BEA WebLogic Integration™

Introducing Trading Partner Integration

Version 8.1
Document Revised: July 2003

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

Introduction

Introducing ebXML Solutions

Introducing RosettaNet Solutions

Trading Partner Integration Security

About This Document

This document introduces key concepts and tasks for understanding and building trading partner integration solutions in BEA WebLogic Integration.

This document covers the following topics:

- Chapter 1, "Introduction," introduces concepts, defines terminology, and summarizes tasks for planning, building, and deploying trading partner integration solutions. It contains the following topics:
  - About Trading Partner Integration
  - Trading Partner Management Concepts
  - Trading Partner Business Process Concepts
  - Messaging Concepts
  - Run-Time Monitoring Concepts
  - Summary of Trading Partner Integration Phases
  - Next Steps
- Chapter 2, "Introducing ebXML Solutions," describes concepts and tasks for building ebXML integration solutions using WebLogic Integration. It contains the following topics:
  - About ebXML Solutions
  - ebXML Concepts
  - ebXML Business Processes

  - Tasks for Implementing an ebXML Solution

- Chapter 3, "Introducing RosettaNet Solutions," describes concepts and tasks for building RosettaNet integration solutions using WebLogic Integration. It contains the following topics:
  - About RosettaNet Solutions
  - RosettaNet Concepts
  - RosettaNet Business Processes
  - Tasks for Implementing a RosettaNet Solution
- Chapter 4, "Trading Partner Integration Security," describes concepts and tasks for implementing security for trading partner integration resources. It contains the following topics:
  - Before You Begin
  - Security Framework for Trading Partner Integration
  - Transport-Level Security
  - Message-Level Security
  - Using Proxy Servers with Trading Partner Integration
  - Implementing Security for Trading Partner Integration

What You Need to Know

This document is intended for anyone who wants to understand and use the trading partner integration capabilities of WebLogic Integration.

Product Documentation on the dev2dev Web Site

BEA product documentation, along with other information about BEA software, is available from the BEA dev2dev Web site:

http://dev2dev.bea.com

To view the documentation for a particular product, select that product from the list on the dev2dev page; the home page for the specified product is displayed. From the menu on the left side of the screen, select Documentation for the appropriate release. The home page for the complete documentation set for the product and release you have selected is displayed.

Related Information

Readers of this document may find the following documentation and resources especially useful:

- For information about ebXML, go to the Web site at http://www.ebxml.org.
- For information about RosettaNet, go to the Web site at http://www.rosettanet.org.
- For general information about Java applications, go to the Sun Microsystems, Inc. Java Web site at http://java.sun.com.
- For general information about XML, go to the O'Reilly & Associates, Inc. XML.com Web site at http://www.xml.com.

Contact Us!

Your feedback on the BEA WebLogic Integration documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Integration documentation.

In your e-mail message, please indicate that you are using the documentation for BEA WebLogic Integration WebLogic Integration 2.1.

If you have any questions about this version of BEA WebLogic Integration, or if you have problems installing and running BEA WebLogic Integration, contact BEA Customer Support at http://support.bea.com. You can also contact Customer Support by using the contact information provided on the quick reference sheet titled "BEA Customer Support," which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.
`monospace text`	Indicates *user input*, as shown in the following examples: • Filenames: `config.xml` • Pathnames: *`BEAHOME`*`/config/examples` • Commands: `java -Dbea.home=BEA_HOME` • Code: `public TextMsg createTextMsg(`
	Indicates *computer output*, such as error messages, as shown in the following example: `Exception occurred during event dispatching:java.lang.ArrayIndexOutOfBoundsException: No such child: 0`
`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `java utils.MulticastTest -n` *`name`* `[-p` *`portnumber`*`]`
|	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed. *Example*: java weblogic.deploy [list|deploy|update]

Convention	Item
. . .	Indicates one of the following in a command line: • That an argument can be repeated several times in a command line • That the statement omits additional optional arguments • That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name] [-f "file1.cpp file2.cpp file3.cpp . . ."`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER 1

Introduction

This topic describes basic concepts, architecture, and tasks for creating WebLogic Integration solutions for trading partner integration. It contains the following sections:

- About Trading Partner Integration
- Trading Partner Management Concepts
- Trading Partner Business Process Concepts
- Messaging Concepts
- Run-Time Monitoring Concepts
- Summary of Trading Partner Integration Phases
- Next Steps

For a comprehensive overview of WebLogic Integration, see *Introducing WebLogic Integration* at the following URL: http://edocs.bea.com/wli/docs81/overview/index.html. For a hands-on walkthrough of building and running example ebXML and RosettaNet solutions in WebLogic Integration, see *Tutorials for Trading Partner Integration*, which is available at:

- Documentation and Example Files

  `http://dev2dev.bea.com/code/wli.jsp`

- Documentation Only

  `http://edocs.bea.com/wli/docs81/tptutorial/index.html`

About Trading Partner Integration

WebLogic Integration allows you to automate and manage relationships with your trading partners so that you can streamline your business processes (with customers, suppliers, distributors, and other partners) and get a top-down view of business transactions across the value chain. Trading partner integration is also known as business-to-business (or B2B) integration.

WebLogic Integration provides the following trading partner integration capabilities:

- Visual Public/Private Process Integration
- Support for Leading Industry Protocols and Standards
- Trading Partner Management (TPM) and Repository Access
- Easy Access to Run-Time Information
- High Performance and Availability
- High Security, Auditing, and Non-Repudiation
- Trading Partner Enablement

Visual Public/Private Process Integration

WebLogic Integration leverages the unified programming model and run-time framework of WebLogic Workshop to provide end-to-end business process integration via easily implemented controls and templates.

Support for Leading Industry Protocols and Standards

WebLogic Integration supports the following B2B protocols and standards:

- ebXML 1.0 and 2.0, which is described in Chapter 2, "Introducing ebXML Solutions."
- RosettaNet 1.1 and 2.0, which is described in Chapter 3, "Introducing RosettaNet Solutions."
- Web Services, which is described in "TPM Control For Business Processes and Web Services" on page 1-16, "TPM Repository Lookups Via Process and Service Broker Controls" on page 1-16, and in "Building Web Services" in the WebLogic Workshop Help, which is available at the following URL:

  `http://edocs.bea.com/workshop/docs81/doc/en/workshop/guide/navBuildingWebServices.html`

Trading Partner Management (TPM) and Repository Access

WebLogic Integration provides sophisticated trading partner management capabilities through the unified WebLogic Integration Administration Console, which enables administrators to easily manage a central repository of trading partner profile information, including protocol bindings used for secure message exchanges between trading partners, services representing public processes, security, and bulk import / export capabilities. Authorized business processes and web services can dynamically access trading partner information via easily implemented controls. In addition to the Administration Console, MBean APIs are also provided so that third-party MBean clients can be written to access the TPM repository, as described in "MBean APIs for Third-Party Access" on page 1-11.

Easy Access to Run-Time Information

WebLogic Integration provides flexible run-time tracking, audit, and reporting capabilities to show a top-down view of trading partner activities and business transactions across the value chain.

High Performance and Availability

WebLogic Integration provides fast and reliable business message exchanges between trading partners, supporting the clustered configuration for scalability and fail-over, message persistence for recovery, low-level acknowledgements and receipts, and transactional integrity.

High Security, Auditing, and Non-Repudiation

WebLogic Integration ensures the private, secure, and reliable business message exchanges among trading partners using transport level security with SSL and message level security with digital signature and encryption. The certificates and private keys used for various purposes are kept in protected keystores while the passwords are kept in encrypted forms in the WebLogic Integration PasswordStore.

Trading Partner Enablement

WebLogic Integration works with WebLogic Integration – Business Connect, a lightweight B2B server that is designed for small trading partners who do not have their own B2B server. For trading partners who want a zero-install solution, WebLogic Integration can be easily extended to offer a browser or FTP interface.

Trading Partner Management Concepts

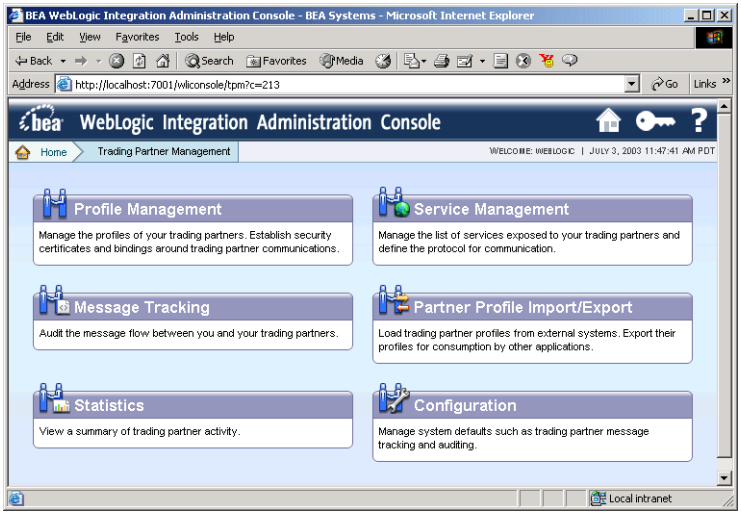
The basic building blocks of trading partner integration are trading partner profiles, services, and service profiles. This topic introduces the concepts you need to understand regarding trading partner management. It contains the following sections:

About Trading Partner Management

All trading partner profile, service, and service profile information resides in the *Trading Partner Management (TPM) repository*, which is stored in a relational database. Administrators use the WebLogic Integration Administration Console to maintain the TPM repository.

The following figure shows the Trading Partner Management home page in the WebLogic Integration Administration Console, which allows administrators to manage trading partner profiles, security certificates, protocol bindings, services, message tracking and auditing, trading partner activity, system defaults, and importing / exporting trading partner profile information.

Figure 1-1 Trading Partner Management in the WebLogic Integration Administration Console



For more information about the Trading Partner Management module in the WebLogic Integration Administration Console, see Trading Partner Management in *Managing WebLogic Integration Solutions*, including the following topics:

- "About Trading Partner Management"
- "Overview of the Trading Partner Management Module"
- "Configuring Trading Partner Management"

Trading Partners

In WebLogic Integration, a *trading partner* is understood to be an entity that has an agreement with another entity to participate in a specific business transaction, or service, by playing a predefined role associated with a distinct business purpose. Trading partner applications form the nodes in system-to-system interactions among business partners.

Types of Trading Partners

A group of trading partners can:

- Exist entirely within a company, spanning multiple corporate departments (the business purpose for such a community might be inventory management, for example)

- Span multiple companies across firewalls and over the Internet (the business purpose might be supply chain management or multistep purchasing interactions, for example)
- Include trading partners both within a company and in other companies (one or more of the trading partners within a company communicates with trading partners in other companies across the Internet)

Trading Partner Profiles

A trading partner profile includes the trading partner's identifying information, and any certificates or protocol bindings required to conduct the business transactions. You use the WebLogic Integration Administration Console to manage trading partner information. For more information about managing trading partner profiles, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Adding Trading Partner Profiles"
- "Defining Trading Partner Profiles"
- "Deleting Trading Partner Profiles"
- "Deleting Trading Partner Profiles and Services Using Bulk Delete"
- "Viewing and Changing Trading Partner Profiles"
- "Listing and Locating Trading Partners"

Basic and Extended Properties

By default, each trading partner has the following set of *basic properties*:

Table 1-1 Basic Trading Partner Properties

Property	Description
Business name	Name of the trading partner.
Business ID	Used for uniquely identifying trading partners in business processes.
Business ID type	Categorizes the type of business ID, such as a DUNs number, customer or vendor ID number, and so on.
Type	Designates whether this trading partner is local to the host system or a remote trading partner.

Table 1-1 Basic Trading Partner Properties

Property	Description
Status	Makes the trading partner visible (enabled) or hidden (disabled) for certain operations, such as business process or web service access to the trading partner information in the TPM repository. For more information, see "Enabling and Disabling Trading Partner and Service Profiles" in Trading Partner Management in *Managing WebLogic Integration Solutions*.
Description	Brief description of the trading partner.
Default Trading Partner	If selected, then the trading partner is designated the default trading partner for sending or receiving messages for the local host system in the absence of specific trading partner information.
Contact information	Email, address, phone, and fax information

In addition to these default properties, you can add custom extensions (extended properties) for individual trading partners in the TPM repository to support application-specific requirements. For example, you might want to include additional contact information, bank account information for electronic transfers, or internal vendor IDs to your trading partners. You can retrieve these properties from business processes and web services using the TPM control and also by navigating subtrees within an XML document. For more information about managing extended properties in the WebLogic Integration Administration Console, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Adding a Custom Extension to a Trading Partner"
- "Viewing and Changing a Custom Extension"
- "Deleting Certificates, Bindings, or Custom Extensions"

Digital Certificates

WebLogic Integration uses digital certificates associated with trading partners to authenticate their identity during message exchanges. For more information about how digital certificates are stored and used in WebLogic Integration, see "Transport-Level Security" on page 4-12. For more information about managing certificates in the WebLogic Integration Administration Console, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Adding Certificates to a Trading Partner"

- “Viewing and Changing Certificates”
- “Deleting Certificates, Bindings, or Custom Extensions”

Services, Service Profiles, and Protocol Bindings

This topic describes services, service profiles, and protocol bindings.

Services

A *service* represents a business process that is either offered by a local trading partner, or a business process that is being called via a control on a remote trading partner.

- In the case of a service *offered* by a local trading partner, this element directly corresponds to a web service or process type deployed in the local domain.
- In the case of a service *called* by a local trading partner, the service corresponds to a control in the local domain that is used to invoke the remote service.

For more information about managing services in the WebLogic Integration Administration Console, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- “Adding Services”
- “Listing and Locating Services”
- “Viewing and Changing Services”
- “Deleting Services”
- “Deleting Trading Partner Profiles and Services Using Bulk Delete”

Service Profiles

Service profiles encapsulate the concept of an agreement between two trading partners on the service bindings to be used. Service profiles specify the *protocol binding* and URL endpoints for the local and remote trading partners that offer and call the service. For more information about managing service profiles in the WebLogic Integration Administration Console, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- “Adding Service Profiles to a Service”
- “Viewing and Changing Service Profiles”

- "Deleting Service Profiles from a Service"
- "Enabling and Disabling Trading Partner and Service Profiles"

Protocol Bindings

Protocol bindings specify the business protocol (ebXML 1.0 or 2.0, or RosettaNet 1.1 or 2.0), transport protocol (such as HTTP 1.0, HTTP 1.1, or HTTPS 1.1), URL end-point, time-outs, number of retries, retry intervals, digital certificates, and other information. For more information about managing protocol bindings in the WebLogic Integration Administration Console, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Defining Protocol Bindings"
- "Viewing and Changing Bindings"
- "Adding Protocol Bindings to a Trading Partner"
- "Deleting Certificates, Bindings, or Custom Extensions"

Exchanging Data in the TPM Repository

WebLogic Integration provides bulk management utilities that simplify the tasks of managing the TPM repository and exchanging trading partner and service information with other trading partners or porting the information to other servers. Data is exchanged via an intermediary XML data file that conforms to the `tpm.xsd` schema that ships with WebLogic Integration. For trading partners that use WebLogic Integration – Business Connect, you can also import a single trading partner profile exported from WebLogic Integration – Business Connect or from WebLogic Integration using the business connect format.

To perform export, import, or bulk delete operations, you can use either the WebLogic Integration Administration Console or a bulk loader command line utility. For more information about bulk management of the TPM repository, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "TPM Schema"
- "Using the Bulk Loader"
- "Importing Management Data"
- "Exporting Management Data"
- "Deleting Trading Partner Profiles and Services Using Bulk Delete"

- Documentation for WebLogic Integration – Business Connect at the following URL:

  `http://e-docs.bea.com/wlibc/docs81/index.html`

When you export TPM data using the console or the bulk loader utility, a file suitable for import is created. To learn more about the required structure, and how the file is used in import, export, and bulk delete operations, see Using the Bulk Loader in *Managing WebLogic Integration Solutions*.

Default TPM Repository Settings

When you create a new WebLogic Integration domain using the BEA WebLogic Platform Configuration Wizard, the Configuration Wizard automatically populates the Trading Partner Management (TPM) repository with default trading partners and protocol bindings. For more information about the Configuration Wizard, see *Creating WebLogic Configurations Using the Configuration Wizard* in the WebLogic Platform documentation.

Default Trading Partners

The WebLogic Integration domain provides two preconfigured trading partners for development and testing:

Table 1-2 Default Trading Partner Configuration in WebLogic Integration Domain

Trading Partner Name	Trading Partner ID	Description
`Test_TradingPartner_1`	`TP1-id`	Default local trading partner. In the tutorials, this trading partner is usually the *initiator* of conversations.
`Test_TradingPartner_2`	`TP2-id`	In the tutorials, this trading partner is usually the *participant* in conversations.

Default Protocol Bindings

Each default trading partner comes with the following preconfigured protocol bindings:

- ebXML 1.0
- ebXML 2.0
- RosettaNet Implementation FrameWork (RNIF) 1.1

- RNIF 2.0

Each protocol binding (except ebXML 1.0) is marked as default. At run-time, the default binding can be used automatically in the absence of specific protocol information. If you are using ebXML 1.0, you need to configure protocol bindings explicitly in the WebLogic Integration Administration Console.

MBean APIs for Third-Party Access

WebLogic Integration supports user-written applications that use Java Management Extensions (JMX) Management Beans (MBeans) to access the TPM repository:

- Configuration MBean APIs are used to configure settings in the TPM repository.
- Monitoring MBean APIs are used to retrieve run-time statistics.

For more information about the MBean APIs, see the WebLogic Integration Javadoc.

Trading Partner Business Process Concepts

This topic introduces the concepts you need to understand regarding trading partner business processes. It contains the following sections:

- About Business Processes for Trading Partner Integration
- Conversations and Roles
- Types of Business Processes

About Business Processes for Trading Partner Integration

In WebLogic Integration, trading partners communicate with each other via WebLogic Workshop business processes that collaborate using the leading B2B protocols—ebXML or RosettaNet. You build, test, and run business processes using WebLogic Workshop's unified programming model and run-time framework. WebLogic Workshop provides controls, templates, and other mechanisms so that you can implement such business processes easily and quickly. For an introduction to building business processes in WebLogic Workshop, see the following topics in the WebLogic Workshop Help:

- Tutorial: Building Your First Business Process
- Tutorial: Building Your First Data Transformation

- Guide to Building Business Processes
- Using Integration Controls

For more information about building business processes for particular business protocols, see:

- "ebXML Business Processes" on page 2-7
- "RosettaNet Business Processes" on page 3-16

Conversations and Roles

This topic describes conversations between trading partners and the roles that trading partners play in conversations.

Conversations

When trading partners exchange business messages for a business purpose, they participate in a *conversation*. A conversation is a series of one or more business messages exchanged between trading partners. The nature of each conversation is determined by its business purpose—whether it is a complex or simple conversation, whether it is a long-running or short-lived conversation, and so on.

Roles: Initiators and Participants

The business messages that can be exchanged between participants in the conversation are determined by the *roles* that the trading partners play in the conversation. Each conversation always involves the following two roles:

- **Initiator**—The trading partner who begins the business exchange by requesting some service of a trading partner.
- **Participant**—The trading partner who is responsible for providing the service and responding to the initiator.

Role-Based Design Patterns

These two roles are fundamental to all business processes involved in conversations, as the design patterns for initiator and participant business processes are very different. In a conversation, the business processes perform very different work but in a collaborative manner—the initiator business process sends a request to the participant, the participant business process receives the request and sends the response back to the initiator, and so on.

Each trading partner that participates in the conversation in a given role must implement the collaborative business process required for its role. Collaborative business process encapsulate the processes required to handle the right business messages at the right time for a given trading partner's role in a conversation.

Role Naming

These roles are often named according to the nature of the conversation. For example, if the conversation involves a supplier sending an invoice to a buyer and getting a confirmation that the invoice was received, then the supplier is the initiator of the conversation and the buyer is the participant. The invoice is the request document and the confirmation is the response document.

