BEA WebLogic Integration™

How Do I?

Version 8.1 Service Pack 3
Document Date: June 2004

Copyright

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

Restricted Rights Legend

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

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

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

Trademarks or Service Marks

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

All other trademarks are the property of their respective companies.

Contents

How Do I . . . ?

Getting Started

Using Keyboard Shortcuts

Importing Files into the Schemas Project

Publishing and Subscribing to Channels

Calling Business Processes

CHAPTER 1

How Do I . . . ?

These topics provide step-by-step instructions for accomplishing common tasks to get started building business processes.

Topics Included in This Section

Getting Started

Learn how to start and stop WebLogic Server, create new applications and files, get started using the WebLogic Workshop graphical design environment to create your business processes, and so on.

Using Keyboard Shortcuts

Learn about the keyboard shortcuts available to you as you design your business processes in WebLogic Workshop.

Importing Files into the Schemas Project

Learn about importing XML Schemas, MFL files, and Channel files into a Schemas project in your integration application.

Publishing and Subscribing to Channels

Learn how to publish and subscribe to Message Broker channels.

Calling Business Processes

Learn about the different approaches that clients can use to communicate with business processes, including using a JPD Proxy. You can use a JPD Proxy to communicate with a business process from any Java client, including JSP pages, servlets, EJBs, and standalone Java clients.

Managing Conversations

Business processes and clients can communicate multiple times to complete a single task. Conversations keep track of data between calls to ensure that a service always responds to the correct client. Learn how to manage conversations in your applications.

CHAPTER 2

Getting Started

To learn how to get started with WebLogic Workshop to create business processes, see the following topics:

- How Do I: Start WebLogic Workshop?
- How Do I: Start and Stop WebLogic Server?
- How Do I: Create a New Application?
- How Do I: Create a New Project?
- How Do I: Create a New Business Process File?
- How Do I: Open an Existing Business Process?
- How Do I: Use the Design View?
- How Do I: Learn More About XQuery?

How Do I: Start WebLogic Workshop?

To start WebLogic Workshop, complete the procedure appropriate for your operating system:

- To Start WebLogic Workshop on Microsoft Windows
- To Start WebLogic Workshop on Linux

To Start WebLogic Workshop on Microsoft Windows

Invoke WebLogic Workshop from the **Start** menu by choosing:

Start→All Programs→BEA WebLogic Platform 8.1→WebLogic Workshop **8.1**

If this is the first time WebLogic Workshop is started since it was installed, the *samples* project, which contains sample services installed with WebLogic Workshop, is displayed. Otherwise, the project which was opened last is displayed.

To Start WebLogic Workshop on Linux

1. Using a file system browser or shell, locate the `Workshop` start script at:

   `$HOME/bea/weblogic81/workshop/Workshop.sh`

2. Start WebLogic Workshop using one of the following methods:
   - Use the following command from the command line:

     `sh Workshop.sh`
   - From a file system browser, double-click `Workshop.sh`.

Related Topics

How Do I: Start and Stop WebLogic Server?

How Do I: Create a New Application?

How Do I: Start and Stop WebLogic Server?

It is not required that you run WebLogic Server while you are designing a business process. However, before you run and test your business process, you must start WebLogic Server. Business Processes you create run on WebLogic Server on your development machine (at least until you deploy them).

Check the status bar at the bottom of WebLogic Workshop to determine whether WebLogic Server is running:

- If WebLogic Server is running, the following icon is displayed:

- If WebLogic Server is not running, the following icon is displayed:

To start WebLogic Server, from the WebLogic Workshop menu, choose **Tools→WebLogic Server→Start WebLogic Server**.

You can minimize the command prompt windows, leaving only a dialog box displaying a progress bar. When the server starts, the status bar at the bottom of WebLogic Workshop indicates that the server is running.

Note: You can view and edit the configuration for your instance of WebLogic Server from the WebLogic Workshop menu by choosing **Tools→Application Properties** or **Tools→WebLogic Server→Server Properties**.

To stop WebLogic Server, from the WebLogic Workshop menu, choose **Tools→WebLogic Server→Stop WebLogic Server**.

Related Topics

How Do I: Create a New Application?

How Do I: Create a New Application?

A WebLogic Workshop application—displayed as the top-level (parent) node on the **Application** tab—contains one or more *projects*, which in turn contain the folders and files that make up your application.

Application—The components of the application you are creating are represented in a hierarchical tree structure on the **Application** tab in your WebLogic Workshop environment. J2EE applications and their components are deployed on WebLogic Server as Enterprise Application Archive (EAR) files. The name you specify for the application becomes the name of the EAR file that you use to deploy your application.

Projects—Projects that are contained in your application represent WebLogic Server web applications. That is, when you create a project, you are creating a web application. The name of your project will be included in the URL your clients use to access your application. For example, if you create a business process (`HelloWorld.jpd`) in a project named **World**, clients can access your business process via the following URL:

`http://host:port/WorldWeb/HelloWorld.jpd`

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

Note: When you name your project folder, the word **Web** is automatically appended to the end of the project folder name and used as the Web application name.

Schemas—To make the XML and MFL schemas in your application available in your business process, you must place them in the **Schemas** folder. This project folder is also where you must

place your Channel files. The Schemas folder is a child folder of your business process application folder in WebLogic Workshop. To learn more about projects and applications, see Creating a Business Process Application.

When you add MFL and XML schemas to the Schemas folder in your business process project, representations of these schemas are available in some of the panes of WebLogic Workshop. In addition, Java interfaces for accessing the data represented in the schemas are generated. To learn more about XML and MFL schemas, see Importing Schemas and Importing Files into the Schemas Project.

To Create a New Application

1. From the WebLogic Workshop menu, choose **File→New→Application**.

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

2. To create a business process application, in the left pane of the dialog box, select **Process** or **Tutorial**. In the right pane, select one of the following options:

   – **Process Application**—Creates a new business process application that contains a basic business process project. A basic process (`process.jpd`) contains a Start node, a Starting Event placeholder, and Finish node. Folders used in your application, such as a Schemas project folder, are also created.

   – **Tutorial: Process Application**—Creates an application containing components for the Business Process and Data Transformation tutorials. To learn about completing the tutorial, see *Tutorial: Building Your First Data Transformation*.

   – **Tutorial: Hello World Process Application**—Creates an application folder, a project folder (`helloWeb`), and a sample business process (`HelloWorld.jpd`). A **Schemas** folder is also created in the application. To learn about adding XML Schemas to your business process applications, see Importing Files into the Schemas Project. These template business process applications are provided to help you quickly get started creating business process applications.

