Oracle® WebLogic Integration

Guide to Building Business Processes

10*g* Release 3 (10.3.1)

January 2010

ORACLE®

Oracle WebLogic Integration Guide to Building Business Processes, 10*g* Release 3 (10.3.1)

Contents

Interacting With Resources Using Controls

Receiving Multiple Events

Creating Parallel Paths of Execution

Defining Conditions For Branching

Creating Case Statements

Writing Custom Java Code in Perform Nodes

Creating Looping Logic

Looping Through Items in a List

Specifying Endpoints in Your Business Process

Grouping Nodes in Your Business Process

Handling Exceptions

Adding Message Paths

Adding Timeout Paths

Running and Testing Your Business Process

Business Process Variables and Data Types

Validating Schemas

Versioning Business Processes

Building Stateless and Stateful Business Processes

Building Synchronous and Asynchronous Business Processes

Transaction Boundaries

Business Process Source Code

Building ebXML Participant Business Processes

Building RosettaNet Participant Business Processes

Building and Deploying Integration Applications

Calling Business Processes

Guide to Building Business Processes

Oracle 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 Oracle Workshop for WebLogic 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 **Node Palette** pane 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 Oracle Workshop for WebLogic, and provides step-by-step instructions for creating a business process project in Oracle Workshop for WebLogic. 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. Oracle Workshop for WebLogic 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 21, "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 Oracle Workshop for WebLogic versioning feature.

Chapter 20, "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 Oracle Workshop for WebLogic writes to a business process file (a Process.java 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 Oracle Workshop for WebLogic.

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 Oracle Workshop for WebLogic.

Chapter 28, "Building and Deploying Integration Applications"

Describes how to work in iterative development mode and deploy applications in EAR format.

Chapter 29, "Calling Business Processes"

Describes how Business Processes can expose their functionality to clients in several ways, including through WSDL files, Process Controls, Service Broker Controls, and JPD Proxies.

Related Topics

Oracle Workshop For WebLogic User's Guide

Creating a Business Process Application

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

This section describes the components of a WLI application, the steps you follow to create an application in Oracle Workshop for WebLogic, 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.

Application—The components of the application you are creating are represented in a tree structure on the **Package Explorer** pane in your Oracle Workshop for WebLogic environment. If the **Package Explorer** pane is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Other > Java > Package Explorer** from the Oracle Workshop for WebLogic menu. An example of **Package Explorer** pane is shown in Figure 2-1.

Figure 2-1 Sample Package Explorer
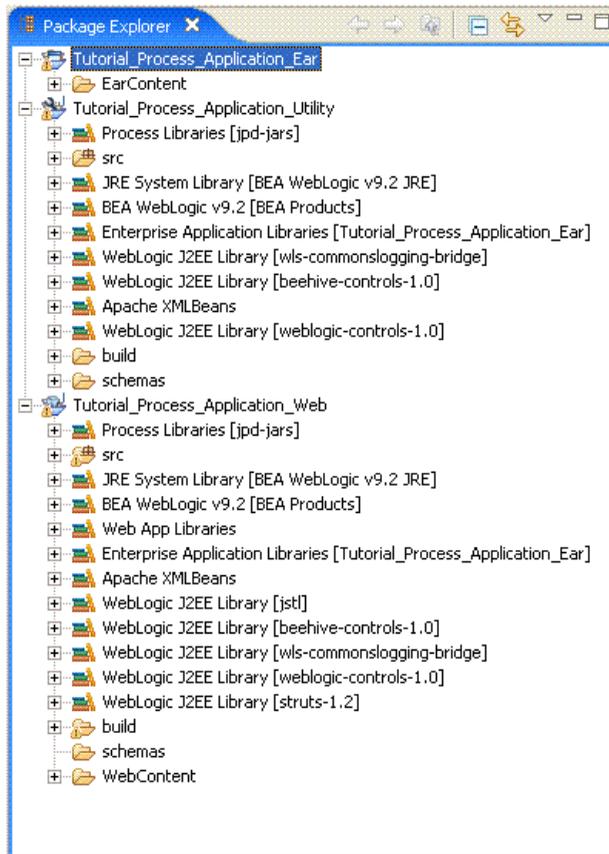
A WLI application will have the following projects (For example, the preceding figure represents an application named **Tutorial_Process_Application** and includes an Enterprise Application Project, Web Project, and Utility Project):

- **Enterprise Application (EAR) Project**— A project that contains JAR files and deployment descriptors build files and auto-generated files. J2EE Applications and their components are deployed on the Oracle WebLogic Server as EAR files.

- **Utility Project**— An optional project that contains the XML Schemas and the Message Broker channel file. When you create your process application through the wizard or using a template, the **Utility** project is created as part of the business process application folder .The Utility Project has a **schemas** folder in it. 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, Oracle Workshop for WebLogic 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.

– **Web Project**— A project that contains the Oracle WebLogic Server Web applications. In other words, when you create a project, you are creating a Web application. The name of your project "**Tutorial_Process_Application_Web"** which in turn contains a business process named **RequestQuote.java**. Clients can access your business process via the following URL:

```
http://host:port/Tutorial_Process_Application_Web/requestquote/RequestQ
uote.jpd
```

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

When a project is created, various information is assembled to specify the type of project, add standard libraries, set compiler options, control publishing tasks, set build path, and/or add an annotation processor. This information is specified by choosing facets during project creation. Facets can also be added and deleted from a project after its initial creation. To edit a project facets, select **Project > Properties > Project Facets**. For the business process (Process.java) and Worklist application, the Project facets has already been configured.

Note: A process can be created only in a WEB project which has Oracle WebLogic Integration Process facet (process-enabled) added to it. Similarly a Worklist task plan can be created in an EAR project which has Worklist Integration Worklist Application Module facet (worklist-enabled) added to it.

Schemas folder exported as a jar needs to be added to a Web Project or a Util project and not to an EAR project. Due to Eclipse and WTP restrictions, schema, channel builders, and controls are associated only with Web or Util project and not with EAR project. If you export or add the Schemas folder as a jar, into the module dependencies of the EAR project, the channel files will not be declared and the controls of the business process will not be availabe in the Data Palette.

Designing Your Application

You build your application in Oracle Workshop for WebLogic 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. From the Oracle Workshop for WebLogic menu, click **File > New > Other...**.

   The **Select a wizard** dialog box is displayed.

2. Expand WebLogic Integration and select **Process Application** and click **Next**.

3. In the **Process Application** dialog box, type the details as shown in the following example:

   a. In the **EAR Project Name** field, enter `Process_Application_EAR`.

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

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

4. Select **Add WebLogic Integration System and Control Schemas in Utility Project** check box.

   This adds the system schemas to the **Utility Project/schemas** folder.

5. Click **Finish**.

   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.java`).

   Note: If you select **Tutorial: Process Application** instead of **Process Application**, Oracle Workshop for WebLogic creates an application containing components for the Business Process. To learn about taking the tutorial, see Tutorial: Building Your First Business Process.You can also build an ebXML or RosettaNet participant business process in Oracle Workshop for WebLogic 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.

6. In the displayed **Open Associated Perspective?** dialog box, click **Yes** to switch from **J2EE Perspective** to **Process Perspective**.

   Note: The **Open Associated Perspective?** dialog box appears when creating a Business Process for the first time in a Workspace.

7. The Process Application is created and displayed in the **Package Explorer** pane, as shown in Figure 2-2.

Figure 2-2 Package Explorer
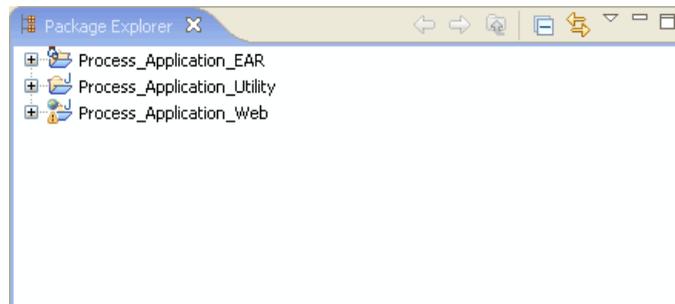
The Tutorial:Getting Started with Workshop briefly describes the components and tools you use to design your business process in the Oracle Workshop for WebLogic graphical design environment.

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 Process.java 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 **JPD Configuration** pane.

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 **JPD Configuration** pane is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Other > General > JPD Configuration** from the Oracle Workshop for WebLogic menu bar.

   Note: The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.

   To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

   In the **JPD Configuration** pane the following properties are displayed: general and process.

general

- **name—**This is the name of your business process, which is displayed throughout the Oracle Workshop for WebLogic application, including the Oracle 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 **notes** and then clicking ▢ to open the **Notes** dialog box. Enter your note and click **OK**. Notes entered in the editor will be also be displayed in the Oracle 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 Oracle 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 Oracle WebLogic Integration Administration Console, see Process Instance Monitoring in *Using the Oracle WebLogic Integration Administration Console.*

- **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.

- **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 Oracle 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 Oracle WebLogic Server *Administration Console Online Help*.

- **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.

Setting the Business Process Annotations

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

To Set the Business Process Annotations

1. Select the **Start** node of the business process for which you want to configure the annotations.

2. If the **Properties** pane is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Properties** from the Oracle Workshop for WebLogic menu bar.

   In the **Properties** pane, the following annotation are displayed:

BeanInfo

- **customizerclass**— This annotation defines which bean properties are public, and which getter/setter methods should be used to access them. This class must reside in the same package and be similarly named to the tag (or bean) with BeanInfo appended to it.

ClassReliable

- **messageTimeToLive—** This annotation specifies how long messages are maintained on the server, in order to detect duplicate messages.

Ebxml

For information, see ebXML annotation.

FeatureInfo

For information about FeatureInfo annotation, see
http://commons.apache.org/modeler/commons-modeler-1.0/docs/api/org/apache/commons/modeler/FeatureInfo.html.

Handler

- **callback**— This annotation specifies which handlers to run on outbound SOAP callbacks.

- **file—** This annotation specifies the handler configuration file to use, if handler chain is not specified inline.

- **operation—** This annotation specifies which handlers to run on incoming SOAP messages.

ManifestAttribute

For information about ManifestAttribute annotation, see
http://beehive.apache.org/docs/1.0/apidocs/classref_controls/org/apache/beehive/controls/api/packaging/ManifestAttribute.html.

ManifestAttributes

For information about ManifestAttribute annotation, see
http://beehive.apache.org/docs/1.0.2/controls/apidocs/javadoc/org/apache/beehive/controls/api/packaging/ManifestAttributes.html.

Protocol

For information, see Protocol annotation.

Rosettanet

For information, see RosettaNet properties.

Schemas

- **value**— This annotation specifies the one or more schema files whose types are referenced in the component class. This annotation can be added to the Process or Service Broker Control. Multiple name value entries are defined and each entry refers to one definition of a schema.

Stateful

- **maxAge**— This annotation specifies the amount of time elapsed since a stateful process started, the conversation may remain active before it is ended by the Oracle WebLogic Server.

- **maxIdleTime**— This annotation specifies the amount of time a stateful process remains idle before it is ended by the Oracle WebLogic Server.

- **runAsStartUser**— This annotation specifies that the continue and finish methods will be run as the user who started the conversation.

- **singlePrincipal**— This annotation specifies that only the principal who started the conversation can continue and finish the conversation.

TargetNamespace

- **value**— This annotation specifies the XML namespace used for outgoing XML messages and generated WSDL files.

Version

- **Strategy**—This describes how to invoke sub processes when differerent 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 the subprocess is invoked.

    – Select **tightly-coupled** if you want the subprocess version to be set at the time the parent process is invoked.

WSSecurityCallback

- **file**— This annotation specifies the location of the WS-Security file.

WSSecurityService

- **file**— This annotation specifies the path to the WS-Security policy file (WSSE file) used by the web service.

Wsdl

- **value**— This annotation specifies a WSDL file that is implemented by a web service.

XmlNamespaces

- **Value**— This annotation provide the namespace value and prefix value.

Com.bea.wli.common.XQuery

- **Prolog**—This specifies the namespace and function declaration.

- **Version**—This specifies the XQuery version used in the Process file for inline XQueries.

Com.bea.wli.jpd.Process

- **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**.

- **name**— This is the name of you process.

- **process**— This is a read-only annotation and displays the content of the process annotation.

  For more information on properties, see Javadocs.

  To learn about ebXML and RosettaNet business processes, see Building ebXML Participant Business Processes and Building RosettaNet Participant Business Processes.

Note: WS-Security policy (WSSE) files are not supported for business processes (JPDs). Therefore, the following annotations are not supported for JPD files: `com.bea.wli.common.WSSecurityCallback` and `com.bea.wli.common.WSSecurityService`.

If you want to use WS-Security, then you must front-end the JPD with a JWS. The client would invoke a JWS using WS-Security, then the JWS would locally invoke the JPD via a Process Control.

Related Topics

Handling Exceptions

Process Instance Monitoring

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 or 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

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 **Start Event** of your process can contain any combination of **Client Request**, **Client Request with Return**, or **Subscription** nodes. You design the **Start Event** of your process by double-clicking the *Start 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 *Start Event* place holder, and a **Finish** node, as shown in the following figure Figure 3-1.

Figure 3-1 Start Node
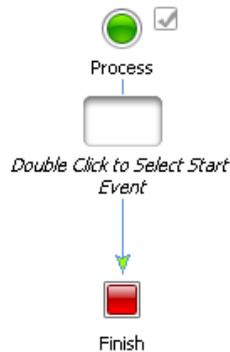
The first action in the business process is specified at the **Start** node. That is, you specify how the business process is started at run time by defining a **Start Event**. The empty node attached to the **Start** node, as well as the gray check box ☑, shown in the preceding figure, indicate that 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, Oracle Workshop for WebLogic 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 *Start Event* placeholder on the **Start** node in the **Design** view to display the Start node builder.

    The node builder displays, as shown in Figure 3-2.

Figure 3-2 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 Event Generator, which facilitate publishing events to Message Broker channels.

   Note: In Oracle 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 Javadocs.

   – **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 Event Generator, 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 **Close**.

    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

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 Figure 3-3.

Figure 3-3 Client Request Start Node

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

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 Figure 3-4.

Figure 3-4 Client Request with Return start Node
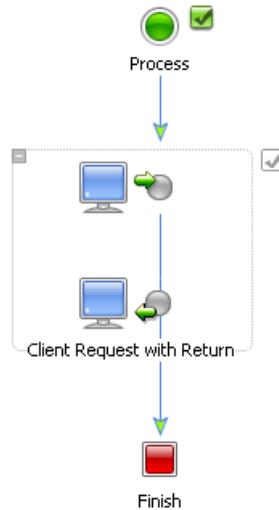
Note the following properties for the **Client Request with Return** group node:

- ☑ 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 design 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 node builder is invoked. 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**. A panel, which shows the data types is displayed.

4. 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.

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

   – **Simple Types**

     Lists Java primitive and classes data types

   – **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 XML Schema.

– **Non-XML Types**

Lists the Message Format Language (MFL) files available in your business process project and the untyped RawData data type. Oracle 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 Oracle Workshop for WebLogic and is also available in the XML Types listing.

For more detailed descriptions of the data types, see Working with Data Types.

6. After you select the data type, click **OK**. The parameter specification you made is displayed in the **General Settings** tab in the node builder.

   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   Creating and Importing Schema Files.

7. 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.

8. To remove a parameter from the node builder pane, select the parameter in the list and then click **Remove**.

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

   – **Variable Assignment**- Use this mode when you want to assign the data received from the client to a variable of the same data type. By default, the **Receive Data** tab opens on the **Variable Assignment** panel.