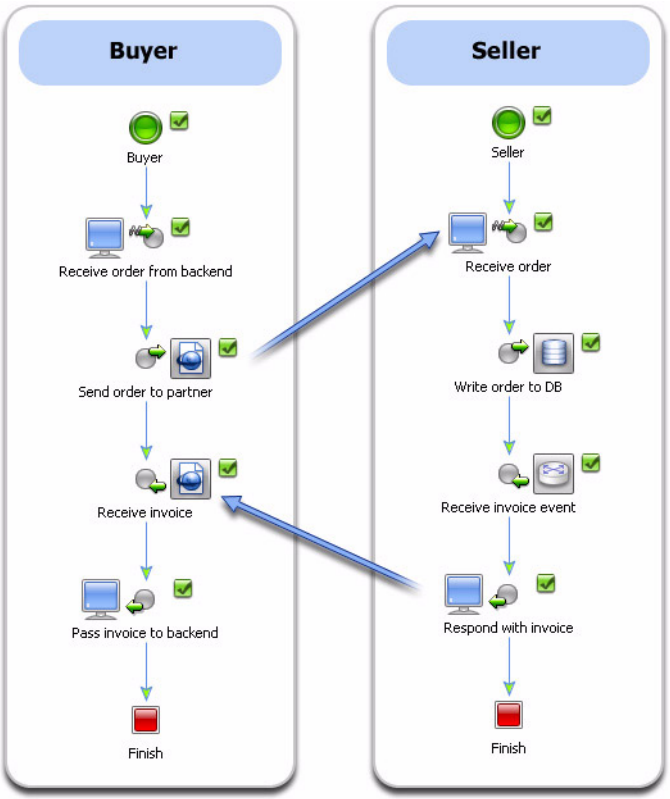
Collaborative Business Processes

In the WebLogic Integration environment, a *collaborative business process* is a business process that implements a role in a conversation for a trading partner. The message choreography for a conversation is defined by collaborative business process.

Sample Conversation

The following figure shows basic interactive business processes between trading partners. The Buyer business process sends an order to the Seller using an agreed-upon business protocol (ebXML or RosettaNet). The Seller business process receives the request, writes the order to a database, receives an invoice from an internal back-end system, and then sends the invoice to the Buyer using the same business protocol.

Figure 1-2 Collaborative, Role-Based Business Processes In a Conversation



Types of Business Processes

This section describes different categories of business processes within trading partner conversations.

Initiator and Participant Business Processes

This topic describes the WebLogic Workshop controls and templates that you can use for initiator and participant business processes.

Initiator Business Processes and WebLogic Workshop Controls

The following table describes the WebLogic Workshop controls that you use in initiator business processes to handle communications with participants:

Table 1-3 WebLogic Workshop Controls Used in Initiator Business Processes

Control Name	Description
ebXML Control	The ebXML control enables WebLogic Workshop business processes to exchange business messages and data with trading partners via ebXML. The ebXML control supports both the ebXML 1.0 and ebXML 2.0 messaging services. You use ebXML controls in *only* initiator business processes to manage the exchange of ebXML business messages with participants. For more information about using the ebXML control, see ebXML Control in the WebLogic Workshop Online Help and "ebXML Initiator Business Processes" on page 2-10.
RosettaNet Control	Enables WebLogic Workshop business processes to exchange business messages and data with trading partners via RosettaNet. You use RosettaNet controls *only* in initiator business processes to manage the exchange of RosettaNet business messages with participants. For more information about using the RosettaNet control, see RosettaNet Control in the WebLogic Workshop Online Help and "RosettaNet Initiator Business Processes" on page 3-17.

Participant Business Processes and WebLogic Workshop Templates

The following table describes the WebLogic Workshop templates that you use in participant business processes to handle communications with conversation initiators:

Table 1-4 WebLogic Workshop Templates Used in Participant Business Processes

Business Protocol	Description
ebXML participant business process file	Template provides a head start for building public participant business processes for ebXML conversations. Although this file is not required to build ebXML participant business processes, it includes the nodes and business process annotations needed to integrate easily with ebXML initiator business processes. For information about using the participant business processes file, see Building ebXML Participant Business Processes and @jpd:ebxml Annotation in the WebLogic Workshop Online Help.
RosettaNet participant business process file	Template provides a head start for building public participant business processes for RosettaNet conversations. Although this file is not required to build RosettaNet participant business processes, it includes the nodes and business process annotations needed to integrate easily with RosettaNet initiator business processes. For information about using participant business processes, see Building RosettaNet Participant Business Processes and @jpd:rosettanet Annotation in the WebLogic Workshop Online Help.

TPM Control For Business Processes and Web Services

The TPM (Trading Partner Management) control provides WebLogic Workshop business processes and web services with query (read-only) access to trading partner and service information stored in the TPM repository. Access to the TPM repository is restricted to active trading partners, services, and active service profiles and their children. For more information about using the TPM control in business processes and web services, see TPM Control in the WebLogic Workshop Online Help. For more information about using web services, see "Building Web Services" in the WebLogic Workshop Help.

TPM Repository Lookups Via Process and Service Broker Controls

Business processes and web services can also access data in the TPM repository via Process and Service Broker controls. These controls provide the `lookupTPMProperties` XQuery function that can be used in selectors. For more information about these controls, see Process Control and Service Broker Control in the WebLogic Workshop Online Help.

Public and Private Business Processes

Business processes can span multiple applications, corporate departments, and business partners (trading partners) behind a firewall and over the Internet. An enterprise's business processes can be divided into two broad categories: *public* and *private*.

Public Business Processes

Public processes are *interface* processes. Their definitions and designs are known, understood, and agreed upon by the organizations using them, and may be customized or standardized across an industry or industry segment, as in the case of RosettaNet Partner Interface Processes (PIPs). They are part of a formal contract between trading partners that specifies the content and semantics of message interchanges. These processes can be implemented in different ways by different trading partners.

In the context of trading partner integration, when collaborative business processes are intended to be reused in multiple conversations with different trading partners, the business processes should be designed as public processes.

Private Business Processes

Participants in a conversation can also implement private, non-collaborative business processes, which can integrate their back-end processing. Private processes are the business processes conducted within an organization. Their definitions and designs are specific to that organization and are not visible outside it. Within trading partner enterprises, private processes interface with public processes and with back-end business systems. In the context of public processes, private processes can be thought of as sub-business processes or subprocesses that implement tasks that are part of the public business process. For example, a trading partner may implement a private business process that works in conjunction with a collaborative business process and that implements the processes that occur locally to a trading partner, but that are not necessarily dictated by the service agreement.

Success and Failure Paths

In a conversation, business processes have two ultimate outcomes—success or failure:

- The success path involves sending a business message and receiving an acknowledgement from the remote trading partner that the business message was received.
- The failure path handles problem scenarios, such as:
  - The remote trading partner did not receive a business message that was sent.
  - The local trading partner did not receive an acknowledgement that the remote trading partner received the business message.
  - The remote trading partner sends an error (the business message was rejected or some other error occurred).
  - The remote trading partner takes too long to reply (the business process expires).

Business processes need to fully implement and account for all of these possible scenarios using such WebLogic Workshop mechanisms as timer controls, retry loops, parallel branches, and so on.

Each failed business message is saved to a file and the file URI and other information (MessageID, FromTP, ToTP, ProcessURI, ProcessInstance, and Timestamp), if available, is enqueued to a dedicated JMS queue named `wli.b2b.failedmessage.queue`. Custom queue listeners or event generator can be created to handle message failures.

Messaging Concepts

This topic introduces the concepts you need to understand how WebLogic Integration handles the delivery of business messages to and from trading partners. It contains the following sections:

- Messaging Services for Trading Partner Integration
- Business Protocols
- Business Messages
- Run-Time Processing of Business Messages
- Message Tracking

Messaging Services for Trading Partner Integration

WebLogic Integration provides reliable, role-based XML messaging that supports enhanced send and receive capabilities, including support for large messages. To support the fast and reliable exchange of business messages among trading partners, WebLogic Integration provides the following messaging services at run-time:

- Generating and processing all non-content message headers (for RosettaNet) or envelopes (for ebXML)
- MIME packing and unpacking of the message
- Message encryption / decryption
- Digital signature generation / validation
- XML validation
- Message persistence for recovery purposes

- Duplicate message detection
- Maintaining a history of messaging activity
- Support for a cluster environment for scalability

Business Protocols

A business protocol is associated with a business process, which governs the exchange of business information between trading partners. It specifies the structure of business messages, how to process the messages, and how to route them to the appropriate recipients. A business protocol may also specify characteristics of messages related to persistence and reliability.

Business Messages

A business message is the basic unit of communication among trading partners and is exchanged as part of a conversation. A business message contains one or more XML business documents, one or more attachments, or a combination of both.

Business Message Formats

The contents and format of a business message depend on the business protocol chosen for the conversation. For more information about specific message formats, see the following topics:

- "RosettaNet Business Messages" on page 3-10
- "ebXML Business Messages" on page 2-4

Attachments

Business messages can include attachments in XML and non-XML formats. WebLogic Workshop business process support the following Java types for attachments:

Table 1-5 Types of Attachments in Business Messages

Data Type	Description
`XmlObject`	Default. Represents data in non-typed XML format. The XML data is not specified at design time.

Table 1-5 Types of Attachments in Business Messages (Continued)

Data Type	Description
`RawData`	Represents any non-XML structured or unstructured data. Examples include PDF, DOC, GIF, JPG, and other binary files.
`MessageAttachment[]`	Array containing one or more parts of an ebXML or RosettaNet business message. Message parts can be non-typed XML data (`XmlObject` data type) or non-XML data (`RawData` data type). Used when sending different kinds of payloads (XML and non-XML) in the same message. The actual number of message parts might not be known until processed. To learn about working with `MessageAttachment` objects, see Message Attachments in the WebLogic Workshop Help.

Note: Attachments can also be typed XML or typed MFL data as long as you specify the corresponding XML Bean or MFL class name in the parameter.

To learn more about these data types, see Working with Data Types in the WebLogic Workshop Help.

To expedite the processing of business messages in JMS queues at run-time, WebLogic Integration stores larger attachments temporarily in a Document Store database.

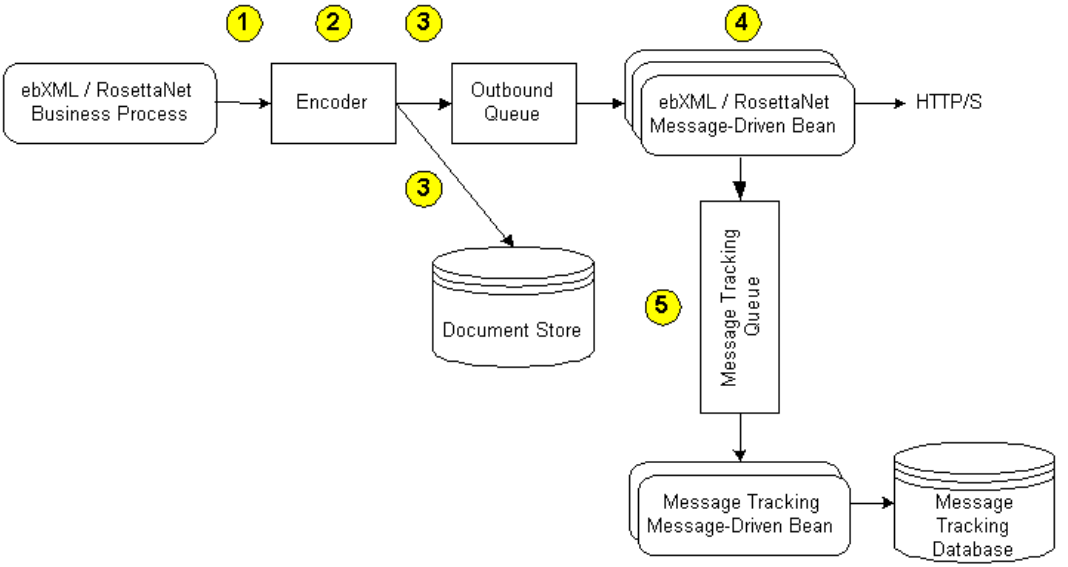
Run-Time Processing of Business Messages

This section describes how ebXML and RosettaNet business messages are processed at runtime. Inbound and outbound business messages traverse different paths. However, the inbound and outbound paths are nearly identical for initiator and business processes alike. The overall flow is the same regardless of the underlying business protocol, although WebLogic Integration's messaging infrastructure provides specialized underlying components to handle ebXML and RosettaNet business messages separately.

Outgoing Message Path

The following figure shows the path of an outgoing business message:

Figure 1-3 Path for Outgoing Trading Partner Messages



The preceding figure illustrates the following process flow:

1. A business message is sent from a WebLogic Integration business process.
   - For initiator business processes, the business message is sent via a particular method on the applicable (ebXML or RosettaNet) control.
   - For participant business processes, the business message is sent via a method on the business process callback.
2. The applicable encoder (RosettaNet or ebXML) packages the message appropriately using input from:
   - the outgoing business message
   - any protocol-specific properties specified in annotations, such as the protocol name and version
   - trading partner and service information retrieved from the TPM repository

   Packaging differs based on the business protocol used and settings configured in the TPM repository. For example, the encoder handles encryption (for RosettaNet), digital signatures, generation of required message headers, MIME packaging, message persistence, and so on.

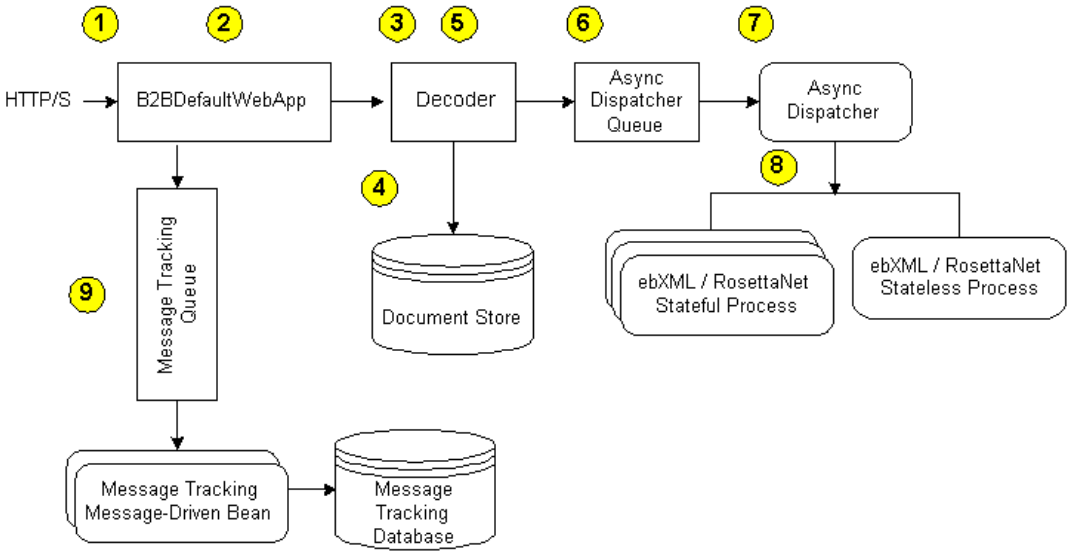
3. The message is persisted in the WebLogic Integration document store and forwarded it to the applicable JMS queue:
   - ebXML: `wli.internal.b2b.ebxmlencoder.queue`
   - RosettaNet: `wli.internal.b2b.rosettanetencoder.queue`
4. The message-driven bean associated with the queue sends the message out asynchronously over HTTP(S) to the endpoint URL for the remote trading partner.
   - ebXML: `WLI-B2B ebXML`
   - RosettaNet: `WLI-B2B RosettaNet`
5. Message tracking information for the outbound message is sent to the message tracking queue (`wli.internal.msgtracking.queue`). A message-driven bean that listens to this queue updates the various message tracking tables based on the tracking level set in the Trading Partner Management module of the WebLogic Integration Administration Console.

   For configuration instructions, see "Configuring the Mode and Message Tracking" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Path for Incoming Trading Partner Messages

The following figure shows the path of an incoming business message:

Figure 1-4 Incoming Message Path



The preceding figure illustrates the following process flow:

1. An incoming trading message, sent to the local trading partner, is received by the Transport Servlet Filter, which is specified in `B2BdefaultWebApp/WEB-INF/web.xml`.
2. The Transport Servlet Filter inspects the URL and determines whether the incoming request is a TPI URL / request.
   - If it is destined for TPI, then the Transport Servlet Filter authenticates the remote trading partner (ensuring that the trading partner name is valid by retrieving its value from a valid certificate associated with the trading partner), and then forwards the message to the Transport Servlet, WLI-B2B HTTP Transport (packaged in `b2b.war`).
   - If it is not destined for TPI, the message continues on to other filters and the final destination servlet.
3. The Transport Servlet sends the message to the applicable decoder (RosettaNet or ebXML), which unpacks the message.

Unpacking differs based on the business protocol used and settings configured in the TPM repository. For example, the decoder disassembles the various MIME parts of the message, validating the XML components (for RosettaNet), handles any required decryption (for RosettaNet), handles any digital signature verification, and so on.

4. The decoder persists the message (payload in ebXML, or Service Content in RosettaNet), plus any attachments, in the WebLogic Integration Document Store.
5. The decoder determines the destination and originator parties, the service name, and other relevant information that helps in dispatching the message.
6. The decoder dispatches the message to the Async Dispatcher Queue.
   - If the decoder determines that this message is part of a new exchange, a new process instance will be requested.
   - If this message is part of an ongoing exchange, the decoder will request that the message be dispatched to a particular receive node within an existing process instance.

   The message parts will be packaged as appropriate for the receive node's method signature.
7. For RosettaNet, the decoder also responds to the sending trading partner that the business message was received.
8. The Async Dispatcher Module dispatches the message to the appropriate receive node on the business process.
9. Message tracking information for an inbound message is sent to the Message Tracking queue (`wli.internal.msgtracking.queue`). The message-driven bean listening to this queue will update the various message tracking tables based on the tracking level set in the Trading Partner Management module of the WebLogic Integration Administration Console. For configuration instructions, see "Configuring the Mode and Message Tracking" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Run-Time Monitoring Concepts

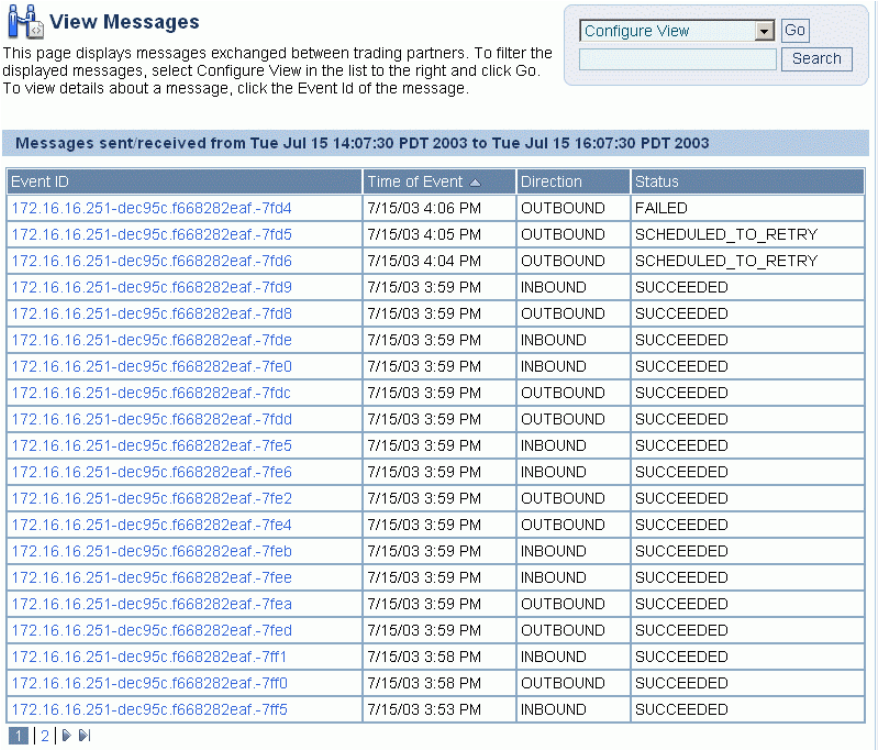
WebLogic Integration the following run-time monitoring capabilities for trading partner integration:

- Message Tracking
- Viewing Run-Time Statistics

Message Tracking

WebLogic Integration tracks the flow and state information of business messages that are sent to, or retrieved from, other trading partners. You can view run-time data and statistics on the Message Tracking module in the WebLogic Integration Administration Console to perform real-time monitoring, process analysis, troubleshooting, and reporting for business messages.