   You can also build an ebXML or RosettaNet participant business process in WebLogic Workshop by using specially created templates. For more information about how to create these participant processes, see Building ebXML Participant Business Processes and Building RosettaNet Participant Business Processes.

3. In the **Directory** field, specify the directory in which to create the Application folder.

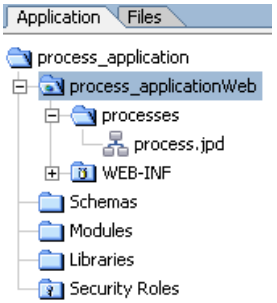
4. In the **Name** field, specify a name for your new application.

5. In the **Server** field, select the integration sample domain or any other WebLogic Integration domain. Click **Browse** to find a WebLogic Server configuration file.

   To learn about creating domains, see Tutorials: Using the Configuration Wizard at `http://edocs.bea.com/platform/docs81/confgwiz/tutorials.html`.

6. Click **Create**.

   Your application is displayed in the **Application** tab in WebLogic Workshop. An example of an application created using the **New Business Process Application** option is displayed in the following figure.



Related Topics

Tutorial: Building Your First Business Process

Tutorial: Building Your First Data Transformation

How Do I: Create a New Project?

How Do I: Create a New Business Process File?

Guide to Building Business Processes

Building ebXML Participant Business Processes

Building RosettaNet Participant Business Processes

Tutorials: Using the Configuration Wizard

How Do I: Create a New Project?

When you create a project in WebLogic Workshop, you are also creating a WebLogic Server web application. The name you specify for your project is visible to clients when they access the services you provide, so the name should describe the collective purpose of the services in the project.

An application can include one or more projects. Every project must be inside an application.

To Create a New Project

1. Select the **Application** tab.

   Note: If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.

2. Right-click the top-level (parent) folder of the application in which you want to add the project, then choose **New→Project**.

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

   Note: When you right-click an application, you can also choose **Import Project**. Select **Import Project** if you want to import a project from outside the application in which you are currently working. You must select the **Copy into Application directory** check box to import Process or Schema projects.

3. In the right pane, select the category of project, such as **Schema**.

4. In the left pane, select the type of project, such as **Schema Project**.

5. In the **Project name** field, enter a name for your project. Give your project a name that reflects its function.

   Note: When you create a project in WebLogic Workshop, you are creating a WebLogic Server web application. The name of your project will be included in the URL your clients use to access your application. To learn more about the components of your application, see Components of Your Application.

6. Click **Create**.

Related Topics

How Do I: Create a New Business Process File?

Guide to Building Business Processes

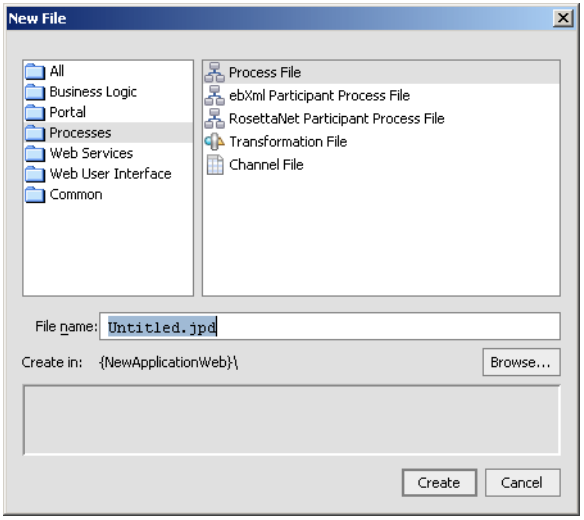
How Do I: Create a New Business Process File?

To create a new business process in a project that you have already created, complete the following steps:

To Create a New Business Process File in an Existing Project

1. Select the **Application** tab.
2. Select the project to which you want to add the new business process.
3. From the menu, choose **File→New→Process File**.

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



4. Select **Process file** from the dialog box as the type of file to create.

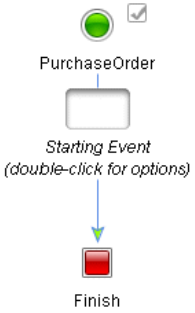
   Other business process-related files you can create include business process DTF files and Channel files. To learn about these files, see *Guide to Data Transformation*, Designing Start Nodes, Building ebXML Participant Business Processes, Building RosettaNet Participant Business Processes, and How Do I: Create Message Broker Channels?

5. Enter a name for your business process in the **File name** field. Give your business process a name that reflects its function, for example, `RequestQuote.jpd` or `ProcessPurchaseOrder.jpd`.

   Note: As indicated by the file extension in the **New File** dialog box, you create a new JPD file when you create this business process file. A JPD file is a Java file in that it contains code for a Java class. JPD files also contain the metadata that describes the business process logic. Because a file with a JPD extension contains the implementation code intended specifically for a business process class, the extension gives it special meaning in the context of WebLogic Server.

6. Click **Create**.

   The new JPD file, which for now consists only of a Start, Starting Event Placeholder, and an Finish node, is created and displayed in the **Design View**.



Related Topics

Guide to Building Business Processes

How Do I: Open an Existing Business Process?

Building ebXML Participant Business Processes

Building RosettaNet Participant Business Processes

How Do I: Open an Existing Business Process?

1. From the WebLogic Workshop menu, choose **File→Recent Application→YourApplication**.
2. If your application is not in the **Recent Applications** menu, choose **File→Open**.

   The **Open Workshop Application** dialog box is displayed.
3. Browse to an `YourApplication.work` file, then click **Open**.

   The application you selected is displayed as a node in the **Application** tab.
4. To display one or more project folders, if necessary, expand the application node. Each node contains one or more business processes (JPD files).
5. To display any business process in the **Design View**, double-click its business process JPD file on the **Application** tab.

Related Topics

How Do I: Use the Design View?

How Do I: Use the Design View?

Designing a business process in WebLogic Workshop involves creating a graphical representation of the business process that meets the business requirements for your project. You represent the business processes you wish to model by drawing and connecting shapes that comprise the flow. Program control is represented visually by these shapes (nodes) and the connections between them. In effect, you create a drawing of your business process and its interactions with clients and other resources.

To help you design your business processes graphically, this section provides a brief overview of the components provided in the WebLogic Workshop graphical design environment. To learn how to design specific business process patterns, see *Guide to Building Business Processes*.

