSDL

```
<service name="syncAsync">
  <port name="syncAsyncSoap" binding="s0:syncAsyncSoap">
    <soap:address
location="http://localhost:7001/SyncAsyncWeb/processes/syncAsync.jspd"/>
  </port>
```

Synchronous WSDL

```
<service name="syncAsync">
  <port name="syncAsyncSoap" binding="s0:syncAsyncSoap">
    <soap:address
location="http://localhost:7001/sync2AsyncIM/SyncAsyncWeb/processes/syncAs
ync.sync2JPD"/>
  </port>
```

To learn how to generate the sync-to-async WSDL, see [To Generate a Sync-to-Async WSDL File](#).

Note: To see an example of a synchronous client that invokes an asynchronous business process and waits (blocks) for the process to return information, see [Solution Samples](#), which is located at the following URL:

http://e-docs.bea.com/wli/docs81/sol_samples/index.html

To Create a Synchronous Client for an Asynchronous Processes

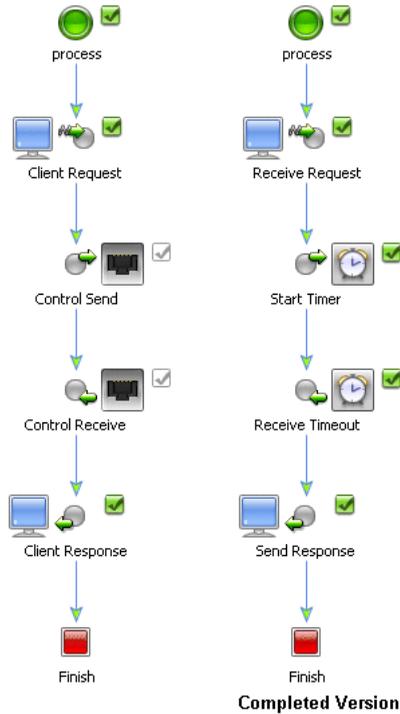
Note: Before designing your business process, be sure to read the [Limitations](#) section.

1. In **Design View**, create a business process with a **Client Request** as the **Starting Event**.
2. From the **Palette**, drag-and-drop the following nodes between the **Client Request** and **Finish** nodes:

Note: If the **Palette** is not visible in WebLogic Workshop, from the WebLogic Workshop menu, choose **View** → **Windows** → **Palette**.

- a.  **Control Send**
- b.  **Control Receive**
- c.  **Client Response**

The business process should now resemble the business process on the left side of the following figure.



3. Configure each node as required by your design.
4. In the **Design View**, click the **Client Request** node.
5. In the **Property Editor**, click the **sync/async callback name** field, then enter the name of the callback method. You can find this name in the **method name** property in the associated **Client Response** node.

If you enter the wrong name, an  appears next to the **Client Request** node.

Note: If the **Property Editor** is not visible, from the menu bar, choose **View** → **Property Editor**.

6. Generate a Sync-to-Async WSDL file, as described in [To Generate a Sync-to-Async WSDL File](#).
7. To learn about security for these processes, see [Synchronous-Asynchronous Security](#).

To Generate a Sync-to-Async WSDL File

WSDL files are used to communicate interface information between web service producers and consumers. A WSDL description allows a client to utilize a web service's capabilities without knowledge of the implementation details of the web service. To learn more about WSDL files, see [WSDL Files: Web Service Descriptions](#).

Note: Before you can generate a WSDL file, you must first set the **sync/async callback name** attribute property on the **Client Request** node.

1. In the **Application** pane, right-right click the business process (JPD) for which you want to generate the WSDL file.

Note: If the **Application** pane is not visible in WebLogic Workshop, from the menu bar, choose **View Application**.

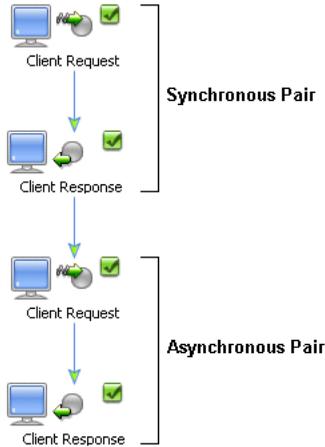
2. From the drop-down menu, choose **Generate Sync/Async WSDL File**.

WebLogic Workshop generates the WSDL file and displays it in the **Application** pane directly below the JPD file. If the name of the JPD is `HelloWorld.jpd`, the name of the WSDL file would be `HelloWorldSyncContract.wsdl`.

Limitations

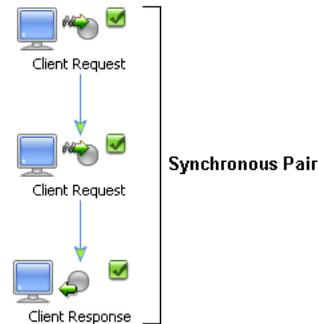
Mixing of Synchronous and Asynchronous Pairs is Not Allowed

You cannot mix synchronous and asynchronous Client Request and Client Response pairs in the same business process. Mixing synchronous and asynchronous pairs causes an error when generating the Sync/Async WSDL for the business process.



A Client Request within a Synchronous Pair is Not Allowed

You cannot place a Client Request node inside of a synchronous pair.



A Synchronous Client Cannot Call an Asynchronous Process with SOAP Attachment in a Client Request Node

Attachments are supported only on HTTP SOAP 1.1 and HTTP SOAP 1.2 bindings. Calling an asynchronous process from a synchronous process requires that JMS SOAP to be set as a binding. Subsequently, this scenario is not supported because of the conflicting requirements.

Synchronous-Asynchronous Security

The `SyncAsyncTransportServlet` is a transport object in the web tier. It provides HTTP protocol support for invoking WebLogic Integration components that are synchronously invoked from asynchronous internal and external clients.