The following example shows a summary of business messages:



View Messages

This page displays messages exchanged between trading partners. To filter the displayed messages, select Configure View in the list to the right and click Go. To view details about a message, click the Event Id of the message.

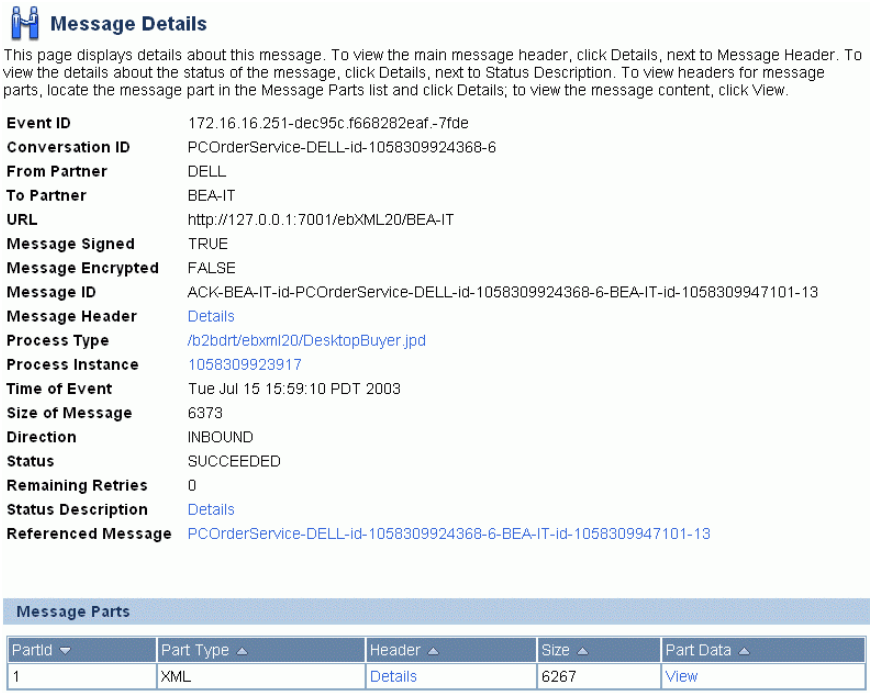
Configure View | Go | Search

Messages sent/received from Tue Jul 15 14:07:30 PDT 2003 to Tue Jul 15 16:07:30 PDT 2003

Event ID	Time of Event	Direction	Status
172.16.16.251-dec95c.f668282eaf.-7fd4	7/15/03 4:06 PM	OUTBOUND	FAILED
172.16.16.251-dec95c.f668282eaf.-7fd5	7/15/03 4:05 PM	OUTBOUND	SCHEDULED_TO_RETRY
172.16.16.251-dec95c.f668282eaf.-7fd6	7/15/03 4:04 PM	OUTBOUND	SCHEDULED_TO_RETRY
172.16.16.251-dec95c.f668282eaf.-7fd9	7/15/03 3:59 PM	INBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fd8	7/15/03 3:59 PM	OUTBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fde	7/15/03 3:59 PM	INBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fe0	7/15/03 3:59 PM	INBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fdc	7/15/03 3:59 PM	OUTBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fdd	7/15/03 3:59 PM	OUTBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fe5	7/15/03 3:59 PM	INBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fe6	7/15/03 3:59 PM	INBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fe2	7/15/03 3:59 PM	OUTBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fe4	7/15/03 3:59 PM	OUTBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7feb	7/15/03 3:59 PM	INBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fee	7/15/03 3:59 PM	INBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fea	7/15/03 3:59 PM	OUTBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7fed	7/15/03 3:59 PM	OUTBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7ff1	7/15/03 3:58 PM	INBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7ff0	7/15/03 3:58 PM	OUTBOUND	SUCCEEDED
172.16.16.251-dec95c.f668282eaf.-7ff5	7/15/03 3:53 PM	INBOUND	SUCCEEDED

1 | 2

The following example shows the details of a single business message:



Message Details

This page displays details about this message. To view the main message header, click Details, next to Message Header. To view the details about the status of the message, click Details, next to Status Description. To view headers for message parts, locate the message part in the Message Parts list and click Details; to view the message content, click View.

Event ID	172.16.16.251-dec95c.f668282eaf.-7fde
Conversation ID	PCOrderService-DELL-id-1058309924368-6
From Partner	DELL
To Partner	BEA-IT
URL	http://127.0.0.1:7001/ebXML20/BEA-IT
Message Signed	TRUE
Message Encrypted	FALSE
Message ID	ACK-BEA-IT-id-PCOrderService-DELL-id-1058309924368-6-BEA-IT-id-1058309947101-13
Message Header	Details
Process Type	/b2bdrt/ebxml20/DesktopBuyer.jpd
Process Instance	1058309923917
Time of Event	Tue Jul 15 15:59:10 PDT 2003
Size of Message	6373
Direction	INBOUND
Status	SUCCEEDED
Remaining Retries	0
Status Description	Details
Referenced Message	PCOrderService-DELL-id-1058309924368-6-BEA-IT-id-1058309947101-13

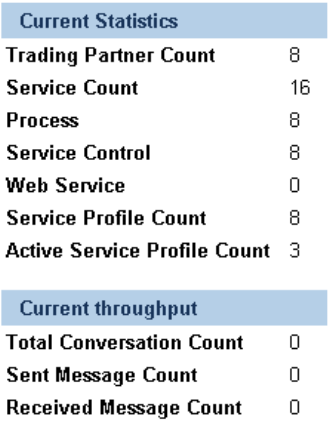
Message Parts

PartId	Part Type	Header	Size	Part Data
1	XML	Details	6267	View

WebLogic Integration maintains message history information in the run-time repository. You configure the message tracking level for individual services profiles, as described in "Adding Service Profiles to a Service" in Trading Partner Management in *Managing WebLogic Integration Solutions*. WebLogic Integration maintains message history information in the run-time database, for which you can schedule periodic archives and purges. For more information about message tracking, see "Monitoring Messages" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Viewing Run-Time Statistics

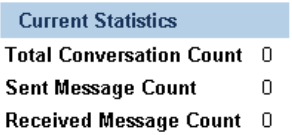
The Statistics module in the WebLogic Integration Administration Console displays run-time statistics for the system and for service profiles. The following example shows system summary statistics:



Current Statistics	
Trading Partner Count	8
Service Count	16
Process	8
Service Control	8
Web Service	0
Service Profile Count	8
Active Service Profile Count	3

Current throughput	
Total Conversation Count	0
Sent Message Count	0
Received Message Count	0

The following example shows service profile statistics:



Current Statistics	
Total Conversation Count	0
Sent Message Count	0
Received Message Count	0

For more information, see "Viewing Statistics" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Summary of Trading Partner Integration Phases

This topic provides an overview of the phases, tasks, and tools involved in implementing trading partner integration solutions. It includes the following sections:

- Phase 1: Plan the Solution
- Phase 2: Design, Build, and Test the Solution
- Phase 3: Deploy the Solution
- Phase 4: Administer and Tune the Solution

Phase 1: Plan the Solution

The first phase is to plan the solution, which involves:

- Determining the trading partners with which you want to integrate.
- Define the business processes that you want to implement with each trading partner.
- Determine the business protocol(s) used to communicate with each trading partner.

For more information, see the following topics:

- *Designing WebLogic Integration Solutions*, which is available at the following URL:

  `http://edocs.bea.com/wli/docs81/design/index.html`
- "Planning the RosettaNet Solution" on page 3-19
- "Planning the ebXML Solution" on page 2-12

Phase 2: Design, Build, and Test the Solution

Once you understand the high-level requirements for your trading partner integration solution, you design, build, and test the solution using the following tools:

- WebLogic Workshop, which provides a unified programming model and run-time framework to provide end-to-end business process integration via easily implemented controls and templates. Before you begin building the solution, it is recommended that you complete the following tutorials so that you know how to build business processes and create data transformations in WebLogic Workshop:
  - Tutorial: Building Your First Business Process, which is available at:

http://edocs.bea.com/workshop/docs81/doc/en/integration/tutorial/tutWLIProcessIntro.html

  - Tutorial: Building Your First Data Transformation, which is available at:

    http://edocs.bea.com/workshop/docs81/doc/en/integration/dttutorial/tutWLIDataTransIntro.html

- BEA WebLogic Platform Configuration Wizard, which you use to create a domain for your RosettaNet solution based on the WebLogic Integration domain template. The Configuration Wizard automatically populates the Trading Partner Management (TPM) repository with default trading partners and protocol bindings (described in "Default TPM Repository Settings" on page 1-10) and default security settings (described in "Default Domain Security Configuration" on page 4-8). For more information about the Configuration Wizard, see *Creating WebLogic Configurations Using the Configuration Wizard* in the WebLogic Platform documentation.

- XMLSPY (optional), which is packaged with WebLogic Platform, to convert the RosettaNet PIP DTDs to XSD files, if you want to use the WebLogic Workshop visual mapper tools for data transformation. For more information, see "Converting RosettaNet DTD Schemas to XSD Schemas" in "Tutorial: Building RosettaNet Solutions" in *Tutorials for Trading Partner Integration*, which is available at:

  http://dev2dev.bea.com/code/wli.jsp

For more information about the tasks associated with each business protocol, see the following topics:

- "Building the RosettaNet Solution" on page 3-20
- "Building the ebXML Solution" on page 2-12

Phase 3: Deploy the Solution

Once you have designed, built, and tested the trading partner integration solution, you deploy it in a production environment using the following tools:

- WebLogic Integration Administration Console, which provides sophisticated trading partner management capabilities, enabling administrators to easily manage a central repository of trading partner profile information, including protocol bindings used for secure message exchanges between trading partners, services representing public processes, security, and bulk import / export capabilities. For detailed instructions on using the information about using the WebLogic Integration Administration Console, see *Managing WebLogic Integration Solutions*, which is available at the following URL:

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

- WebLogic Server Administration Console, which provides crucial functionality for configuring and managing WebLogic domains, including clustering, high availability / fail-over, security, application deployment, and other key services. For detailed information, see the "Administration Console Online Help" at the following URL:

  `http://edocs.bea.com/wls/docs81/ConsoleHelp/index.html`

- Bulk Loader, which provides import / export capabilities for trading partner information stored in the TPM repository. For instructions on using the bulk loader, see "Bulk Loader" in *Managing WebLogic Integration Solutions* at the following URL:

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

For more information, see the following topics:

- For comprehensive information about deploying WebLogic Integration, see *Deploying WebLogic Integration Solutions*, which is available at the following URL:

  `http://edocs.bea.com/wli/docs81/deploy/index.html`

- For detailed information about resources that you deploy in the production environment, see "Trading Partner Integration Resources" in the "Introduction" to *Deploying WebLogic Integration Solutions*, which is available at the following URL:

  `http://edocs.bea.com/wli/docs81/deploy/intro.html`

- For more information about the tasks associated with each business protocol, see the following topics:
  - "Deploying the RosettaNet Solution" on page 3-21
  - "Deploying the ebXML Solution" on page 2-14

Phase 4: Administer and Tune the Solution

Once you have deployed a trading partner integration solution in a production environment, you use the same tools—the WebLogic Integration Administration Console and the WebLogic Server Administration Console—to monitor and tune your deployment.

For more information, see the following topics:

- *Managing WebLogic Integration Solutions*, which is available at the following URL:

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

- "Managing the RosettaNet Solution" on page 3-23
- "Managing the ebXML Solution" on page 2-15

Next Steps

At this point, you can proceed to the following topics:

- For an introduction to using the ebXML business protocol for trading partner integration, see Chapter 2, “Introducing ebXML Solutions.”
- For an introduction to using the RosettaNet business protocol for trading partner integration, see Chapter 3, “Introducing RosettaNet Solutions.”
- For an overview of security in trading partner integration solutions, see Chapter 4, “Trading Partner Integration Security.”

CHAPTER 2

Introducing ebXML Solutions

This topic provides an introduction to implementing ebXML solutions with WebLogic Integration. It contains the following sections:

- About ebXML Solutions
- ebXML Concepts
- ebXML Business Processes
- Tasks for Implementing an ebXML Solution

This topic focuses on information that is relevant to ebXML solutions only. Before you begin, be sure to read the Chapter 1, "Introduction" to learn basic concepts for integrating trading partners using WebLogic Integration. In addition, for a hands-on walkthrough of building example ebXML solutions, see "Tutorial: Building ebXML Solutions," in *Tutorials for Trading Partner Integration*, which is available at:

- Documentation and Example Files

  `http://dev2dev.bea.com/code/wli.jsp`

- Documentation Only

  `http://edocs.bea.com/wli/docs81/tptutorial/ebxml.html`

About ebXML Solutions

An *ebXML solution* is any WebLogic Integration solution that involves exchanging business messages with trading partners using the ebXML business protocol. This topic describes ebXML and how it is supported in WebLogic Integration. It contains the following sections:

- About ebXML
- ebXML Support in WebLogic Integration
- Interoperability with WebLogic Integration – Business Connect

About ebXML

The ebXML business protocol is sponsored by UN/CEFACT and OASIS. The ebXML Web site (`http://www.ebxml.org`) describes ebXML as "a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet. Using ebXML, companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes."

ebXML Specifications

WebLogic Integration supports the following ebXML specifications:

- *ebXML Message Service Specification v2.0*
- *ebXML Message Service Specification v1.0*

These specification defines the message envelope and header document schema used to transfer ebXML messages with a communication protocol such as HTTP. They provide a set of layered extensions to the base Simple Object Access Protocol (SOAP) and SOAP Messages with Attachments specifications. The ebXML Message Service provides security and reliability features that are not provided in the specifications for SOAP and SOAP Messages with Attachments. For more information about these and other ebXML specifications, see the `http://www.ebxml.org/specs/index.htm` page on the ebXML web site.

SOAP Specifications

Information about SOAP, including the following documents, can be found at the World Wide Web Consortium (W3C) Web site (`http://www.w3c.org`):

- *Simple Object Access Protocol (SOAP) 1.1*at the following URL:

http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

- *SOAP Messages with Attachments* at the following URL:

  http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211

ebXML Support in WebLogic Integration

This topic describes supported and unsupported ebXML features in this release of WebLogic Integration.

Supported ebXML 1.0 and 2.0 Features

WebLogic Integration supports the following ebXML 1.0 and ebXML 2.0 features:

Table 2-1 WebLogic Integration Support for ebXML 1.0 and 2.0 Features

Feature	Support
Packaging	For ebXML 1.0 and 2.0—SOAP, SOAP headers, and attachments
Security	• Transport-level security: HTTP/S • Message-level security: digital signatures (XML DSig) for protection of the entire message from message tampering (ebXML 1.0 and 2.0). For more information, see Chapter 4, "Trading Partner Integration Security."
Reliable Messaging	• ebXML 1.0: Best effort, and Once and only once • ebXML 2.0: At least once, At most once, Best effort, and Once and only once For more information, see "Reliable Messaging" on page 2-6.
Clustering, High Availability, and Recovery	For ebXML 1.0 and 2.0

Unsupported ebXML 2.0 Features

This release of WebLogic Integration does not support certain optional features of ebXML2.0, including:

- Message Status Service

- Synchronous reply mode of communication
- Message Order Module
- Multi-Hop Module

This release does not provide XML DSIG at each payload level of an ebXML message.

Interoperability with WebLogic Integration – Business Connect

WebLogic Integration – Business Connect is a lightweight client that implements the ebXML business protocol. WebLogic Workshop business processes can exchange ebXML business messages with trading partners that use WebLogic Integration – Business Connect. For more information about WebLogic Integration – Business Connect, see the following URL:

`http://edocs.bea.com/wlibc/docs81/index.html`

ebXML Concepts

This topic describes ebXML concepts that you need to understand before implementing ebXML business processes in WebLogic Integration. It contains the following sections:

- ebXML Protocol Layer
- ebXML Business Messages
- Reliable Messaging

ebXML Protocol Layer

The ebXML protocol layer provides the ability to send and receive messages via the Internet according to the ebXML Message Service specifications for transport, message packaging, and security. The ebXML 1.0 and 2.0 message service specifications are independent of the communication protocol used. WebLogic Integration supports the HTTP(S) communication protocol.

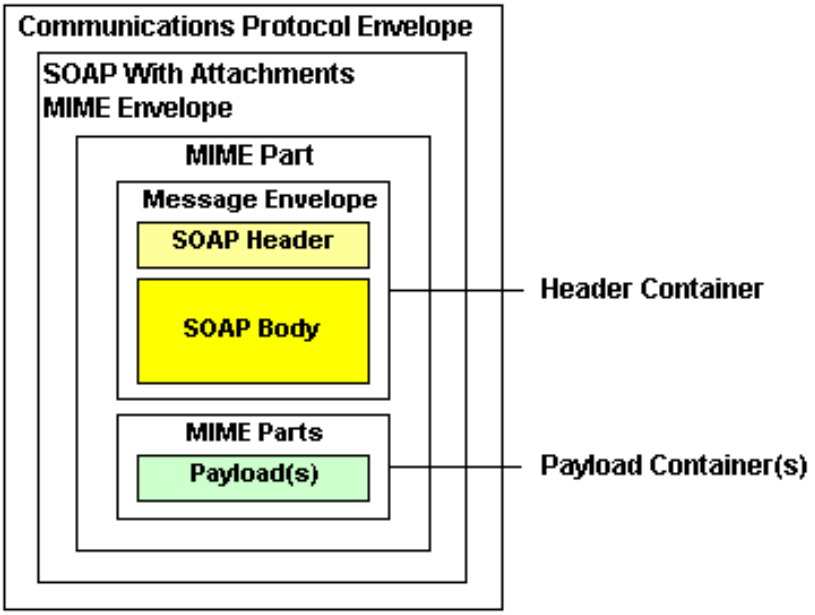
ebXML Business Messages

A business message is the basic unit of communication between trading partners. Business messages are exchanged as part of a conversation. The roles in a conversation are implemented by business processes, which choreograph the exchange of business messages.

Diagram of an ebXML Business Message

The following figure represents the structure of a business message exchanged in a conversation based on the ebXML business protocol.

Figure 2-1 ebXML Business Message



An ebXML business message contains one XML business document and one or more attachments. An ebXML message is a communication protocol-independent MIME/Multipart message envelope, referred to as a *message packag*e. All message packages are structured in compliance with the SOAP Messages with Attachments specification.

Logical MIME Parts of an ebXML Business Message

The message package shown in the preceding figure illustrates the following logical MIME parts:

- **Header Container**—Logical container in which one SOAP 1.1-compliant message is stored. This SOAP message is an XML document; its root element is the SOAP Envelope, which, in turn, contains the following elements:
  - **SOAP Header**—Contains ebXML-specific header elements, including the ebXML `MessageHeader` element that specifies details such as from and to business IDs, service that relates to the business process, and action that relates to a node in the

business process. The SOAP Header is a generic mechanism for adding features to a SOAP message.

  - **SOAP Body**—Container for message service handler control data and information related to the payload parts of the message.

- **Payload Container**—Zero or more payloads. Each payload can contain XML or non-XML (binary) data.

WebLogic Integration provides a mechanism in WebLogic Workshop business processes for retrieving the ebXML message envelope that relates to the Header container from incoming business messages. For more information, see jpd:ebxml-method Tag and jc:ebxml Tag in the WebLogic Workshop Help.

Message Attachments

An ebXML message can have any combination of payloads. The payloads can be all binary, all XML, or a mixture of both. Any payload is sent as a MIME attachment to the SOAP message—the SOAP body is not used to carry the payload.

WebLogic Integration provides a `MessageAttachment[]` data type that business processes can use to retrieve payloads from an ebXML message, particularly when payloads consist of mixed data types or when the number of payloads or the order of payloads is not known in advance. It provides methods for determining the content of a payload (`isXmlObject` and `isRawData`) and retrieving the contents of the payload (`getXmlObject` and `getRawData`) as non-typed XML data (`XmlObject` data type) or non-XML data (`RawData` data type).To learn about working with `MessageAttachment` objects, see Message Attachments in the WebLogic Workshop Help.