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

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, click **OK**. Your new variable is created and displayed in the **Receive Data** tab.

3. If the data types of your method parameters and your variables are different, you can use the XQuery Mapper Tool included in Oracle Workshop for WebLogic to map between heterogeneous data types. The data transformations you create using the tool are stored in data transformation files. When the data transformation files containing your data transformations are built, they are built as controls. The controls expose transformation methods, which business processes invoke to map disparate data types.

   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 parameter from the node builder pane, select the parameter 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. If an instance of a Transformation control (defined by a data transformation file) already exists in your project, then that instance is used 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 **XQuery Mapper** tool window. This automatically applies changes to the builder and opens a transformation editor in a new window.

The XQuery Mapper 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 XQuery Mapper Tool, see the *Guide to Data Transformation*.

Note: To return to node builder, in the **Package Explorer** pane, double-click the Process.java 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 **Close**.

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] 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.

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:

   – **Simple Types**

   – Lists Java primitive and classes data types.

   – **XML Types**

   Lists the XML Schemas that are available in your business process project and the untyped XMLObject and XMLObjectList data type.

   – **Non-XML Types**

   Lists the Message Format Language (MFL) files available in your business process project and the untyped RawData data type. Oracle 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 Oracle Workshop for WebLogic and is also available in the XML Types listing.

   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. By default, the **Send Data** tab opens on the Variable Assignment panel.

   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 **Close**.

3. If the data types of your return value and your variables are different, you can use the XQuery Mapper tool included in Oracle Workshop for WebLogic to map between heterogeneous data types. The data transformations you create using the tool are stored in data transformation files. When data transformation files containing your data transformations are built, they are built as controls. The controls expose transformation methods, which business processes invoke to map disparate data types.

    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. If an instance of a Transformation control already exists in your project, then that instance is used 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 **XQuery Mapper** tool window. This automatically applies changes to the builder and opens a transformation editor in a new window.

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

    Note: To return to node builder, in the **Package Explorer** pane, double-click the Process.java 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 **XQuery Mapper tool** window, click **Close**.

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 **Close**.

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

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 Figure 3-5.

Figure 3-5 Subscription Start Node

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 Oracle 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.

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.

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.

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 (see Figure 3-6).

Figure 3-6 Channel Name

Note: If no appropriate channels are available for you to select, you must create a Channel
file that specifies the Message Broker channels for your application.

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 **Close** to close the node builder.

6. If the data types of your method parameters and your variables are different, you can use the XQuery Mapper tool included in Oracle WebLogic Integration to map between heterogeneous data types. The data transformations you create using the tool are stored in data transformation files. When the data transformation files containing your data transformations are built, they are built as controls. The controls expose transformation methods, which business processes invoke to map disparate data types.

   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 data transformation files 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.

Note: To return to node builder, in the **Package Explorer** pane, double-click the Process.java 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 **Close**.

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 **Close**.

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

Sending Messages to Clients

Handling Exceptions

Client Operations and Control Communication Methods

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 Figure 3-7.

Figure 3-7 Subscription Start (Synchronous) Node
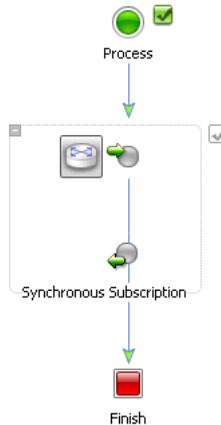
A synchronous static subscription to a Message Broker channel is defined on the **Synchronous Subscription** node. Your business process is started as the result of receiving a synchronous message from a Message Broker channel.

Note: In Oracle 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.

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.

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 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 channel file that specifies the Message Broker channels for your application.

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

   The data type and the 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 **Ok**.

6. If the data types of your method parameters and your variables are different, you can use the XQuery Mapper tool included in Oracle WebLogic Integration to map between heterogeneous data types. The data transformations you create using the tool are stored in data transformation files. When data transformation files containing your data transformations are built, they are built as controls. The controls expose transformation methods, which business processes invoke to map disparate data types.

   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. If an instance of a Transformation control already exists in your project, then that instance is used 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 **XQuery Mapper** 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.

      Note: To return to node builder, in the **Package Explorer** pane, double-click the Process.java 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 **Close**.

> **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

Client Operations and Control Communication Methods

Message Broker

Event Generators

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 Figure 3-8.

Figure 3-8 Event Choice Start Node
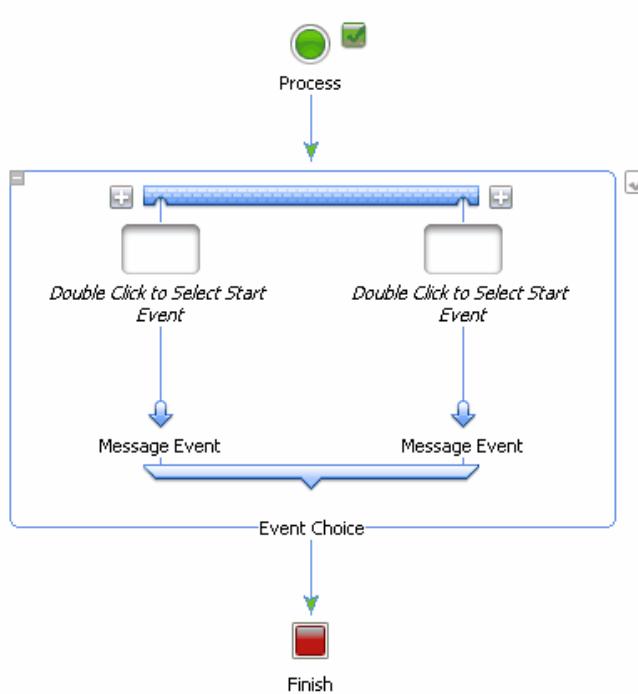
By default, **Event Choice** nodes are created with two branches. Click ⊞ to create additional branches. A new branch is added on the left or right of the existing branches.

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 *Start 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 *Start Event* placeholder to invoke the node builder, see Figure 3-9.

Figure 3-9 Node Builder Event
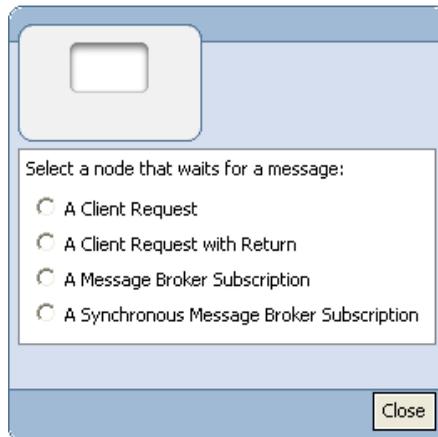
2. From the node builder, select the event for which this branch waits:

    – **A Client Request**

    – **A Client Request with Return**

    – **A Message Broker Subscription**

    – **A Synchronous Message Broker Subscription**

3. Click **Close**.

    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

Business Process Source Code

Adding Message Paths

Adding Timeout Paths

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 **Package Explorer** pane, double-click the business process (Process.java file) you want to design. Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Drag and drop **Client Request** from the **Node 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, Oracle Workshop for WebLogic 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:

   – **Simple Types**

   Lists Java primitive and classes data types.

> – **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 Creating and Importing Schema Files.
>
> – **Non-XML Types**
>
>   Lists the Message Format Language (MFL) files available in your business process project and the untyped RawData data type. Oracle 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 Oracle Workshop for WebLogic and is also available in the XML Types listing.

For more detailed descriptions of the data types, see Working with Data Types.

4. Click **OK**.

   The parameter specifications you made is displayed in **General Setting** tab in the node builder.

   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 Creating and Importing Schema Files.

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

      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, then under **Select variable to assign**, click the arrow and 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 **Ok**. Your new variable is created and displayed in the **Receive Data** tab.

3. If the data types of your method parameters and your variables are different, you can use the Transformation tool included in Oracle Workshop for WebLogic to map between heterogeneous data types. The data transformations you create using the tool are stored in data transformation files. When data transformation files containing your data transformations are built, they are built as controls. The controls expose transformation methods, which business processes invoke to map disparate data types.

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 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** XQuery Mapper 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 **Package Explorer** pane, double-click the Process.java 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 **Ok**.

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

   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

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 **Package Explore** pane, click the Process.java file you want to design. Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Click **Client Response** in the **Node 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, Oracle Workshop for WebLogic 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:

   – **Simple Types**

     Lists Java primitive and classes data types.

   – **XML Types**

     Lists the XML Schemas that are available in your business process project. To learn how to import a Schema into your project.

   – **Non-XML Types**

     Lists the Message Format Language (MFL) files available in your business process project. Oracle 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 Oracle Workshop for WebLogic and is also available in the XML Types listing.

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. By default, the **Send Data** tab opens on the **Variable Assignment** panel.

      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, then under **Select variable to assign**, click the arrow and select **Create new variable...**To Create a New Variable in the Node Builder.

   d. If the data types of your method parameters and your variables match, click **Ok**. Your new variable is created and displayed in the **Send Data** tab.

3. If the data types of your method parameters and your variables are different, you can use the XQuery Mapper tool included in Oracle Workshop for WebLogic to map between heterogeneous data types.The data transformations you create using the tool are stored in data transformation files. When data transformation files containing your data transformations are built, they are built as controls. The controls expose transformation methods, which business processes invoke to map disparate data types.

   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 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.

Note: To return to node builder, in the **Package Explorer** pane, double-click the Process.java 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 **Close**.

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 Oracle 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 **Properties** pane, in the **xquery** field under the **ReturnXml** section, enter the Dynamic Selector XQuery data.

For information about how to configure the security information associated with your dynamic callback property, see Adding or Changing Dynamic Client Callback Selectors in *Using The Oracle WebLogic Integration Administration Console*.

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

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 **Properties** pane, in the **message buffer** section do the following:

    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**.

Interacting With Clients

Interacting With Resources Using Controls

Oracle Workshop for WebLogic 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 Oracle Workshop for WebLogic 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 and Annotations

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 **Package Explorer** pane, double-click the business process (Process.java file) you want to design.Your business process is displayed in the **Design** view.

2. 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 Oracle Workshop for WebLogic, choose **Window** > **Show View** > **Data Palette** from the Oracle Workshop for WebLogic menu.

   b. If you have already added an instance of your control to your business project (see 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, Oracle Workshop for WebLogic 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 **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

b. If you have not yet created a control, click the control on the **Node Palette** that fits the action you want to create:

➜ **Control Send**—Choose 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 **Node 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**.

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

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. If the **Data Palette** pane is not visible in Oracle Workshop for WebLogic, choose **Window** > **Show View** > **Data Palette** from the menu bar.

2. Click ▽ on the **Data Palette**. A drop-down list of controls that represent the resources with which your business process can interact is displayed. Instances of controls already available in your project are displayed in the **Controls** tab.

3. Select a control from the **Integrations Controls** drop-down menu.

   Note: Table 5-1contains information about the standard controls used in Oracle WebLogic Integration. Other custom and plug-in controls may be available.

Table 5-1 Controls in Oracle WebLogic Integration

Type of Control	Description
ALSB	The ALSB control enables a business process to invoke a ALSB proxy service via RMI with security and transaction context propagation support.
Dynamic Transformation	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.
ebXML	The ebXML control enables Oracle Workshop for WebLogic 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.
Email	The Email control enables Oracle Workshop for WebLogic 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.
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.

Table 5-1 Controls in Oracle WebLogic Integration

Http	This control enables Oracle Workshop for WebLogic 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.
MB Publish	Message Broker (MB) Publish controls allow your business process to publish messages to Message Broker channels.
	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.
MB Subscription	Message Broker (MB) Subscription controls allow your business process to dynamically register for and receive messages from a Message Broker channel.
	In Oracle 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.
	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.
MQSeries Control	The MQSeries control enables Oracle Workshop for WebLogic 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.
Process	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.
	To create a Process control, on the **Package Explorer** pane, right-click a business process (Process.java) file to display a drop-down menu. Select **Generate > Process** Control from the drop-down menu. Oracle Workshop for WebLogic creates a Business Process control file (.java file) in your project

Table 5-1 Controls in Oracle WebLogic Integration

RosettaNet	The RosettaNet control enables Oracle Workshop for WebLogic 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.
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.
TPM	The TPM (Trading Partner Management) control provides read-only access to trading partner information stored in the TPM repository.
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 Process.java file like most other WLI controls and is generated specifically to interact with Task Plan.
Task Batch	The Task Batch replaces the Task Worker control from Oracle WebLogic Integration 8.x. A Task Batch helps to manage multiple tasks in its operations and is generated specifically for a Task Plan.
Tibco	The Tibco control helps you to send and receive messages in XML, String, and Tibco proprietary Rendezvous Message (TibrvMsg) formats.
Transformation	Use a Transformation control to achieve data transformations for data in your business processes. A Transformation control can be created and edited from within communication nodes in a business process and via the **File** > **New** > **Transformation** option on the Oracle Workshop for WebLogic menu.
WLI JMS	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 Oracle Workshop for WebLogic control.
WLI Timer Control	A Timer control notifies your application when a specified period of time has elapsed or when a specified absolute time has been reached.
XML MetaData Cache	The XML MetaData Cache control only allows you to retrieve XML metadata from the XML cache. The XML cache is managed using the Oracle WebLogic Integration Administration Console.

4. After you select a type of control from the **Integration Controls** list.

   An **Insert Control** dialog box, which contains tasks specific for the control you selected, is displayed.

   Note: For WLI Timer Control and WLW Timer Control, an Insert Control dialog box do not appear.

5. In the **Insert Control** dialog box, enter the information specific for the control you want to create in the **Field Name** and click **Next**.

6. Enter the required information in **Create Control** wizard and click **Finish**.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.

7. If you want to use an existing control created in an earlier business process (Process.java file), you can do it in the following ways:

   – Drag an existing control from the **Package Explorer** pane to the **Data Palette**, **Controls** folder.

   – Click **Menu** tab, on the **Data Palette** and select **Local** control. From the drop-down list select an exiting control.

Note About Transformations

Transformations handle mapping heterogeneous data types in your application. Oracle Workshop for WebLogic provides a data mapping tool to map between heterogeneous data types. The data transformations you create using the tool are stored in data transformation files. They can hold multiple transformations and are designed to enable packaging, sharing and reuse of transformation formats. When data transformation 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:

- In a Process perspective or XQuery Transformation perspective, by choosing **File > New > Transformation** from the Oracle Workshop for WebLogic menu.

- By choosing **File > New > Other > WebLogic IntegrationTransformation** from the Oracle Workshop for WebLogic menu.

- During the design of a node builders for any **Control** or **Client** 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 Edit or Delete an Instance of a Control

In the **Controls** tab, right-click on the 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.

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, see Figure 5-1.

Figure 5-1 Control Nodes

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 **Close**.

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 **Close** 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 Oracle Workshop for WebLogic to map between heterogeneous data types. The data transformations you create using the tool are stored in data transformation files. When data transformation files containing your data transformations are built, they are built as controls. The controls expose transformation methods, which business processes invoke to map disparate data types.

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 data transformation 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.

>
> **Note:** To return to node builder, in the **Package Explorer** pane, double-click the Process.java 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**.

Note: In the Advanced Option window, if you enter the right input parameter and it matches with the signature of the transformation method, the Ok button will be enabled.

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

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

Guide to Data Transformation

Setting Control Properties and Annotations

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 **JPD Configuration** pane and **Properties** pane.

To View and Edit Properties for Control Types

To View and Edit Annotations for Control Instances

To View and Edit Properties for Control Types

Double-click the control type on the **Package Explorer** pane.