Design View and Source View

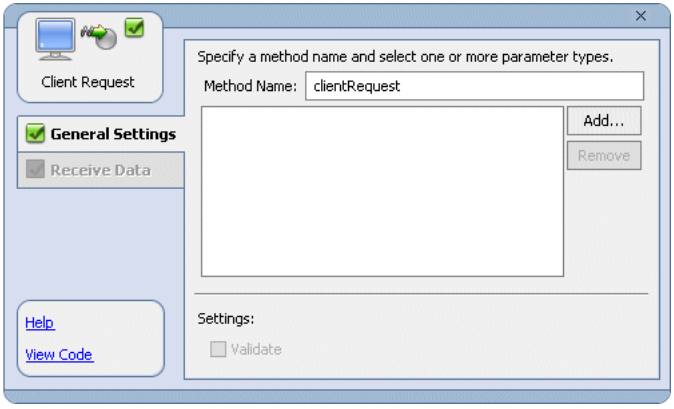
Two tabs are provided on your main work space in the Business Process development environment: **Design View** and **Source View**. You use the Design View as your main work space when you develop business processes graphically. Your source code is updated in the Source View to reflect your work in the Design View. Two way editing is supported—in other words, changes you make in the Source View are reflected in the Design View also. The source code is commented to help you edit the source correctly. To learn more about the components of your business process source code (JPD files), see Business Process Source Code.

Node Builders

Node builders provide a task-driven interface that allow you to specify the logic required at nodes in your business process. You drag nodes from the **Palette** onto your business process to represent the business logic required at points in your business process. Double-click a node to invoke a node builder, which contains tasks appropriate for the node. Node builders open in place in your business process—you can open several at once and move back and forward between them to design your business process logic.

For example, you can define methods that are invoked by clients to start your business process, and methods by which the business process responds to clients on Client nodes. You can specify controls with which your business process node interacts and the data exchanged in the interaction. You can also invoke the transformation tool to transform heterogeneous data as it is

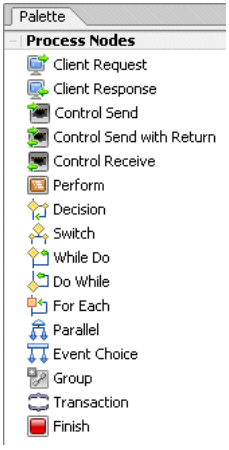
exchanged between your business process and resources. For example, the following figure shows the node builder for a Client Request node:



Palette

In the **Design View**, you add business process nodes to create the graphical representation of your business process that meets the business requirements for your project. To add nodes to the business process, drag a node from the **Palette** onto the business process in the **Design View**.

If the **Palette** tab is not visible in WebLogic Workshop, choose **View→Windows→Palette** from the menu bar. The **Palette** displays business process nodes you can use to design the patterns required for your business process.



As you drag a node from the **Palette** onto the **Design View**, targets appear on your business process. Each target represents a valid location in the flow where you can place the node. As you drag the node near a valid 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 onto the business process at the location indicated by the active target. Note that if you create a node at an invalid location (that is, if you create invalid logic in your business process flow) that node is marked with the following icon: . Move your mouse pointer over the error icon to see a message that describes the error.

Data Palette

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

The **Data Palette** displays the following tabs: **Variables** and **Controls**. The **Variables** tab displays the business process variables created in your business process, and allows you to create new variables. To learn how to use the **Variables** tab, see Business Process Variables and Data Types.

The **Controls** tab displays the instances of controls in your project, and allows you to create new instances. Business Processes interact with resources such as web services, databases, EJBs, and so on, via controls. To learn about using controls in your business process, see Interacting With Resources Using Controls.

Property Editor

The **Property Editor** allows you to view and set properties for the different components of your application such as controls and business processes, as well as nodes and groups of nodes in your business process. Each **Property Editor** looks different depending on the component. Most components contain the **notes** property; any notes you enter in the **Property Editor** are associated with and displayed for that component throughout the WebLogic application.

If the **Property Editor** is not visible in WebLogic Workshop, from the menu bar, choose **View→Property Editor**.

Related Topics

Guide to Building Business Processes

How Do I: Learn More About XQuery?

You can use WebLogic Workshop to create queries (written in the XQuery language) for transformations in business processes and web services.

For more information on transformations in web services (where they are also called XQuery maps), see Transforming XML Messages with XQuery Maps.

If your application requires more complex queries than what can be generated by WebLogic Workshop, you can edit and enhance the generated queries directly in WebLogic Workshop.

When WebLogic Workshop is used to create business processes, queries are generated for the following cases:

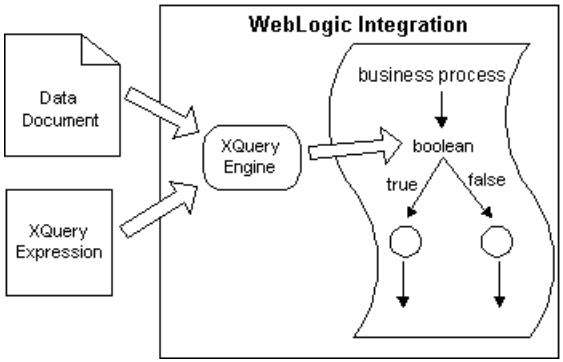
- Branching in Business Processes
- Data Transformation
- Generating a Set of Typed Data Elements for a For Each Loop

These queries are written in the XQuery language. The XQuery language, defined by the World Wide Web Consortium (W3C), provides a vendor independent language for the query and retrieval of XML data.

To learn about the XQuery language, see the XQuery 1.0: An XML Query Language Specification - W3C Working Draft 16 August 2002 at `http://www.w3.org/TR/2002/WD-xquery-20020816`. The WebLogic Integration XQuery engine conforms to the August 16, 2002 draft of the XQuery Specification.

Branching in Business Processes

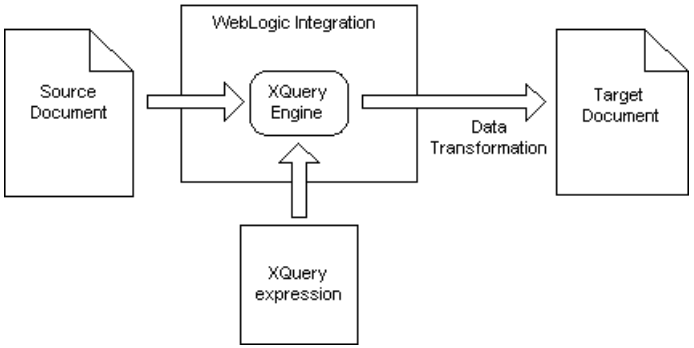
As part of the process of adding a **Decision** node to business process, a customized XQuery expression is generated by WebLogic Workshop. When the business process is run, the generated XQuery expression is run against the data document and based on the result of that XQuery, as shown in the following diagram:



The branching occurs in the **Decision** node of a business process.