Reliable Messaging

The ebXML business protocol supports reliable messaging, an optional but important capability that allows you to configure different levels of quality of delivery service. Reliable messaging has a reliability versus performance trade-off, as increasing the level of guarantee increases run-time requirements on system resources. You configure reliable messaging in the WebLogic Integration Administration Console, as described in “Defining an ebXML 1.0 or 2.0 Binding” in “Defining Protocol Bindings” in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Of the eight qualities of service policies defined in the ebXML 2.0 Specification, WebLogic Integration supports the following four (non-multihop) policies, which determine how acknowledgements and duplicate messages are handled:

Table 2-2 Supported Reliable Messaging Policies

Policy	Description
Best Effort (ebXML 1.0 and 2.0)	Best effort. No reliable messaging (no acknowledgement or duplicate elimination).
Once and Only Once (ebXML 1.0 and 2.0)	Requires acknowledgement and duplicate elimination. If a message is not acknowledged, it is resent. If a duplicate message is received, it is ignored. If this policy is selected, you can specify: • Number of Retries—Number of times WebLogic Integration resends a message in specific situations, such as timeouts, network failures, and so on. • Interval—Number of seconds that WebLogic Integration waits before trying to resend a message. • Persistence Duration—(Optional) Amount of time the message will be kept in the repository. After this time, the message will be deleted and any subsequent message sent with same ID will *not* be considered a duplicate. If specified, the configured persistence duration must be greater than the internally set TimeToLive, which is calculated as: ((Retries +1) * RetryInterval) < TimeToLive < Persistence Duration
Atleast Once (ebXML 2.0 only)	Requires acknowledgement, but not duplicate elimination. If selected, you can specify the number of retries and persistence duration.
Atmost Once (ebXML 2.0 only)	Requires duplicate elimination, but not acknowledgement.

ebXML Business Processes

This topic describes WebLogic Workshop business processes that implement ebXML conversations. It contains the following sections:

- Guidelines for Building ebXML Business Processes
- ebXML Initiator Business Processes
- ebXML Participant Business Processes

Guidelines for Building ebXML Business Processes

When designing business processes for ebXML solutions, consider the following guidelines:

- You should thoroughly understand the content, choreography, roles, and other aspects of the business processes that you want to implement. For example, you should understand:
  - the design pattern
  - the number and nature of business messages exchanged
  - the contents of each business message
  - the success path for the conversation (such as request / response sends and receipt acknowledgements)
  - the possible failure paths, such as retries, timeouts, and errors
- You should thoroughly understand data transformations, if any, that occur between the ebXML business message and back-end systems. For each transformation, you use a transformation control to convert the service content of the ebXML business message to / from the format(s) used in the private processes. For more information, see "Creating a Transformation Control and a Transformation Method" in the WebLogic Workshop Help.
- Initiator business processes can have multiple conversations with different participants. Within a single business process, *each* ebXML conversation requires its own *separate* ebXML control instance. If the conversation needs to involve a different participant, or if it involves the same participant with different reliable messaging or security, create a different ebXML control JCX file. Each ebXML control JCX in the initiator business process and each business process JDP on participant side represents a service in the TPM repository.
- Decide which action mode you want to use for the initiator and participant business processes. The action mode determine the value specified in the `eb:Action` element in the message header of the ebXML message, which becomes important in cases where multiple message exchanges occur within the same conversation. You can use one of the following values in the `ebxml-action-mode` business process property:
  - `default`—Sets the `eb:Action` element to `SendMessage` (default name) and `onMessage` is the control callback method name.
  - `non-default`—Sets the `eb:Action` element to the name of the method (on the ebXML control) that sends the message in the initiator business process. For sending a message from the initiator to the participant, this name must match the method name of the **Client Request** node in the corresponding participant business process. For sending a message from the participant to the initiator, the method name in the callback

interface for the client callback node in the participant business process must match the method name (on the ebXML control) in the control callback interface in the initiator business process.

Using `non-default` is recommended to ensure recovery and high availability.
If unspecified, the `ebxml-action-mode` element is set to `non-default`.

- Determine how you want to specify the business process properties:
  - Statically—using hard-coded annotations (jc:ebxml Tag on the ebXML control in initiator business processes and the @jpd:ebxml Annotation in participant business processes).
  - Dynamically—using pass-in values, such as business IDs, and calling the `setProperties` method.

  Note: To ensure proper routing, the ebXML service name specified in the initiator and participant business processes must match. In addition, for non-default action mode, the method names in the ebXML control instance in the initiator business process must match the method names in the corresponding participant business process.

- To explicitly handle acknowledgements and errors, you can use `onAck` and `onError` nodes.
  - For default Action mode, you must use an Event Choice for these nodes, as the order in which acks and messages arrive is not guaranteed. For more information about these constructs, see "Tutorial: Building ebXML Solutions" in *Tutorials for Trading Partner Integration* (available at `http://dev2dev.bea.com/code/wli.jsp`).
  - For non-default Action mode, these nodes can be modeled in any form, including sequentially.

- To retrieve message envelopes for incoming business messages, use the jpd:ebxml-method Tag (for participant business processes) and jc:ebxml Tag (on the ebXML control JCX for initiator business processes). To retrieve message envelopes for outgoing business messages, you specify the return value of the send method as `XMLObject` and provide explicit casting in the business process, as in the following example:
  - For the initiator business process:

```
public XmlObject send( messageAttachement msg )
XmlObject envelope = (XmlObject) control.send(msg)
```

  - For the participant business process:

```
public XmlObject send( messageAttachement msg )
XmlObject envelope = (XmlObject) callback.send(msg)
```

- Thoroughly test the implementation by simulating the choreography between the initiator and participant business processes with sample data. By default, the Trading Partner Management module runs in Test (development) mode, which allows you to run business processes on the same machine (collocated) using preconfigured trading partners and a default binding that uses the ebXML 2.0 protocol. The endpoints for both partners use `127.0.0.1:7001`. The delivery-semantics is Once and Only Once. The ebxml service name can be anything as long as it is same for both the initiator and participant business processes. For more information, see "Default TPM Repository Settings" on page 1-10.

ebXML Initiator Business Processes

In WebLogic Integration, initiator business process use an ebXML control to enable WebLogic Workshop business processes to exchange business messages and data with trading partners via ebXML. You use ebXML controls *only* in initiator business processes to manage the exchange of ebXML business messages with participants. The ebXML control provides methods for sending business messages, acknowledgements, and errors, and it provides callback methods to handle responses from participants.

For detailed information about using the ebXML control in business processes, see the following topics in ebXML Control in the WebLogic Workshop Help:

- Overview: ebXML Control
- Creating an ebXML Control
- Using an ebXML Control
- EBXMLControl Interface
- jc:ebxml Tag

For an introduction to initiator business processes, see "Initiator and Participant Business Processes" on page 1-14.

ebXML Participant Business Processes

In WebLogic Integration, you can easily create a new ebXML participant business process using a WebLogic Workshop template, the ebXML participant business process file. This template provides a head start for building public participant business processes for ebXML conversations. Although this file is not required to build ebXML participant business processes, it includes the nodes and business process annotations needed to integrate easily with ebXML initiator business processes, as well as standard choreography patterns such as acknowledgements, time-outs,

retries, and errors. For information about using participant business processes, see Building ebXML Participant Business Processes and @jpd:ebxml Annotation in the WebLogic Workshop Online Help. For an introduction to participant business processes, see "Initiator and Participant Business Processes" on page 1-14.

Tasks for Implementing an ebXML Solution

This topic provides a high-level, end-to-end overview of the tasks involved with implementing an ebXML solution. It includes the following topics:

- Before You Begin
- Planning the ebXML Solution
- Building the ebXML Solution
- Deploying the ebXML Solution
- Managing the ebXML Solution

Note: This topic describes, in a general way, the tasks that are *typically* involved in implementing an ebXML solution. The process of implementing ebXML solutions is iterative, and it can vary in scope and sequence depending on your unique business requirements and environment.

Before You Begin

Before you begin implementing an ebXML solution, we recommend that you review the following documents:

- Chapter 1, "Introduction," to learn basic concepts for integrating trading partners using WebLogic Integration.
- This chapter, to learn basic concepts that are unique to ebXML solutions.
- "Tutorial: Building ebXML Solutions" in *Tutorials for Trading Partner Integration*, which is available at:

  `http://dev2dev.bea.com/code/wli.jsp`

Planning the ebXML Solution

Once you have decided to use ebXML as the business protocol for your trading partner integration (as described in "Phase 1: Plan the Solution" on page 1-28), you need to plan the solution by determining certain factors in your implementation:

- With which trading partners will you integrate?
- For each trading partner, you need to determine:
  - Which business process(es) will you integrate?
  - What information about that trading partner is required for your implementation (for example, their business ID and other basic profile information). For more information, see "Trading Partner Profiles" on page 1-6.
- For each business process, you need to determine:
  - Which ebXML version (1.0 or 2.0) is required for exchanging business messages?
  - What role (initiator or participant role) will your organization play in the business message exchange?
  - Do you need to create business processes for just your role, or for both roles in the business process?
  - What are the back-end integration requirements for each business process? Will you use private business processes, and if so, what private business processes are required? What is the data format of, and connections with, associated back-end systems?
  - Does your implementation need to meet legal standards for nonrepudiation, which is described in "NonRepudiation" on page 4-37?
  - What qualities of service are required for reliable messaging, as described in "Reliable Messaging" on page 2-6?

For more information about planning an integration solution, see *Designing WebLogic Integration Solutions*, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/design/index.html`

Building the ebXML Solution

After planning your ebXML solution, you build the business processes that implement the design patterns you are using. For more information about design-time tools, see "Phase 2: Design,

Build, and Test the Solution" on page 1-28. For conceptual information about ebXML business processes, see "ebXML Business Processes" on page 2-7.

The "Tutorial: Building ebXML Solutions" in *Tutorials for Trading Partner Integration* (available at http://dev2dev.bea.com/code/wli.jsp) provides a detailed walkthrough of the typical tasks and design patterns that are required to build an ebXML solution.

To build an ebXML solution, you would typically complete the following steps:

1. Using the BEA WebLogic Platform Configuration Wizard, create a new domain based on the WebLogic Integration domain template. For instructions, see *Creating WebLogic Configurations Using the Configuration Wizard* in the WebLogic Platform documentation.
2. From the *Tutorials for Trading Partner Integration*, copy the example implementation associated with the design pattern that you want to use, if available.
3. Change the ebXML annotations for the new business processes:
   - For initiator business processes, you change the `ebxml-service-name` attribute (and others if needed) in the @jc:ebXML Tag, which is described at the following URL:

     `http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsebXMLTag.html`
   - For participant business processes, you change the `ebxml-service-name` attribute (and others if needed) in the @jpd:ebxml Annotation, which is described at the following URL:

     `http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refJpdAnnotations_ebxml.html`
4. Rename the JPD and JCX files and change the names of other components to be more descriptive of the new business process implementation, if you want.
   - For more information about modifying the ebXML control in initiator business processes, see ebXML Control in the WebLogic Workshop Online Help, which is available at:

     `http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsebXML.html`
   - For more information about building ebXML participant business processes, see Building ebXML Participant Business Processes in the WebLogic Workshop Help, which is available at the following URL:

     `http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideEbXML.html`

5. Ensure that the ebXML Service name is the same for both the initiator and participant business processes.
6. For non-default Action mode, ensure that method names are the same for both the initiator and participant business processes.
7. Change the implementation of any back-end integration as needed.
8. Make any other changes to the business processes as needed.
9. Test the ebXML solution using the default TPM repository configuration (described in "Default TPM Repository Settings" on page 1-10) and security settings (described in "Default Domain Security Configuration" on page 4-8).

Deploying the ebXML Solution

Once you have built and tested your ebXML solution, you deploy the solution in a production environment. For more information about deployment tools, see "Phase 3: Deploy the Solution" on page 1-29. For detailed information about deploying WebLogic Integration solutions, see *Deploying WebLogic Integration Solutions*, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/deploy/index.html`

To deploy an ebXML solution, you would typically complete the following steps:

1. If necessary, using the BEA WebLogic Platform Configuration Wizard, create a new domain based on the WebLogic Integration domain template. For instructions, see *Creating WebLogic Configurations Using the Configuration Wizard* in the WebLogic Platform documentation.
2. Start WebLogic Server in production mode.
3. Using the WebLogic Integration Administration Console, complete the following tasks:

   Note: If you have already defined trading partner information in your development environment, you can export this information to an external file, and then import this file into the production environment. For instructions, see "Exporting Management Data" and "Importing Management Data" in Trading Partner Management.

   a. Add trading partners to the TPM repository, including basic profile information, such as trading partner IDs. For instructions, see "Adding Trading Partner Profiles" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

   b. For each trading partner, define the protocol bindings (including ebXML version—1.0 or 2.0—and other settings) to be used for message exchanges with this trading partner.

For instructions, see "Defining an ebXML 1.0 or 2.0 Binding" in Trading Partner Management.

c. For protocol bindings, you can (optionally) define signature transforms, as described in "Configuring Signature Transforms for ebXML Bindings" in Trading Partner Management.

d. Define the services that you will use in your deployment. For instructions, see "Adding Services" in Trading Partner Management.

e. For each trading partner, define the service profiles (protocol bindings and URL endpoints for local and remote trading partners) associated with each service. For instructions, see "Adding Service Profiles to a Service" in Trading Partner Management.

4. Using the WebLogic Server Administration Console and WebLogic Integration Administration Console, implement security for your deployment, as described in "Implementing Security for Trading Partner Integration" on page 4-48.

5. Configure your domain (clustering, high availability, load balancing, fail-over, and so on) as needed for your production environment. For instructions, see *Deploying WebLogic Integration Solutions*, which is available at the following URL:

   `http://edocs.bea.com/wli/docs81/deploy/index.html`

6. Deploy the application and other WebLogic Integration resources associated with your ebXML solution. For instructions, see *Deploying WebLogic Integration Solutions*, which is available at the following URL:

   `http://edocs.bea.com/wli/docs81/deploy/index.html`

Managing the ebXML Solution

Once you have deployed your ebXML solution, you would typically monitor run-time performance, message volumes, resource utilization, and other factors to ensure optimum operation on your solution. For more information about monitoring tools, see "Phase 4: Administer and Tune the Solution" on page 1-30.

For instructions on monitoring trading partner integration resources, see the following topics in in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Viewing Statistics"
- "Monitoring Messages"

CHAPTER 3

Introducing RosettaNet Solutions

This topic provides an introduction to implementing RosettaNet solutions with WebLogic Integration. It contains the following sections:

- About RosettaNet Solutions
- RosettaNet Concepts
- RosettaNet Business Processes
- Tasks for Implementing a RosettaNet Solution

This topic focuses on information that is relevant to RosettaNet solutions only. Before you begin, be sure to read the Chapter 1, "Introduction" to learn basic concepts for integrating trading partners using WebLogic Integration. In addition, for a hands-on walkthrough of example RosettaNet solutions, see "Tutorial: Building RosettaNet Solutions," in *Tutorials for Trading Partner Integration*, which is available at:

- Documentation and Example Files

  `http://dev2dev.bea.com/code/wli.jsp`

- Documentation Only

  `http://edocs.bea.com/wli/docs81/tptutorial/rosettanet.html`

About RosettaNet Solutions

A *RosettaNet solution* is any WebLogic Integration solution that involves exchanging business messages with trading partners using the RosettaNet business protocol. This topic describes RosettaNet and how it is supported in WebLogic Integration. It contains the following sections:

About RosettaNet

RosettaNet is a business protocol that enables enterprises to conduct business over the Internet. The RosettaNet Consortium (`http://www.rosettanet.org`) is an independent, nonprofit consortium of major information technology, electronic component, and semiconductor manufacturing companies working to create and implement industry-wide, open process standards. These processes are designed to standardize the electronic business interfaces used between participating supply chain partners. The *RosettaNet Implementation Framework* (RNIF) specification (available at `http://www.rosettanet.org/rnif`) is a guideline for applications that implement RosettaNet *Partner Interface Processes* (PIPs). These PIPs are standardized electronic business processes used between trading partners. For a list of PIPs, see `http://www.rosettanet.org/pipdirectory`.

Understanding RosettaNet

The following RosettaNet documents are necessary reading if you want to implement your own PIP using the support for RosettaNet provided by WebLogic Integration, and recommended reading if you want to fully understand the sample RosettaNet PIP implementations. These documents are available at `http://www.rosettanet.org/standards`:

- RosettaNet Implementation Framework v1.1 (RNIF 1.1) —The RNIF is an open, common networked-application framework designed to allow RosettaNet supply chain and solution partners to collaborate in executing RosettaNet PIPs.
- RosettaNet Implementation Framework v2.0 (RNIF 2.0)
- RNIF Technical Advisories—RNIF Technical Advisories are updates and additional information for RNIF 1.1 and RNIF 2.0.
- RNIF Technical Recommendations—Technical Recommendations describe features or enhancements not yet available in a published version of the RNIF v1.1. Implementation of Technical Recommendations is optional.

- RNIF Business Signals, Service Header & Preamble—The RNIF business signals, service header & preamble document contains message guidelines and XML document type definitions (DTDs) for the RNIF business signals, service header, and preamble.
- Understanding a PIP Blueprint—reference for PIP blueprint components and evaluation. Available under Supporting Documents in the Standards Section.
- PIPs of interest—PIPs are specialized system-to-system, XML-based dialogs that define business processes between supply chain companies. Each PIP includes a technical specification based on the RosettaNet Implementation Framework (RNIF), a Message Guideline document with a PIP-specific version of the Business Dictionary, and XML document type definitions (DTDs) for PIP-specific messages.

RosettaNet Support in WebLogic Integration

This topic describes supported and unsupported RosettaNet features in this release of WebLogic Integration.

Supported RosettaNet 1.1 and 2.0 Features

WebLogic Integration supports the following RosettaNet 1.1 and RosettaNet 2.0 features:

Table 3-1 WebLogic Integration Support for RosettaNet 1.1 and 2.0 Features

Feature	Supported (X=Yes)	
	RNIF 1.1	RNIF 2.0
Packaging		
• RosettaNet Objects (RNO)	X	
• RosettaNet Business Message (RBM) with standard multipart MIME / headers		X
• Attachments	X	X
Security		
• Transport-level security (HTTP/S)	X	X
• Message-level security (digital signatures) to protect messages from tampering	X	X
• Standard S/MIME and encryption for message privacy		X

Table 3-1 WebLogic Integration Support for RosettaNet 1.1 and 2.0 Features (Continued)

Feature	Supported (X=Yes)	
	RNIF 1.1	RNIF 2.0
Message Handling		
• Duplicate message removal	X	X
• Message persistence	X	X
• High-level ACK	X	X
• High-level retry	X	X
• Message status returned	X	X
• Explicit choreography.	X	X
Performance and Availability		
• Clustering	X	X
• High Availability	X	X
• Recovery	X	X

Unsupported RosettaNet Features

This release of WebLogic Integration does not support the following RNIF 2.0 features:

Table 3-2 Unsupported RosettaNet Features

Feature	Support
Synchronous response messages	This WebLogic Integration release supports asynchronous one-action and two-action activities only. It does not support synchronous one-action and two-action activities.
Use of SMTP transport with RNIF 2.0	While strongly biased toward HTTP transport, the RNIF 2.0 is transport independent and includes documentation showing how SMTP transport might be used. This WebLogic Integration release supports HTTP and HTTPS transport but not SMTP.

RosettaNet Concepts

This topic describes RosettaNet concepts that you need to understand before implementing PIPs in WebLogic Integration. It contains the following sections:

RosettaNet Protocol Layer

The RosettaNet protocol layer provides the ability to send and receive messages by way of the Internet according to the RNIF 1.1 and RNIF 2.0 specifications for transport, message packaging, and security.

When a WebLogic Integration trading partner receives a RosettaNet message, the transport servlet forwards the message to the RosettaNet decoder. The RosettaNet decoder processes the protocol-specific message headers, identifies the trading partner that sent the message, and forwards the RosettaNet message to a JMS queue. When a WebLogic Integration trading partner sends a RosettaNet message, the RosettaNet encoder takes the message from the send-side JMS outbound event queue and forwards it to the transport service.