Synchronous invocations to business processes from asynchronous clients that arrive via HTTP are received by the `SyncAsyncTransportServlet` transport. You can set basic authentication security on specific business process URLs that are invoked with the J2EE Web application `web.xml` and `weblogic.xml` deployment descriptors of the `SyncAsyncTransportServlet`.

To learn about security and basic authentication, see “Developing Secure Web Applications” in [Securing Web Applications](#) in *Programming WebLogic Security* at http://e-docs.bea.com/wls/docs81/security/thin_client.html and [Basic Authentication](#) in the *WebLogic Workshop Security Overview*.

To Configure Basic Authentication for Resources Accessed via `SyncAsyncServlet`

The `SyncAsyncTransportServlet` is packaged within WebLogic Integration System EJBs. The deployment descriptor (`web.xml`) for this servlet is contained in the `wli-ejbs.ear` file. This file is located in the `BEA_HOME\weblogic81\integration\lib` directory, where `BEA_HOME` is the directory in which you installed the WebLogic Platform. The `web.xml` is located in the `transport/http/WEB-INF` directory in the `wli-ejbs.ear` file.

1. To add basic authentication security settings for the business process URLs that are invoked from the `SyncAsyncTransportServlet`, you must modify the `web.xml` for the servlet.
2. Follow the guidelines in [Basic Authentication](#) in the *WebLogic Workshop Security Overview*.

Related Topics

[Transaction Boundaries](#)

[Starting Your Business Process](#)

[Building Stateless and Stateful Business Processes](#)

[Web Service Features in Business Processes](#)

[WSDL Files: Web Service Descriptions](#)

[Handling Exceptions](#)

[Calling Business Processes](#)

Building Synchronous and Asynchronous Business Processes

[Securing Web Applications](#)

[Basic Authentication](#)

Transaction Boundaries

Business processes in WebLogic Integration are transactional in nature. Every step of a process is executed within the context of a JTA transaction. A transaction ensures that one or more operations execute as an atomic unit of work. If one of the operations within a transaction fails, then all of them are *rolled-back* so that the application is returned to its prior state. Depending on whether you design your business process logic such that your process is stateful or stateless (see, [Building Stateless and Stateful Business Processes](#)), there may be one or more transactions within the context of a given business process.

When you are building a business process, *implicit transaction* boundaries are formed based on where in the process you place blocking elements. The transaction boundaries within a business process change as you add process nodes to the business process. You can also create *explicit transaction* boundaries by selecting contiguous nodes and declaring them to be in a transaction separate from those created implicitly by the application. Resources accessed by a business process may also be part of the transaction, depending on the nature of the resource and the control that provides the access.

Implicit transactions are implicit both because their behavior is automatically determined (or *implied*) by your business process logic and because they are not visible in your process diagram. In the section, [An Implicit Transaction Boundary Example](#), the implicit transaction boundaries in the diagrams are added for illustration; implicit transaction boundaries are not visible in the WebLogic Workshop graphical design environment. *Explicit transactions*, on the other hand, are explicit because they are defined by you and they are visible in the business process diagram in WebLogic Workshop.

This following sections deal specifically with transactions in the context of WebLogic Integration and business processes:

- [Implicit Transaction Boundary Rules](#)
- [An Implicit Transaction Boundary Example](#)
- [Explicit Transaction Boundaries](#)
- [Handling Exceptions in Transaction Blocks](#)

Note: To learn about how WebLogic Workshop transactions work, see [Default Transactional Behavior in WebLogic Workshop](#).

Implicit Transaction Boundary Rules

Recall that *implicit transaction* boundaries are formed based on where in the process you place blocking elements and that these boundaries change as you add process nodes to the business process. Additionally, a business process is stateless by default, and blocking elements that change transaction boundaries can change the process to stateful (see, [Building Stateless and Stateful Business Processes](#)). For more information about The following rules apply to transaction boundaries when you are building a business process:

- Adding any receive (blocking) nodes (Client Request or Control Receive nodes) to a business process changes the transaction boundaries:
 - Unless the node is in the beginning of a business process, the new receive node marks the beginning of a new transaction.
 - The node immediately preceding the new receive node marks the end of the preceding transaction.
- Adding a Parallel group node to a business process changes the transaction boundaries:
 - The node immediately preceding the Parallel group node marks the end of the preceding transaction.
 - The beginning of each branch in a Parallel group node marks the beginning of a new transaction.
 - The end of each branch in a Parallel group node marks the end of the new transaction.
 - The node immediately following the Parallel group node marks the beginning of the next transaction.

Note: By default, the beginning and the end of a parallel group node mark the boundaries of new transactions. However, you can specify that the active transaction is continued when entering and exiting a parallel block. To use this functionality, in **Design View**,

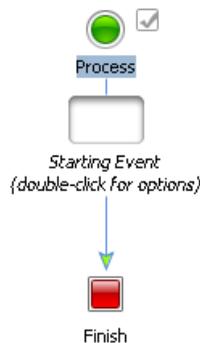
select the for a parallel node in your business process, then, in the **Property Editor**, change the **continue transaction** property to true.

- Adding an Event Choice node to a business process changes the transaction boundaries:
 - The node immediately preceding the Event Choice group marks the end of the preceding transaction.
 - The new group node marks the beginning of a new transaction.
 - Unlike the case of a Parallel node, in which each branch in the Parallel node has its own transaction context, the end of the Event Choice group does not mark the end of the transaction. Execution continues in the same transaction after the Event Choice group until a node that forces an implicit transaction boundary or an explicit boundary is reached.
- Existing transaction boundaries are unaffected by adding one or more nodes (within those boundaries) that do not themselves force a transaction boundary.

An Implicit Transaction Boundary Example

The following example illustrates the rules listed in the [Implicit Transaction Boundary Rules](#) section. In each of the figures in this section, the transaction boundaries are added to illustrate where they are implied in WebLogic Integration business processes.