Data Transformation

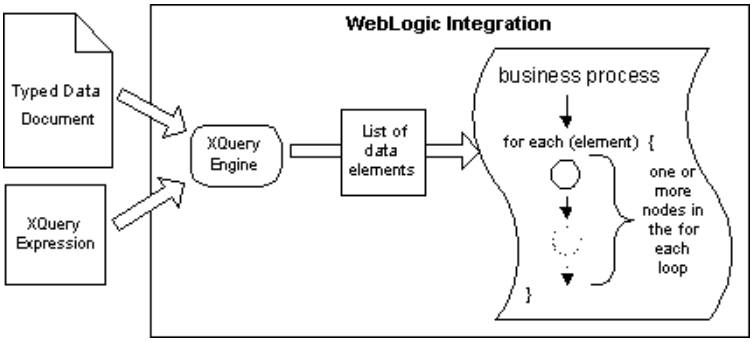
A customized XQuery expression is generated by WebLogic Workshop for the transformation of data. Using XQueries, WebLogic Integration can perform complex data transformation from typed non-XML (MFL) to XML, XML to non-XML, or even non-XML to non-XML. When the business process is run, the generated XQuery expression is run against the source document, which transforms the data from the original document (valid to the source schema if typed) to the resulting document (valid to the target schema if typed), as shown in the following diagram:



Generating a Set of Typed Data Elements for a For Each Loop

A customized XQuery expression is generated by WebLogic Workshop as part of the process of adding a **For Each** node. When the business process is run, the customized XQuery expression creates a list of typed data elements for the For Each loop. The For Each loop in the business

process iterates through the set of typed data elements produced by the XQuery. In each iteration, the node(s) contained in the For Each loop are executed, as shown in the following diagram:



Related Topics

Transforming Data Using XQuery

Guide to Building Business Processes

Tutorial: Building Your First Data Transformation

Transforming XML Messages with XQuery Maps

Using Keyboard Shortcuts

To learn how to use keyboard shortcuts as you design your business processes in WebLogic Workshop, see the following topics:

How Do I: Use Cut/Copy/Paste Shortcuts?

The **cut**, **copy**, and **paste** shortcuts are as follows:

- **Ctrl+X**—If you select an item and then enter **Ctrl+X** (or select **Cut** from a right-click or the **Edit** menu), the item you selected is copied to the clipboard and removed from its original place. If your item is a process node, you can paste it elsewhere within the current business process, or another business process. If you selected text, you can paste this text elsewhere within the WebLogic Workshop or any other application.

Warning: When you choose to cut a process node, all non-referenced associated process language and methods are removed as well. If a method is referenced by more than one node, the method will stay in the source code in its original location until all nodes that reference it are deleted. Also, if you have modified the node in the Source View, the node will remain in the source code in its original location.

- **Ctrl+C**—If you select an item and then enter **Ctrl+C** (or select **Copy** from a right-click or the **Edit** menu), the item you selected is copied to the clipboard. If your item is a process node, you can paste it in another location within the current business process, or another business process. If you selected text, you can paste this text elsewhere within the WebLogic Workshop or any other application.
- **Ctrl+V**—If you have previously copied something to the clipboard, either by using a cut or copy command, you can paste the contents of the clipboard elsewhere in the WebLogic Workshop application or any other application (if the clipboard contains text only). To paste an item, click the location where you want to paste the item, then enter **Ctrl+V**. To paste into a group, you must first expand the group. The new node will be pasted in the beginning of the block. To move it, simply drag it to the location you want.

  Note: When you paste a Client Request, Client Response, or Process node after copying it to the clipboard, the new node and the corresponding methods will be duplicated and have a number appended to their names. If you want two nodes to reference the same method, you can explicitly specify the method name in the node builder.

Related Topics

How Do I: Use Undo/Redo Shortcuts?

How Do I: Use Arrow Keys?

How Do I: Use Delete/Enter Keys?

How Do I: Use Print Shortcuts?

Guide to Building Business Processes

How Do I: Use Undo/Redo Shortcuts?

The **undo** and **redo** shortcuts are as follows:

- **Ctrl+Z**—Entering **Ctrl+Z** (or selecting **Undo** from the **Edit** menu) removes the last change you applied to the application. If you added a node, then the node and all corresponding generated code is removed. If you deleting a node, then the node and all corresponding generated code is returned. Multiple undos are supported.
- **Ctrl+Y**—Entering **Ctrl+Y** (or selecting **Redo** from a right-click or the **Edit** menu) undoes your latest undo. Multiple redos are supported.

Note: The **undo** and **redo** commands are based on the last action that took place in the **Source View**, not necessarily the last action that took place in the **Design View.**

Related Topics

How Do I: Use Cut/Copy/Paste Shortcuts?

How Do I: Use Arrow Keys?

How Do I: Use Delete/Enter Keys?

How Do I: Use Print Shortcuts?

Guide to Building Business Processes

How Do I: Use Arrow Keys?

The rules for using the tab and arrow keys are as follows:

- **Up and Down Arrow keys**—You can use the up and down arrow keys to navigate up and down the path of your business process. The downward path of the business process is as follows:
  - The main node.
  - The node group.
  - The left most branch of the node, if any.
  - The remaining branches counting from left to right, if any.
  - Any exception, message, or timeout path (starting from the left to right).
  - The next node.

  Note: Holding down the **Shift** key while using the up or down arrows, selects multiple nodes or group members.
- **Left and Right Arrow Keys**—Use the left arrow key to collapse a node and the right arrow key to expand a node.
- **Alt Left and Right Arrow Keys**—Hold down the **Alt** key, then use the left and right arrow keys to toggle between **Design View** and **Source View**.

Related Topics

How Do I: Use Undo/Redo Shortcuts?

How Do I: Use Cut/Copy/Paste Shortcuts?

How Do I: Use Delete/Enter Keys?

How Do I: Use Print Shortcuts?

Guide to Building Business Processes

How Do I: Use Delete/Enter Keys?

The **Delete** and **Enter** keys have the following functions:

- **Delete**—Pressing the **Delete** key at any time, deletes the item which was selected at the time the key was pressed.

Warning: When you choose to delete a process node, all non-referenced associated process language and methods are removed as well. If a method is referenced by more than one node, the method will stay in the source code in its original location until all nodes that reference it are deleted. Also, if you have modified the node in the Source View, the node will also remain in the source code in its original location.