Partner Interface Processes (PIPs)

RosettaNet PIPs define the *public processes* in which trading partners participate while performing transactions. A PIP defines the roles, choreography, contents of business messages, and other design details for a particular RosettaNet message exchange. For example, PIP 3A2 (Query Price and Availability) defines the process that a *Customer* trading partner performs with a *Product Supplier* trading partner to get information about the price and availability of goods that the *Customer* wants to buy and the *Product Supplier* wants to sell. Trading partners participating in PIPs need to implement the public process defined by their role in the PIP, and they need to connect their internal systems, as well as their private processes and business processes, to the public process.

Public and Private Business Processes

RosettaNet business processes follow the RosettaNet design strategy of separating public and private business processes. Public business processes provide for the exchange of business messages between trading partners, while private business processes interact with internal, back-end systems. Public business processes have standardized, highly structured PIP choreographies, while private business processes are highly customized to a trading partner's internal environment. Private processes communicate with public processes via well-defined interfaces, typically based on JMS queues. For more information, see "Public and Private Business Processes" on page 1-16.

PIP Design Patterns

RosettaNet PIPs follow one of the following design patterns:

- Asynchronous single-action activity
- Asynchronous two-action activity
- Synchronous single-action /two-action activity

In WebLogic Integration, RosettaNet PIPs are implemented as public business processes. Because the RosettaNet PIPs follow just a few general design patterns, once you have implemented a single PIP, you can easily implement other PIPs with similar design patterns by copying the implementation and making a few changes (such as changing the business message schema definitions and business process properties).

For more information about RosettaNet design patterns and choreography, see the *RosettaNet Implementation Framework Core Specification* (version V02.00.01) at `http://www.rosettanet.org/rnif`.

Asynchronous Single-Action Activity

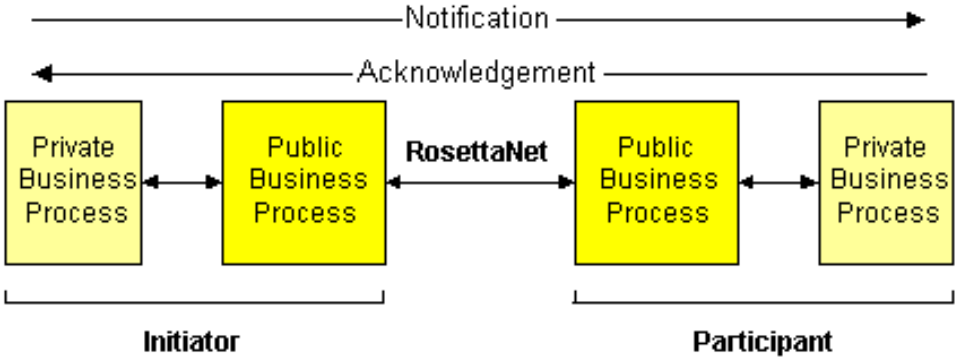
The asynchronous single-action (or one-action) activity design pattern involves a single action (business message) from the sender and a signal (a protocol response, such as an acknowledgement, ack-reject, or error) from the recipient back to the sender:

1. The initiator sends a business message to the participant.
2. Upon receiving the request business message, the participant sends a receipt acknowledgement to the initiator.

This design pattern is typical of one-way sends with acknowledgements, such as notifications from one trading partner to another. An example of this design pattern in PIP 3B2 (Advance Shipment Notice Example), which is described at http://www.rosettanet.org/PIP3B2.

The following figure shows an example of how, in WebLogic Integration, public and private business processes might be involved in an asynchronous one-action collaboration.

Figure 3-1 Public and Private Business Processes in an Asynchronous One-Action Activity Design Pattern



In this sample scenario, the message flow proceeds along the following path:

1. The initiator's private business process creates the notification document (in some internal data format) and submits it to the public business process.
2. The initiator's public business process receives the notification document, converts it to the appropriate PIP format, and sends it to the participant trading partner.

   Note: For outbound and inbound messages, WebLogic Integration automatically handles the packaging of non-payload portion of RosettaNet business messages (such as the version, content length, headers, and digital signatures), as well as transport-level security and message-level security.
3. On the participant side, WebLogic Integration handles the process of receiving the RosettaNet business message from the initiator, validating the contents of the inbound business message, and forwarding the document to the appropriate public business process.
4. The participant's public business process receives the notification document from the initiator and sends an acknowledgement of receipt to the initiator.

5. The participant's public business process converts the document from the PIP format to the appropriate internal data format, and then submits the document to the private business process.

Alternatively, the private business processes on either end could handle the conversion from the private data format to the appropriate RosettaNet PIP. In addition, private and public business processes might use internal means to indicate whether or not the overall process succeeded.

Asynchronous Two-Action Activity

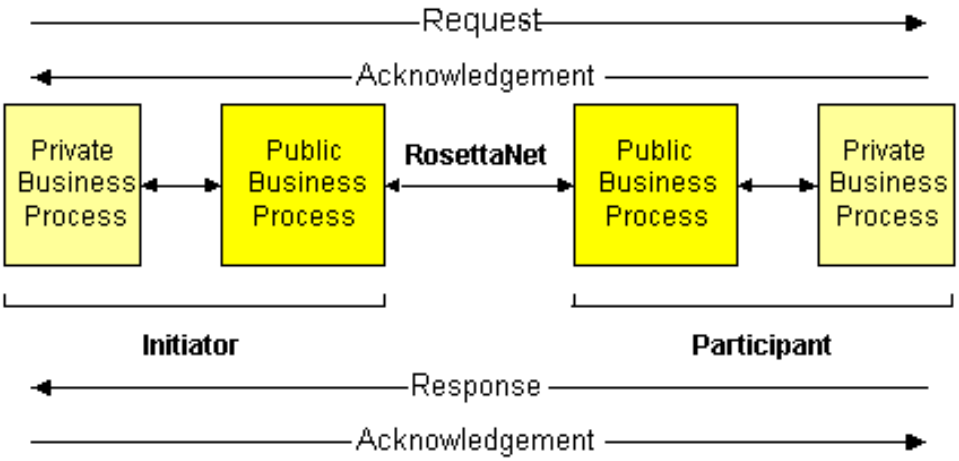
The asynchronous two-action activity design pattern involves two actions—a single action (business message) from the sender to the recipient, and a single action (business message) from the recipient back to the sender, and their corresponding signals (protocol responses, such as acknowledgements, ack-rejects, or errors) to each other:

1. The initiator sends a business message request to the participant.
2. The participant sends a receipt acknowledgement to the initiator.
3. The participant sends a business message response to the initiator.
4. The initiator sends a receipt acknowledgement to the participant.

This design pattern is typical of two-way, bi-directional communications between trading partners, such as a request / reply activities, that require mutual confirmation. An example of this design pattern in PIP 3A4 (Purchase Order Request and Confirmation), which is described at `http://www.rosettanet.org/PIP3A4`.

The following figure shows an example of how, in WebLogic Integration, public and private business processes might be involved in an asynchronous two-action collaboration.

Figure 3-2 Public and Private Business Processes in an Asynchronous Two-Action Activity Design Pattern



In this sample scenario, the message flow proceeds along the following path:

1. The initiator's private business process creates the request (in some internal data format) and submits it to the public business process.
2. The initiator's public business process receives the request, converts it to the appropriate PIP format, and sends it to the participant trading partner.

   Note: For outbound and inbound messages, WebLogic Integration automatically handles the packaging of non-payload portion of RosettaNet business messages (such as the version, content length, headers, and digital signatures), as well as transport-level security and message-level security.
3. On the participant side, WebLogic Integration handles the process of receiving the RosettaNet business message from the initiator, validating the contents of the inbound business message, and forwarding the document to the appropriate public business process.
4. The participant's public business process receives the request document from the initiator and sends an acknowledgement of receipt to the initiator.
5. The participant's public business process converts the request document from the PIP format to the appropriate internal data format, and then submits the request to the private business process.
6. The participant's private business process handles the request, creates a response in some internal format, and sends the response to the participant's public business process.

7. The participant's public business process converts the response from the internal data format to the PIP format, and then sends the response document to the initiator.
8. On the initiator side, WebLogic Integration handles the process of receiving the RosettaNet business message from the initiator, validating the contents of the inbound business message, and forwarding the document to the appropriate public business process.
9. The participant's public business process receives the response document and sends an acknowledgement of receipt to the participant.
10. The initiator's public business process converts the response document from the PIP format to the appropriate internal data format, and then submits the response to the private business process.

As with the single-action activity design pattern, the private business processes on either end could handle the conversion from the private data format to the appropriate RosettaNet PIP. In addition, private and public business processes might use internal means to indicate whether or not the overall process succeeded.

Synchronous One-Action / Two-Action Activity

RosettaNet also specifies synchronous versions of the asynchronous design patterns, in which an immediate response is required. The current release of WebLogic Integration does not support synchronous design patterns.

RosettaNet Business Messages

WebLogic Integration supports sending and receiving RosettaNet messages according to the RosettaNet Implementation Framework (RNIF) 1.1 and 2.0. A business message exchanged via the RosettaNet 1.1 protocol is called a *RosettaNet Object* (RNO). The business message exchanged via the RosettaNet 2.0 protocol is called a *RosettaNet Business Message* (RBM).

Note: In this document, we refer to both RNOs and RBMs as RosettaNet business messages.

The RNIF provides exchange protocols for implementation of the PIPs. The RNIF specifies information exchange between trading partner servers using XML, covering transport, routing and packaging, security, signals, and trading partner agreements. Some elements of RosettaNet messages are common across all RosettaNet messages, while other elements are unique to specific PIPs. To ensure that RosettaNet messages are structured and processed in a consistent manner, each PIP comes with a message guideline and XML document type definition (DTD).

Note: WebLogic Integration supports character encoding for sending messages. WebLogic Integration uses UTF-8 character encoding in all RosettaNet messages.

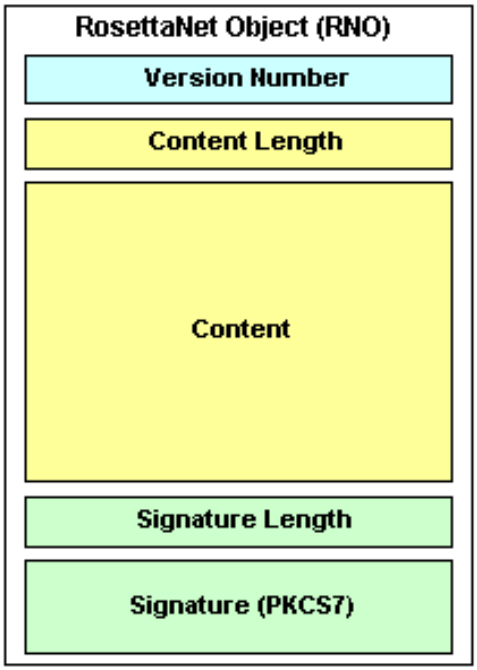
Components of a RosettaNet Business Message

This section describes the components of RosettaNet business messages.

RosettaNet Object (RNO) for RNIF 1.1

The following figure shows the components of a RosettaNet business message for RNIF 1.1.

Figure 3-3 Components of a RosettaNet Object (RNO) for RNIF 1.1



The following table describes the components of an RNO:

Table 3-3 Components of an RNO

Component	Description
Version	Specifies the RNIF version (1.1), in binary format.
Content Length	Length of the multi-part MIME message, in binary format.

Table 3-3 Components of an RNO (Continued)

Component	Description
Headers	Includes the following headers: • Preamble Header • Service Header
Payload	Includes the following components: • Service Content—contains either an action or a signal message. • Attachments—Optional. Can contain one more attachments, which consist of XML and non-XML (binary) data. Examples of possible attachments include Word documents, GIF images, PDF files, and so on. The information for each attachment is contained in the Service Header of the message.
Digital Signature (optional)	Optional. Digital signature. • Length of the signature in binary format. • Signature (PKCS7) in binary format.

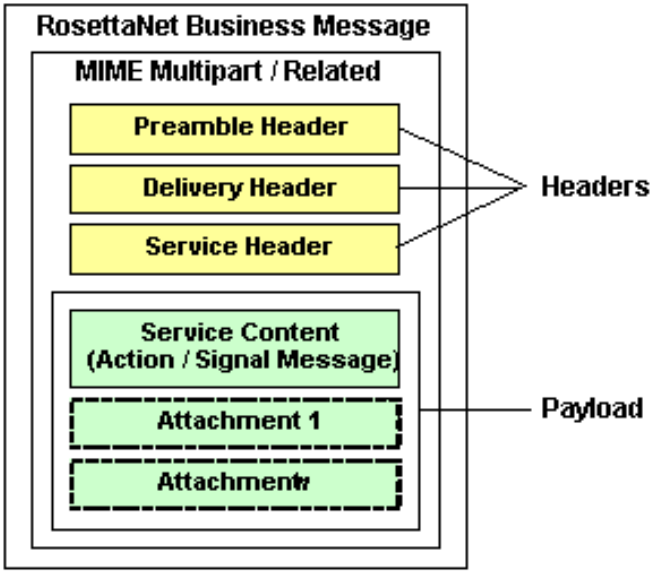
RosettaNet Business Message (RBM) for RNIF 2.0

The RosettaNet Implementation Framework 2.0 introduced the following notable differences in the composition of a RosettaNet Business Message (RBM):

- In RNIF 2.0, the Delivery Header was added.
- In RNIF 2.0 (but not in RNIF 1.1), the Service Header and Content can be encrypted. Using the WebLogic Integration Administration Console, you can configure the system to encrypt the Service Content, Service Header, and any attachments when a message is sent. For more information about encryption, see "Encryption—PKCS7 Enveloped Data for RosettaNet 2.0" on page 4-44.

The following figure shows the components of an RBM:

Figure 3-4 Components of a RosettaNet Business Message for RNIF 2.0



The following table describes the components of a RBM:

Table 3-4 Components of an RBM

Component	Description
Headers	Includes the following headers: • Preamble Header • Delivery Header • Service Header
Payload	Includes the following components: • Service Content—Contains either an action or a signal message. • Attachments—Optional. Can contain one more attachments, which consist of XML and non-XML (binary) data. Examples of possible attachments include Word documents, GIF images, PDF files, and so on. The information for each attachment is contained in the Service Header of the message.

WebLogic Integration Handles the Non-Payload Portion of RosettaNet Messages

When sending and receiving RosettaNet business messages, WebLogic Integration automatically handles the non-payload portion (version, content length, headers, and digital signatures) of the business message, as well as packaging, transport-level security, and message-level security, so that WebLogic Workshop business processes can focus on Service Content and attachments.

Validation of RosettaNet Business Messages

The RosettaNet PIP definitions contain detailed validation rules for messages exchanged in the PIP. These rules are significantly more stringent than the validation expressed within an XML Document Type Definition (DTD). The required validation rules are expressed in XML schema documents (XSD), which are based on the World Wide Web Consortium (W3C) 2000 XML Schema Definitions (XSD) schema.

WebLogic Integration provides message validation services for both RNIF 1.1 and RNIF 2.0 messages. The validation performed is dependent on the values specified for the `validateServiceContent`, `validateServiceHeader`, and `useDTDValidation` business process variables.

Validation Options for RosettaNet Messages

WebLogic Integration supports the following validation options:

Table 3-5 Validation Options for RosettaNet Business Messages

Option	Description
None	No validation is performed if `validateServiceHeader` and `validateServiceContent` are set to false.
Validate Service Header	If `validateServiceHeader` set to true, the service header of all messages sent and received for a template are validated. The type of validation performed is dependent on the setting for the `useDTDValidation` variable.
Validate Service Content	If `validateServiceContent` set to true, all service content of all messages sent and received for a template are validated. The type of validation performed is dependent on the setting for the `useDTDValidation` variable.

Table 3-5 Validation Options for RosettaNet Business Messages (Continued)

Option	Description
DTD	DTD validation is performed if `useDTDValidation` is set to true. The DTD files are supplied as part of the PIP specification and can be obtained from `www.rosettanet.org`.
XSD Schemas	XSD schema validation is performed if `useDTDValidation` is set to false, or is absent from the business process template. In order to use XSD validation, you must create your own schema files from the DTD for the PIP.

Configuring RosettaNet Message Validation

To configure these options in the WebLogic Integration Administration Console:

1. Add a service for RosettaNet, as described in "Adding Services" in Trading Partner Management.
2. Edit this service, as described in "Viewing and Changing Services" in Trading Partner Management.
3. On the View and Edit Service Details page, click Add Defaults.
4. Configure validation options for RosettaNet business messages.

Further Reading on RosettaNet Message Validation

For more information about RosettaNet message validation, see the following documents:

- The *RosettaNet Implementation Framework* (RNIF) specification, which provides an explanation of the exception handling process, is available at the following URL:

  `http://www.rosettanet.org/rnif`

- Information about XML schema tools, usage, specifications, and development is available at the following URL:

  `http://www.w3.org/XML/Schema`

- The *XML Schema Part 0: Primer*, which provides useful descriptions of the features and capabilities of XML schema, is available at the following URL:

  `http://www.w3.org/TR/xmlschema-0/`

RosettaNet Business Processes

This topic describes WebLogic Workshop business processes that implement RosettaNet PIPs. It contains the following sections:

- Guidelines for Designing RosettaNet Business Processes
- RosettaNet Initiator Business Processes
- RosettaNet Participant Business Processes

Guidelines for Designing RosettaNet Business Processes

The specification for each RosettaNet PIP is highly structured. It specifies roles, message choreography, business message structure, and other design details. Implementing a RosettaNet business process requires adherence to the PIP specification for that PIP role. Therefore, when designing business processes for RosettaNet solutions, consider the following guidelines:

- You should thoroughly understand the schema, choreography, roles, and other aspects of the PIP that you want to implement. For example, you should understand:
  - the design pattern (asynchronous one-action and two-action activities)
  - the number and nature of business messages exchanged
  - the schema for each business message
  - the success path for the message exchange (such as request / response sends and receipt acknowledgements)
  - the possible failure paths, such as retries, timeouts, errors, rejections, and notifications of failure

  For a list of PIPs, see `http://www.rosettanet.org/pipdirectory`.
- You should obtain every provided DTD schema that the PIP provides for business messages. You might need to convert this DTD schema to an XSD schema for WebLogic Integration, as described in "Converting RosettaNet DTD Schemas to XSD Schemas" in "Tutorial: Building RosettaNet Solutions" in *Tutorials for Trading Partner Integration*, which is available at:

  `http://dev2dev.bea.com/code/wli.jsp`
- You should thoroughly understand the data transformation between any private processes and the Service Content exchanged in the public (PIP) business processes. For each transformation, you use a transformation control to convert the Service Content and

attachments of the RosettaNet business message to / from the format(s) used in the private processes. For more information, see Creating a Transformation Control and a Transformation Method in the WebLogic Workshop Help.

- Determine how you want to specify the business process properties:
  - Statically—using hard-coded annotations (jc:rosettanet Tag on the RosettaNet control in initiator business processes and the @jpd:rosettanet Annotation in participant business processes). This is recommended for specifying certain static information about the PIP, such as the PIP name and version.
  - Dynamically—using passed-in values, such as DUNS numbers, for initiator and participant IDs, and then calling the `setProperties` method. You can also use XQuery selectors to extract the business IDs and other information from incoming messages.
- Thoroughly test the implementation by simulating the choreography between the initiator and participant business processes with sample data. By default, WebLogic Integration runs in Test (development) mode, which allows you to run business processes on the same machine (collocated) using preconfigured trading partners and a default binding that uses either the RNIF 1.1 or 2.0 protocol. The endpoints for both partners use `127.0.0.1:7001`. For more information, see "Default TPM Repository Settings" on page 1-10.

RosettaNet Initiator Business Processes

In WebLogic Integration, initiator business processes use a RosettaNet control to enable WebLogic Workshop business processes to exchange business messages and data with trading partners via RosettaNet. You use RosettaNet controls *only* in initiator business processes to manage the exchange of RosettaNet business messages with participants. The RosettaNet control provides methods for sending business messages, acknowledgements, and errors, and it provides callback methods to handle responses from participants.

For detailed information about using the RosettaNet control in business processes, see the following topics in RosettaNet Control in the WebLogic Workshop Help:

- Overview: RosettaNet Control
- Creating a New RosettaNet Control
- Using a RosettaNet Control
- RosettaNetControl Interface
- jc:rosettanet Tag

For an introduction to initiator business processes, see "Initiator and Participant Business Processes" on page 1-14.