1. Start with an empty business process, as illustrated in the following figure.



2. If we configure the Starting Event to be a Client Receive node, the following transaction boundaries are implied.

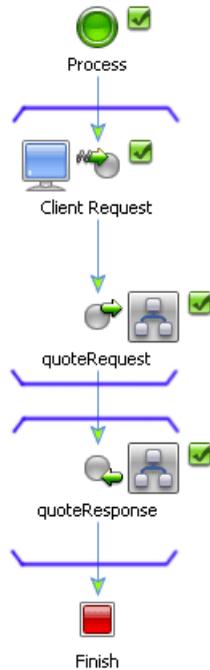
Transaction Boundaries



3. When we add a Control Send node, the implied transaction boundary extends to include the new node.

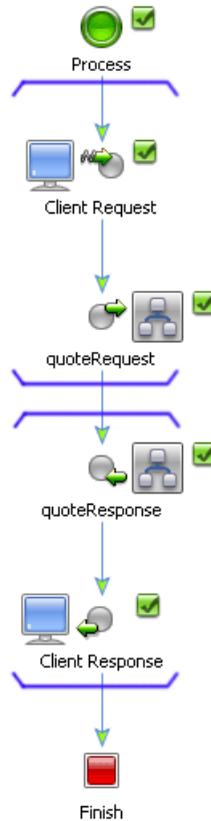


- By adding a Control Receive node, we add a blocking element to the business process and therefore create a new transaction.



Since the business process now contains two transactions, the Start node icon changes to indicate that the business has changed from Stateless  to Stateful . For more information about Stateless and Stateful business processes, see [Building Stateless and Stateful Business Processes](#).

- If we add a Client Response node to the business process, the second transaction's boundaries expands to include the new node.



This concludes the implicit boundaries example, you can also create transactions by adding explicit transaction boundaries to your business process. For information about how to do this, see [Explicit Transaction Boundaries](#).

Explicit Transaction Boundaries

Recall that you define explicit transaction boundaries and that they are visible in the business process diagram in WebLogic Workshop. You can create explicit transaction boundaries in your business process by selecting contiguous nodes and declaring them to be within their own transaction. The following rules apply for explicit transaction boundaries:

- The selected nodes must be contiguous.
- The selected nodes cannot include a Client Request or Control Receive node.

- The selected nodes cannot include a Parallel or Event Choice group node where including them in an explicit transaction would nest the transaction for their branches.
- The selected nodes cannot be inside an existing explicit transaction.

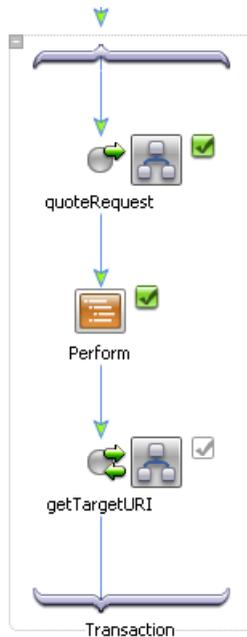
If you violate any of these rules when you create your business process, the application displays the transaction boundaries, but the offending nodes are marked with a . If you place your cursor over this icon, WebLogic Workshop will display a message about the violation.

Creating Explicit Transaction Boundaries

To Create an Explicit Transaction—Alternative 1

1. Select the nodes that you want to include in your transaction by clicking and dragging your mouse around them, or holding down your Ctrl key while clicking them.
2. Right-click one of the selected nodes and select **Create Transaction** from the drop-down menu.

Explicit transaction boundaries are drawn around the nodes you selected.



You can rename your transaction block by right-clicking **Transaction** and selecting **Rename** from the drop-down menu.

To Create an Explicit Transaction—Alternative 2

1. In the Design View, drag and drop  **Transaction** from the **Palette** onto the business process, placing it on the business process at the point at which you want to create explicit transaction boundaries.

Transaction boundaries are created in the business process.

2. Drag and drop nodes from the **Palette** onto the business process, placing them within the transaction boundaries.

Setting the Explicit Transaction Properties

After you create an explicit transaction, you can set the properties for the transaction in the Property Editor.

To Set the Transaction Properties

1. Select the transaction for which you want to set the properties.

The related properties are displayed in the **Property Editor**. If the **Property Editor** is not visible in WebLogic Workshop, choose **View—Property Editor** from the menu bar.

2. In the **Property Editor**, set the following properties:
 - **name**—Enter the name you want displayed in the WebLogic Workshop for this transaction.
 - **notes**—Enter any notes you want associated with this transaction.
 - **retry count**—Specify how many times, after the first attempt, the process engine should try to execute the node or group of nodes contained in the transaction.
 - **retry delay**—Enter the amount of time (in seconds) you want to pass before a retry is attempted.

Handling Exceptions in Transaction Blocks

To learn about exception handling in business processes, see [Handling Exceptions](#). How exceptions are handled in transaction blocks is described in [Handling Exceptions in Transaction Blocks](#).

Related Topics

[Implicit Transactions in WebLogic Workshop](#)

[Transaction Basics](#)

[Default Transactional Behavior in WebLogic Workshop](#)

[Building Stateless and Stateful Business Processes](#)

[Configuring and Managing Transactions at](#)

<http://e-docs.bea.com/wls/docs81/jta/admtrx.html>

Transaction Boundaries

Business Process Source Code

As you design business processes using the graphical tools in WebLogic Workshop, WebLogic Workshop writes source code to a business process file (a JPD file), in keeping with your work in the **Design View**. A JPD file is a Java file in that it contains code for a Java class. However, because a file with a JPD extension contains the implementation code intended specifically for a business process class, the extension gives it special meaning to the WebLogic Workshop compiler.

You can access the source code for business processes you are creating in the **Design View**, by clicking the **Source View** tab.