- **Enter**—After selecting a process node, pressing the **Enter** or **Return** key invokes the node builder. After selecting collapsed node or group, pressing the **Enter** or **Return** key expands the node.

Related Topics

How Do I: Use Undo/Redo Shortcuts?

How Do I: Use Arrow Keys?

How Do I: Use Cut/Copy/Paste Shortcuts?

How Do I: Use Print Shortcuts?

Guide to Building Business Processes

How Do I: Use Print Shortcuts?

You can print your business processes by selecting **Edit→Print**, or by pressing the **Ctrl+P** keys.

Note: You can not print the content of any of the node builders. If you print a business process with a node builder open, the node builder will appear as a large white empty block on your print out.

Related Topics

How Do I: Use Undo/Redo Shortcuts?

How Do I: Use Arrow Keys?

How Do I: Use Delete/Enter Keys?

How Do I: Use Cut/Copy/Paste Shortcuts?

Guide to Building Business Processes

Importing Files into the Schemas Project

To learn about importing XML Schemas, MFL files, and Channel files into the Schemas project in your integration application, see the following topics:

- How Do I: Create a Schemas Project Folder?
- How Do I: Import Files into a Schemas Project Folder?

How Do I: Create a Schemas Project Folder?

When you create a new Process application, by default a project folder named `Schemas` is created in the business process application folder within the **Application** tab of WebLogic Workshop. Additional schema project folders can be created in your application folder.

To Create Additional Schema Project Folders

1. In the **Application** tab, right-click the top-level (parent) application folder.

   Note: If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.

2. From the drop-down menu, select **New→Project...**.

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

3. In the left pane, select **Schema**.

4. In the right pane of the **New Project** dialog box, select one of the following:

- **Schema Project—**Any XSD or MFL schema files imported into this file will automatically generate MFL Classes and XML Bean Classes to be used as schemas representation in WebLogic Workshop. Select this file if you are going to add channel files to your schemas folder.
- **OAG Schemas**—This project folder is for Open Application Group schemas. For more information, see http://www.openapplications.org.
- **WLI System Schemas—**This project folder has the same functionality as a project folder of type Schema Project. In addition, it contains WebLogic Integration System XSD files.

For more information about how to import files into your schemas folder and how to generate MFL Classes and XML Bean Classes, see How Do I: Import Files into a Schemas Project Folder?

5. In the **Project name** field, enter a name, such as `MySchemas`.

   Note: You can name your schemas folder anything you want, except for when you are going to use the folder for application view channels and schemas. In that case, the folder must be named **`Schemas`**. For more information see "Prerequisites for Integration Applications Using WebLogic Workshop" in Application View Control.

6. Click **Create**.

   A new project folder is created in the **Application** tab.

MFL and XSD files can be imported into any schemas project folders. You can have one or more project folders of either schema type in an application folder. For example, you might want to place schemas that infrequently change into the default project folder named `Schemas` and create another **Schema Project** folder called `MySchemas` that contain schemas that change more frequently. If a schema file changes in the project folder, all the schemas in that project folder are rebuilt. Partitioning your schemas in this way can reduce the schema build time. This means if a schema changes in the `MySchemas` project folder, the schemas in the `Schemas` project folder will not be built. For more information about how to import schemas, see How Do I: Import Files into a Schemas Project Folder?

Related Topics

How Do I: Import Files into a Schemas Project Folder?

Importing Schemas

How Do I: Import Files into a Schemas Project Folder?

When a schema (XSD or MFL file) is imported into your application, representations of these schemas are available in various WebLogic Workshop panes. To learn more about XSD and MFL files, see Importing Schemas. In addition, Java interfaces for accessing the data represented in the schemas are generated. To learn more about these Java classes, see "Java Classes Created from Importing Schema" in Programming Transformations.

You also import channel files into schema project folders. This ensures that your Channel files are globally accessible throughout your WebLogic Workshop application. To learn more about how to create channel files and subscribe to Message Broker channels, see Publishing and Subscribing to Channels

To Import a File into a Schemas Project Folder

To make the schemas in XSD, MFL, and Channel files available in your business process application, you must import them into a **Schemas** project folder.

1. In the **Application** tab, right-click a **Schemas** folder.

   Note: If the **Application** tab is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.

2. From the drop-down menu, select **Import**.

   The **Import Files** dialog box is displayed.

3. Browse the file system to select the file that you want to import.

4. Click **Import**.

When a XSD or MFL file is imported, a build of the current Schemas project folder is triggered. The build verifies that the schema file is well formed. For XSD files, it also verifies that the element and attribute names in the XML Schema do not conflict with the XSD files that have previously imported into the current Schemas project folder. To learn more about what gets generated when you import schemas, see Importing Schemas.

Related Topics

How Do I: Create a Schemas Project Folder?

Publishing and Subscribing to Channels

Java Classes Created From Importing Schemas

Importing Schemas

CHAPTER 5

Publishing and Subscribing to Channels

The Message Broker resource provides a publish and subscribe message-based communication model for WebLogic Workshop business processes, and includes a powerful message filtering capability.

The Message Broker provides typed channels to which messages can be published and to which services can subscribe to receive messages. You can design a business process to subscribe to specific channels, using XML Beans for type-safe methods.

Subscribers to Message Broker channels can filter messages on the channels using XQuery filters. Business processes can filter documents on channels, based on the type of document, or on a specific instance of a document type. For example, you can design a filter that filters on stock symbols in a document, or one that filters on a specific purchase order number.

In addition to business processes that can publish messages to Message Broker channels, WebLogic Integration supports event generators that can publish external events to Message Broker channels. WebLogic Integration provides native event-generators, including JMS, Email, File, and Timer event generators. WebLogic Integration also works with Application View event generators that work with J2EE-CA connectors.

To learn about publishing and subscribing to Message Broker channels, see the following topics:

- How Do I: Create Message Broker Channels?
- How Do I: Publish to Message Broker Channels?
- How Do I: Subscribe to Message Broker Channels?

How Do I: Create Message Broker Channels?

Channel files define the Message Broker channels available in a WebLogic Integration application. Channel files must be placed in a Schema project in your application. Otherwise, they will not be built when you build your application or be visible to your application components.

This topic includes the following sections:

To Create Channels Files in Your Application

1. Locate the **Schemas** project in the **Application** tab.
2. Right-click the **Schemas** project and choose **New→Channel File** from the drop-down menu.

   The **New File** dialog box is displayed.
3. In the left pane, select **Processes**, then in the right pane, select **Channel file**.
4. Enter a name for the file in the **File name** field.

   Note: As indicated by the file extension in the **New File** dialog box, the Channel file is automatically appended with **channel**.
5. Click **Create**.