RosettaNet Participant Business Processes

In WebLogic Integration, you can easily create a new RosettaNet participant business process using a WebLogic Workshop template (the RosettaNet participant business process file). This template provides a head start for building public participant business processes for RosettaNet message exchanges. Although this file is not required to build RosettaNet participant business processes, it includes the nodes and business process annotations needed to integrate easily with RosettaNet initiator business processes, as well as standard choreography patterns such as acknowledgements, time-outs, retries, and errors. For information about creating participant business processes, see:

- Building RosettaNet Participant Business Processes and @jpd:rosettanet Annotation in the WebLogic Workshop Online Help
- "Tutorial: Building RosettaNet Solutions" in *Tutorials for Trading Partner Integration* (available at `http://dev2dev.bea.com/code/wli.jsp`)

For an introduction to participant business processes, see "Initiator and Participant Business Processes" on page 1-14.

Tasks for Implementing a RosettaNet Solution

This topic provides a high-level, end-to-end overview of the tasks involved with implementing a RosettaNet solution. It includes the following topics:

- Before You Begin
- Planning the RosettaNet Solution
- Building the RosettaNet Solution
- Deploying the RosettaNet Solution
- Managing the RosettaNet Solution

Note: This topic describes, in a general way, the tasks that are *typically* involved in implementing a RosettaNet solution. The process of implementing RosettaNet solutions is iterative, and it can vary in scope and sequence depending on your unique business requirements and environment.

Before You Begin

Before you begin implementing a RosettaNet solution, we recommend that you review the following documents:

- Chapter 1, "Introduction," to learn basic concepts for integrating trading partners using WebLogic Integration.
- This chapter, to learn basic concepts that are unique to RosettaNet solutions.
- "Tutorial: Building RosettaNet Solutions" in *Tutorials for Trading Partner Integration*, which is available at:

  `http://dev2dev.bea.com/code/wli.jsp`

Planning the RosettaNet Solution

Once you have decided to use RosettaNet as the business protocol for your trading partner integration (as described in "Phase 1: Plan the Solution" on page 1-28), you need to plan the solution by determining certain factors in your implementation:

- With which trading partners will you integrate?
- For each trading partner, you need to determine:
  - Which business process(es) will you integrate?
  - What information about that trading partner is required for your implementation (for example, their DUNS number and other profile information). For more information, see "Trading Partner Profiles" on page 1-6.
- For each business process, you need to determine:
  - Which RosettaNet PIP(s) are used? Which version of the PIPs will be implemented?
  - Which RNIF version (1.1 or 2.0) is required for exchanging business messages?
  - What role (RosettaNet role name) will your organization play in the PIP?
  - Do you need to create business processes for just your role, or for both roles in the business process?
  - What are the back-end integration requirements for each PIP? What private business processes are required? What is the data format of, and connections with, associated back-end systems?

  - Does your implementation need to meet legal standards for nonrepudiation, which is described in "NonRepudiation" on page 4-37?

For more information about planning an integration solution, see *Designing WebLogic Integration Solutions*, which is available at the following URL:

`http://edocs.bea.com/wli/docs81/design/index.html`

Building the RosettaNet Solution

After planning your RosettaNet solution, you build the business processes that implement the PIP(s) according to the RosettaNet PIP specifications. For more information about design-time tools, see "Phase 2: Design, Build, and Test the Solution" on page 1-28. For conceptual information about RosettaNet business processes, see "RosettaNet Business Processes" on page 3-16.

The "Tutorial: Building RosettaNet Solutions" in *Tutorials for Trading Partner Integration* (available at http://dev2dev.bea.com/code/wli.jsp) provides a detailed walkthrough of the typical tasks that are required to build a RosettaNet solution.

To build a RosettaNet solution, you would typically complete the following steps:

1. Using the BEA WebLogic Platform Configuration Wizard, create a new domain based on the WebLogic Integration domain template. For instructions, see *Creating WebLogic Configurations Using the Configuration Wizard* in the WebLogic Platform documentation.
2. Download the PIP distribution, including the specification and any DTDs, from the RosettaNet web site at `http://www.rosettanet.org/`.
3. Optionally, convert any DTDs to XSD files, as described in "Converting RosettaNet DTD Schemas to XSD Schemas" in "Tutorial: Building RosettaNet Solutions" in *Tutorials for Trading Partner Integration*.
4. From the *Tutorials for Trading Partner Integration*, copy the example PIP implementation associated with the design pattern that you want to use. For more information about design patterns, see "PIP Design Patterns" on page 3-6.
5. In WebLogic Workshop, import the schema for the new PIP into the project and then change the schema definition to the new PIP.
6. Change the RosettaNet annotations for the new PIP:
   - For *public* initiator business processes, you change the `pip` and `pip-version` attributes (and others if needed) in the @jc:rosettanet Tag, which is described at the following URL:

http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsRosettaNetTag.html

– For *public* participant business processes, you change the `pip-name` and `pip-version` attributes (and others if needed) in the @jpd:rosettanet Annotation, which is described at the following URL:

http://edocs.bea.com/workshop/docs81/doc/en/integration/reference/refJpdAnnotations_rosettanet.html

7. Rename the JPD and JCX files and change the names of other components to be more descriptive of the new PIP implementation, if you want.

   – For more information about modifying the RosettaNet control in initiator business processes, see RosettaNet Control in the WebLogic Workshop Online Help, which is available at:

   http://edocs.bea.com/workshop/docs81/doc/en/integration/controls/controlsRosettaNet.html

   – For more information about building RosettaNet participant business processes, see Building RosettaNet Participant Business Processes in the WebLogic Workshop Help, which is available at the following URL:

   http://edocs.bea.com/workshop/docs81/doc/en/integration/wfguide/wfguideRosettaNet.html

8. Change the implementation of any *private* business processes as needed.

9. Make any other changes to the business processes as needed.

10. Test the RosettaNet solution using the default TPM repository configuration (described in "Default TPM Repository Settings" on page 1-10) and security settings (described in "Default Domain Security Configuration" on page 4-8).

Deploying the RosettaNet Solution

Once you have built and tested your RosettaNet solution, you deploy the solution in a production environment. For more information about deployment tools, see "Phase 3: Deploy the Solution" on page 1-29. For detailed information about deploying WebLogic Integration solutions, see *Deploying WebLogic Integration Solutions*, which is available at the following URL:

http://edocs.bea.com/wli/docs81/deploy/index.html

To deploy a RosettaNet solution, you would typically complete the following steps:

1. If necessary, using the BEA WebLogic Platform Configuration Wizard, create a new domain based on the WebLogic Integration domain template. For instructions, see *Creating WebLogic Configurations Using the Configuration Wizard* in the WebLogic Platform documentation.
2. Start WebLogic Server in production mode.
3. Using the WebLogic Integration Administration Console, complete the following tasks:

   Note: If you have already defined trading partner information in your development environment, you can export this information to an external file, and then import this file into the production environment. For instructions, see "Exporting Management Data" and "Importing Management Data" in Trading Partner Management.

   a. Add trading partners to the TPM repository, including basic profile information, such as the DUNS number used for trading partner IDs. For instructions, see "Adding Trading Partner Profiles" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

   b. For each trading partner, define the protocol bindings (including RNIF version—1.1 or 2.0—and other settings) to be used for message exchanges with this trading partner. For instructions, see "Defining a RosettaNet 1.1 or 2.0 Binding" in Trading Partner Management.

   c. Define the PIP Notification of Failure roles, as described in "Configuring PIP Notification of Failure Roles for RosettaNet Bindings" in Trading Partner Management.

   d. Define the services that you will use in your deployment. For instructions, see "Adding Services" in Trading Partner Management.

   e. For each trading partner, define the service profiles (protocol bindings and URL endpoints for local and remote trading partners) associated with each service. For instructions, see "Adding Service Profiles to a Service" in Trading Partner Management.
4. Using the WebLogic Server Administration Console and WebLogic Integration Administration Console, implement security for your deployment, as described in "Implementing Security for Trading Partner Integration" on page 4-48.
5. Configure your domain (clustering, high availability, load balancing, fail-over, and so on) as needed for your production environment. For instructions, see *Deploying WebLogic Integration Solutions*, which is available at the following URL:

   `http://edocs.bea.com/wli/docs81/deploy/index.html`

6. Deploy the application and other WebLogic Integration resources associated with your RosettaNet solution. For instructions, see *Deploying WebLogic Integration Solutions*, which is available at the following URL:

   `http://edocs.bea.com/wli/docs81/deploy/index.html`

Managing the RosettaNet Solution

Once you have deployed your RosettaNet solution, you would typically monitor run-time performance, message volumes, resource utilization, and other factors to ensure optimum operation on your solution. For more information about monitoring tools, see "Phase 4: Administer and Tune the Solution" on page 1-30.

For instructions on monitoring trading partner integration resources, see the following topics in in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Viewing Statistics"
- "Monitoring Messages"

CHAPTER 4

Trading Partner Integration Security

This topic describes security concepts and tasks in WebLogic Integration trading partner integration solutions. It includes the following sections:

- Before You Begin
- Security Framework for Trading Partner Integration
- Transport-Level Security
- Message-Level Security
- Using Proxy Servers with Trading Partner Integration
- Implementing Security for Trading Partner Integration

Before You Begin

Before you begin implementing WebLogic Integration security for trading partner integration, be sure to read the following documents:

- "Using WebLogic Integration Security" in *Deploying WebLogic Integration Solutions* at the following URL:

  `http://edocs.bea.com/wli/docs81/deploy/secure.html`

- *Introduction to WebLogic Security* at the following URL:

  `http://edocs.bea.com/wls/docs81/secintro/index.html`

- Security summary page, which provides a compendium of WebLogic security topics at the following URL:

  `http://edocs.bea.com/wls/docs81/security.html`

- To learn about the basic concepts for trading partner integration in WebLogic Integration, see the Chapter 1, "Introduction."

Security Framework for Trading Partner Integration

This topic describes the WebLogic Integration security framework for trading partner integration. It includes the following sections:

- Summary of WebLogic Security Features
- WebLogic Server Default Security Configuration
- Components of Trading Partner Integration Security
- Default Domain Security Configuration
- Credential Stores
- Trading Partner Integration Resources Requiring Security Policies

Summary of WebLogic Security Features

WebLogic Integration leverages certain functionality from the WebLogic Server security framework and provides additional features for trading partner integration. WebLogic Integration supports secure business transactions between trading partners through:

- Transport-level security, including authentication via userID/passwords and digital certificates, and role and policy based access control to WebLogic Integration resources
- Message-level security, including the use of digital signatures, encryption (for RosettaNet 2.0 only), and support for other non-repudiation features such as secure audit logs and timestamp providers
- Integrated trading partner management and security management capabilities in the WebLogic Integration Administration Console
- Default security configurations in WebLogic Integration domains
- Embedded LDAP support

WebLogic Server Default Security Configuration

WebLogic Integration security is based on the WebLogic Server security framework—namely, the Default Security Configuration. For more information, see "The Default Security Configuration in WebLogic Server" in "Overview of Security Management" in *Managing WebLogic Security* at the following URL:

http://edocs.bea.com/wls/docs81/secmanage/overview.html

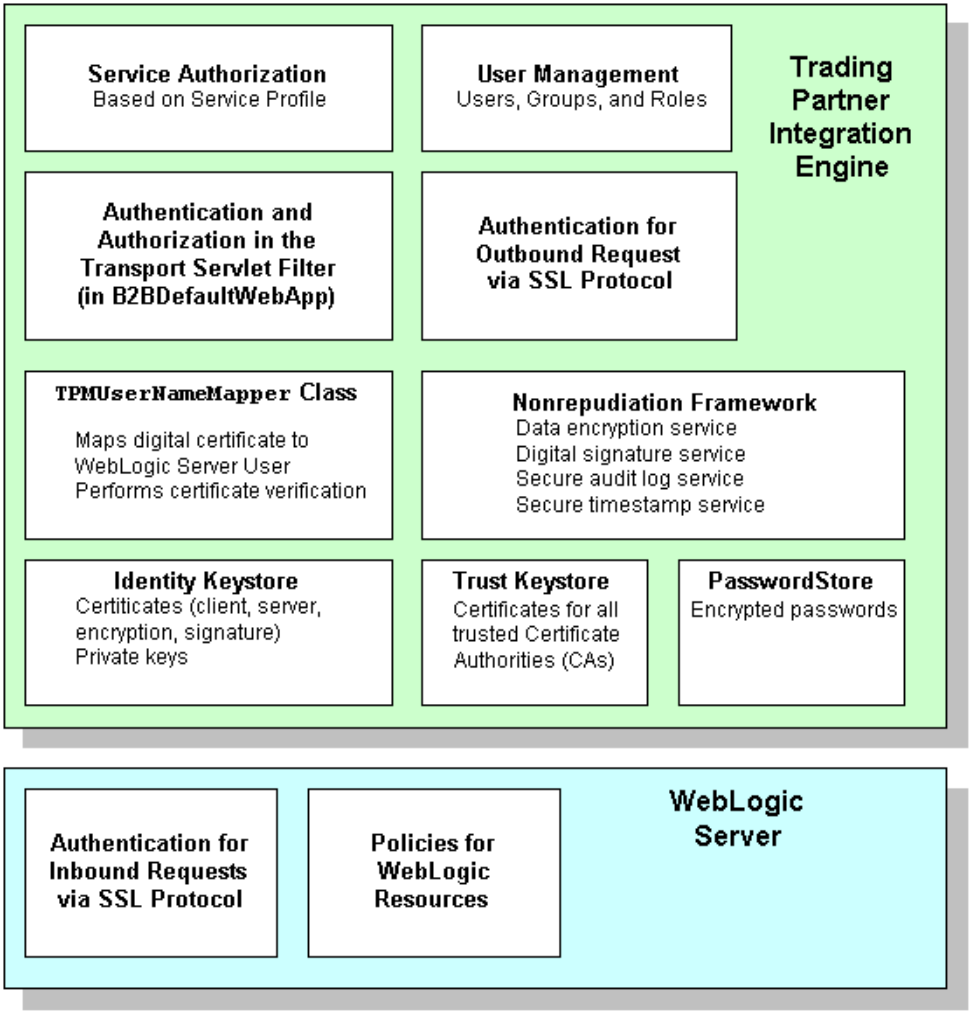
Components of Trading Partner Integration Security

The WebLogic security model for trading partner integration:

- Uses the security features of the underlying BEA WebLogic Server™ platform to perform authentication and authorization of principals before granting access to trading partner integration resources.
- Is extensible by allowing you to incorporate your own or third-party vendor tools to verify trading partner digital certificates and implement nonrepudiation support, which is a requirement for critical business messages.

The following figure shows the entities and features in WebLogic Server and WebLogic Integration that support trading partner integration.

Figure 4-1 WebLogic Security Model for Trading Partner Integration



The following table describes the features in this security model.

Table 4-1 Components in the Trading Partner Integration Security Model

Component	Description
Service authorization	The service profile defines the business messages that a given trading partner may send and receive. When a business message arrives for a trading partner, WebLogic Integration, as part of the business message authorization process, examines the contents of the business message to validate it against the service profile. WebLogic Integration verifies that the content of the incoming business message is consistent with the business messages that the trading partner is bound, by the service profile, to either send or receive. This authorization scheme ensures that only the business messages that are consistent with the relevant service profile have access to trading partner integration resources. For more information, see "Service Authorization" on page 4-33.
User management	The *User Management* module of the WebLogic Integration Administration Console to manage the users, groups, and roles defined in the default security realm. For instructions on configuring users, groups, and roles in the WebLogic Integration Administration Console, see User Management in *Managing WebLogic Integration Solutions*.
Data encryption service	The data encryption service encrypts business messages for the business protocols that require it. Data encryption works by using a combination of the sender's certificate, private key, and the recipient's certificate to encode the business message. The message can then be decrypted only by the recipient using the recipient's private key. For details about using the data encryption service, see "SSL Protocol" on page 4-12.

Table 4-1 Components in the Trading Partner Integration Security Model

Component	Description
Authentication and authorization in the Transport Servlet Filter (in B2BDefaultWebApp)	The Transport Servlet Filter is a WebLogic Integration-specific servlet filter that serves as the entry point for both HTTP and HTTPS access to trading partner integration resources, including: • WebLogic Workshop business processes • JDBC connection pools and MBeans that are used to access the TPM repository • JMS destinations For more information about these resources, see "Trading Partner Integration Resources Requiring Security Policies" on page 4-11. The Transport Servlet Filter performs either basic or mutual authentication and it authorizes access to URL (Web) resources. A Transport Servlet Filter is dynamically registered in the WebLogic Server environment for trading partners bound to a specific service profile. For more information, see: • "Authenticating Trading Partner Messages" on page 4-21 • "URL (Web) and EJB (Enterprise JavaBean) Resources" and "Application Resources" in "Types of WebLogic Resources" in *Securing WebLogic Resources* at the following URL: `http://edocs/wls/docs81/secwlres/types.html`
Authentication for outbound request via the SSL protocol	WebLogic Integration authenticates the recipient for all outbound messages (using the SSL certificate obtained in the SSL handshake) to ensure that the messages are consistent with the relevant service profile to which they are bound. For more information, see "Authentication" on page 4-12.
`TPMUserNameMapper` class	The `TPMUserNameMapper` class maps a trading partner certificate to the corresponding WebLogic Server user that has been configured for the trading partner. The `TPMUserNameMapper` class implements the `weblogic.security.providers.authentication.UserNameMapper` interface. You can configure this class or write your own class. For more information, see "Authenticating Remote Users in Two-Way Authentication" on page 4-23 and reference information for the UserNameMapper interface at the following URL: `http://e-docs.bea.com/wls/docs81/javadocs/weblogic/security/providers/authentication/UserNameMapper.html`

Table 4-1 Components in the Trading Partner Integration Security Model

Component	Description
Nonrepudiation framework	The trading partner integration security system provides a means to implement nonrepudiation support. Nonrepudiation is the ability of a trading partner to prove or disprove having previously sent or received a particular business message to or from another trading partner. Nonrepudiation requires the following services: • Digital signatures • Secure timestamps • Secure audit log WebLogic Integration provides Service Provider Interfaces (SPIs) that allow you to incorporate your own or a trusted third-party's implementation. In addition, WebLogic Integration provides an audit log provider that can be used for demo / development purposes. For more information about nonrepudiation, see "NonRepudiation" on page 4-37.
Identity keystore	Store private keys for local trading partners and certificates for both the local and remote trading partners. These certificates are of the following types: • Client certificate—Digital certificate of the remote or local trading partner. • Server certificate—Digital certificate of the remote trading partner. • Signature certificate—Used for each trading partner business message if digital signature support is required. • Encryption certificate—Used for each trading partner if business message encryption is required. For more information about these certificates, see "Types of Digital Certificates" on page 4-17. For more information about the identity keystore, see "Keystore for Private Keys and Certificates" on page 4-9.
Trust keystore	Store all the trusted CA certificates associated with any certificate used in WebLogic Integration in this KeyStore. For more information, see "Keystore for Private Keys and Certificates" on page 4-9.
PasswordStore	All passwords are kept in encrypted form in the WebLogic Integration PasswordStore, which uses the JCE security provider. For more information, see "WebLogic Integration PasswordStore for Encrypted Passwords" on page 4-9.

Table 4-1 Components in the Trading Partner Integration Security Model

Component	Description
Authentication for inbound requests via SSL protocol	When an inbound trading partner message arrives, both the trading partner and the WebLogic Server system exchange certificates to establish each other's identity. When the SSL handshake is completed, the trading partner's network connection to the WebLogic Server system is established. For information about configuring the SSL protocol in WebLogic Server to provide mutual authentication, see "SSL Protocol" on page 4-12.
Policies for WebLogic resources	A *security policy* is an association between a WebLogic resource and one or more users, groups, or security roles that is designed to protect the WebLogic resource against unauthorized access. For more information, see "Security Policies" in Securing WebLogic Resources at the following URL: `http://e-docs.bea.com/wls/docs81/secwlres/sec_poly.html`

Default Domain Security Configuration

When you create a new domain using the WebLogic Platform Configuration Wizard and the WebLogic Integration template, the new domain contains default security settings, including:

- Default WebLogic Integration roles and groups. Default security policies define the roles authorized to access specific WebLogic Integration resources. For more information, see "Default Groups, Roles, and Security Policies" in User Management in *Managing WebLogic Integration Solutions*.
- B2BDefaultWebApp, on which you can configure policies for access control in trading partner authorization, which is described in "Authenticating Trading Partner Messages" on page 4-21. For configuration instructions, see "URL (Web) and EJB (Enterprise JavaBean) Resources" and "Application Resources" in "Types of WebLogic Resources" in *Securing WebLogic Resources* at the following URL:

  `http://edocs/wls/docs81/secwlres/types.html`
- PasswordStore, which is described in "WebLogic Integration PasswordStore for Encrypted Passwords" on page 4-9.
- Default identity and trust keystores, which are described in "Keystore for Private Keys and Certificates" on page 4-9.