The file is displayed in the **Source** view, and its properties are displayed in the **JPD Configuration** pane. The properties you see and edit in the **JPD Configuration** pane depend on the control you are using.

Values you specify for the properties in the **JPD Configuration** pane are written to the file. In other words, the **Source** view is updated in keeping with the work you do in the **JPD**

Configuration pane. Properties you specify for the control are inherited by any instances of the control you create based on this type.

To View and Edit Annotations for Control Types

The file is displayed in the **Source** view, and its annotations are displayed in the **Properties** pane. The properties you see and edit in the **Properties** pane depend on the control you are using.

Values you specify for the annotations in the **Properties** pane are written to the file. In other words, the **Source** view is updated in keeping with the work you do in the **Properties** pane. Annotations you specify for the control are inherited by any instances of the control you create.

Follow the above, to view and edit annotations for control instances.

To View and Edit Annotations for Control Instances

Double-click the control instance in the **Data Palette** to display its properties in the **Properties** pane. The annotations you can see and edit depend on the control you are using. Note that when you open the **Properties** pane for an instance of a control, the annotations for that instance, are listed at the top of the **Properties** pane. You can edit the referenced control properties by opening the file as described in To View and Edit Properties for Control Types.

Note: Follow the above instructions for the annotations.

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.

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 **Package Explorer** pane, click the business process (Process.java file) you want to design.Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Click ⥮ **Event Choice** in the **Node 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 Figure 6-1.

Figure 6-1 Event Choice Node
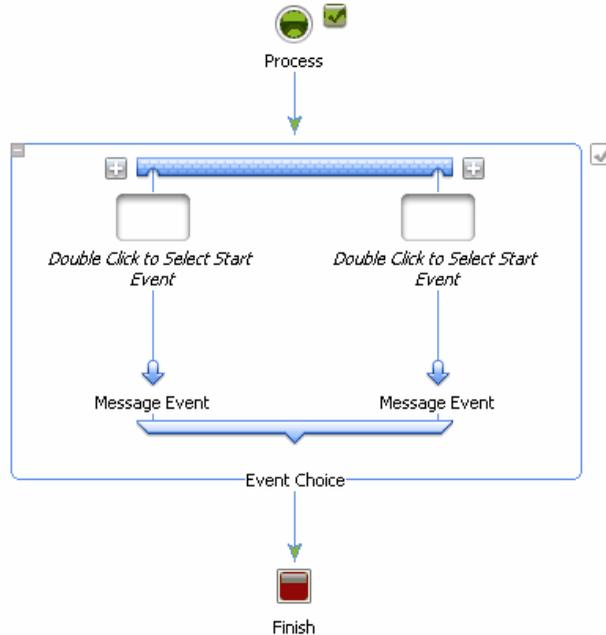
Note the following characteristics of the **Event Choice** node:

– 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 **Properties** pane. 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 *Start 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 **Start 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 subsrciptions 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**.

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 (**Start 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 **Close**. 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 Figure 6-2, is added to the **Event Choice** node:

Figure 6-2 Time Branch
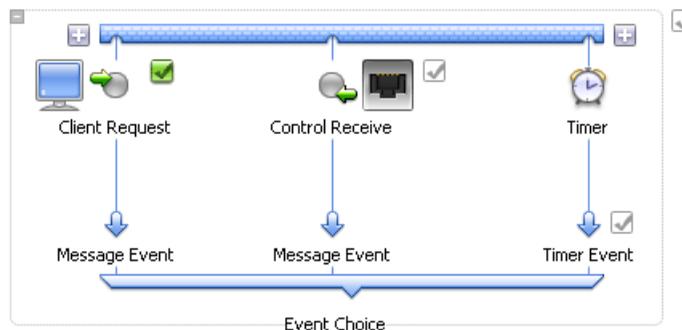
2. You can set the properties for the Timer branch (and other properties for this group of nodes) in the **Properties** pane.

   a. If the **JPD Configuration** pane is not visible in the **Design** view, choose **Window** > **Show View** > **Other > General > JPD Configuration** from the Oracle Workshop for WebLogic menu.

      Note: The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.

To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

b. Select the Timer branch. The **JPD Configuration** pane for the **Event Choice** node appears as shown in Figure 6-3.

Figure 6-3 Time Branch Property
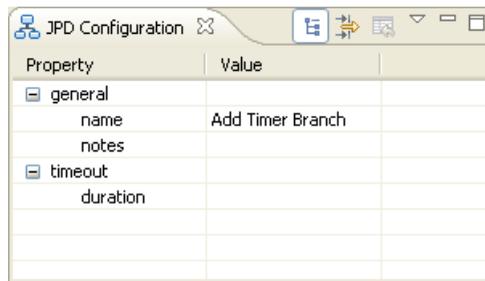
c. In the **timeout** property, select the **duration** attribute, then specify the number of seconds before the timer path is triggered. (The expected format is Xs, for example 7s.)

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**.

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.

Create a Parallel Node in Your Business Process

To Add A Parallel Node to Your Business Process

1. On the **Package Explorer** pane, double-click the business process (Process.java file) you want to design.Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Click Parallel in the **Node 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 Figure 7-1.

Figure 7-1 Parallel Node
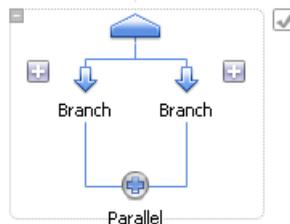
Note the following characteristics of the **Parallel** node:

– 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**.

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 and annotations for a group of nodes using the **JPD Configuration** pane and **Properties** pane.

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 **JPD Configuration** pane.

Note: If the **JPD Configuration** pane is not visible in the **Design** view, choose **Window > Show View > Other > General > JPD Configuration** from the Oracle Workshop for WebLogic menu.

The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.

To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

2. To change the value of the Join Condition from **AND** (the default) to **OR**, in the **Properties** pane, 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** pane, 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 **Node Palette**, click a **Node** that represents the type of logic you want to add to the business process.

2. Drag and drop the node from the **Node Palette** onto the appropriate branch, see Figure 7-2.

Figure 7-2 Process Nodes
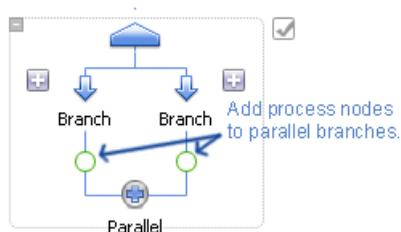
3. Complete the design of the nodes added on each branch. In this way, you create the activities appropriate for the business logic defined by your business process.

Note: You can create nested **Parallel** nodes in your business process by dragging a **Parallel** node from the **Node 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

Grouping Nodes in Your Business Process

Handling Exceptions

Business Process Source Code

Adding Message Paths

Adding Timeout Paths

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, Oracle 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 **Package Explorer** pane, click the business process (Process.java file) you want to design. Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Click **Decision** on the **Node 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 Figure 8-1.

Figure 8-1 Decision Node
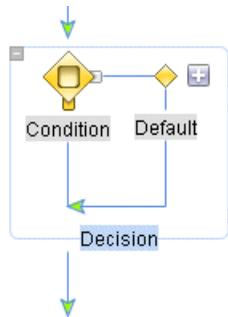
Note the following characteristics of the **Decision** node:

- 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 **JPD Configuration** pane and **Properties** pane. To learn about groups, see Grouping Nodes in Your Business Process.

Related Topics

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.

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 as shown in Figure 8-2.

Figure 8-2 Select Expression Node
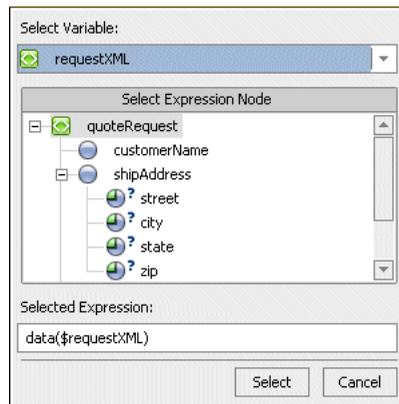
The elements and attributes of an XML document, assigned to this variable, are represented as nodes in a hierarchical representation, as shown in the preceding figure. Note that the schema in the example (`QuoteRequest.xsd`) specifies a root element (`quoteRequest`), and child elements: `customerName`, `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:

    `fn: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, `fn: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 Process.java file; it is indicated by the `@com.bea.wli.commom.XQuery` 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. Click **Close** 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 Process.java 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 Process.java file:

```
@com.bea.wli.common.Xquery(prolog =
" declare namespace ns0 =\http://ww.example.org./quote\";"+
" declare function cond_requestXML_1($requestXML) as  xs:boolean {"+
" fn:data($requestXML/customerName) = \"BEA\""+
and" fn:data($requestXML/customerName) = \"Avitek\""+
```

"};")

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 Process.java file at which the Java method is written.

3. Edit your Java method.

4. To return to the **Design** view, click the tab.

5. Close the condition builder by clicking **Close**.

    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, you are ready to define the actions on the paths in the conditions.

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

Figure 8-3 shows an example **Decision** node in the **Design** view.

Figure 8-3 Decision Node
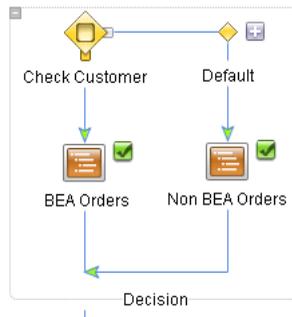
Building on the QuoteRequest example used in building the Variable (Schema) condition, two **Perform** nodes are added to the paths on the **Decision** node. At run time, the following sequence represents the flow of control in this decision node:

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

   ``` " fn:data($requestXML/customerName) = \"BEA\""+ ```

   ``` and" fn:data($requestXML/customerName) = \"Avitek\"" ```

   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

Grouping Nodes in Your Business Process

Handling Exceptions

Adding Message Paths

Adding Timeout Paths

Business Process Source Code

Interacting With Resources Using Controls

Defining Conditions For Branching

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 **Package Explorer** pane, right-click the business process (Process.java file) you want to design. Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Click ⚛ **Switch** in the **Node 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 Figure 9-1.

Figure 9-1 Switch Node
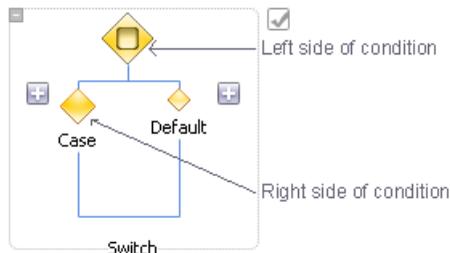
Note the following characteristics of the **Switch** node:

- 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 and annotations of your **Switch** node by clicking the outline of the group to select it, then viewing the group properties and annotations in the **JPD Configuration** pane and **Properties** pane respectively. To learn about groups, see Grouping Nodes in Your Business Process.

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: Variable 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 select **Create new variable...**, to 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 as shown in Figure 9-2.

Figure 9-2 Select Variable
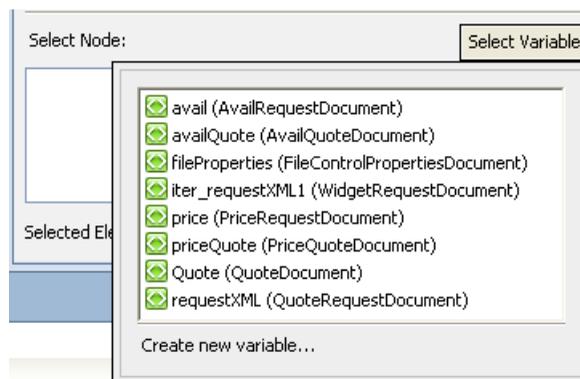
Note: (To learn about creating business process variables and importing schemas to your project, see Business Process Variables and Data Types and Creating and Importing Schema Files.)

When you select a variable, a representation of the XML Schema associated with that variable is displayed in the **Select Node** pane.

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 **Simple**, **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, Figure 9-3 shows the XML Schema represented when the `requestXML` variable is selected:

Figure 9-3 XML Schema

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 `$requestXML/ns0:shipAddress/@street)` is displayed in the **Selected Element** field.

6. Click **Close** 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 left side of the **Java Method Name** field.

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

   The **Source** view is displayed at the line of code in your Process.java file at which the Java method is written.

3. Edit your Java method and click the **Design** tab to return to the **Design** view.

4. Click **Close**, 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 **Close**.

To Add Activities to the Paths in Your Switch Node

After you define the condition, 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 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

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 Process.java 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 **Package Explorer** pane, double-click the business process (Process.java file) you want to add the **Perform** node to. Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Click ▦ **Perform** in the **Node 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 left-hand corner of the **Perform** builder.

The **Source** view is displayed at the line of code in your Process.java 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** tab to return to the **Design** view.

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

    Your Process.java 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.

While Do, **Do While**, 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 **While Do** and **Do While** 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, **While Do** and **Do While** 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 **Package Explorer** pane, click the business process (Process.java file) you want to add the While group node to.Your business process is displayed in the **Design View**.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu bar.

3. Determine whether you need *While Do* or *Do While* looping logic, and click **While Do** or **Do While** in the **Node 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 as shown in Figure 11-1.

Figure 11-1 While Node Groups
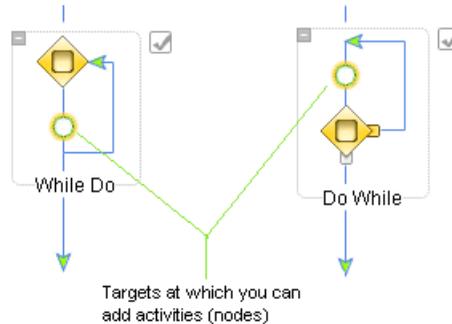
Note the following characteristics of the **While** groups:

- Each **While** group represents a placeholder for one or more additional nodes, a looping

  mechanism, and a condition builder ![icon]. 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.

- ![icon] indicates that the design of a group is incomplete. When you complete the design of the
  group, ![icon] is replaced by ![icon]. **While** groups are completed when the condition group is
  properly configured.

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 **JPD Configuration** pane.

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 11-2.

Figure 11-2 Collapsed Groups

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

CHAPTER **12**

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 **Package Explorer** pane, click the business process (Process.java file) you want to add the **For Each** node to.Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Click 📑 **For Each** in the **Node 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 Figure 12-1.

Figure 12-1 For Each Node
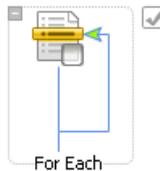
Note the following characteristics of the **For Each** node:

– 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.

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 Creating and Importing Schema Files.

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 as shown in Figure 12-2.

Figure 12-2 Select Variable
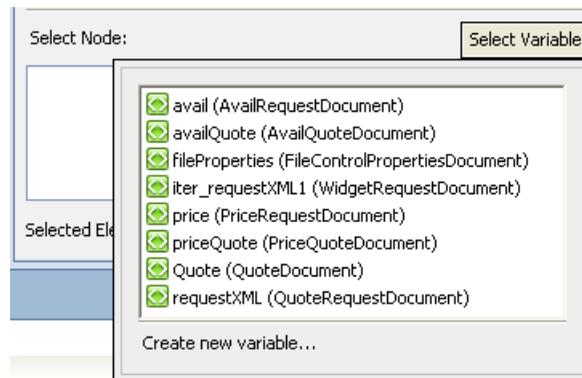
   Note: To learn about creating business process variables and importing schemas to your project, see Business Process Variables and Data Types.

   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 **Non-XML** 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.