This section describes the source code in a business process (JPD) file, and how it is related to the work you do while creating your business process graphically in the **Design View**. It includes the following topics:

- [Overview](#)
- [Business Process Language](#)
- [Variables](#)
- [Control Declarations](#)
- [Client Operations and Control Communication Methods](#)
- [Perform Methods](#)
- [XQuery Statements](#)

Overview

To organize the source code in a JPD file, the code generated for you as you work in the **Design View** is hidden in collapsible regions in the **Source View**. Methods that you write for conditions in **Decision**, **For Each**, **While** nodes, and so on, are shown inside the **Business Process Designer generated code region**—each inside their own collapsible regions in the JPD file in the **Source View**. Methods you create for **Perform** nodes are created outside the **Business Process Designer generated code region**.

Specific regions in the **Source View** represent variable declarations, control declarations, XQuery annotations, and methods associated with client operations and communication with controls. In the **Source View**, you can expand these collapsed regions of code to add or edit the contained code. The WebLogic Workshop environment supports two-way editing of your business process (Java) class—the extent to which you can add code or change the code generated by WebLogic Workshop is indicated by comments in the source code and described in the following sections.

Business Process Language

To view the business process annotation that describes the business process you created in the **Design View**, expand the region of code indicated with  `Process Language`.

This annotation contains the business process definition, created for you as you add nodes to your business process in the **Design View**. The Java methods and variables defined in this JPD file can be referenced by the flow logic described in annotation.

The `<process>` element is the top-level container for the business process logic. A business process is composed of a set of activities with defined ordering. The business process element contains a **name** attribute, which specifies the name of your business process. Lines of XML describe the nodes in your business process. A line of XML is written in this area of code for each node you add to your business process in the **Design View**.

Two-way editing is supported for the process language. In other words, changes you make to the code in this region of the JPD file appear in the **Design View**. For example, you can:

- Create a new line of XML to describe a node in your business process. In the **Design View**, the business process is updated with the new node, in keeping with your work in the **Source View**. If the XML you add is not *well formed*, a new node is not added in the **Design View**. Instead, WebLogic Workshop displays a compiler error.
- Write a line of process language that references a method. *However*, the method is not created automatically; you must create it in the JPD file.

- Edit the process language that was already created for you. Your changes are reflected in the **Design View**.
- Delete lines that correspond to a business process node. The business process in the **Design View** is updated accordingly.

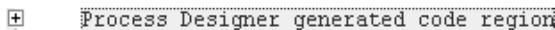
If the line of process language you delete references a method, which is already written in the file, the method is not deleted. You can leave the method in your file—if it is not referenced in the process language at run time, it is ignored by the run-time engine. Delete only methods that you are sure are not referenced in your process language. If you delete referenced methods, errors will be generated in your application.

Note: WebLogic Workshop flags errors you make in the process language with red, wavy underlines and error messages visible in mouseover text.

Warning: If you add any text within XML comment tags, or comment out sections of code, those lines of comments will disappear from the **Source View** the next time you make changes in the **Design View**. This only applies to comments, any other code changes remains.

Variables

Business process variables are defined within the region of code in the **Source View** shown in the following figure.

 Process Designer generated code region

Two-way editing is supported for variables. In other words, changes you make to the code in this region of the JPD file appear in the **Design View**, specifically the **Variables** tab on the **Data Palette**.

You can create, edit, or delete a business process variable in the **Source View**. The **Variables** tab on the **Data Palette**, is updated to reflect your changes. If the variable is not declared correctly, the error is identified in the **Source View** with red, wavy, underlines, and the variable does not appear on the **Variables** tab.

Warning: Ensure that you update a given variable in all locations where it is used. If you do not, changes you make to the business process variable generates errors in your application.

Control Declarations

Declarations for controls are defined in the region of code in the **Source View** shown in the following figure.

 `Process Designer generated code region`

Two-way editing is supported for control declarations. In other words, changes you make to the code in this region of the JPD file appear in the **Design View**.

You can create, edit, or delete a control declaration in the **Source View**. The **Design View**, specifically the **Controls** tab on the **Data Palette**, is updated to reflect your changes. If the control is not declared correctly, the error is identified in the **Source View** with red, wavy, underlines, and the control does not appear on the **Controls** tab.

Warning: Changing declarations for controls already in use by your application generates errors in your application if you do not remove or update references to the control.

To learn about working with controls in the **Design View**, see [Interacting With Resources Using Controls](#).

Client Operations and Control Communication Methods

Every client operation and communication method associated with a control is defined in its own collapsed region of code in the **Source View**. Code in these regions is generated automatically. Comments in the code, within these methods, identify the *protected sections* of code.

Can You Edit Code in Protected Sections?

In the **Source View**, you can add and edit the code before and after the protected sections within the blocks of code that specify client operations and control methods.

```
public void quoteRequest(org.example.request.QuoteRequestDocument requestXML)
{
    ///#START: CODE GENERATED - PROTECTED SECTION - you can safely add code above this comment in this method. ///
    ///input transform
    ///parameter assignment
    this.requestXML = requestXML;
    ///#END : CODE GENERATED - PROTECTED SECTION - you can safely add code below this comment in this method. ///
}
```

After you add or edit the code before or after the protected sections, the following icon is associated with the appropriate node (**Client Request**, **Client Response**, **Control Send**, **Control Receive**, **Control Send with Return**) in the **Design View**: . Two-way editing is still supported. In other words, you can continue to design the node in either the **Source View** or the **Design View**. The icon () in the **Design View** is a visual reminder that you edited the code in the **Source View**.

If you try to edit the code *within* the protected sections, you are prompted with a warning message and a question whether you want to continue with your edit, which in effect removes the protected

sections. If you edit the code in the protected sections, the node builders in the **Design View** can no longer interpret the code. Consequently, the node appears unconfigured in the **Design View**.