   Your new channel file is created and displayed in the **Schemas** folder in the **Application** tab. This file is a template file. The contents of the template file is displayed in the **Source View**, which you can edit to define the Message Broker channels for your application. For information about how to edit your channel file template, see To Configure Your Template Message Broker Channel File.

   Channel files are XML files and are valid against an XML Schema. The Schema is available at the following location in your WebLogic Platform installation:

   ```
   BEA_HOME\weblogic81\integration\lib\xmlschema\config\ChannelFile.xsd
   ```

   In the preceding line, *BEA_HOME* represents the directory in which you installed WebLogic Platform.

To Configure Your Template Message Broker Channel File

1. If your channel file is not visible, double-click the channel file in your **Schemas** project. The file is displayed in the **Source View**:
2. Edit this code specific to your channel needs.

Note the following characteristics of the channel file:

- `channels xmlns="http://www.bea.com/wli/broker/channelfile"`

  This is a namespace that references the names used in the channel file schema. Do not edit this line.

- `channelPrefix="/SamplePrefix"`

  The **`channelPrefix`** uniquely identifies the channels in your file across the entire domain. To identify your channels, you can change the **`channelPrefix`**. If you want to ensure that you do not accidentally send or receive messages from other applications, we recommend that you use a **`ChannelPrefix`** that is the same as your Workshop application name. However, if you want to use the Message Broker for inter-application communication, these applications should use channels with the same prefix, such as "`/sharedChannels`".

- `xmlns:eg="http://www.bea.com/wli/eventGenerator"`

  This namespace refers to schemas that are used for event generator metadata references. The schemas are created for you when you create a new process application or a new Schemas project folder. This line can safely be removed if your channels do not publish messages via event generators. To learn more about how to create schemas, see To Create Additional Schema Project Folders. To learn about creating and managing event generators, see the following:

  – Using Event Generators to Publish to Message Broker Channels

  – Event Generators at `http://edocs.bea.com/wli/docs81/manage/evntgen.html`

- `xmlns:dp="http://www.bea.com/wli/control/dynamicProperties"`

  This namespace refers to schemas that are used by the file event generator to pass payload for pass-by-filename. The schemas are created for you when you create a new process application or a new schemas project folder. This line can safely be removed if you are not using your channels to publish messages via file event generators. To learn more about how to create schemas, see To Create Additional Schema Project Folders.

- `xmlns:oag="http://www.openapplications.org/003_process_po_007"`

  This namespace refers to schemas that are used by some of the sample channels. The schemas are created for you when you create a new OAG Schemas project folder. This line

can safely be removed if none of the channels in your file refer to the OAG schemas. To learn more about how to create schemas, see To Create Additional Schema Project Folders.

Warning: Make sure that the namespaces you reference in your channel files exist in your application. If they do not, the channel file compiles without problems, but you will get a run-time error when you try to run your application.

- `<channel name ="Samples" messageType="none">`

  Edit the attributes of this element as follows:

  – **`name`**—Enter the name of your channel.

  – **`messageType`**—Specify the message type that will be routed through your channel, possible choices are:

  -**`xml`**: For channels routing XML data.

  -**`rawData`**: For channels routing Raw Data.

  -**`string`**: For channels routing String messages.

  -**`none`**: For channels that are not going to be used to route messages but are used as place holders only.

  For example, a simple channel routing XML data would look like:

  ```
  <channel name ="MyXMLChannel" messageType="xml">
  ```

  There are several additional attributes that you can add to the `<channel>` element:

  – **`qualifiedMessageType`**—Add this attribute specify the message type qualifier. For example, if the message on a channel were element `PROCESS_PO_007` in the `"http://www.openapplications.org/003_process_po_007"` (prefixed by `oagpo`) namespace, the **`qualifiedMessagetype`** attribute would look like: `qualifiedMessageType="oagpo:PROCESS_PO_007"`.

  – **`qualifiedRawDataMessageType`**—If your message type is RawData, use this attribute to specify the message type qualifier. the name must begin with `"urn:"` and refer to an MFL file in a schema. For example, if the messages on a channel were described by `StockQuotes.mfl` in a schemas project folder, the **`qualifiedRawDataMessageType`** attribute would look like: `qualifiedRawDataMessageType="urn:StockQuotes.mfl"`. To learn more about MFL files, see "Using Format Builder to Create Format Schemas (MFL Files)" in Transforming Non-XML Data.

  – **`qualifiedMetadataType`**—When you are using event generators (or applications that send metadata) to route messages through your channel, use this attribute to specify the

metadata. All event generators and some applications send metadata with messages. For example, an email event generator always sends element "EmailEventGenerator" in the "http://www.bea.com/wli/eventGenerator" (prefixed by eg:) namespace. The **qualifiedMetadataType** attribute for different event generator channels looks like:

-Email event generator: qualifiedMetadataType="eg:EmailEventGenerator".

-File event generator: qualifiedMetadataType="eg:FileEventGenerator"

-JMS event generator: qualifiedMetadataType="eg:JmsEventGenerator"

-Timer event generator: qualifiedMetadataType="eg:TimerEventGenerator"

To learn about creating and managing event generators using the WebLogic Integration Administration Console, see Event Generators in *Managing WebLogic Integration Solutions* at the following URL:

http://edocs.bea.com/wli/docs81/manage/evntgen.html

Dead Letter Channels

If the Message Broker is unable to resolve the URI to which it is sending a message, the message is not discarded. Rather, it is routed to one of three dead-letter channels, depending on the data type. For example, when no subscribers are found, or registered subscribers do not match because of unsatisfied filter conditions, the message is send to the appropriate dead-letter channel. WebLogic Integration provides the following dead-letter channels:

- /deadletter/xml
- /deadletter/string
- /deadletter/rawData

For example, an unmatched messages published to an XML channel (that is, a channel that has messageType = "xml") is routed to the /deadletter/xml channel.

At design time, the dead-letter channels are available when you create MB Publish and MB Subscription controls. Your business processes can publish and subscribe to the dead-letter channels. You can use the dead-letter channels when you design error handling. For example, you can create a business process that includes static subscriptions to the dead-letter channels and design error-handling code to handle the unmatched messages published to those channels.

The Message Broker module in the WebLogic Integration Administration Console allows you to monitor and manage all the Message Broker channels in your application, including the dead-letter channels. To learn about accessing information about the dead-letter channels and

dead-letter counts in your deployed application, see Message Broker in *Managing WebLogic Integration Solutions* at the following URL:

```
http://edocs.bea.com/wli/docs81/manage/msgbroker.html
```

Related Topics

Understanding the Message Broker Channels in Your Tutorial Application

Using Event Generators to Publish to Message Broker Channels

Transforming Non-XML Data

How Do I: Publish to Message Broker Channels?