A representation of the XML in the `requestXML` variable is displayed in the **Select Node** panel, see Figure 12-3.

Figure 12-3 XML Representation
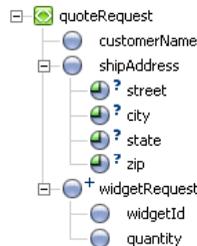
Note the following characteristics of the `QuoteRequest.xsd` schema as displayed in the preceding figure:

– 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`, `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 ⊕⁺ 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 ⊕⁺ in the GUI representation of the schema. A repeating XML element that specifies *zero or more* occurrences is represented by ⊕*.

3. 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.

4. Click **Close**, 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.

5. 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 **Node Palette**, click a node that represents the logic you want to add to the business
   process.

2. Drag and drop the node from the **Node 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.

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 **Package Explorer** pane, click the business process (Process.java file) you want to design. Your business process is displayed in the **Design** view.

2. If the **Node Palette** is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Node Palette** from the Oracle Workshop for WebLogic menu.

3. Click ▇ **Finish** in the **Node 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**.

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, 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. For example if you drag a Client Response node to a Group node throws an assertion error.

To Create Groups—Alternative 2

1. In the Design View, drag and drop **Group** from the **Node 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 **Node 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:

Figure 14-1 Exception Path

For more information about exception paths and how to configure them, see Handling Exceptions. To learn about the settings in the **JPD Configuration** pane, see Creating Exception Handler Paths.

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

Figure 14-2 Message Path
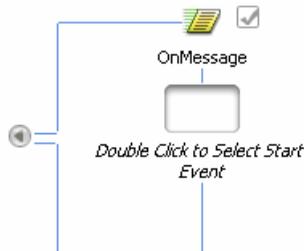
For information about how to configure your message path, see Adding Message Paths.

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

Figure 14-3 Time Path

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

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 exception path is not applicable to the following: `<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, as shown in Figure 15-1.

Figure 15-1 Exception Path

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 **Properties** view.

To Configure an Exception Path

1. Select the exception path which you want to configure.

The related properties are displayed in the **JPD Configuration** view. If the **JPD Configuration** view is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Other > General > JPD Configuration** from the Oracle Workshop for WebLogic menu bar.

Note: The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.

To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

2. In the **JPD Configuration** view, configure the following properties:

general

   – **name**—Enter the name you want to be displayed in the graphical design environment for this exception path.

   – **notes**—Enter any notes you want to be associated with this exception path. These notes can then be accessed through the Oracle WebLogic Integration Administration Console.

exception

– **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.

Note: when there is an exception path attached to a node and the exception is rethrown (`afterExecute="rethrow"`) the context_onException() will not be called.

– **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.

– **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.

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:

Table 15-1 Exception Path

	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 **Node 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.

Table 15-2 Collapse and Expanded View

	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 Oracle 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. In 8.x, Supports Global Transaction with Emulate Two-Phase Commit was the default setting, however, now Supports Global Transaction is off by default, so you need to enable it, when using the non-XA drivers.

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, 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, see Create a Client Request Node in Your Business Process.

Compensation Example

Figure 15-2 shows an example of how to use an exception path for compensation.

Figure 15-2 Exception Path for Compensation
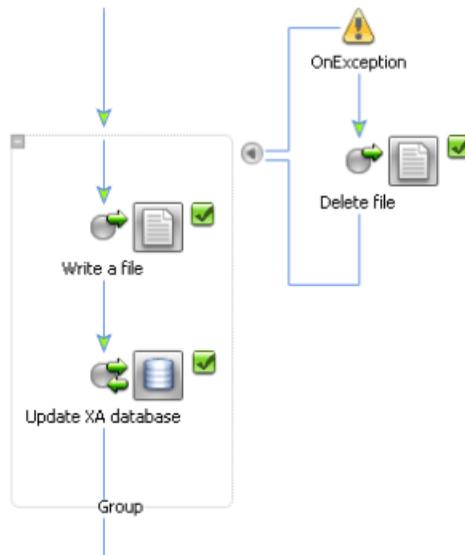
In the example, two nodes are running in the same transaction. The first node writes to a file (non-transactional) and the second node updates a database (transactional). Both nodes are within a group that has an associated exception handler. The exception handler **execute on rollback** property is 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. Figure 15-3 shows an example of an explicit transaction with an exception handler path with compensation logic.

Figure 15-3 Explicit Transaction
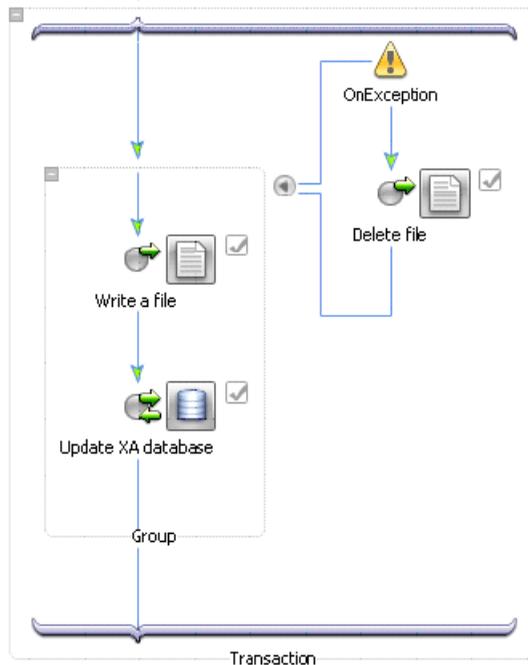
Unhandled Exceptions

If you do not handle exceptions in a business process, the exception will be wrapped in one of the following:

- `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()`

Related Topics

Grouping Nodes in Your Business Process

Writing Custom Java Code in Perform Nodes

Adding Message Paths

Adding Timeout Paths

Transaction Boundaries

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.

A Message path can be associated with a node that can receive messages. For example, a client request node, a control receive node, or a client request with return node.

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 shown in Figure 16-1.

Figure 16-1 Message Path
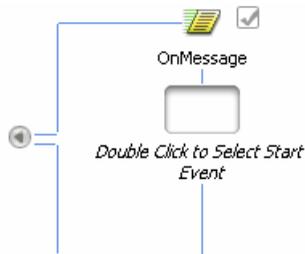
You can rename the path anything you like by double-clicking **OnMessage** and entering the new name. You can also change the name in the **name** field of the **JPD Configuration** pane.

To Configure a Message Path

1. Double-click **Start** node 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 **Close**, 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 **JPD Configuration** view. If the **JPD Configuration** view is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Other > General > JPD Configuration** from the Oracle Workshop for WebLogic menu bar.

> **Note:** The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.
>
> To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

5. In the **JPD Configuration** pane, configure the following properties:

general

– **name**—Enter the name you want displayed in the Oracle Workshop for WebLogic for this path.

– **notes**—Enter any notes you want associated with this message path. These notes can then be accessed through the Oracle WebLogic Integration Administration Console.

message

– **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.

exclusive— Blocks process execution until the handler completes.

– **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.

8. To configure the annotations, go to **Properties** view. If the **Properties** view is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Properties** from the Oracle Workshop for WebLogic menu bar.

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:

Table 16-1 Message Path
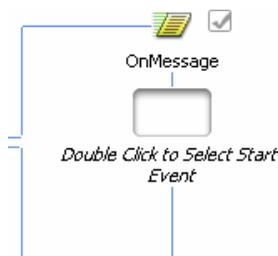
This icon represents the message path in your business process. In this case, the path appears empty, indicating that the logic to execute when a message is received is not defined yet.

To define the exception handling logic, add business process nodes by dragging the nodes from the **Node 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.

Table 16-2 Collapsed and Expanded View

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 Message Paths

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 (see Figure 17-1).

Figure 17-1 Timeout Path

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** pane.

Note: You cannot add a Timeout Path to stateless JPD

To Configure a Timeout Path

1. Select the Timeout path which you want to configure.

   The related properties are displayed in the **JPD Configuration** view. If the **JPD Configuration** view is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Other > General > JPD Configuration** from the Oracle Workshop for WebLogic menu bar.

   Note: The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.

   To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

2. In the **JPD Configuration** pane, configure the following properties:

general

- **name**—Enter the name you want to be displayed in the Oracle Workshop for WebLogic for this path.

- **notes**—Enter any notes you want to be associated with this timeout path. These notes can then be accessed through the Oracle WebLogic Integration Administration Console.

timeout

- **after execute**—Select the action you want to take place after a timeout path is executed.

  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).

- **duration**—Specify the number of seconds, minutes, days, month, or year that should lapse before the path is triggered. (Expected format is Xs, for example 5s.

- **exclusive**— Block process execution until the handler completes.

- **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.

4. To configure the annotations, choose **Window > Show View > Properties** from the Oracle Workshop for WebLogic menu bar. For information, see Annotations.

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:

Table 17-1 Timeout Path
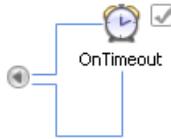
This icon represents the timeout path in your business process. In this case, the path appears empty, indicating that the logic to execute when a timeout is received is not defined yet.

To define the exception handling logic, add business process nodes by dragging the nodes from the **Node Palette** and dropping them on the timeout path.

You can collapse the view of any timeout pate (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.ode

Table 17-2 Collapsed and Expanded

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

Writing Custom Java Code in Perform Nodes

Handling Exceptions

Adding Message Paths

Transaction Boundaries

Running and Testing Your Business Process

Oracle Workshop for WebLogic 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 Oracle Workshop for WebLogic 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 **Package Explorer** pane, click the business process (Process.java file) you want to test.Your business process is displayed in the **Design** view.

2. If the **Server** view is not visible in Oracle Workshop for WebLogic menu, choose **Window** > **Show View** > **Other > Server** > **Servers**, and click **Ok**. A **Server** view is displayed.

3. On the **Package Explorer** pane, select and right-click the business process (Process.java file) you want to test.

4. Click **Run As**, and **Run On Server**.

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

6. In the **WebLogic Server v10.3** dialog box, to manually define a server, click **Browse**, and select the samples integration domain directory from the product installation directory available at `BEA_HOME\wlserver_10.3\samples\domains\integration`, where BEA_HOME represents the directory in which you installed Oracle WebLogic Integration. Click **Finish**.

7. The samples domain integration server is started, and the application is deployed on it. When Oracle WebLogic Server is running, the following indicator is visible in the **Servers** view (see Figure 18-1).

Figure 18-1 Server Status

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

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.

The Overview tab displays public information about the web service, including:

- A link to the WSDL that describes the public contract for the web service. Clients of the web service use the WSDL file to determine how to call the web service and what data will be returned. The WSDL describes both methods and callbacks on the web service.

- The callback WSDL for clients that cannot handle the callbacks described in the complete WSDL. Clients wishing to receive callbacks by implementing a callback listening service can communicate with the web service using this WSDL.

- The Java source code for a Oracle Workshop for WebLogic service control for the web service. A developer using Oracle Workshop for WebLogic who wishes to call your web service from their Oracle Workshop for WebLogic application can use this source to construct a Service control.

- The Java proxy for calling the web service from a Java client. A Java client can call your web service by creating a class from the Java proxy and invoking its methods.

- The description of the web service, which lists the web service's methods and callbacks.

- Links to useful information such as specifications for WSDL and SOAP.

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 Oracle Workshop for WebLogic 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, Oracle Workshop for WebLogic will offer to download and install it for you.

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.

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. Click **Test Form** tab.

   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. To upload a file to test data, click **Browse** beside the **xml requestXML (file value)** field to open the file browser, then select the file that contains the test data you want to use.

   Note: An input xml file URI with space is not supported in the Test Form.

   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.

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 or Java Batik SVG Viewer (for more information, see Requirement for the Interactive Graph). The first time you

open the **Process Graph** tab, you will be asked if you would like to accept the **Software License Agreement** for the Adobe Viewer :

This viewer is not available for some configurations that the Oracle WebLogic Integration supports. For details, please see Process Monitoring.

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 SVG Viewer displays the interactive view. The Process Graph Visual cues are provided to indicate node status as described in the following table:

Table 18-1 Mode Details

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
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.

   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.

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.

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:7051/samples/myBusinessProcess.jpd`

In the preceding line:

- `http://localhost:7051/`—Represents the machine name and its default listening port (7051). Specifically, this means that the request (the call to the business process) from your Test browser is intercepted by Oracle WebLogic Server on port number 7051 of your local machine.

- `samples/`— This refers to the Web application of which the service is a part. When you create a project in Oracle Workshop for WebLogic, you are also creating a Oracle 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.jpd`— "myBusinessProcess" is the name of the business process. ".jpd" extension is retained for the URL and it maps to "myBusinessProcess.java". Oracle 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.

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

There are two ways of creating variables for your business process. Variables can be created in the **Data Palette** by clicking **Menu** tab on the **Data Palette** menu and scroll down the list, select **Add a 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 **JPD Configuration** view.

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 Oracle WebLogic Integration 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 Oracle WebLogic Integration schemas project.

Before you can create a variable of a Java class, the Java class file must be available in the Workshop project.

To Create a New Variable in the Data Palette

You can create variables using the **Add a Variable...** option on the **Menu** tab. To learn how to create variables in the node builder, see To Create a New Variable in the Node Builder.

1. If the **Data Palette** pane is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Data Palette** from the menu bar.

   The **Data Palette**, which contains a **Menu** drop-down button, is displayed in the Oracle Workshop for WebLogic.

2. In the **Menu** tab, scroll down the list and click **Add a Variable...**, the **Create Variable** dialog box is displayed, as shown in Figure 19-1.

Figure 19-1 Create Variable
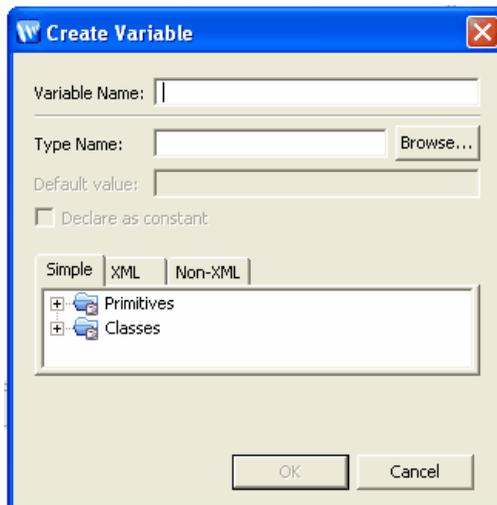
p

A description of the possible variable types options are as follows:

– **Simple Types**

Lists Java primitive and classes data types.

– **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. Oracle 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 Oracle Workshop for WebLogic and is also available in the XML Types listing.

For more detailed descriptions of the data types, see Working with Data Types.

3. From the menu, select the appropriate variable types from the selected option: **Simple, XML**, or **NonXML**.

4. In the **Variable Name** field, enter a name for your variable.

5. In the **Type Name** field, select or in the **Type Name** field, enter a variable type:

– For a Simple class type variable, enter the full package name of the class in the **Type Name** 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 **Type Name** 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 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**.

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 **Properties** view.

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.

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, click the arrow and 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:

   – **Simple Types**

     Lists Java primitive and classes data types

   – **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. Oracle 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 Oracle Workshop for WebLogic and is also available in the XML Types listing.

     For more detailed descriptions of the data types, see Working with Data Types.

   The **Type Name** field is populated with the variable type you selected.

5. Select or enter a variable type:

- For a Simple class type variable in the **Type Name** 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**.

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 **Properties** pane.

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
- Simple 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.