- Default trading partners and services profiles, described in "Default TPM Repository Settings" on page 1-10, which you can use for development and testing (in Test mode). You can configure one trading partner as local and the other as remote, and you can add test certificates to these trading partners for testing purposes, as described in "Keystore for Private Keys and Certificates" on page 4-9.

Credential Stores

WebLogic Integration provides the following credential stores to store passwords, private keys, and certificates:

- WebLogic Integration PasswordStore for Encrypted Passwords
- Keystore for Private Keys and Certificates

WebLogic Integration PasswordStore for Encrypted Passwords

All passwords are kept in encrypted form in the PasswordStore. WebLogic Integration does not require clear-text passwords. The PasswordStore uses the JCE security provider. Configuration of passwords is controlled through an MBean API and passwords are accessed using password-aliases.

You use the WebLogic Integration Administration Console to manage passwords in the PasswordStore. For more information, see the following topics in System Configuration in *Managing WebLogic Integration Solutions*:

- "Adding Passwords to the Password Store"
- "Listing and Locating Password Aliases"
- "Changing the Password for a Password Alias"
- "Deleting Passwords from the PasswordStore"

Keystore for Private Keys and Certificates

WebLogic Integration requires that you use keystores to store all private keys and certificates. A keystore is a protected database that holds keys and certificates. If you have keys and certificates and use message encryption, digital signatures, or SSL, you must use a keystore for storing those keys and certificates and make the keystore available to applications that might need it for authentication or signing purposes.

Types of Keystores

When you set up a WebLogic Integration domain for trading partner integration, the following keystores are configured:

Table 4-2 Types of Certificates Used in WebLogic Integration

Type of Certificate	Description
Identity keystore	Stores private keys for local trading partners and certificates for both the local trading partner and remote trading partners. Certificates are of the following types: client, server, signature, and encryption certificates. WebLogic Integration retrieves private keys and certificates from this keystore to use for SSL, message encryption, and digital signatures. For more information about certificates, see "Digital Certificates" on page 4-15.
Trust keystore	WebLogic Server uses the trust keystore to locate trusted CAs for SSL. WebLogic Integration uses it to locate the trusted CAs while verifying signature and encryption.

Default Keystores for the Test Environment

When you create a new domain using the WebLogic Platform Configuration Wizard and the WebLogic Integration template, the new domain contains Demo Keystores of type JKS.

- Utilizes the JDK bundled Java KeyStore (JKS) provider, which implements the keystore as a file
- Protects each private key with an individual password
- Protects the entire keystore with a password

Keystores in a Production Environment

You can use the Demo keystores in a development / testing environment, but you must explicitly create new identity and trust keystores for a production environment. To create a keystore and make it available for trading partner integration:

1. If the identity and trust keystores do not already exist in your domain, create them according to the instructions in "Storing Private Keys, Digital Certificates, and Trusted Certificate Authorities" in "Configuring SSL" in *Managing WebLogic Security* at the following URL:

   `http://edocs.bea.com/wls/docs81/secmanage/ssl.html`

2. Configure the keystores using the WebLogic Server Administration Console according to the instructions in "Configuring KeyStores" in "Configuring SSL" in *Managing WebLogic Security* at the following URL:

   `http://edocs.bea.com/wls/docs81/secmanage/ssl.html`

3. Add trading partner certificates to the identity keystore.

4. Add trusted certificate authority certificates to the trust keystore.

For information about refreshing the keystore using the WebLogic Integration Administration Console, see "Refreshing the Keystore" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Note: In a clustered domain, you need to create and configure a separate keystore for each WebLogic server.

Trading Partner Integration Resources Requiring Security Policies

You can configure security policies for trading partner integration resources, such as:

- B2BDefaultWebApp and endpoint URIs of protocol bindings of local trading partners in the TPM repository
- WebLogic Workshop business processes
- JDBC connection pools (`bpmarchpool` and `cgpool`) and MBeans that are used to access the TPM repository
- JMS destination (for message tracking, asynchronous dispatcher queues for trading partner integration business processes)

For more information, see *Securing WebLogic Resources* at the following URL:

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

Transport-Level Security

Transport-level security involves authentication and encryption at the transport level (HTTP/HTTPS) and authorization at the endpoint. This topic describes transport-level security concepts and tasks for trading partner integration. It contains the following sections:

- Authentication
- Authenticating Remote Users in Two-Way Authentication
- Verifying Certificates in Two-Way Authentication
- Authorization

Authentication

In WebLogic Integration, *authentication* is the process of verifying an identity claimed by—or for—a system entity. Authentication is concerned with who an entity is—it is the association of an identity with an entity. WebLogic Integration examines and validates digital certificates against security information stored in the TPM repository.

For trading partner integration, WebLogic Integration uses the following approaches to authentication:

- **Username and password**—Human users (administrators) as well as trading partners use usernames and passwords as credentials to prove their identity. For more information, see "Types of Authentication" on page 4-13.
- **Digital certificates**—Trading partners use digital certificates to prove their identity to WebLogic Integration. For more information, see "Digital Certificates" on page 4-15.

Trading partner authentication is configured in the protocol bindings that define communications between trading partners. For more information, see "Defining Protocol Bindings" in Trading Partner Management in *Managing WebLogic Integration Solutions.*

SSL Protocol

The SSL protocol provides secure connections by enabling two applications linked through a network connection to authenticate the other's identity and by encrypting the data exchanged between the applications. An SSL connection begins with a handshake during which the applications exchange digital certificates, agree on the encryption algorithms to use, and generate encryption keys used for the remainder of the session.

The SSL protocol provides the following security features that WebLogic Integration supports:

- **Server authentication**—The server uses its digital certificate, issued by a trusted certificate authority, to authenticate itself to clients.
- **Client authentication**—Optionally, clients might be required to authenticate themselves to the server by providing their own digital certificates. This type of authentication is also referred to as mutual authentication. The authentication model in Trading Partner Integration uses mutual authentication.

Both types of authentication can be used while exchanging business messages between trading partners. In addition, administrators can use HTTPS fro ma Web Browser to access the WebLogic Integration Administration Console and WebLogic Server Administration Console.

Administrators use a Web browser to access the WebLogic Integration Administration Console. You can use the Hypertext Transfer Protocol with SSL (HTTPS) to secure this type of network communication. For more information about SSL, see "Configuring SSL" in *Managing WebLogic Security* at the following URL:

`http://edocs.bea.com/wls/docs81/secmanage/ssl.html`

Types of Authentication

WebLogic Integration supports the following types of authentication:

Table 4-3 Trading Partner Certificates Configured in Trading Partner Integration

Authentication Type	Description
Basic	With *basic authentication*, a user ID and password are requested from the user and sent to WebLogic Server. WebLogic Server checks the information and, if it is trustworthy, grants access to the protected WebLogic resource. For example: • WebLogic Integration will provide information for outgoing messages Configuration—TPM repository and Web Application Servlets • WLS will authenticate for incoming messages Configuration—User management in the WebLogic Integration Administration Console, deployment descriptor of B2BDefaultWebApp
One-Way (Server-Side) Authentication	With *one-way (server-side) authentication*, the server provides its identity to the client via a certificate. The client is not authenticated, and therefore the application does not have any access to the identity of the client. This mechanism is primarily used for transport-level encryption only to provide privacy of the message. You might want to use server-side authentication, however, if you do not want to require certificate-based authentication among your trading partners. This is the default authentication for WebLogic Integration domains.
One-Way (Server-Side) Plus Basic Authentication	With *one-way (server-side) plus basic authentication*, the server provides its identity to the client via a certificate, and the client provides its identity to the server via a user ID and password. You would use this type of authentication to have client authentication as well as transport-level encryption.
Two-Way (Mutual) Authentication	With *two-way (mutual) authentication*, both the client and the server are required to authenticate themselves to each other by means of digital certificates or other forms of proof material.

Authentication Levels

Trading Partner Integration incorporates a two-level authentication process:

- Verification of the trading partner certificate, which is described in "Authenticating Remote Users in Two-Way Authentication" on page 4-23.
- Authentication of the incoming trading partner message, which is described in "Authenticating Trading Partner Messages" on page 4-21.

Both levels are required for end-to-end access control (authorization) on WebLogic Integration resources. After a trading partner business message has passed both levels of authentication, WebLogic Integration performs the authorization process on the business message. To protect

trading partner integration resources, the authorization process requires at least basic or mutual authentication, which are described in "Types of Authentication" on page 4-13.

Digital Certificates

Digital certificates are electronic documents used to verify the unique identities of principals and entities over networks such as the Internet. A digital certificate securely binds the identity of a user or entity, as verified by a trusted third party (known as a certificate authority), to a particular public key. The combination of the public key and the private key provides a unique identity to the owner of the digital certificate.

Digital certificates allow verification of the claim that a specific public key does in fact belong to a specific user or entity. The recipient of a digital certificate can verify that the certificate, including the public key of the subject, was issued and signed by a trusted certificate authority (CA). The recipient does this by using the trusted certificate authority's public key to ensure that the digital signature was created using the corresponding CA private key. If such verification is successful, this chain of reasoning provides assurance that the corresponding private key is held by the subject named in the digital certificate, and that the digital signature was created by that particular certificate authority.

Digital certificates are stored in the identity keystore. For more information, see "Keystore for Private Keys and Certificates" on page 4-9.

Information in Digital Certificates

A digital certificate typically includes a variety of information, such as:

- The name of the subject (holder, owner) and other identification information required to uniquely identify the subject, such as a URL or an e-mail address
- The subject's public key
- The name of the certificate authority that issued the digital certificate
- A serial number
- The validity period (or lifetime) of the digital certificate (defined by a start date and an end date)

The most widely accepted format for digital certificates is defined by the ITU-T X.509 international standard. Thus, digital certificates can be read or written by any application complying with the X.509 standard. The public key infrastructure (PKI) in WebLogic Server

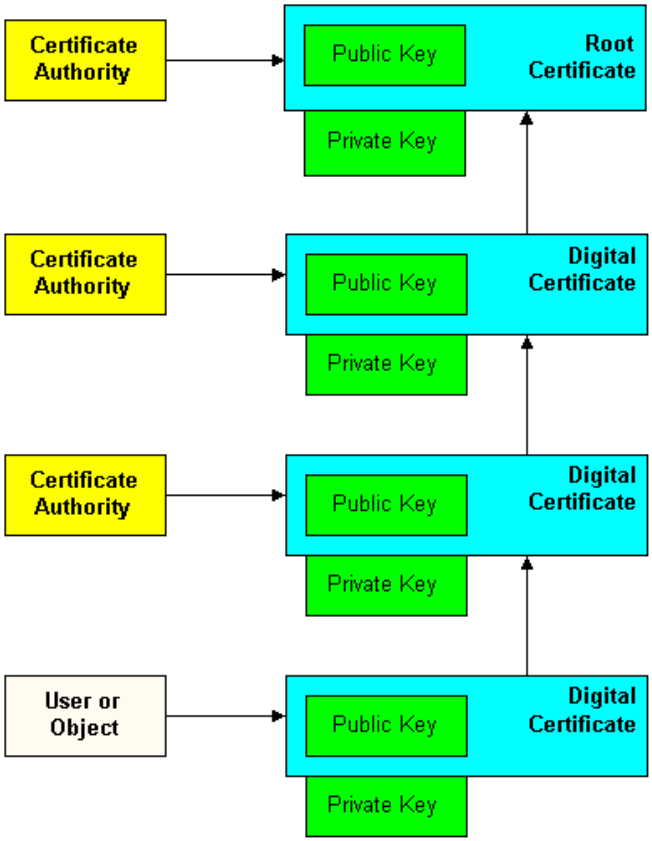
recognizes digital certificates that comply with X.509 version 1 (X.509v1) or version 3 (X.509v3).

Certificate Authorities

Digital certificates are issued by a certificate authority. Any trusted third-party organization or company that is willing to vouch for the identities of those to whom it issues digital certificates and public keys can be a certificate authority. When a certificate authority creates a digital certificate, the certificate authority signs it with its private key, to ensure the detection of tampering. The certificate authority then returns the signed digital certificate to the requesting subject.

The subject can verify the signature of the issuing certificate authority by using the public key of the certificate authority. The certificate authority makes its public key available by providing a digital certificate issued from a higher-level certificate authority attesting to the validity of the public key of the lower-level certificate authority. This hierarchy of certificate authorities is terminated by a self-signed digital certificate known as the root certificate, as shown in the following figure.

Figure 4-2 Certificate Authority Hierarchy



Before you use a digital certificate, verify a digital signature, or decrypt a business message, make sure that the digital certificate is issued by a trusted certificate authority. Regardless of who encrypts the business message, the digital certificate of the business message must be trusted, which is established by the certificate authority.

Types of Digital Certificates

WebLogic Integration supports the following types of certificates in trading partner integration:

Table 4-4 Certificates Supported in Trading Partner Integration

Type	Description
Client certificate	Digital certificate of the remote or local trading partner. Configuring the client certificate is required when using the SSL protocol. **Certificate is:** • Type X.509 version 1 or 3 • Privacy Enhanced Mail (PEM) or Definite Encoding Rules (DER) encoded. (The filename extension specifies the encoding type: `.pem` or `.der`.) • Required for all trading partner types when HTTPS with mutual authentication is used. **Private Key is:** • PEM or DER encoded. (The filename extension specifies the encoding type: `.pem` or `.der`.) • Required only for local trading partner type • Password-protected or plain text **Note:** When importing a plain-text private key using the WebLogic Integration Administration Console, use the password of the identity keystore.
Server certificate	Digital certificate of the remote trading partner. Configuring the server certificate is required when using the SSL protocol. **Certificate is:** • Type X.509 version 1 or 3 • PEM or DER encoded. (The filename extension specifies the encoding type: `.pem` or `.der`.) • Required for remote trading partner types when HTTPS is used with mutual authentication

Table 4-4 Certificates Supported in Trading Partner Integration (Continued)

Type	Description
Signature certificate	Certificate required of each trading partner if digital signature support, a requirement for nonrepudiation, is configured. Used in message-level security. For a description of digital signature support, see "Digital Signatures" on page 4-34. **Certificate is:** • Type X.509 version 1 or 3 • DER encoded (ebXML or RosettaNet) or PEM encoded (ebXML only) • Read by using the RSA CertJ package (for RosettaNet) or RSA/DSA for (ebXML XMLDSIG) • Required for all trading partner types that use a digital signature service **Private Key is:** • Presented only in PKCS8 format • Always password-protected.
Encryption certificate	Certificate required of each trading partner when business message encryption is configured. Used in message-level security. Note that encryption support is available only with the RosettaNet protocols. For a description of message encryption, see "SSL Protocol" on page 4-12. **Certificate is:** • Type X.509 version 1 or 3 • DER encoded • Read by using the RSA CertJ package • Required for all trading partner types that use an encryption service **Private Key is:** • Presented only in PKCS8 format • Always password-protected.

Guidelines for Using Trading Partner Certificates

Note the following general rules about configuring trading partner certificates:

- Each trading partner may have one client certificate and an unlimited number of encryption and signature certificates. A remote trading partner also has a server certificate for the system on which it is hosted. The name of this server certificate must be specified when you configure that trading partner.

- For each certificate, there is a trading partner type: Local or Remote. In the WebLogic Integration Administration Console, configuration options differ between local and remote trading partners. For example, the tab for configuring a remote trading partner does not contain fields for entering information about private keys because information about private keys should be set only for local trading partners.
- For local trading partners, you do not configure a server certificate in the Trading Partner Management section of the WebLogic Integration Administration Console. However, the Server PrivateKey alias and pass phrase should be configured in the WebLogic Server Administration Console.
- Passwords are required for private keys of the local trading partner. If no password is provided, WebLogic Integration uses the KeyStore password to store the private keys in the identity keystore.
- You can configure certificates using the default trading partners described in "Default Domain Security Configuration" on page 4-8. For example, you can configure TP1 as the local trading partner and TP2 as the remote trading partner. If you configure TP2 as the remote trading partner, you can configure certificates on the local machine and export them to the different machine (using the TPM import / export features described in "Exporting Management Data" and "Importing Management Data" in Trading Partner Management in *Managing WebLogic Integration Solutions*). However, before importing on the remote machine, you must first create the private key in the keystore on the remote machine—you cannot copy the private key configuration to the remote machine.

Digital Certificates for Local and Remote Trading Partners

Configuration requirements regarding digital certificates differ between local and remote trading partners.

- For local trading partners, you configure:
  - client certificates plus private keys
  - signature certificates plus private keys
  - encryption certificates plus private keys

  You do not configure a server certificate for a local trading partner. A client certificate, as well as encryption and signature certificates, are required if mutual authentication with SSL is used. All of these certificates require associated private keys. You can use the same certificate and private key pair for all of these functions, as long as the key-usage in the certificate covers these functions.

- For remote trading partners, you configure the following certificates only:
  - client certificates
  - server certificates
  - signature certificates
  - encryption certificates

  You do not specify private keys for remote trading partner certificates. If you are using mutual authentication with SSL, then encryption and signature, client, and server certificates are required.

Configuring Digital Certificates

You use the WebLogic Integration Administration Console to configure digital certificates. Digital certificates are stored in the identity keystore. Before you can configure digital certificates, the trading partner must be defined in the TPM repository and the identity keystore must be configured. For more information, see "Keystore for Private Keys and Certificates" on page 4-9.

For configuration instructions, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Adding Certificates to a Trading Partner"
- "Viewing and Changing Certificates"
- "Deleting Certificates, Bindings, or Custom Extensions"

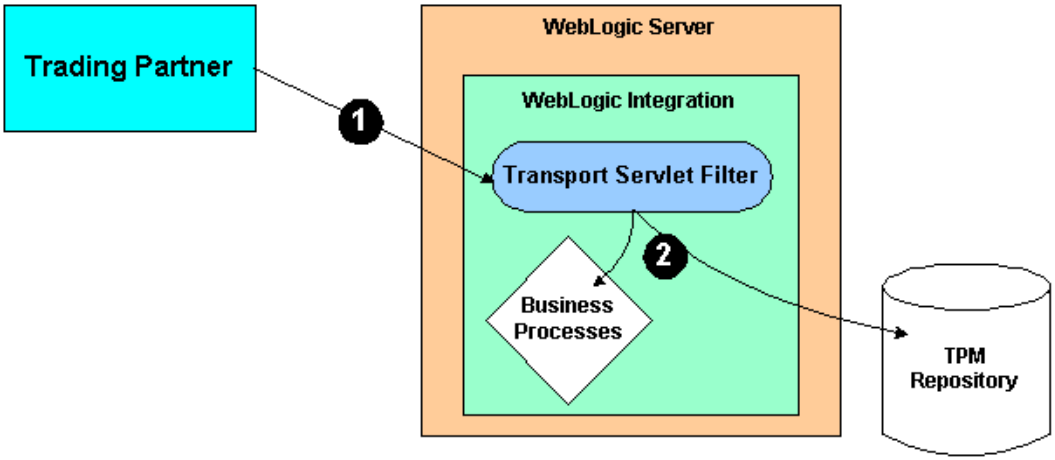
Note: WebLogic Integration does not validate any of the trading partner certificates against a trusted Certificate Authority as you load them into the keystore.

Authenticating Trading Partner Messages

As described in "Authentication Levels" on page 4-14, after a trading partner's certificate has been validated by WebLogic Server, WebLogic Integration needs to authenticate the trading partner message before the message itself can be serviced. Authenticating the trading partner message involves verifying that the sender of the business message is a valid trading partner listed in the TPM repository. After a trading partner message has been authenticated, the trading partner's identity is recognized and access to various trading partner integration resources is provided—based on the configured policies—while processing that message.

The following figure shows the process of authenticating a trading partner message.