For example your business process can include a **Client Receive** node that you configured using the **Client Receive** node builder. In the **Design View**, the node is displayed as shown in the following figure:



If you right-click the node and select **View Code** from the drop-down menu, your view is switched to the **Source View** at the appropriate method. In this case, your source code resembles that in the following figure:

```
public void quoteRequest(org.example.request.QuoteRequestDocument requestXML)
{
    /**START: CODE GENERATED - PROTECTED SECTION - you can safely add code above this comment in this method. /**
    // input transform
    // parameter assignment
    this.requestXML = requestXML;
    /**END : CODE GENERATED - PROTECTED SECTION - you can safely add code below this comment in this method. /**
}
```

The *protected sections* of code are clearly marked. You can add any valid code you want. For example, say you add a `validate()` method to validate the incoming XML document against an XML schema, as shown in the following figure:

```
public void quoteRequest(org.example.request.QuoteRequestDocument requestXML)
{
    /**START: CODE GENERATED - PROTECTED SECTION - you can safely add code above this comment in this method. /**
    // input transform
    // parameter assignment
    this.requestXML = requestXML;
    /**END : CODE GENERATED - PROTECTED SECTION - you can safely add code below this comment in this method. /**

    if (requestXML.validate()) {
        //
    } else {
        // Handle error
    }
}
```

After you add your custom code, you can open the **Design View** by clicking the **Design View** tab. Note that the representation of the node associated with this code changed from:



You can make subsequent changes to the design of the node using either the **Design View** or the **Source View**. To learn about designing client and control operations in the **Design View**, see [Interacting With Clients](#) and [Interacting With Resources Using Controls](#).

Perform Methods

Methods you create for **Perform** nodes and methods you write for conditions in **Decision**, **For Each**, or **While** nodes are shown outside (and below) the collapsed regions of code in the JPD file in the **Source View**.

```
public void perform() throws Exception {  
}
```

You can write code (perform methods) in the **Source View** to perform any logic you want. The **Design View** for your business process is not updated in keeping with your work on such perform methods until you create a reference to the methods in the [Business Process Language](#).

To learn about creating **Perform** nodes in the **Design View**, see [To Create a Perform Node in Your Business Process](#).

XQuery Statements

XQuery statements are written to the JPD file in the region of code in the **Source View** shown in the following figure.

A screenshot of a JPD file showing an XQuery prologue annotation. The text is "@jpd:xquery prologue::" and is enclosed in a dashed rectangular box. To the left of the box is a small square icon containing a plus sign.

The XQuery statements are preceded by the following annotation:

```
@jpd:xquery prologue::
```

For example, when you select a repeating XML node using the **For Each** node builder, as described in [Designing For Each Nodes](#), an XQuery expression is created in your JPD file. The expression returns the set of XML elements over which the **For Each** node iterates. XQuery expressions are also written in your JPD file when you create conditions on **Decision** nodes. XQuery expressions also define the transformations you create between disparate data types using the mapping tool.

For more information about XQuery, see [XQuery Reference](#).

To learn more about **For Each** nodes, **Decision** nodes, and data transformations, see the following topics:

- [Looping Through Items in a List](#)
- [Defining Conditions For Branching](#)
- [Guide to Data Transformations](#)
- [Tutorial: Building Your First Data Transformation](#)

Related Topics

[Jpd Context Interface](#)

[ControlContext Interface](#)

[@jpd:xquery Annotation](#)

Business Process Source Code

Building ebXML Participant Business Processes

The ebXML protocol (Electronic Business using eXtensible Markup Language) is a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. It is sponsored by UN/CEFACT and OASIS. To learn about ebXML, see the following URL:

<http://www.ebXML.org>

This topic focuses on *participant* business processes for ebXML. For *initiator* business processes, you use the ebXML control, which provides methods for sending and receiving ebXML messages in a conversation. To learn about designing initiator business processes for ebXML conversations, see *Introducing Trading Partner Integration* at the following URL:

<http://edocs.bea.com/wli/docs81/tpintro/index.html>

This topic describes the template that you can use to build an ebXML participant business process in WebLogic Workshop. It contains the following sections:

- [About the ebXML Participant Business Process File](#)
- [Creating an ebXML Participant Business Process](#)
- [Customizing an ebXML Participant Business Process](#)

Related Topics

[@jpd:ebxml Annotation](#)

[@jpd:ebxml-method Annotation](#)

[ebXML Control](#)

[ebXML Control Interface](#)

Introducing Trading Partner Integration at

<http://edocs.bea.com/wli/docs81/tpintro/index.html>

[Interacting With Resources Using Controls](#)

About the ebXML Participant Business Process File

The generated ebXML participant business process file provides a head start for building public participant business processes for ebXML conversations. Although this file is not required to build ebXML participant business processes, it includes the nodes and business process annotations needed to integrate easily with ebXML initiator business processes.

The generated ebXML participant business process file consists of the following nodes, which are linked in the following sequence:

Node Name	Node Type	Method Name	Description
Start	Start		To learn about Start nodes, see Starting Your Business Process .
Receive Request	Client Request	<code>request</code>	Starts the ebXML participant business process upon receiving an ebXML message from the initiator. To learn about Client Request nodes, see Receiving Messages From Clients .
Respond to Request	Client Response	<code>response</code>	Sends the response document back to the initiator. To learn about Client Response nodes, see Sending Messages to Clients .
Finish	Finish		Ends the ebXML participant business process. To learn about Finish nodes, see Specifying Endpoints in Your Business Process .

Related Topics

[Creating an ebXML Participant Business Process](#)

[Customizing an ebXML Participant Business Process](#)

Creating an ebXML Participant Business Process

To create an ebXML participant business process

1. If you have not already done so, create a new application or a new project within an existing application. To learn more about projects and applications, see [Getting Started](#).
2. From the WebLogic Workshop menu, choose **File**—**New**—**Process File**.
3. In the **New File** dialog, select **Processes**, and then select **ebXML Participant Process File**.
4. In the **File name** field, enter the name of the JPD file.
5. If you want to create the JPD file in a directory other than the one displayed in the **Create in** field, click the **Browse** button and select the target directory.
6. Click the **Create** button.