Note: To make the Schemas in your project available in your business process, you must place them in the **schemas** folder. To learn about the Application and project folders in the **Design** view, see Components of Your Application and Designing Your 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, Oracle Workshop for WebLogic 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. In addition when creating a variable or an xquery transformation method, xsd simple types would be mapped to java type names.

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.

The XML untyped data 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.

Note: When creating a schema file supporting two imports with the same `namespace`, the XQuery Mapper ignores the second import.

Non-XML Types

Oracle 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. 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 Oracle Workshop for WebLogic. 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.xsd** is listed in **Non XML Types**, and the corresponding XML Schema representation, **mydata.mfl**, 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.

Note: Non-XML variables are equivalent to *Binary* variables in prior versions of Oracle 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.

Although both XMLObject and RawData are both *untyped* data types in Oracle 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.

RawData type supports the following file type:

- ASCII file types (for example, .html .shtml .php .pl .cgi .txt .css etc).
- XML types
- MFL types.

Simple Types

Contains the following Java data types:

Primitive Data Types—boolean, byte,char, double, float, int, long, and short.

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 to assign** pane in the node builders. You must explicitly specify the Java class in the **Type Name** 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.

Java class variables can be used in business processes without any conversion. When you use Java classes in data transformations, Oracle 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.

Related Topics

Validating Schemas

Variables in Business Process Source Code

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 Oracle Workshop for WebLogic. 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 Oracle 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 Oracle Workshop for WebLogic 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 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, as shown in Figure 19-2.

Figure 19-2 Processing Logic
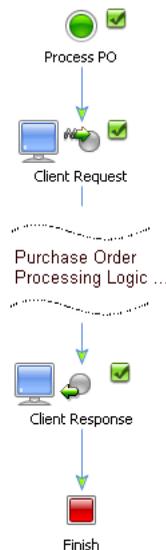
The following steps describe how to design the **Client Request** and **Client Response** nodes to do the MFL-to-XML and XML-to-MFL assignments required for the scenario described in this example.

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 Simple Types are also available. To display the Non-XML and Java data types, select the **NonXML** or **Simple** tab on the panel, see Figure 19-3.

Figure 19-3 Data Types

> **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 Oracle Workshop for WebLogic, an XML Schema representation of your MFL data is also available in the XML Types list.

4. Select **NonXML** to display the Non-XML Types (Typed and Untyped) in your application, as shown in Figure 19-4.

Figure 19-4 Non-XML
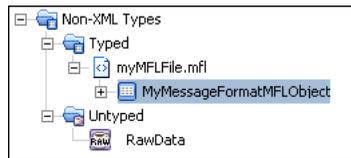
5. Click the + associated with the name of the MFL file that represents the type of the request the client makes to the business process. In this example, we click the + beside **myMFLFile.mfl** to display the root element **MyMessageFormatMflObject**.

6. Click the root element—in this case, **MyMessageFormatMflObject**. The data type is displayed in the **Type** field.

7. 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:

8. 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**:

9. 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. Select the **XML** option.

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**.

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 **Close**, 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 **Close**, 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.

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 Oracle WebLogic Integration MflObject interface.

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:

```
 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 instance of 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

Working with Data Types

Versioning Business Processes

By using the Oracle WebLogic Integration versioning feature in the Oracle Workshop for WebLogic 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 **Package Explorer** pane, right-click the business process (Process.java file) for which you want to create a new version and select **Process Versioning > Create Version**.

   The **Create Version** window opens.

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 Oracle Workshop for WebLogic.

3. Click **OK**.

   The Create Version window closes and the new version of your business process is added to the **Package Explorer** pane (see Figure 21-1).

Figure 21-1 Version
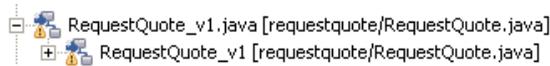
The ![icon] indicates that this version of the business process is the active version of the process. By default, the first version of a process becomes the new version since and the original version becomes a virtual URI which points to the active version of the process. All currently running instances 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 and the Process.java file is not visible.

### To Create a New Version of Your Process

1. On the **Package Explorer** pane, right-click the business process (Process.java file) which you want to create a new version for and select **Process Versioning > Create Version**.

    The **Create New Version File** window opens.

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 Oracle Workshop for WebLogic.

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 **Package Explorer** pane.

    The ![icon] 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 **Package Explorer** pane, right-click the business process (Process.java file) that you want to set to active and select **Process Versioning > Make Active Version**.

   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 **Package Explorer** pane by changing the icon of the process.java file(s) to .

  Note: The error icon will not appear if the business process is not save. Go to File and click Save All.

- 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 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 **Package Explorer** pane, right-click the business process (Process.java file) that you want to delete and select **Process Versioning > Delete Version**.

   The version of the business process you selected is deleted and removed from the **Package Explorer** 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 Process file and the **Package Explorer** pane 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, Oracle 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, Oracle recommends the following design practices:

- Split long-running business processes into multiple subprocesses.

- Specify the version strategy as **loose-coupling** (between business processes and sub processes). 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 Oracle WebLogic Integration Administration Console.

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 **Properties** view, select the **version strategy** method that you want to use for the sub processes 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 (Process.java) from another application, the the versioning relationship is not preserved in the imported business process (Process.java). You need to manually edit the `wlw-config.xml`configuration file, which is located in the `\Web\WEB-INF` folder of your application.

Building Stateless and Stateful Business Processes

Business Processes are either Stateless or Stateful, depending on the transactions 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.

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 view indicates whether a business process is Stateless or Stateful in two different ways:

- The **Stateless** property in the **JPD Configuration** view of your **Start** node.

- The icon of the **Start** node in the **Design** view.

The following table summarizes the ways in which Oracle Workshop for WebLogic indicates if your Business Process is Stateless or Stateful.

Table 22-1 Stateless or Stateful Business Process

	Stateless	Stateful
JPD Configuration Pane	stateless = **true**	stateless = **false**
Start Node Icon		

You can use the **persistence** property in the **JPD Configuration** view to set how a stateful business process is persisted. For more information about the Start node JPD Configuration view, 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

Setting the Business Process Properties

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 Oracle Workshop for WebLogic 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 Oracle 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. 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 **JPD Configuration** view in the Oracle Workshop for WebLogic 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 **JPD Configuration** view is not visible, from the menu bar, choose **Window > Show View > Other > General > JPD Configuration**.

   The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.

To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

2. In the **JPD Configuration** view, 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 (Java file) 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 Oracle WebLogic Server documentation.

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 Figure 23-1.

Figure 23-1 Asynchronous Subprocesses
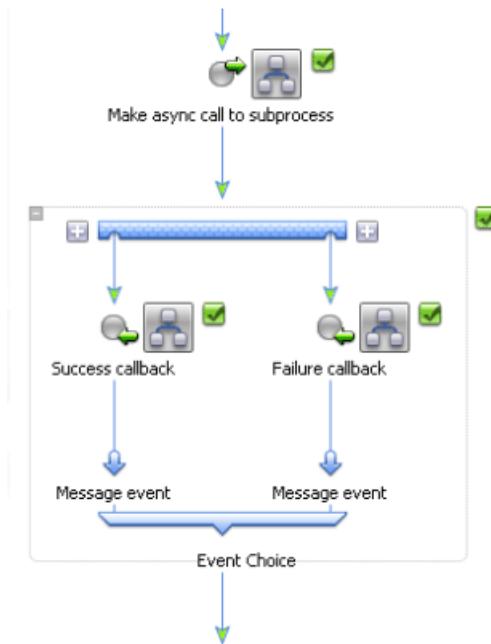
Synchronous Clients for Asynchronous Business Processes

You can enable synchronous clients to interact with business processes that have asynchronous interactions with resources. For example, a synchronous Oracle Workshop for WebLogic 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.Figure 23-2 demonstrates this scenario.

Figure 23-2 Synchronous Clients for Asynchronous Business Processes
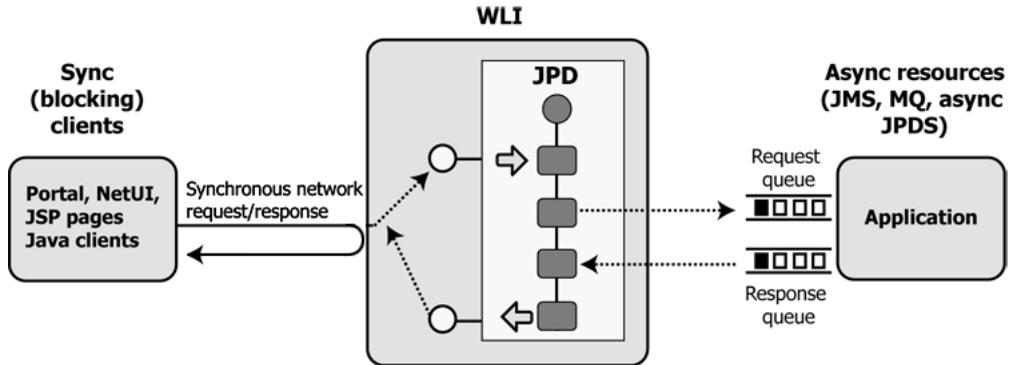
While it is not possible to create a synchronous process that requires asynchronous operations within a **Client Request with Return** node, as shown in the following figure (and indicated by ⊗ on the **Control Receive** node), you can create an asynchronous process with **Client Request** and **Client 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 Oracle WebLogic Server `clientgen` utility or with a Oracle Workshop for WebLogic entity that uses the Service Broker control. See Figure 23-3.
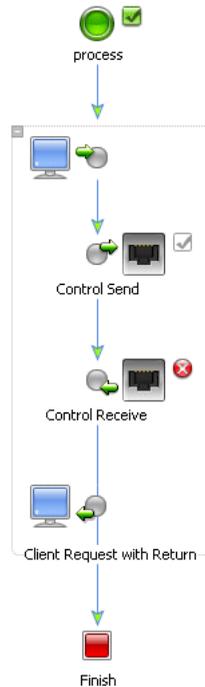
Figure 23-3 Asynchronous Operations

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.jpd"/>
   </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.

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 **Node Palette**, drag-and-drop the following nodes between the **Client Request** and **Finish** nodes:

   Note: If the **Node Palette** is not visible in Oracle Workshop for WebLogic, from the Oracle Workshop for WebLogic menu, choose **Window > Show View > Node Palette**.

   a. **Control Send**

   b. **Control Receive**

   c. **Client Response**

   The business process should now resemble the business process on the left side as shown in Figure 23-4.

Figure 23-4 Synchronous Client for an Asynchronous Processes

Completed Version
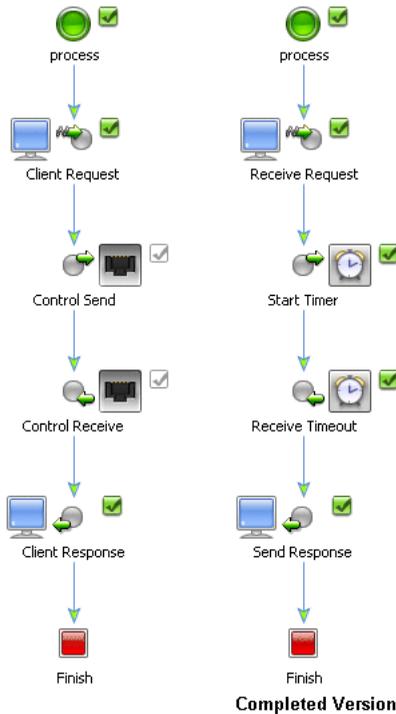
3. Configure each node as required by your design.

4. In the **Design** view, click the **Client Request** node.

5. In the **JPD Configuration** pane, do the following:

    – 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.

    – Change the **jmsSoap** property field value to true, the default value is false.

    Note: If the **JPD Configuration** view is not visible, from the menu bar, choose **Window > Show View > Other > General > JPD Configuration**.

    The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.

To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

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.

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 **Package Explorer** pane, right-right click the business process (Process.java file) for which you want to generate the WSDL file.

2. From the drop-down menu, choose **Generate > Sync/Async WSDL**.

   Oracle Workshop for WebLogic generates the WSDL file and displays it in the **Package Explorer** pane directly below the Process.java file. If the name of the Java file is `HelloWorld.java`, 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.

Figure 23-5 Synchronous and Asynchronous Pairs
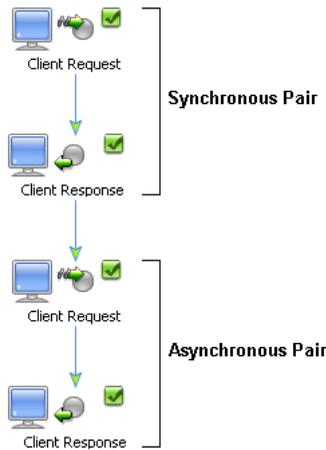
A Client Request within a Synchronous Pair is Not Allowed

You cannot place a Client Request node inside of a synchronous pair.

Figure 23-6 Synchronous pair
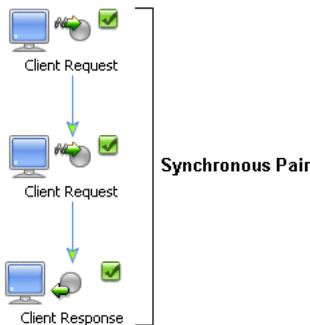
A Synchronous Client Cannot Call an Asynchronous Process with SOAP Attachment in a Client Request Node

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 Oracle 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 Configure Basic Authentication for Resources Accessed via SyncAsyncServlet

The SyncAsyncTransportServlet is packaged within Oracle WebLogic Integration System EJBs. The deployment descriptor (`web.xml`) for this servlet is contained in the `jpd-ejbs.ear` file. This file is located in the `BEA_HOME\wli_10.3\lib` directory, where `BEA_HOME` is the directory in which you installed the Oracle WebLogic Integration. The `web.xml` is located in the `transport/http/WEB-INF` directory in the `jpd-ejbs.ear` file. 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.

Related Topics

Transaction Boundaries

Starting Your Business Process

Building Stateless and Stateful Business Processes

Handling Exceptions

Transaction Boundaries

Business processes in Oracle 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 Oracle Workshop for WebLogic 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 Oracle Workshop for WebLogic.

This following sections deal specifically with transactions in the context of Oracle WebLogic Integration and business processes:

- Implicit Transaction Boundary Rules

- An Implicit Transaction Boundary Example

- Explicit Transaction Boundaries

- Handling Exceptions in Transaction Blocks

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 a Parallel node in your business process, then, in the **JPD Configuration** pane, 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 Oracle WebLogic Integration business processes.

1. Start with an empty business process, as shown in Figure 24-1.

Figure 24-1 Empty Business Process
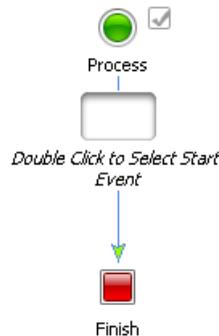
2. If we configure the Starting Event to be a Client Receive node, the following transaction boundaries are implied, as shown in Figure 24-2.

Figure 24-2 Transaction for Client Receive Node
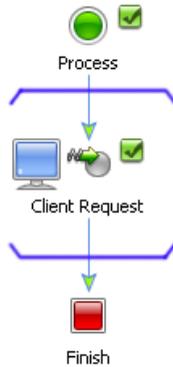
3. When we add a Control Send node, the implied transaction boundary extends to include the new node (see Figure 24-3).

Figure 24-3 Implied Transaction Boundary
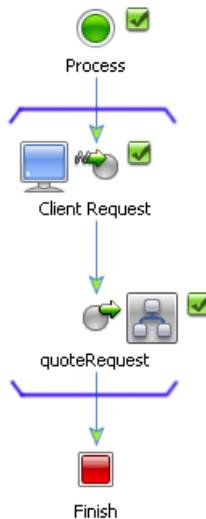
4. By adding a Control Receive node, we add a blocking element to the business process and therefore create a new transaction (see Figure 24-4).

Figure 24-4 New Transaction Boundaries
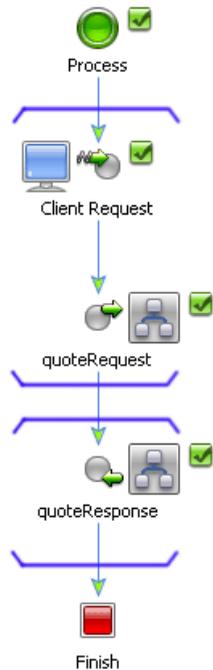
Since the business process now contains two transactions, the Start node icon changes to indicate that the business has changed from Stateless ⬤ to Stateful ⬤. For more information about Stateless and Stateful business processes, see Building Stateless and Stateful Business Processes.

5. If we add a Client Response node to the business process, the second transaction's boundaries expands to include the new node as shown in Figure 24-5.

Figure 24-5 Client Response Node Transaction node
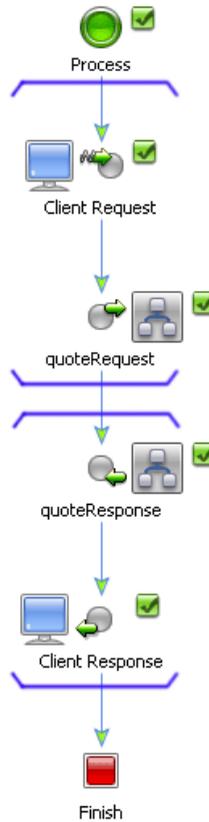
This concludes the implicit boundaries example, you can also create transactions by adding explicit transaction boundaries to your business process. For information about how to do this, see Explicit Transaction Boundaries.

Explicit Transaction Boundaries

Recall that you define explicit transaction boundaries and that they are visible in the business process diagram in Oracle Workshop for WebLogic.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, Oracle Workshop for WebLogic will display a message about the violation.

To invoke a JWS using JMS transport using request/response paradigm the caller must not be within a transaction. When business process is the caller, user must explicitly suspend the current business process transaction before calling the service control that sends the message to the JWS and resume the transaction after the invocation. The implications of suspending and resuming a transaction is as follow:

```
Transaction savedTxn =

TransactionHelper.getTransactionHelper().getTransactionManager().forceSusp
end();

//call service control

TransactionHelper.getTransactionHelper().getTransactionManager().forceResu
me(savedTxn);
```

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 as shown in Figure 24-6.

Figure 24-6 Explicit Transaction Boundaries
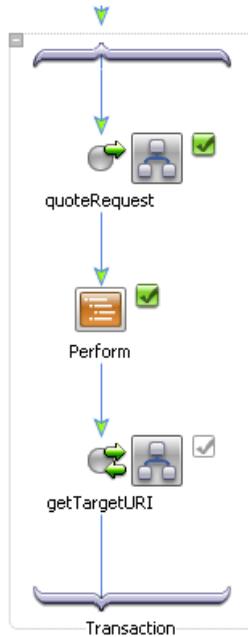
You can rename your transaction block by right-clicking **Transaction** and selecting **Rename** from the drop-down menu.

To Create an Explicit Transaction—Alternative 2

1. In the **Design** view, drag and drop ⬜ **Transaction** from the **Node 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 **Node 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 **Properties** view.

To Set the Transaction Properties

1. Select the transaction for which you want to set the properties.

   The related properties are displayed in the **JPD Configuration** view. If the **JPD Configuration** view is not visible in Oracle Workshop for WebLogic, choose **Window > Show View > Other > General > JPD Configuration** from the Oracle Workshop for WebLogic menu bar.

   Note: The default Process view displays only the Properties pane. The JPD Configuration view and the Properties pane can not be used together in the Process perspective.

   To modify values in the JPD Configuration pane, you need to first close the Properties pane and then open the JPD Configuration pane and reload the JPD. To return to the Properties pane, close the JPD Configuration pane, open the Properties pane and reload the JPD.

2. In the **JPD Configuration** pane, set the following properties:

general

   – **name**—Enter the name you want displayed in the Oracle Workshop for WebLogic for this transaction.

   – **notes**—Enter any notes you want associated with this transaction.

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

Building Stateless and Stateful Business Processes

Transaction Boundaries

Business Process Source Code

As you design business processes using the graphical tools in Oracle Workshop for WebLogic, it writes source code to a business process file (Process. java file), in keeping with your work in the **Design** view. This Process.java contains annotations and the implementation code intended specifically for a business process.

You can access the source code for business processes you are creating in the **Design** view, by clicking the **Source** tab.

This section describes the source code in a business process (Java) 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 Java 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 their own collapsible regions in the Java file in the **Source** view.

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 Oracle Workshop for WebLogic 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 Oracle Workshop for WebLogic 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, choose **Window > Show View > Properties**.

The Process 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 Java 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 Process.java 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, Oracle Workshop for WebLogic 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 Java 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: Oracle Workshop for WebLogic flags errors you make in the process language with red, wavy underlines and error messages visible in mouseover text.

Variables

Business process variables are defined within the region of code in the **Source** view.

Two-way editing is supported for variables. In other words, changes you make to the code in this region of the Java file appear in the **Design** view, specifically the **Variables** section 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.

Control Declarations

Declarations for controls are defined in the region of code in the **Source** view.

Two-way editing is supported for control declarations. In other words, changes you make to the code in this region of the Process.java 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.

Can You Edit Code?

In the **Source** view, you can add and edit the code within the blocks of code that specify client operations and control methods.

After you add or edit the code, 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.

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 Figure 25-1.

Figure 25-1 Client Receive

Client Requests Quote

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.

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:

Client Requests Quote   to   Client Requests Quote

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 Java 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 Java file in the region of code in the **Source** view.

The XQuery statements are preceded by the following annotation:

```
@com.bea.wli.common.XQuery
```

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 Java file. The expression returns the set of XML elements over which the **For Each** node iterates. XQuery expressions are written in your Process.java file when you create conditions on **Decision** nodes and it also define the transformations you create between disparate data types using the mapping tool.

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

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.

This topic describes the template that you can use to build an ebXML participant business process in Oracle Workshop for WebLogic. It contains the following sections:

- About the ebXML Participant Business Process
- Creating an ebXML Participant Business Process
- Customizing an ebXML Participant Business Process

About the ebXML Participant Business Process

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:

Table 26-1 ebXML Nodes

Node Name	Node Type	Method Name	Description
Start	Start		To learn about **Start** nodes, see Starting Your Business Process.
Receive Request	Client Request	`request`	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	`response`	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.

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.

1. From the Oracle Workshop for WebLogic menu, choose **File > New > Other. The Select a wizard** dialog box is displayed.

2. Expand WebLogic Integration and select **ebXml Participant Process** and click **Next**.

3. In the **Name** field, enter the name of the Java file.

4. If you want to create the Java file in a directory other than the one displayed in the **Source folder** field, then click the **Browse** button and select the target directory and click **Ok**.

5. Click **Finish**,

   Oracle Workshop for WebLogic creates a new ebXML participant process Java file and displays it in the **Design** view pane.

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

   After you create the process.java file, the name of the Java file becomes available as a service on the Services tab in the Oracle WebLogic Integration Administration Console.

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 Annotations (Required)

- Customizing Names and Argument Types (Optional)

- Retrieving the ebXML Message Envelope (Optional)

Depending on your implementation requirements, you might make additional customization 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 Annotations (Required)

The generated ebXML participant business process file specifies the following default annotations:

```
ebxmlActionMode="non-default,ebxmlServiceName="eBXML"
"protocol-name="ebxml"
```

These properties are set in the **Properties** view that is visible when you have the Start node of your business process selected. Review and edit (if needed) the following annotations:

Table 26-2 ebXML Annotations

Property	Default	Description
ebxmlActionMo de	non-default	Action mode for this business process. Determines the value specified in the eb:Action 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: • default—Sets the eb:Action element to SendMessage (default name). • non-default—Sets the eb:Action 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 non-default is recommended to ensure recovery and high availability. If unspecified, the ebxml-action-mode is set to non-default.
service-name	*serviceName*	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 ebxml-service-name annotation on the ebXML control in the initiator business process). You provide this service name to your trading partners.
protocolName	ebxml	Do not change.

> **Note:**   If the **Properties** view is not visible in the **Design** view, choose **Window > Show View > Properties** from the Oracle Workshop for WebLogic menu.

To learn more, see ebXML annotations. 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 **Node 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:

Table 26-3 Data Type

Data Type	Description
XmlObject	Default. Represents data in untyped XML format. The XML data is not specified at design time.
XmlObject[]	An array containing one or more XmlObject elements.
RawData	Represents any non-XML structured or unstructured data for which no MFL file (and therefore no known schema) exists.
RawData[]	An array containing one or more RawData elements.
MessageAttachment[]	Array containing one or more parts of an ebXML 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.

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:

```
/**
*com.bea.wli.jpd.EbXMLMethod 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.

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 Oracle Workshop for WebLogic.

The following sections are included:

● About the RosettaNet Participant Business Process

● 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.

About the RosettaNet Participant Business Process

The RosettaNet participant business process 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:

Table 27-1 RosettaNet Nodes

Example Node Name	Example Node Types	Description
Start	Start	This marks the beginning of your business process. In the **Annotation** pane of the Start node, you can define the following properties: • pipName • pipRole • pipVersion • protocolName • protocolVersion To learn more about these annotation, see Configuring Business Process Annotations (Required). To learn about Start nodes, see Starting Your Business Process.
On Error Message (global error handling)	Message Path	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.
On Error (Global message path)	Client Request	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

Example Node Name	Example Node Types	Description
Alert local administrator (Global message path)	Perform	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
Receive Message	Client Request	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.
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.

Example Node Name	Example Node Types	Description
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.
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

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.

2. From the Oracle Workshop for WebLogic menu, choose **File > New > Other**. The **Select a wizard** dialog box is displayed.

3. Expand WebLogic Integration and select **RosettaNet Participant Process** and click **Next**.

4. In the **Process** dialog box enter a valid java class name for the Java file in the **Name** field.

   Note: This name is used as the default value for the `pip-name` attribute in the `com.bea.wli.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 Java file in a directory other than the one displayed in **Source folder**, then click the **Browse** button and select the target directory and click **Ok**.

6. Click **Finish**.

   Oracle Workshop for WebLogic creates a new RosettaNet participant process Java file and displays it in the **Design** view pane.

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

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 Annotations (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 Annotations (Required)

The RosettaNet participant business process file specifies the following default annotations:

```
com.bea.wli.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 **Properties** view that is visible when you have the Start node of your business process selected. Review and edit (if needed) the following properties:

Table 27-2 Business Process Annotations

Property	Default	Description
pipName	*processName*	Change to the RosettaNet PIP code, such as 3B2. Must be a valid PIP code as defined in http://www.rosettanet.org.
pipRole	Seller	Change to the RosettaNet name of the participant's role as defined in the PIP specification (example: Receiver for PIP3B2).
pipVersion	1.0	Change to your RosettaNet PIP version (example: v01.01 for PIP3B2). Must be a valid version number associated with the PIP.
protocolName	rosettanet	Do not change.
protocolVersion	2.0	Change to 1.1 if you are using RNIF (RosettaNet Implementation Framework) version 1.1 instead.

> **Note:** If the **Properties** view is not visible in **Design** view, choose **Window > Show View > Properties** from the Oracle Workshop for WebLogic menu.

To learn more, see Annotations. 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:

Table 27-3 Data Description

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.

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.

- 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 Transforming Data Using XQuery Mapper.

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. Oracle Workshop for WebLogic 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.

- **Perform** nodes for non-Oracle 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 Oracle Workshop for WebLogic, 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 Oracle Workshop for WebLogic, you can use a **Perform** node instead. To learn more about **Perform** nodes, see Writing Custom Java Code in Perform Nodes.

Building and Deploying Integration Applications

This topics includes the following sections:

- Working in Iterative Development Mode
- Deploying Applications in EAR Format

Working in Iterative Development Mode

If you start Oracle WebLogic Server when you work in iterative development mode in Oracle Workshop for WebLogic, your application is, by default, deployed on the server as an exploded directory. Changes to those source files are dynamically noted by the server and affect the running process.

The Oracle WebLogic Integration Administration Console allows you to monitor and manage the entities and resources for your Oracle WebLogic Integration applications.

Note: If you delete a business process (Process.java file) from your application, the data associated with instances of that process remain in the run-time database after the application is redeployed. You can purge the data for completed and terminated process instances using the Oracle WebLogic Integration Administration Console. However, before the data can be purged from the database, any instances of the deleted process that did not run to completion must be terminated. Terminate the instances of a business process in one of the following pages in the administration console: **Process Instance Detail** page, **Process Instance Summary** page. To learn how, see "Suspending, Resuming, Terminating, or Unfreezing Process Instances" in Process Instance Monitoring in *Using the Oracle WebLogic Integration Administration Console.*

If the purge process is scheduled to run regularly, tracking data, which includes process history, task history, and trading partner integration message history, is purged from the run-time database according to the specified schedule. You can also run the purge process manually from the **System Configuration** module in the administration console. To learn how, see System Configuration in *Using the Oracle WebLogic Integration Administration Console.*

Deploying Applications in EAR Format

You can also deploy your applications in enterprise application archive (EAR) format. To do so, you must build an EAR file for your application and deploy it to Oracle WebLogic Server. To learn how to do so for applications that you create in Oracle Workshop for WebLogic, see Deploying WLI Applications.

Related Topics

Deploying Oracle WebLogic Integration Solutions

Calling Business Processes

How you call a business process from another application depends on your business requirements, including whether the client is in the same application as the business process it calls, and whether the client is a Oracle Workshop for WebLogic component, such as a Web service (JWS), business process (JPD), or pageflow (JPF). To learn about invoking business processes, see the following topics:

- How Do I: Call Business Processes?

- How Do I: Use a JPD Proxy to a Call Business Process?

How Do I: Call Business Processes?

Business Processes can expose their functionality to clients in several ways, including through WSDL files, Process Controls, Service Broker Controls, and Business Process (Process.java) Proxies.

You can only use Process controls and Service Broker controls between Oracle Workshop for WebLogic components: Web services (JWS), business processes (Process.java), or pageflows (JPF).

The Process control allows a Web service, business process, or pageflow to send requests to, and receive callbacks from, a business process. Process control invocations are Java Remote Method Invocation (RMI) calls. The target business process must be hosted on the same Oracle WebLogic Server domain as the caller. The Process control is typically used to call a subprocess from a parent business process. Transaction contexts are propagated from the parent processes to

the subprocesses over the Process control calls. In other words, the target business process runs in the same transaction as the caller.

The Service Broker control allows a business process or 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; it can be a business process, Web service, or remote (non-Workshop) Web service. Because the transport used is HTTP or JMS, the transaction contexts are not propagated over the Service Broker control calls. Typically, the Service Broker control calls are to remote services.

To call business processes from non-Workshop clients, use a JPD Proxy. You can use a JPD Proxy to communicate with a business process from any Java code. When you invoke a business process using a JPD Proxy, the calls are Java RMI calls. Transaction contexts are propagated from the client to the business process over the JPD Proxy calls. In other words, if the client has a transaction context, the target business process runs in the same transaction as the client. JPD Proxies are typically used by non-Workshop J2EE clients or standalone Java clients to invoke business processes.

Depending on the nature of the client and where it is with respect to the target business processes, clients call the business processes in different ways. The following sections describe the scenarios:

- Workshop Client Invokes a Business Process in a Different Domain

- Workshop Client Invokes a Business Process in the Same Workshop Application

- Workshop Client Invokes a Business Process in a Different Workshop Application, in the Same Domain

- A Non-Workshop Java Client (EJB, servlet or JSP) Invokes a Business Process

Workshop Client Invokes a Business Process in a Different Domain

If the client is a Workshop client (a Web service, a business process, or a pageflow) and the target business process is in a different domain than the client, use a Service Broker control. In this case, create a Service Broker control from the target business process and call the business process using that control. To learn how to create and use Service Broker controls, see Service Broker Control.

You can also use a JPD Proxy in this case. To learn how, see How Do I: Use a JPD Proxy to a Call Business Process?

Workshop Client Invokes a Business Process in the Same Workshop Application

If the client is a Workshop client (a Web service, a business process, or a pageflow) and the target business process is in the same Oracle Workshop for WebLogic application, we recommend that you use a Process control. That is, create a Process control from the target business process and call the business process using that control. To learn how to create and use Process controls, see Process Control.

You can also use a Service Broker control in this case. To learn how, see Service Broker Control.

Workshop Client Invokes a Business Process in a Different Workshop Application, in the Same Domain

- If the client is a Workshop business process or a pageflow and the target business process is in another Oracle Workshop for WebLogic application in the same domain, we recommend that you use a Process control. You can also use a Service Broker control in this scenario.

- If the client is a Workshop Web service and the target business process is in another Oracle Workshop for WebLogic application in the same domain, you must use a Service Broker control.

A Non-Workshop Java Client (EJB, servlet or JSP) Invokes a Business Process

If the client is a standalone Java program, a non-workshop J2EE client (EJB, servlet, or JSP), use a JPD Proxy to call the target business process. To learn how, see How Do I: Use a JPD Proxy to a Call Business Process?

Because JPD Proxy calls are RMI calls, the client and the target business process must be in the same organization.

Warning: Business processes that include client callbacks send those callbacks to the client that started the process. JPD Proxies cannot receive callbacks from business processes. An error will occur in your business process if a client uses a JPD Proxy to start a business process that includes client callbacks; the business process fails at run time when it tries to send the callback to the client (the JPD Proxy) that started it.

How Do I: Use a JPD Proxy to a Call Business Process?

You can use a JPD Proxy to call any business process (synchronous and asynchronous, stateful, and stateless) from any Java client, including standalone Java applications, EJBs, JSPs and Servlets. Using a Java proxy for a business process requires different steps depending on whether the client application that uses the proxy is in the same JVM as the target business process.

This topic includes the following sections:

- What is a JPD Proxy?
- How Do I: Get a JPD Proxy for a Business Process?
- How Do I: Use a JPD Proxy From a Java Client?
- How Do I: Use a JPD Proxy From a JSP?
- How Do I: Use a JPD Proxy From an EJB?

What is a JPD Proxy?

A JPD Proxy is an RMI client to a business process (JPD). An interface that matches a business process' client requests is associated with each business process. This interface is called the *JPD public contract*. Each method on the JPD public contract has the same signature as the corresponding client request. A JPD Proxy is a JAR that contains the compiled class file for the JPD contract. You can use the class file to access the JPD as though it were a local Java class. JPD Proxy calls are over Java RMI. JPD Proxy calls propagate the transaction context from the clients to the business processes.

You can download the JPD Proxy JAR file from the **JPD Proxy** link on the Oracle Workshop for WebLogic Test Browser **Overview** page (see How Do I: Get a JPD Proxy for a Business Process?).

Warning: Business processes that include client callbacks send those callbacks to the client that started the process. JPD Proxies cannot receive callbacks from business processes. An error will occur in your business process if a client uses a JPD Proxy to start a business process that includes client callbacks; the business process fails at run time when it tries to send the callback to the client (the JPD Proxy) that started it.

The `JpdProxy` class is a factory class for proxies to a Oracle WebLogic Integration business process type. Clients call one of the `create()` methods on the class to get a proxy instance. The `create()` methods take the JPD contract class (`java.lang.Class`) as input.

An example JPD contract interface for a business process named `RequestQuote.java` is shown in the following listing.

Listing 29-1   Example JPD Contract Interface