How Do I: Subscribe to Message Broker Channels?

Event Generators at `http://edocs.bea.com/wli/docs81/manage/evntgen.html`

Message Broker at `http://edocs.bea.com/wli/docs81/manage/msgbroker.html`

How Do I: Publish to Message Broker Channels?

You must first create a Message Broker Publish control in your project, then bind a method from the control to a node in your business process. To do so, complete the following steps:

1. In the **Application** tab, double-click the applicable business process to ensure that it is displayed in the **Design View**.
2. On the **Data Palette Controls** tab, click **Add→Integration Controls→MB Publish**. The **Insert Control** dialog box is displayed.

   Note: If the **Data Palette** is not visible in WebLogic Workshop, click **View→Windows→Data Palette** from the menu bar.
3. In **Step 1**, enter the variable name for this control.
4. In **Step 2**, do one of the following:
   a. To use an existing MB Publish control, select **Use a MB Publish control already defined by a JCX file**, then enter the name of a JCX file or use **Browse** to select the JCX file.
   b. To create a new MB Publish control, select **Create a new MB Publish control to use**, then, in the **New JCX name** field, enter the name for the new JCX file.
5. In **Step 3**, select the applicable channel from the **channel-name** field. This specifies the channel to which your business process publishes the messages it receives from clients.

Note: The message type field is populated with the data type of the message that is published to the channel. To learn how the channel is defined, see How Do I: Create Message Broker Channels?.

6. Click **Create**.

   The **Insert Control** dialog box closes and the JCX file is created and displayed in the **Application** tab. The instance of the MB Publish control you created is displayed in the **Data Palette**.

7. To view the individual methods in the **Data Palette**, click the + beside the MB Publish control.

8. In the **Data Palette**, select the publish method in the control:

   For example, `void publish(RawData value).`

9. Drag-and-drop the method onto the applicable business process in the **Design View**.

   A **Control Send** node is created. By default, the node is named **publish**.

10. Double-click the **publish** node to open its node builder. The node builder opens on the **General Settings** tab with the **Control** and the **publish** method are already selected.

11. Click **Send Data**.

    In this tab, you can specify the message to be published to the selected Message Broker channel. The **Control Expects** field is populated with the data type of the parameter expected by the control.

12. In the **Select variables to assign** field, click the arrow to display the variables in your project. Then select the applicable variable or create a new variable.

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

Related Topics

How Do I: Create Message Broker Channels?

How Do I: Subscribe to Message Broker Channels?

Message Broker Publish Control

Using Event Generators to Publish to Message Broker Channels

Dead Letter Channels

How Do I: Subscribe to Message Broker Channels?

You design a subscription to a Message Broker channel in one of two ways in your business processes: you create an instance of a Message Broker Subscription control and design the subscription using it, or for the special case of a process that is started as a result of a subscription to a Message Broker channel, you design the subscription on the process' Start node.

To learn how to design the Start-node subscriptions, see Subscription Start (Asynchronous) and Subscription Start (Synchronous).

This section describes how to create a Message Broker Subscription control in your project, then bind a method from the control to a node in your business process.

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

To Subscribe to a Message Broker Channel Using a Subscription Control

1. In the **Application** tab, double-click the applicable business process to ensure that it is displayed in the **Design View**.
2. On the **Data Palette Controls** tab, click **Add→Integration Controls→MB Subscription**.

   The **Insert Control** dialog box is displayed.
3. In **Step 1**, enter the variable name for this control.
4. In **Step 2**, do one of the following:

   a. To use an existing MB Subscription control, select **Use a MB Subscription control already defined by a JCX file**, then enter the name of a JCX file or use **Browse** to select the JCX file.

   b. To create an existing MB Subscription control, select **Create a new MB Subscription control to use**, then, in the **New JCX name** field, enter the name for the new JCX file.
5. In **Step 3**, select the applicable channel from the **channel-name** field. This specifies the channel to which your business process subscribes (and the channel to which the Validation service publishes messages when it determines that an incoming message is invalid).

   Note: The message-type field is populated with the data type of the message that is published to the selected channel. To learn how the channel is defined, see Understanding the Message Broker Channels in Your Tutorial Application.

6. If you intend to use filters on the channel, select **This subscription will be filtered**.
7. Click **Create**.

   The **Insert Control** dialog box closes and the new JCX file is created and displayed in the **Application** tab. The instance of the Message Broker Subscription control you created is displayed in the **Data Palette**.
8. In the **Data Palette**, select the subscribe method in the control:

   ```
   void subscribe()
   ```
9. Drag-and-drop the method onto the business process in the **Design View**.

   A **Control Send** node is created. By default, the node is named **subscribe**. Note that the ✅ indicates that the specifications on this node are complete. No further work is required to design this node, unless you selected to use a filter. If so, proceed to the To Specify a Filter Value for Your Channel section.

Note: When you create your control, a callback method is created for you (`void OnMessage(`*`message`*`)`). You can drag-and-drop this method from the **Data Palette** onto your business process. This creates a Control Receive node that you can configure to handle your callback. For an example, see Step 14: Designing a Message Path for Your Business Process in the *Tutorial: Building Your First Business Process*.

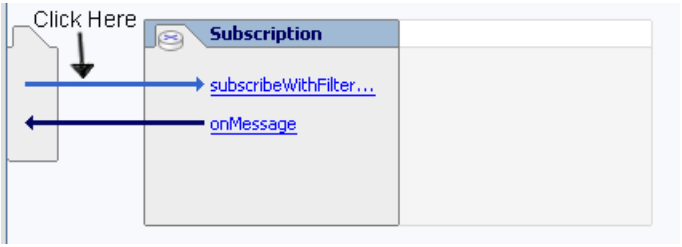
To Specify a Filter Value for Your Channel

If you selected the **This subscription will be filtered** check box in step 5. in the above procedure, you must enter a filter value for your `subscribeWithFilterValue(`*`value`*`)` before your filter will work properly.

1. In the **Application** tab, double-click the subscription **JCX** file that you created earlier.

   Note: If the **Application** tab is not visible, select **View→Application**.

   The JCX file is displayed in Design View.
2. Click the arrow pointing to your `subscribeWithFilterValue` method, as shown in the figure below.



3. In the **Property Editor**, beneath the **mb-subscription-method** property, in the **filter-value-match** field, click ....

   The **Property Text Editor** window opens.