WebLogic Workshop creates a new ebXML participant process JPD file and displays it in the Design View pane.

7. To save your work, select **File**—**Save**.

After you create the JPD file, the name of the JPD file becomes available as a service on the Services tab in the WebLogic Integration Administration Console.

Related Topics

[About the ebXML Participant Business Process File](#)

[Customizing an ebXML Participant Business Process](#)

Customizing an ebXML Participant Business Process

After you create an ebXML participant business process, you must customize it for the associated ebXML conversation. Common customization tasks include:

- [Configuring Business Process Properties \(Required\)](#)
- [Customizing Names and Argument Types \(Optional\)](#)
- [Retrieving the ebXML Message Envelope \(Optional\)](#)

Depending on your implementation requirements, you might make additional customizations to the participant business process as needed. For example, participant business processes typically

invoke other controls (such as the File, JMS, or Application View controls), or a subprocess, to accomplish the necessary backend integration.

Configuring Business Process Properties (Required)

The generated ebXML participant business process file specifies the following default annotations:

```
@jpd:ebxml protocol-name="ebxml" ebxml-service-name="serviceName"
ebxml-action-mode="non-default"
```

These properties are set in the **Property Editor** that is visible when you have the Start node of your business process selected. Review and edit (if needed) the following properties:

Property	Default	Description
protocol-name	ebxml	Do not change.
service-name	<i>serviceName</i>	Name of the ebXML service associated with this business process. The name specified here must match the service name specified on the initiator side (for example, in the <code>ebxml-service-name</code> annotation on the ebXML control in the initiator business process). You provide this service name to your trading partners.
action-mode	non-default	<p>Action mode for this business process. Determines the value specified in the <code>eb:Action</code> element in the message header of the ebXML message, which becomes important in cases where multiple message exchanges occur within the same conversation. Select one of the following values:</p> <ul style="list-style-type: none"> <code>default</code>—Sets the <code>eb:Action</code> element to <code>SendMessage</code> (default name). <code>non-default</code>—Sets the <code>eb:Action</code> element to the name of the method (on the ebXML control) that sends the message in the initiator business process. For sending a message from the initiator to the participant, this name must match the method name of the Client Request node in the corresponding participant business process. For sending a message from the participant to the initiator, the method name in the callback interface for the Client Callback node in the participant business process must match the method name (on the ebXML control) in the control callback interface in the initiator business process. Using <code>non-default</code> is recommended to ensure recovery and high availability. <p>If unspecified, the <code>ebxml-action-mode</code> is set to <code>non-default</code>.</p>

Note: If the **Property Editor** is not visible in the **Design View**, choose **View—Property Editor** from the WebLogic Workshop menu.

To learn more about ebXML annotations, see [@jpd:ebxml Annotation](#). To learn about configuring business process properties, see [Business Process Language](#).

Customizing Names and Argument Types (Optional)

By default, the ebXML participant business process includes a Client Request node, named **Receive request**, to handle an incoming ebXML request message from an initiator. For business processes that involve multiple round-trips, you need to create additional Client Request nodes for any other operations that involve that receive an ebXML message from the initiator. To add the node to a business process, drag  **Client Request** from the **Data Palette** onto the business process.

After creating a **Client Request** node, for the `request` method, specify the attachment and its Java data type for the incoming message. The data type must match the contents of the incoming message and can be one of the following values:

Data Type	Description
<code>XmlObject</code>	Default. Represents data in untyped XML format. The XML data is not specified at design time.
<code>XmlObject[]</code>	An array containing one or more <code>XmlObject</code> elements.
<code>RawData</code>	Represents any non-XML structured or unstructured data for which no MFL file (and therefore no known schema) exists.
<code>RawData[]</code>	An array containing one or more <code>RawData</code> elements.
<code>MessageAttachment[]</code>	Array containing one or more parts of an ebXML business message. Message parts can be untyped XML data (<code>XmlObject</code> data type) or non-XML data (<code>RawData</code> data type). Used when sending different kinds of payloads (XML and non-XML) in the same message. The actual number of message parts might not be known until processed. To learn about working with <code>MessageAttachment</code> objects, see Using Message Attachments .

Note: Attachments can also be typed XML or typed MFL data as long as you specify the corresponding XML Bean or MFL class name in the parameter. To learn more about data types, see [Working with Data Types](#).

The following restrictions apply to payload specifications:

- If an array of any type is used, an argument of the same type cannot follow that array in the argument list. In other words, that array must be the last argument specified.
- If a `MessageAttachment[]` type is one of your arguments, no other array (including a `MessageAttachment[]`) is allowed immediately before or after it.

Retrieving the ebXML Message Envelope (Optional)

You can retrieve the message envelope of an incoming ebXML message by using the `message-envelope` annotation in the `@jpd:ebxml-method` tag, as shown in the following example:

```
/**
 *@jpd:ebxml-method message-envelope="{env}"
 */
public void request(XmlObject payload, XmlObject env) {
}
```

Note: You can rename the default value (`env`) as long as it matches the name of the parameter specified in the method.

To learn more about the `message-envelope` annotation, see [@jpd:ebxml-method Annotation](#).

Related Topics

[About the ebXML Participant Business Process File](#)

[Creating an ebXML Participant Business Process](#)

Building RosettaNet Participant Business Processes

This topic describes how to build public participant business processes for RosettaNet conversations using the RosettaNet participant business process file in WebLogic Workshop.