```
package weblogic.wli.jpdproxy;

import org.example.request.QuoteRequestDocument;

public interface RequestQuote {
    public void quoteRequest(org.example.request.QuoteRequestDocument
requestXML);
    public static final String SERVICE_URI =
    "/myApplication/requestquote/RequestQuote.jpd";
}
```

Note the following characteristics in the preceding example contract interface:

- The class name of the interface matches the JPD class name (in this case, you download a JAR file named `RequestQuoteProxy.jar` that contains a class file named `RequestQuote.class`).

- One method is available: `public void quoteRequest(org.example.request.QuoteRequestDocument requestXML)`.

  Note: When you write your client application, you can determine which client request methods are available for you to use through the JPD Proxy by reviewing the source code for the business process. To do so, ensure that the business process (JPD) is open in the Oracle Workshop for WebLogic graphical design environment, identify the Client Request calls in the **Design** view, then open the **Source** view to view the method names and signatures.

- The JPD contract references a strongly typed XML argument: `requestXML`, which is of type `QuoteRequestDocument`.

- The JPD contract interface includes a `SERVICE_URI` static final field. The String value of the `SERVICE_URI` field is the URI for the business process at the time the JPD Proxy is downloaded from the Oracle Workshop for WebLogic test browser. The client can pass this constant to the `create` method, or can pass a different value.

  A different value for `SERVICE_URI` is required if the business process (JPD) is deployed to a different location after the JPD Proxy JAR was generated. For example, you can create the JPD Proxy from the business process while the process is deployed in a development environment. Subsequently, the business process can be moved to a different location for production. Therefore, the business process is accessible through a different URI; clients must pass the new URI value to the create method.

How Do I: Get a JPD Proxy for a Business Process?

1. Open your business process in Oracle Workshop for WebLogic.

2. On the menu bar, click Run, to run the business process.

   If the Oracle WebLogic Server is not running, a window is displayed asking if you want to start your server. To start the server, click **OK**.

   Note: To learn about generating JPD Proxies for business processes that are versioned, see About Versioned Business Processes.

3. After the **Workshop Test Browser** appears, click the **Overview** tab.

4. On the **Overview** page, under the **Process Clients** section, click **JPD Proxy**. You are prompted to save the file to disk

   Note: By default, the package is `weblogic.wli.jpdproxy`. If you want to specify a different package for the generated JPD Proxy, enter a package name in the **Java package** field associated with the **JPD Proxy** button.

5. Save the file to your disk according to how you want to use the proxy:

   – **To Use the JPD Proxy From a Oracle Workshop for WebLogic Application (an EJB, JSP, or Servlet)**

   Save the JAR file to one of the following directories:

   `WEB-INF/lib`—Save the JAR to the `WEB-INF/lib` directory of the Web application from which you want to use the proxy (the client application). In the Oracle Workshop for WebLogic graphical design environment, the JAR file is displayed in the `WEB-INF/lib` folder on the **Package Explorer** pane.

   `APP-INF/lib`—If you want to use the JPD Proxy JAR from more than one project in your (client) application, save the JAR to the `APP-INF/lib` directory at the root of your application. In the Oracle Workshop for WebLogic graphical design environment, the JAR file is displayed in the `Libraries` folder at the root of your application in the **Package Explorer** pane.

   – **To Use the JPD Proxy From a Standalone Java Application**

   If you are using the JPD Proxy from a standalone Java client (outside of Oracle WebLogic Server), save the JAR to any location that is convenient for your client Java application and add the JAR to the client's `CLASSPATH` environment variable.

   Note: The default name of the JAR file is `<business-process-name>Proxy.jar`, where `business-process-name` represents the name of the business process for which

you are generating the JPD Proxy. Accept the default name unless it conflicts with an existing JAR file.

6. If you plan to use the JPD Proxy from an application running in a different JVM to that in which the target business process is running, append the following JAR files to the client's `CLASSPATH` environment variable:

   – **<business-process-name>Proxy.jar**—The JPD Proxy you downloaded from the Oracle Workshop for WebLogic Test Browser (where *business-process-name* represents the name of the business process for which you generated the JPD Proxy).

   – **jpdproxy_client.jar**—A support JAR that contains business process-independent client-side classes. It is located in the following directory in your Oracle WebLogic installation:

   BEA_HOME\wli_10.3\lib

   In the preceding line, *BEA_HOME* represents the location where you installed Oracle WebLogic Integration.

   This JAR contains an abstract proxy-factory class called `JpdProxy`, a proxy implementation `JpdProxyImpl`, and other client-side run-time classes.

   – **Schemas.jar**—If the JPD Proxy (*<business-process-name>*Proxy.jar) file you downloaded contains references to strongly typed XML or MFL arguments, add the *Schemas*.jar file to the classpath. (*Schemas* represents the name you gave to the Schemas project in your application.) The *Schemas*.jar file is available in APP-INF\lib at the root of your application.

   – **weblogic.jar**—This file is available in the following location in your installation: BEA_HOME\wlserver_10.3\server\lib.

   – **wlcipher.jar**—If you are using a client with two-way SSL, add this JAR to the CLASSPATH. The wlcipher.jar is available in the following location in your installation: BEA_HOME\wlserver_10.3\server\lib.

About Versioned Business Processes

If the target business process is versioned, you can run the active version of the process to invoke the Test Browser (in this case, the Test Browser is opened on the virtual URI) or you can run any other version of the process (in which case the Test Browser is opened on a specific physical URI). To learn about creating versions of business processes, see Versioning Business Processes.

If you subsequently download a JPD Proxy from the Test Browser, its JPD contract interface matches the virtual JPD or the physical JPD, accordingly. When you create a Java client, you pass the JPD contract and a service URI to the proxy factory method. In most cases the JPD contract

interfaces for all versions of a business process are identical, but a specific version of a business process can extend the public interface of the original process. In this case, you must ensure that the service URI and JPD contract interface passed to the proxy factory method are consistent.

How Do I: Use a JPD Proxy From a Java Client?

This section uses example code to describe how to use a JPD Proxy from a Java client. It includes the following topics:

- To Use a JPD Proxy From a Java Client
- To Use a JPD Proxy From a Java Client With Two-Way SSL

To Use a JPD Proxy From a Java Client

This topic describes how to use a JPD Proxy from a Java client. The code listing in Listing 29-2 is an example of a Java client that invokes a business process using a JPD Proxy. This example includes basic username-password authentication. A second example (Listing 29-5) describes how to add two-way SSL to the Java client.

You obtain the JPD Proxy JAR for the business process by first running the purchase order business process in Oracle Workshop for WebLogic to invoke the Test Browser. Then select the **JPD Proxy** link on the **Overview** page of the Test Browser.

The following sections reference the code example in Example Java Client to describe how a JPD Proxy Client is used from a Java client:

- Example Java Client
- To Import the Proxy Classes
- To Use the Proxy Factory (JpdProxy.create()) Method
- To Call the Methods on the Target Business Process
- About Strongly-Typed XML or MFL Arguments in Business Processes
- About Conversation Management
- To Run the Java Client
- Limitation Using JPD Proxies for Business Processes That Include Client Callbacks

Example Java Client

The code listing in Listing 29-2 is an example of a Java client that invokes a business process using a JPD Proxy. It invokes a business process named `PoRequest.java`. This example includes basic username-password authentication. A second example (Listing 29-5) describes how to add two-way SSL to the Java client.

Listing 29-2   Example Java Client

```
package your.package;

// Proxy classes are located in the com.bea.wli.bpm.proxy package.
import com.bea.wli.bpm.proxy.JpdProxy;
import com.bea.wli.bpm.proxy.JpdProxySession;

import weblogic.wli.jpdproxy.PoProcess;

/**
* Import any packages required for your application. For example, if the business
* process uses XML Beans, you must import the appropriate packages.
*/
import requisitionpo.www.purchase.PurchaseDocument;
import requisitionpo.www.purchaserequestreq.PurchaseRequestReqDocument

import javax.naming.Context;
import javax.naming.NamingException;
import javax.naming.InitialContext;
import weblogic.jndi.Environment;
import java.io.*;

public class startPoProcess
{
    public static void main(String[] args)
    {
        try
        {
            PoProcess p = (PoProcess)
                JpdProxy.create(
                    PoProcess.class,
                    PoProcess.SERVICE_URI,

                     new JpdProxy.ContextHandler()
                    {
                        public Context getContext() throws NamingException
                        {
                            Environment env = new Environment();
                            env.setProviderUrl("t3://localhost:7001");
```

```
                    env.setSecurityPrincipal("weblogic");
                    env.setSecurityCredentials("weblogic");
                    return env.getInitialContext();
            }
        });
    PoDocument document = PoDocument.Factory.newInstance();
    Po po = document.addNewPo();
    po.setSku("abc");

    PoReferenceDocument ref = p.processPO(document);
    p.done();
    }
    catch (Exception e) { ... }
  }
}
```

To Import the Proxy Classes

Note that the following packages are imported in our example Java client:

import com.bea.wli.bpm.proxy.JpdProxy;

import com.bea.wli.bpm.proxy.JpdProxySession;

Proxy classes are located in the com.bea.wli.bpm.proxy package. Clients can typecast proxies returned by JpdProxy.create() to JpdProxySession to set and get the conversation ID that is used when a business process is invoked. To learn about setting and getting conversation IDs, see About Conversation Management.

To Use the Proxy Factory (JpdProxy.create()) Method

The proxy factory method (JpdProxy.create()) provides two signatures: one to use when the client is running in the same Oracle WebLogic Server domain as the target business process (JPD), the other to use when the client is running in a different domain than the target business process:

- Method Detail for the create() Method—Use When the Client is Running in the Same Oracle WebLogic Server Domain as the Target JPD

- Method Detail for the create() Method—Use When the Client is Running in a Different Domain Than the Target JPD

Method Detail for the create() Method—Use When the Client is Running in the Same Oracle WebLogic Server Domain as the Target JPD

The `JpdProxy.create()` method creates a client proxy for a business process (JPD). `JpdProxy.create()` accepts the public-contract interface that describes the methods of the JPD as input. The result of this call can be typecast to the public contract class. A service URI uniquely identifies the business process (JPD) on the server.

Use the following method when the client is running on the same WLS domain as the target JPD.

Listing 29-3 JpdProxy.create()