4. In the **Property Text Editor**, replace **{value}** with the value that you want your subscription control to filter on.

5. Click **OK**.

   The **Property Text Editor** closes and the valued you entered will be used as the filter value at run time.

Related Topics

How Do I: Create Message Broker Channels?

How Do I: Publish to Message Broker Channels?

Message Broker Subscription Control

Subscription Start (Asynchronous)

Subscription Start (Synchronous)

Dead Letter Channels

@jpd:mb-static-subscription Annotation

CHAPTER 6

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 WebLogic Workshop 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 JPD Proxies.

You can only use Process controls and Service Broker controls between WebLogic Workshop components: web services (JWS), business processes (JPD), 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 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 WebLogic Workshop 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 WebLogic Workshop 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 WebLogic Workshop application in the same domain, you must use a Service Broker control.

To learn how to create and use process and Service Broker controls, see Process Control and 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.

Related Topics

Starting Your Business Process

Interacting With Clients

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

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?

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 WebLogic Workshop 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 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.jpd` is shown in the following listing.

Listing 6-1 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 WebLogic Workshop 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 WebLogic Workshop 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.

Related Topics

- How Do I: Get a JPD Proxy for a Business Process?
- How Do I: Use a JPD Proxy From a Java Client?

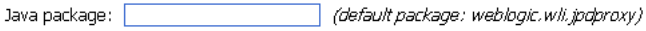
How Do I: Get a JPD Proxy for a Business Process?

1. Open your business process in WebLogic Workshop.
2. On the menu bar, click  to run the business process.

   If the 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 WebLogic Workshop 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 WebLogic Workshop graphical design environment, the JAR file is displayed in the `WEB-INF/lib` folder on the **Application** tab.

   `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 WebLogic Workshop graphical design environment, the JAR file is displayed in the `Libraries` folder at the root of your application in the **Application** tab.

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

If you are using the JPD Proxy from a standalone Java client (outside of 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 WebLogic Workshop 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 WebLogic Platform installation:

   `BEA_HOME``\weblogic81\integration\lib`

   In the preceding line, *`BEA_HOME`* represents the location where you installed WebLogic Platform.

   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 WebLogic Platform installation: `\bea\weblogic81\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 WebLogic Platform installation: *`BEA_HOME`*`\weblogic81\server\lib`

   In the preceding line, *`BEA_HOME`* represents the location at which you installed WebLogic Platform.

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.

Related Topics

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

Starting Your Business Process

Versioning Business Processes

Interacting With Clients

How Do I: Call Business Processes?

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 6-2 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 6-5) 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 WebLogic Workshop 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 6-2 is an example of a Java client that invokes a business process using a JPD Proxy. It invokes a business process named `PoRequest.jpd`. This example includes basic username-password authentication. A second example (Listing 6-5) describes how to add two-way SSL to the Java client.

Listing 6-2 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 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 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 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 JPD on the server.

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

Listing 6-3 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 6-4, and is used in the example code in Listing 6-2.

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 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 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 WebLogic Integration system stateless session bean that handles incoming requests from JPD Proxies. Its scope is the 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 WebLogic Server Administration Console. BEA 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 WebLogic Server documentation:

WebLogic JNDI at `http://e-docs.bea.com/wls/docs81/jndi/jndi.html`

Using JAAS Authentication in Java Clients at `http://e-docs.bea.com/wls/docs81/security/fat_client.html`

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 6-4 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 6-2 shows the `getContext()` method used in the Java client.

  To learn more about the context handler interface, see Interface JpdProxy.ContextHandler in the WebLogic Integration *Javadoc*, which is available at the following URL:

  `http://e-docs.bea.com/wli/docs81/javadoc/com/bea/wli/bpm/proxy/JpdProxy.ContextHandler.html`

  To learn more about creating an initial context, see Class Environment in the WebLogic Integration *Javadoc,* which is available at the following URL:

  `http://e-docs.bea.com/wls/docs81/javadocs/weblogic/jndi/Environment.html`

- 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 WebLogic Workshop 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. To learn more about the Client Request and Client Response methods in business processes, see Designing Start Nodes.

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 6-2, 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 WebLogic Integration *Javadoc*, which is available at the following URL:

`http://e-docs.bea.com/wli/docs81/javadoc/com/bea/wli/bpm/proxy/JpdProxySession.html`

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 6-2.

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

where `-Dbea.home=C:\bea` specifies the location of the BEA license file (`license.bea`).

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 6-5 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 6-5 shows a Java client that uses a JPD Proxy to invoke a business process named `MyProcess.jpd`.

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

  [1]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 (at `http://e-docs.bea.com/wls/docs81/javadocs/weblogic/security/PEMInputStream.html`) reads digital certificates stored in PEM files.

  [2]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 WebLogic Server instance acting as the SSL server. In this example, the URL specifies the t3s protocol which is a WebLogic Server proprietary protocol built on the SSL protocol.

  Note: In addition to the t3 and t3s protocols, WebLogic Server clients can use the RMI over IIOP protocol. To learn about using RMI over IIOP, see Programming WebLogic RMI over IIOP in the WebLogic Server documentation at `http://e-docs.bea.com/wls/docs81/rmi_iiop/index.html`.

  – **`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 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 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 6-5. Setting these options ensures that the two-way SSL handshake can take place between the Java client and 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 WebLogic Server. In this example, WebLogic Server was set up with a Custom Trust. (Other options include the WebLogic Server Demo Trust and Java Standard Trust.)

In the preceding command line:

- **`-Dbea.home=C:\bea`**—Specifies the location of the BEA license file (`license.bea`).
- **`-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 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 (`WL_HOME\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 following WebLogic Server documentation:

Configuring Keystores and SSL at
`http://e-docs.bea.com/wls/docs81/ConsoleHelp/security_7x.html`

Using SSL Authentication in Java Clients at
`http://e-docs.bea.com/wls/docs81/security/SSL_client.html`

Related Topics

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

Using SSL Authentication in Java Clients at
`http://e-docs.bea.com/wls/docs81/security/SSL_client.html`

Weblogic JNDI Environment Class at
`http://e-docs.bea.com/wls/docs81/javadocs/weblogic/jndi/Environment.html`

weblogic.Admin Command-Line Reference at
`http://e-docs.bea.com/wls/docs81/admin_ref/cli.html`

Starting Your Business Process

Interacting With Clients

How Do I: Call Business Processes?

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.

Related Topics

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 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 in WebLogic Workshop, see Programming Enterprise JavaBeans, which is available at the following URL:

```
http://e-docs.bea.com/wls/docs81/ejb/
```

Related Topics

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

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