The following sections are included:

- [About the RosettaNet Participant Business Process File](#)
- [Creating a RosettaNet Participant Business Process](#)
- [Customizing a RosettaNet Participant Business Process](#)

RosettaNet is a consortium of major companies working to create and implement industry-wide, open e-business process standards. These standards form a common e-business language, aligning processes between supply chain partners on a global basis. RosettaNet is a subsidiary of the Uniform Code Council, Inc. (UCC). To learn about RosettaNet, see the following URL:

<http://www.rosettanet.org>

This topic focuses on *public, participant* two-action business processes based on the RosettaNet PIP3A4. For *initiator* business processes, you use the RosettaNet control, which provides methods for sending and receiving RosettaNet messages in a conversation. To learn about designing public initiator business processes and private business processes for RosettaNet conversations, see *Introducing Trading Partner Integration* at the following URL:

<http://edocs.bea.com/wli/docs81/tpintro/index.html>

Related Topics

Introducing Trading Partner Integration at

<http://edocs.bea.com/wli/docs81/tpintro/index.html>

Trading Partner Management at <http://edocs.bea.com/wli/docs81/manage/tpm.html>

[RosettaNet Control](#)

[RosettaNet Control Interface](#)

[Tutorial: Building RosettaNet Solutions](#) at

<http://edocs.bea.com/wli/docs81/tptutorial/rosettanet.html>

[@jpd:rosettanet Annotation](#)

About the RosettaNet Participant Business Process File

The RosettaNet participant business process file provides a head start for building public participant business processes for RosettaNet conversations. Although this file is not required to build RosettaNet participant business processes, it includes the nodes and business process annotations needed to integrate easily with RosettaNet initiator business processes.

The RosettaNet participant business process is intended to serve as an example of the type of processes you can build for RosettaNet message exchange. The file consists of the following nodes:

Example Node Name	Example Node Types	Description
Start	Start	<p>This marks the beginning of your business process. In the Property Editor of the Start node, you can define the following properties:</p> <ul style="list-style-type: none"> • protocol-name • protocol-version • pip-name • pip-version • pip-role <p>To learn more about these properties, see Configuring Business Process Properties (Required).</p> <p>To learn about Start nodes, see Starting Your Business Process.</p>
On Error Message (global error handling)	Message Path	<p>Use the Message Path to interrupt an executing process upon delivery of a message from either a client or a control. This allows the process to halt the current stream of execution and take the specified alternate actions. To learn more about Message Paths, see Adding Message Paths.</p>
On Error (Global message path)	Client Request	<p>Use this node, or any other nodes in its place, to handle the error processing you want to take place when an error message is received. To learn about Client Request nodes, see Receiving Messages From Clients</p>
Alert local administrator (Global message path)	Perform	<p>Use this node, or any other nodes in its place, to send a failure message to an administrator. To learn more about Perform nodes, see Writing Custom Java Code in Perform Nodes</p>
Receive Message	Client Request	<p>Starts the RosettaNet participant business process upon receiving a RosettaNet message from the initiator. To learn about Client Request nodes, see Receiving Messages From Clients.</p>

Example Node Name	Example Node Types	Description
Send receipt acknowledgment.	Client Response	Sends an acknowledgement to the initiator that the request message was received. To learn about Client Response nodes, see Interacting With Resources Using Controls .
Send private message (Invoke private process group)	Perform	Use this node, or any other nodes in its place, to send a request to the private process. To learn more about Perform nodes, see Writing Custom Java Code in Perform Nodes .
Receive private message (Invoke private process group)	Perform	Use this node, or any other nodes in its place, to receive a response from the private process. To learn more about Perform nodes, see Writing Custom Java Code in Perform Nodes .
Send reply (Retry block)	Client Response	Use this node, or any other node in its place, to send the response back to the initiator. To learn about Client Response nodes, see Sending Messages to Clients .
Receive receipt acknowledgment (Retry block)	Client Request	Use this node, or any other node in its place, to listen for an acknowledgment from the initiator process. To learn more about Client Request nodes, see Receiving Messages From Clients .
OnTimeout (Timeout path on Retry block)	Timeout Path	Use the Timeout Path to interrupt the execution of an iteration of the nodes in the Retry block group after a certain amount of time has lapsed. To learn more about grouping nodes, see Grouping Nodes in Your Business Process . To learn more about Timeout Paths, see Adding Timeout Paths .
Check retries (Timeout path on Retry block)	Condition	Use this node, or any other nodes in its place, to select a path of execution based on the evaluation of one or more conditions, in this case, the number of iterations of the Retry block group. To learn more about Decision nodes, see Defining Conditions For Branching .

Example Node Name	Example Node Types	Description
Notification of Failure (Timeout path on Retry block)	Perform	Place this node, or any other node in its place, inside the Decision node to handle failure notifications to the initiator if an iteration of the Retry block group times out. This node is where you would normally invoke a PIP0A1 notification of failure. To learn more about Perform nodes, see Writing Custom Java Code in Perform Nodes . To learn more about customizing this node, see Setting Up the Notification of Failure (Required) .
Finish	Finish	Ends the RosettaNet participant business process. To learn about Finish nodes, see Specifying Endpoints in Your Business Process .

To learn more about how to customize the nodes in the RosettaNet participant template, see [Customizing a RosettaNet Participant Business Process](#).

This business process is modeled on the Two-Action Activity (Asynchronous) choreography that is specified in the *RosettaNet Implementation Framework Core Specification* (version V02.00.01). To learn about this choreography, see the following URL:

<http://www.rosettanet.org>

Related Topics

[Creating a RosettaNet Participant Business Process](#)

[Customizing a RosettaNet Participant Business Process](#)

Creating a RosettaNet Participant Business Process

You can use the RosettaNet participant business process file to create a public participant business process. The RosettaNet participant template is based PIP3A4, but you can use it for other PIPs as well with only slight variations (such as the PIP schema and PIP identifying information in the annotations).

To create a RosettaNet participant business process

1. If you have not already done so, create a new application or a new project within an existing application. To learn more about projects and applications, see [Getting Started](#).