```
public static final Object create(Class publicContract, String serviceUri)
                          throws JpdProxyException
```

In the preceding code listing:

- *publicContract* specifies the public-contract interface of the JPD.

- *serviceUri* specifies the URI of the JPD.

Note: In most cases, the public-contract interfaces for all versions of a business process are identical, but a specific version of a business process can extend the public interface of the original process. In this case, you must ensure that the service URI and JPD contract interface passed to the proxy factory method are consistent. To learn about generating JPD Proxies for versioned business processes, see About Versioned Business Processes.

- The method returns a proxy object that can be cast to the public-contract interface.

- The method throws the `JpdProxyException`, which wraps the checked exceptions that are thrown during construction of the proxy.

Method Detail for the create() Method—Use When the Client is Running in a Different Domain Than the Target JPD

This method signature is shown in Listing 29-4, and is used in the example code in Listing 29-2.

The `JpdProxy.create()` method creates a client proxy for a business process (JPD). `JpdProxy.create()` accepts, as input, the public contract interface that describes the methods of the business process. The result of this call can be typecast to the public-contract class. A service URI uniquely identifies the business process on the server. For the case in which your client is running in a different domain than the target JPD, the `JpdProxy.ContextHandler` is

invoked by the proxy to obtain the JNDI context used to login to the server and lookup server-side resources.

If you use the version of `JpdProxy.create()` that does not take a `ContextHandler`, the client's JNDI context is used to look up the `ProxyDispatcher` EJB.

You need a `ContextHandler` in the following scenarios:

- When the client is running in Oracle WebLogic Server but on a different domain than the target business process.

- When the client is a standalone Java application.

- When the client is running in the same Oracle WebLogic Server as the target business process, but the credentials of the client are not appropriate. (For example, the client may be running as *anonymous*, and the JPD Proxy dispatcher bean or business process requires a different set of credentials.)

Note: The `ProxyDispatcher` EJB is a Oracle WebLogic Integration system stateless session bean that handles incoming requests from JPD Proxies. Its scope is the Oracle WebLogic Server domain. `ProxyDispatcher` is targeted to all managed servers in a cluster. Administrators can set authentication and authorization policies on this EJB using the Oracle WebLogic Server Administration Console. Oracle recommends using the Java Authentication and Authorization Service (JAAS) rather than JNDI to associate a User with a security context. To learn more, see the following Oracle WebLogic Server documentation:

Oracle WebLogic JNDI

Using JAAS Authentication in Java Clients

The JpdProxy implementation does not explicitly authenticate to the server. Instead, it relies on JNDI authentication when it looks up the `ProxyDispatcherHome` with the JNDI context returned by the ContextHandler.

Use the following method when you need a `ContextHandler`:

Listing 29-4 JpdProxy.create()

```
public static final Object create(Class publicContract, String serviceUri,
JpdProxy.ContextHandler ch) throws JpdProxyException
```

In the preceding code listing:

- *publicContract* specifies the public contract interface of the JPD.

- *serviceUri* specifies the URI of the JPD.

- *ch* specifies a context handler. Clients pass an instance of this JpdProxy.ContextHandler interface to the create method and the proxy implementation uses this instance at run time to allocate a JNDI context. This context is used to login to the server and look up server side resources that handle incoming proxy requests.

  Listing 29-2 shows the getContext() method used in the Java client.

  To learn more about the context handler interface, see Interface JpdProxy.ContextHandler in the Oracle WebLogic Integration Javadoc.

  To learn more about creating an initial context, see Class Environment in the Oracle WebLogic Integration *Javadoc.*

- The method returns a proxy object that can be cast to the public contract interface.

- The method throws the JpdProxyException, which wraps checked exceptions thrown during construction of the proxy.

To Call the Methods on the Target Business Process

To determine which client request methods are available for you to use via the JPD Proxy, review the source code for the business process. To do so, ensure that the business process (JPD) is open in the Oracle Workshop for WebLogic graphical design environment, identify the Client Request calls in the **Design** view, then open the **Source** view, where you can view the method names and signatures.

JPD Proxies cannot receive callbacks from business processes. See Limitation Using JPD Proxies for Business Processes That Include Client Callbacks.

About Strongly-Typed XML or MFL Arguments in Business Processes

Business processes can accept (as input) and return typed XML (XML Beans) and typed binary data (MFL). The JPD contract interface generated from such business process references these types. (For an example of an XML type referenced in a JPD contract, see the code listing in What is a JPD Proxy?)

Note that in our example Java client, in Listing 29-2, the following packages are imported to support the XML Bean types used in the **PoProcess** business process:

```
import requisitionpo.www.purchase.PurchaseDocument;
import requisitionpo.www.purchaserequestreq.PurchaseRequestReqDocument
```

About Conversation Management

You can use the JpdProxySession interface to set and get the conversation ID used when a business process is invoked. To use the JpdProxySession interface, clients can simply typecast proxies returned by JpdProxy.create() to JpdProxySession.

The dynamic proxy returned by JpdProxy.create() implements the JpdProxySession interface. The methods on the JpdProxySession interface include:

```
String getConversationID()
```
Returns the current conversation ID in use by the JPD Proxy.

```
void reset()
```
Resets the conversation ID to null.

```
void setConversationID(String conversationID)
```
Sets the conversation ID for future invocations of the same instance of the business process.

The conversation ID is initialized to *null*. If the conversation ID is *null* when a method on a business process is invoked through the JPD Proxy, a unique conversation ID is generated. This unique ID is maintained in the run-time state on the client side. In other words, the value is maintained for subsequent invocations, until the client specifies a new conversation ID or resets it to null.

The same JPD Proxy instance can be used to call methods on different instances of a business process. However, clients should take care to avoid making a call with the wrong conversation ID. Specifically, when a client application is finished invoking an instance of a business process through its JPD Proxy and wants to start a new conversation, it must either explicitly set the conversation ID for the second conversation or call JpdProxySession.reset(), which causes the JPD Proxy to reset the conversation ID to null.

To learn more about the **JpdProxySession** interface, see Interface JpdProxySession in the Oracle WebLogic Integration *Javadoc*.

To Run the Java Client

The following command line describes the options that you must set when you run the example Java client (startPoProcess) shown in Listing 29-2.

```
java -Dbea.home=C:\bea startPoProcess
```

where -Dbea.home=C:\bea specifies the product installation location.

Limitation Using JPD Proxies for Business Processes That Include Client Callbacks

JPD Proxies cannot receive callbacks from business processes. Business processes that include client callbacks send those callbacks to the client that started the business process. If a client uses a JPD Proxy to start a business process that includes client callbacks, the business process fails at run time when it tries to send the callback to the client that started it (the JPD Proxy).

To Use a JPD Proxy From a Java Client With Two-Way SSL

The example described in this section shows you how to add two-way SSL to a Java client. This section also describes the command-line options required to run the Java client so that the two-way SSL handshake can take place between the Java client and the SSL server. This section includes the following topics:

- Example Java Client With Two-Way SSL
- To Run the Java Client

Example Java Client With Two-Way SSL

The following example demonstrates how to add two-way SSL to a Java client. The example code is explained in the text that follows the listing.

Listing 29-5   Example Java Client With Two-Way SSL

```
import weblogic.wli.jpdproxy.MyProcess;
import javax.naming.Context;
import javax.naming.NamingException;
import weblogic.jndi.Environment;
import com.bea.wli.bpm.proxy.JpdProxy;
import java.io.*;
import javax.naming.InitialContext;

public class startMyProcess
{
    public static void main(String[] args)
    {
        try {

            InputStream key = new
FileInputStream("C:\\certcmds\\qa\\pki\\keys\\newParent.key");
            InputStream cert = new
FileInputStream("C:\\keystore\\newParentx509.cer");
```

```
            final InputStream FStream[] = {key,cert};
            MyProcess tm = (MyProcess)
            JpdProxy.create(MyProcess.class,MyProcess.SERVICE_URI,
                new JpdProxy.ContextHandler()
            {

                public Context getContext() throws NamingException
                {
                    Environment env = new Environment();
                    //Use t3s – secure port for ssl
                    env.setProviderUrl("t3s://localhost:7002");
                    //Client Certificate and Private Key for that certificate.
                    env.setSSLClientCertificate(FStream);
                    env.setSSLClientKeyPassword("testing123");
                    return env.getInitialContext();
                }
            });
            String str = tm.requestQuote();
            System.out.println("Return String = " + str);
            }
            catch (Exception ex)
            {
                    //Got an exception
                    System.out.println("Got Exception: " + ex);
                    ex.printStackTrace();
            }
    }
}
```

This example builds on the information provided in the previous section (To Use a JPD Proxy From a Java Client) by demonstrating how to add two-way SSL to the Java client. The example code in Listing 29-5 shows a Java client that uses a JPD Proxy to invoke a business process named `MyProcess.java`.

The following items describe the lines of code used to set up the two-way SSL between the client and Oracle WebLogic Server:

- **`import weblogic.wli.jpdproxy.MyProcess;`**

  This client accesses the business process via proxy classes found in `MyProcess.jar`, in the default package: `weblogic.wli.jpdproxy`.

- **`final InputStream FStream[] = {key,cert};`**

  To pass the digital certificates to JNDI, an array of `InputStreams` opened on files containing DER-encoded[1] digital certificates is created. The first element in the array is a

private key file; it is followed by the Java client's digital certificate file, or files[2]. (The digital certificate file contains the public key for the Java client.)

Note:

If you have PEM-encoded data, you can wrap your `InputStreams` in `PEMInputStream` classes before passing them in. To do so, add the following lines of code after you create instances of the PEM-encoded key and certificates in your file:

```
// wrap input streams if key/cert are in pem files
key = new PEMInputStream(key);
cert = new PEMInputStream(cert);
```
The weblogic.security.PEMInputStream class reads digital certificates stored in PEM files.

The private key is the first input stream in the array; subsequent input streams in the array can be a single certificate (as in our example) or a chain of X.509 certificates.

- **`Environment env = new Environment();`**

  You must create a `new Environment` object for each call to the `getInitialContext()` method. Once you specify a `User` object and security credentials, both the user and their associated credentials remain set in the `Environment` object.

- Specify the following parameters. The WebLogic JNDI `Environment` class creates a hash table to store these parameters:

  – **`env.setProviderURL`**—The client calls this method to specify the URL of the Oracle WebLogic Server instance acting as the SSL server. In this example, the URL specifies the t3s protocol which is a Oracle WebLogic Server proprietary protocol built on the SSL protocol.

    Note: In addition to the t3 and t3s protocols, Oracle WebLogic Server clients can use the RMI over IIOP protocol. To learn about using RMI over IIOP, see Programming Oracle WebLogic RMI over IIOP in the Oracle WebLogic Server documentation.

  – **`env.setSSLClientCertificate`**—Specifies a certificate (or a chain of certificates) to use for the SSL connection. You use this method to specify the input stream array that consists of a private key and a certificate.

  – **`env.setSSLClientKeyPassword`**—Sets the password for an encrypted RSA private key. If you aren't using an encrypted private key then you do not need to set this value.

- **`return env.getInitialContext();`**

When the JNDI `getInitialContext()` method is called, the Java client and Oracle WebLogic Server execute mutual authentication. An exception is thrown if the digital certificates cannot be validated or if the Java client's digital certificate cannot be authenticated in the default (active) security realm. The authenticated user object is stored on the Java client's server thread and is used for checking the permissions governing the Java client's access to any protected Oracle WebLogic resources.

To Run the Java Client

The following command line describes the options that you must set when you run the example Java client (`startMyProcess`) shown in Listing 29-5. Setting these options ensures that the two-way SSL handshake can take place between the Java client and Oracle WebLogic Server.

```
java -Dbea.home=C:\bea
-Djava.protocol.handler.pkgs=com.certicom.net.ssl
-Dweblogic.security.SSL.ignoreHostnameVerification=true
-Dweblogic.security.TrustKeyStore=CustomTrust
-Dweblogic.security.CustomTrustKeyStoreFileName=c:\keystore\trustCA.jks
-Dweblogic.security.CustomTrustKeyStoreType=jks
startMyProcess
```

The command-line options you use depend on the type of trust set up on Oracle WebLogic Server. In this example, Oracle WebLogic Server was set up with a Custom Trust. (Other options include the Oracle WebLogic Server Demo Trust and Java Standard Trust.)

In the preceding command line:

- **`-Dbea.home=C:\bea`**—Specifies the product installation location.

- **`-Djava.protocol.handler.pkgs`**—Specifies the protocol handler.

  Note: *SSL Client License Requirement*—Any stand-alone Java client that uses WebLogic SSL classes (`weblogic.security.SSL`) to invoke an Enterprise Java Bean (EJB) must use the BEA license file. When you run your client application, you must set the `-Dbea.home` and the `-Djava.protocol.handler.pkgs` system properties on the command line:

- **`-Dweblogic.security.SSL.ignoreHostnameVerification`**—Disables host-name verification. Specifically, the client does not verify that the host name, which the SSL server returns in its digital certificate, matches the host name of the URL used to connect to the SSL server. We recommend that you enable hostname verification when you run your application in production.

- **`-Dweblogic.security.TrustKeyStore`**—Specifies the keystore used by the the server instance to which you want to connect. In this example, the server is using a custom keystore: **`CustomTrust`**.

- **-Dweblogic.security.CustomTrustKeyStoreFileName**—Specifies the fully qualified path to the trust keystore.

- **-Dweblogic.security.CustomTrustKeyStoreType**—This optional command-line argument specifies the type of the keystore. The value defaults to the keystore type specified in the JDK's `java.security` file. Generally, the value is `jks`.

Note: If the custom keystore is protected by a password, include the following:
`-Dweblogic.security.CustomTrustKeystorePassPhrase=`*password*.

Our example trusts the CA certificates in a custom keystore. The command-line options you use depend on the type of trust set up on Oracle WebLogic Server. For instance:

- To trust only the CA certificates in the Java Standard Trust keystore (`SDK_HOME\jre\lib\security\cacerts`), you do not need to specify command-line arguments, unless the keystore is protected by a password. If the Java Standard Trust keystore is protected by a password, use the following command-line argument:

  `-Dweblogic.security.JavaStandardTrustKeystorePassPhrase=password`

- To trust both the CA certificates in the Java Standard Trust keystore and in the demonstration trust keystore (`BEA_HOME\wlserver_10.3\server\lib\DemoTrust.jks`), include the following argument:

  `-Dweblogic.security.TrustKeyStore=DemoTrust`

  This argument is required if the server instance to which you want to connect is using the demonstration identity and certificates. If the Java Standard Trust keystore is protected by a password, include the following command-line argument:

  `-Dweblogic.security.JavaStandardTrustKeystorePassPhrase=password`

To learn more about using SSL authentication in Java clients, see the Oracle WebLogic Server documentation:

Related Topics

Weblogic JNDI Environment Class

weblogic.Admin Command-Line Reference

How Do I: Use a JPD Proxy From a JSP?

To Create a JSP file that Calls a Business Process Using the JPD Proxy

1. In your JSP file, add an import statement for the JPD Proxy package, as shown in the following lines:

```
<%@ page import="com.bea.wli.bpm.proxy.JpdProxy"%>
<%@ page import="com.bea.wli.bpm.proxy.JpdProxySession"%>
```

To learn about using the JpdProxySession interface, see To Import the Proxy Classes.

2. Create an instance of the proxy class.

Using the same example as we used in How Do I: Use a JPD Proxy From a Java Client?, the code should resemble the following code:

```
try
{
    PoProcess p = (PoProcess)
        JpdProxy.create(
            PoProcess.class,
            PoProcess.SERVICE_URI,

             new JpdProxy.ContextHandler()
             {
                 public Context getContext() throws NamingException
                 {
                     Environment env = new Environment();
                     env.setProviderUrl("t3://localhost:7001");
                     env.setSecurityPrincipal("weblogic");
                     env.setSecurityCredentials("weblogic");
                     return env.getInitialContext();
                 }
             });
    PoDocument document = PoDocument.Factory.newInstance();
    Po po = document.addNewPo();
    po.setSku("abc");

    PoReferenceDocument ref = p.processPO(document);
    p.done();
}
    catch (Exception e) { ... }
}
%>
</html>
```

Note: To learn about the signatures of the JpdProxy.create() class, see To Use the Proxy Factory (JpdProxy.create()) Method.

How Do I: Use a JPD Proxy From an EJB?

You can use a JPD Proxy to invoke a business process from an EJB in the same way as you use a JPD Proxy from any Java file. To learn how, see How Do I: Use a JPD Proxy From a Java Client?

To learn about developing EJBs, see Programming WebLogic Enterprise JavaBeans.

Calling Business Processes