2. From the WebLogic Workshop menu, choose **File**—**New**—**Process File**.
3. In the **New File** dialog, select **Processes**, and then select **RosettaNet Participant Process File**.
4. In the **File name** field, enter a valid java class name for the JPD file.
Note: This name is used as the default value for the `pip-name` attribute in the `@jpd:rosettanet` annotation. Before you run your RosettaNet participant business process in production mode, you must change the `pip-name` attribute to a valid PIP code. For more information see, [Customizing a RosettaNet Participant Business Process](#).
5. If you want to create the JPD file in a directory other than the one displayed in **Create in**, then click the **Browse** button and select the target directory.
6. Click the **Create** button.
WebLogic Workshop creates a new RosettaNet participant process JPD file and displays it in the Design View pane.
7. To save your work, select **File**—**Save**.

Related Topics

[About the RosettaNet Participant Business Process File](#)

[Customizing a RosettaNet Participant Business Process](#)

Customizing a RosettaNet Participant Business Process

After you create a RosettaNet participant business process, you must customize it for the associated RosettaNet conversation. Common customization tasks include:

- [Configuring Business Process Properties \(Required\)](#)
- [Customizing Argument Types \(Optional\)](#)
- [Configuring Data Transformation \(Required\)](#)
- [Integrating with the Private Participant Process \(Required\)](#)
- [Setting Up the Notification of Failure \(Required\)](#)

Depending on your implementation requirements, you might make additional customizations to the participant business process as needed.

Configuring Business Process Properties (Required)

The RosettaNet participant business process file specifies the following default annotations:

```
@jpd:rosettanet protocol-name="rosettanet" protocol-version="2.0"
pip-name="processName" pip-version="1.0" pip-role="Seller"
```

These properties are set in the **Property Editor** that is visible when you have the Start node of your business process selected. Review and edit (if needed) the following properties:

Property	Default	Description
protocol-name	rosettanet	Do not change.
protocol-version	2.0	Change to 1.1 if you are using RNIF (RosettaNet Implementation Framework) version 1.1 instead.
pip-name	<i>processName</i>	Change to the RosettaNet PIP code, such as 3B2. Must be a valid PIP code as defined in http://www.rosettanet.org .
pip-version	1.0	Change to your RosettaNet PIP version (example: v01.01 for PIP3B2). Must be a valid version number associated with the PIP.
pip-role	Seller	Change to the RosettaNet name of the participant's role as defined in the PIP specification (example: Receiver for PIP3B2).

Note: If the **Property Editor** is not visible in **Design View**, choose **View—Property Editor** from the WebLogic Workshop menu.

To learn more about these annotations, see [@jpd:rosettanet Annotation](#). To learn about configuring business process properties, see [Business Process Language](#).

Customizing Argument Types (Optional)

By default, the RosettaNet participant business process includes a **Client Request** node, named **Receive Message**, to handle an incoming RosettaNet request message from an initiator. For the `response` and `callback.sendReply` methods, you might need to specify the attachment and its Java data type. The data type must match the contents of the incoming message and can be one of the following values:

Data Type	Description
XmlObject	Default. Represents data in untyped XML format. The XML data is not specified at design time.
RawData	Represents any non-XML structured or unstructured data for which no MFL file (and therefore no known schema) exists.
MessageAttachment[]	Array containing one or more parts of a RosettaNet business message. Message parts can be untyped XML data (XmlObject data type) or non-XML data (RawData data type). Used when sending different kinds of payloads (XML and non-XML) in the same message. The actual number of message parts might not be known until processed. To learn about working with MessageAttachment objects, see Using Message Attachments .

Note: Attachments can also be typed XML or typed MFL data as long as you specify the corresponding XML Bean or MFL class name in the parameter.

To learn more about data types, see [Working with Data Types](#).

Configuring Data Transformation (Required)

Public and private business processes often use different document formats. Public business processes use the associated PIP schema. Private processes use whatever format is required by the internal process (XML or binary), such as a proprietary ERP format. If a private business process does not use the PIP format, then the public business process needs to transform data between the PIP format to the format used in the private business process.

To configure data transformation, you need to:

- Import the schemas you need for data transformation into the project, including any schemas associated with the PIP and the format used in the internal process. To learn about importing schemas, see [How Do I: Import Files into a Schemas Project Folder?](#)
- Add a Transformation to the project, add methods to perform the transformations, and then drag these methods into the business process. To learn more about using transformations, see [Creating a Transformation Control and a Transformation Method](#).

Integrating with the Private Participant Process (Required)

After you create a RosettaNet public participant business process, you need to link it to the *private* participant process that handles the initiator's request privately. WebLogic Workshop provides many ways for communicating with other business processes, including:

- **Control Send** and **Control Receive** nodes (for asynchronous communication) or a **Control Send with Return** node (for synchronous communication). To learn more about control nodes, see [Interacting With Resources Using Controls](#).
- **JMS** (Java Message Service) controls. To learn more about using JMS, see [WLI JMS Control](#).
- **Perform** nodes for non-WebLogic Integration systems. To learn more about **Perform** nodes, see [Writing Custom Java Code in Perform Nodes](#).

Setting Up the Notification of Failure (Required)

In this participant business process, if a time-out occurs while awaiting a reply from the initiator to the response document, the participant needs to send a Notification of Failure (PIP0A1) to the initiator. To learn more about PIP0A1, see the following URL:

<http://www.rosettanet.org>

To notify the initiator of the failure, you need to create a separate initiator business process that implements PIP0A1, and then start the initiator business process:

- If the initiator business process is created in WebLogic Workshop, you can use a **Control Send** and **Control Receive** nodes (for asynchronous communication) or a **Control Send with Return** node (for synchronous communication). To learn more about control nodes, see [Interacting With Resources Using Controls](#).
- If the initiator business process is not created in WebLogic Workshop, you can use a **Perform** node instead. To learn more about **Perform** nodes, see [Writing Custom Java Code in Perform Nodes](#).

To learn about initiator business processes, see [Introducing Trading Partner Integration](#) at the following URL:

<http://edocs.bea.com/wli/docs81/tpintro/index.html>

Related Topics

[About the RosettaNet Participant Business Process File](#)

[Creating a RosettaNet Participant Business Process](#)