Figure 4-3 Authenticating a Trading Partner Message



In the preceding figure, note the following:

- The Transport Servlet Filter is the entry point into WebLogic Integration. When the trading partner message arrives in the Transport Servlet Filter, as shown by callout 1, the Transport Servlet Filter verifies the trading partner message, ensuring that the trading partner name is valid by retrieving its value from a valid certificate associated with the trading partner.
- When the trading partner message is authenticated, the trading partner is authorized for access to WebLogic Integration resources, such as business processes, the TPM repository, and other resources described in "Trading Partner Integration Resources Requiring Security Policies" on page 4-11.

Authenticating Remote Users in Two-Way Authentication

This section describes how the `TPMUserNameMapper` class, which helps find the association between a remote trading partner's identity and a WebLogic Server user.

Note: `TPMUserNameMapper` applies only to two-way authentication. If your deployment uses a different authentication mechanism, you can skip this section.

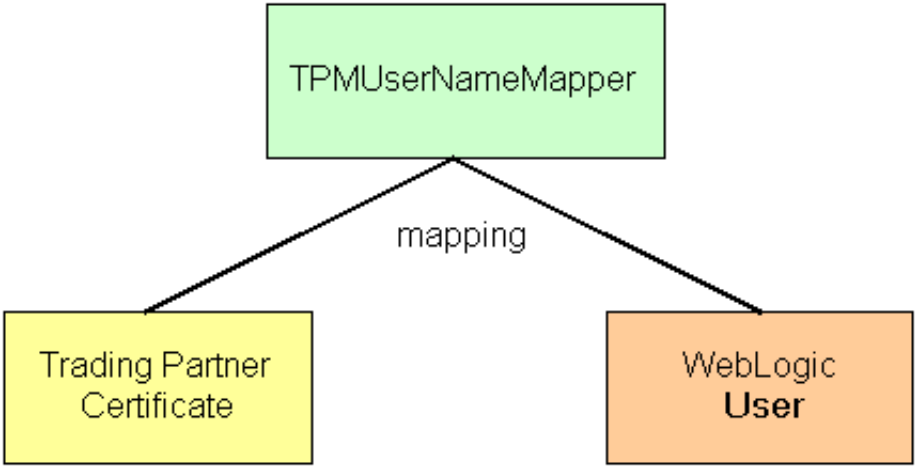
About the TPMUserNameMapper Class

Note: `TPMUserNameMapper` applies only to *remote*—not local—trading partners. When configuring a local trading partner, you do not need to provide a WebLogic Server username for that trading partner.

The `TPMUserNameMapper` class helps find the association between a remote trading partner's identity and a WebLogic Server user. When you configure a trading partner profile in WebLogic Integration, you also specify the trading partner name bound to that profile. To associate a user with a trading partner in the WebLogic Integration Administration Console, specify the trading partner username, which is a WebLogic Server username.

At run time, WebLogic Server maps the digital certificate for that trading partner to the trading partner user, as shown in the following figure.

Figure 4-4 Mapping a Trading Partner Certificate to a WebLogic Server User



If mutual authentication is used (CLIENT-CERT in `web.xml` and SSL configured for mutual authentication in WebLogic Server), the TPMUserNameMapper uses two schemes to find the certificate to the user mapper in the following manner. When a trading partner message arrives

in WebLogic Server from a remote trading partner, TPMUserNameMapper is invoked just before completing the SSL handshake. First, it looks into the TPM repository for a trading partner-WebLogic Server user association using the fingerprint of the client certificate of the remote trading partner. If not found, it tries to map an attribute of the client certificate to the WLS user.

Configuring the DefaultIdentityAsserter to Use TPMUserNameMapper

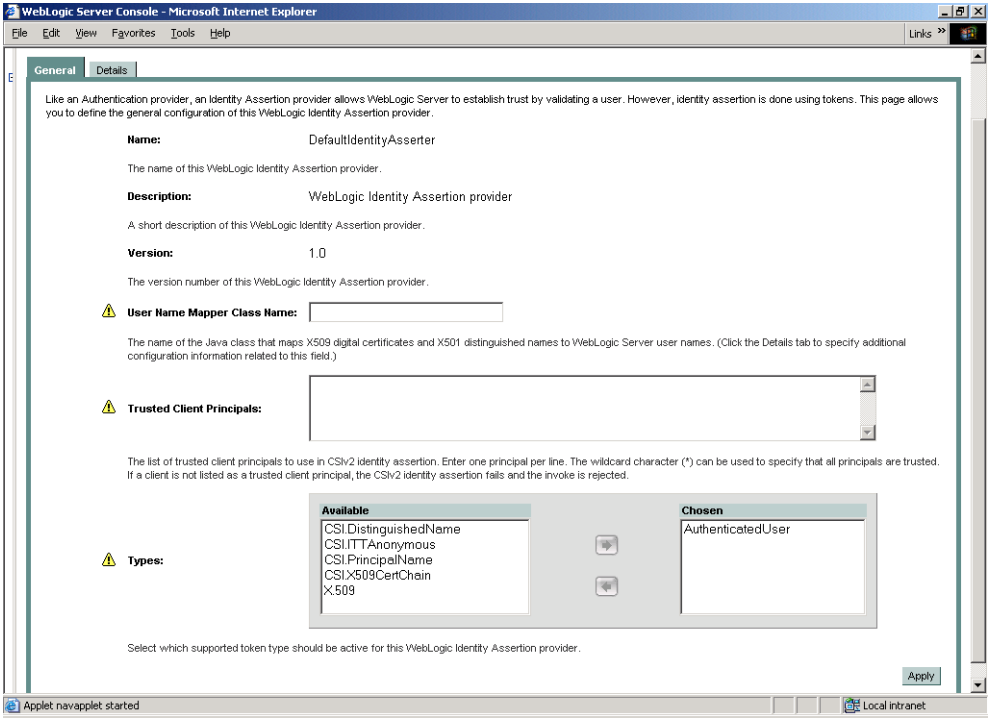
You need to configure the DefaultIdentityAsserter settings in WebLogic Server to use TPMUserNameMapper so that the WebService/SOAP, ebXML and RosettaNet protocols can use the same UserNameMapper in WebLogic Integration.

To configure the TPMUserNameMapper:

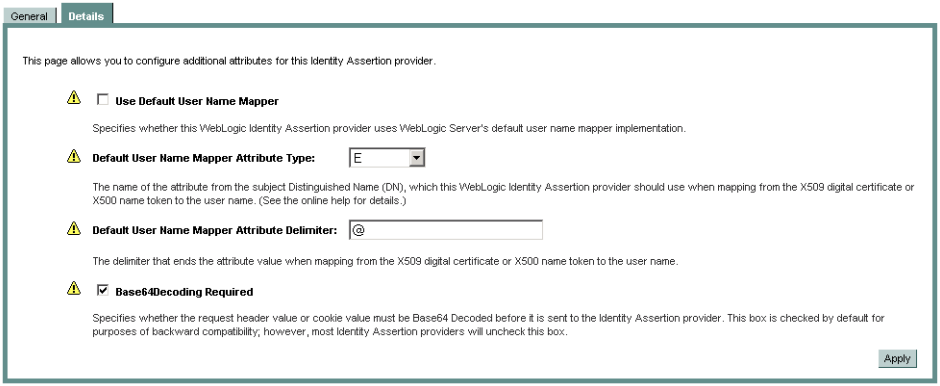
1. Start WebLogic Server in the WebLogic Integration domain you are using for trading partner integration.
2. Start the WebLogic Server Administration Console.
3. In the left navigation pane, navigate to the WebLogic Integration domain you are using for trading partner integration, and then choose:

   Security→Realms→myrealm→Providers→Authentication→DefaultIdentityAsserter

   The WebLogic Server Administration Console displays the configuration tabs for the DefaultIdentityAsserter.



4. In the User Name Mapper Class Name field, type `com.bea.b2b.security.TPMUserNameMapper`.
5. From the list of available types, select `X.509` and click the right arrow.
6. Optionally, you can configure WebLogic Integration to attempt to map an attribute of the client certificate to a WLS user *if* WebLogic Integration cannot find the association in the TPM repository. To configure this functionality:
   a. Click the Details tab.



b. Make sure that the **Use Default User Name Mapper** checkbox is *cleared* (unchecked).

c. Select the Default User Name Mapper Attribute Type (C, CN, E, L, O, or OU), which is the attribute of the subject distinguished name (DN) in a digital certificate used to create a username.

d. Select the Default User Name Mapper Attribute Delimiter, which is the delimiter that ends the user name. (The User Name Mapper uses everything to the left of the delimiter to create a username).

7. Click Apply.

8. Restart WebLogic Server.

Implementing a Custom UserNameMapper

The `com.bea.b2b.security.TPMUserMapper` class implements the WebLogic Server `weblogic.security.providers.authentication.UserNameMapper` interface. You could create your own implementation of this API, if you wanted, but using the `TPMUserNameMapper` class provides you access to the TPM repository as well. For more information, see "Interface UserNameMapper" at the following URL:

`http://e-docs.bea.com/wls/docs81/javadocs/weblogic/security/providers/authentication/UserNameMapper.html`

Verifying Certificates in Two-Way Authentication

To verify a trading partner's digital certificate in WebLogic Integration, you use a certificate verification provider (CVP). The Trading Partner Integration security framework provides a Service Provider Interface (SPI) that allows you to insert a Java class implementing SPI that can call out to a third-party service to verify trading partner certificates. Such an implementation, called a certificate verification provider, can call out to one of the following certificate verification applications:

- A Certificate Revocation List (CRL) implementation
- An Online Certificate Status Protocol (OCSP) implementation that interacts with a trusted third-party entity, such as a certificate authority, for real-time certificate status checking
- Your own certificate verification implementation

If you are using a certificate verification provider (CVP), you need to configure it in the WebLogic Integration Administration Console, as described in "Specifying the Certificate Verification Provider" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Benefits of Certificate Verification

The purpose of trading partner certificate verification is to validate the trading partner's digital certificate. For example, verifying a certificate may involve some or all of the following tasks:

- Traversing the certificate chain to the root certificate authority
- Checking a certificate revocation list (CRL) for all the certificates in the chain to identify any of those that have been revoked
- Performing a real-time certificate check with a trusted vendor, who can verify the certificate
- Checking to make sure all dates in the certificate chain are valid
- Verifying the signature of each certificate in the chain

Configuring and using a CVP implementation is optional, but doing so can provide an additional level of security in the certificate verification process.

When WebLogic Integration Uses the Certificate Verification Provider

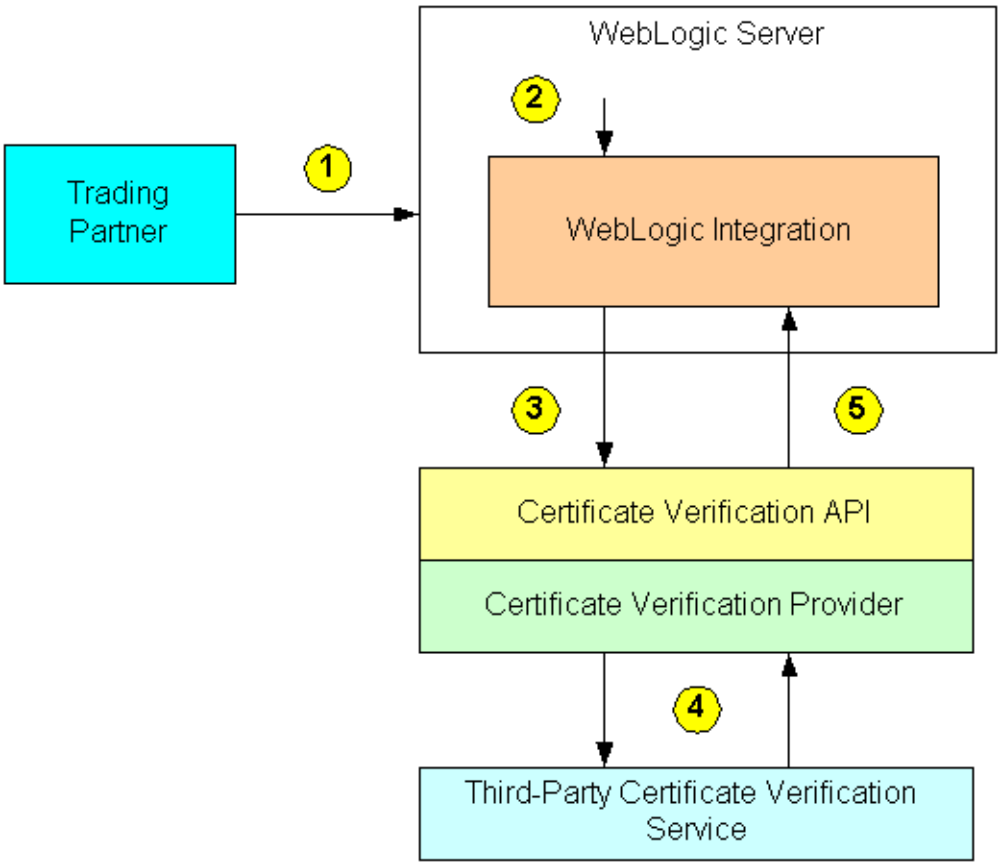
The CVP is used by WebLogic Integration in the following cases:

- inbound SSL and mutual authentication
- outbound SSL and mutual authentication

Certificate Verification Process

The following figure is an example of the sequence of events that occur during the certificate verification process in WebLogic Integration for an incoming message using SSL and mutual authentication.

Figure 4-5 Trading Partner Certificate Verification in WebLogic Integration



Note the following callouts in the preceding figure.

Callout	Description
1	Certificate verification is used in SSL. The trading partner and the WebLogic Server system perform an SSL handshake, during which they exchange certificates to establish each other's identity. The Certificate Authority of the trading partner digital certificate must be trusted in WebLogic Server. During this handshake, WebLogic Server verifies the following: • The Certificate Authority of the trading partner certificate must be one that is trusted in the WebLogic Server environment. • The trading partner certificate has not expired. When the SSL handshake is completed, the trading partner's network connection to the WebLogic Server system is established.
2	WebLogic Server dispatches the message to WebLogic Integration.
3	WebLogic Integration invokes the CVP interface to the implementation that calls out to the third-party certificate verification service.
4	The CVP implementation calls out to the third-party certificate verification service, which returns the status of the trading partner certificate.
5	The CVP implementation returns the appropriate status of the certificate to WebLogic Integration.

Implementing a Certificate Verification Provider

A certificate verification provider (CVP) Java class must implement the `com.bea.wli.security.verification.CertificateVerificationProvider` interface. You have two choices for what a CVP class can call out to:

- A trusted third-party vendor that conforms to the service provider interface, as described in "Using the Service Provider Interface" on page 4-29.
- Your own certificate verification application.

Regardless of which choice you pick, you need to create a Java implementation of the CVP SPI that calls out to the application that performs the actual certificate verification. Creating, compiling, and configuring this CVP application is explained in the subsections that follow.

Using the Service Provider Interface

Trading Partner Integration allows you to implement a CVP via the `com.bea.wli.security.verification.CertificateVerificationProvider` interface,

which provides the CVP service provider interface (SPI). If you implement or use a CVP using the SPI described in this section, you must later configure this CVP in the WebLogic Integration Administration Console so that the CVP is invoked properly during run time.

The `com.bea.wli.security.verification.CertificateVerificationProvider` interface has the following methods, which a CVP application must implement:

- `void init()`

  This method is automatically invoked by WebLogic Integration to invoke any custom initialization processes in the class you create that implements this interface. This method is invoked only once, at the startup of WebLogic Integration.

- `String verify(Certificate[] certs)`

  This method validates the certificate chain obtained during the SSL handshake. It should return one of the following `String` values:

  – `good`—the trading partner certificate is valid and not expired.

  – `revoked`—the trading partner certificate has been revoked by one of the certificate authorities in the certificate chain, or the trading partner certificate has expired.

  – `unknown`—none of the certificate authorities in the certificate chain is able to establish the validity of the trading partner certificate.

  The implementer can choose the validation procedure performed by this method. For example, this method can check certificate revocation lists (CRLs) stored in files, it can check the certificate status in real-time using the Online Certificate Status Protocol (OCSP), or it can use any other mechanism, as appropriate.

Notes: If you implement a CVP, you need to add a default public constructor for the CVP with no arguments. Neither the constructor nor any methods in the class should throw any exceptions.

If you do not configure a CVP, any certificate issued by a trusted certificate authority is considered by WebLogic Integration to be valid.

Compiling the Certificate Verification Provider Class

If you implement a CVP, after you create the CVP Java class, you must compile it and place it in the system `CLASSPATH`.

Configuring a Certificate Verification Provider with Trading Partner Integration

You must configure the CVP via the WebLogic Integration Administration Console or the Bulk Loader utility. After you configure the CVP, restart WebLogic Server so that the CVP can take

effect. If you do not configure a CVP, any certificate issued by a trusted certificate authority is considered by WebLogic Integration to be valid.

You use the WebLogic Integration Administration Console to configure a CVP. For more information, see "Specifying the Certificate Verification Provider" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

After you configure a CVP, restart WebLogic Server so that the CVP can take effect.

Authorization

Authorization pertains to granting an entity permissions and rights to perform certain actions on a given resource. The authorization process is a procedure for granting such rights.

Roles and Policies

Permission to access trading partner resources is assigned through policies and roles—for any resource that needs to be protected, its security policy will be defined based on roles. Individual users/entity will thus be able to get access depending upon the roles that they belong to. Whereas authentication is concerned with *who* an entity is—it is the association of an identity with an entity—authorization is concerned with what that identity is *allowed* to see and do.

Note: Authorization is available (but not required) with basic, one-way plus basic, and mutual authentication.

Authorization to Trading Partner Integration Resources

Authorization determines whether access is provided to trading partner integration resources, such as:

- B2BDefaultWebApp and endpoint URIs of protocol bindings of local trading partners in the TPM repository
- WebLogic Workshop business processes
- JDBC connection pools (`bpmarchpool` and `cgpool`) and MBeans that are used to access the TPM repository
- JMS destination (for message tracking, asynchronous dispatcher queues for trading partner integration business processes)

Authorization Levels

For trading partner integration, WebLogic Integration incorporates two levels of authorization:

- Authorization of the trading partner for access to the Transport Servlet Filter.
- Authorization in the service associated with the trading partner business message.

Trading Partner Authorization

WebLogic Server performs trading partner authorization. When the trading partner message arrives in WebLogic Server, and the trading partner and WebLogic Server complete the mutual or basic (username and password) authentication procedure, authorization is performed by WebLogic Server to access the Transport Servlet Filter.

The preferred way to configure the B2BDefaultWebApp is to use the WebLogic Server Administration Console to set policies on the B2BDefaultWebApp for access to URLs. For instructions, see "URL (Web) and EJB (Enterprise JavaBean) Resources" in "Types of WebLogic Resources" in *Securing WebLogic Resources* at the following URL:

`http://edocs/wls/docs81/secwlres/types.html`

Note: In addition to configuring B2BDefaultWebApp, you can also configure other trading partner integration resources (such as the JDBC connection pool and MBeans used to access the TPM repository, WebLogic Workshop business processes, and JMS destinations) that need to be configured as well. For more information, see "Authorization to Trading Partner Integration Resources" on page 4-31.

Alternatively, you can specify Transport Servlet Filter ACLs in the `web.xml` file. However, this is not the recommended approach. The following example shows a `web.xml` file that specifies the ACLs for a Transport Servlet Filter named `B2BTransport`.

Listing 4-1 Example Transport Servlet Filter ACL

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 1.2//EN
" "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
...
...
<!-- Authentication -->
 <security-constraint>
  <web-resource-collection>
   <web-resource-name>B2BTransport</web-resource-name>
   <url-pattern>*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
   <role-name>PremiumTradingPartner</role-name>
```

```
  </auth-constraint>
 </security-constraint>

 <login-config>
  <auth-method>CLIENT-CERT</auth-method>
 </login-config>

 <security-role>
  <role-name>PremiumTradingPartner</role-name>
 </security-role>
</web-app>
```

In the preceding code example:

- `B2BTransport` is the Transport Servlet Filter whose endpoint is defined in the TPM repository.
- `PremiumTradingPartner` is a WebLogic Server user group in which all the trading partner WebLogic Server users are members.
- `CLIENT-CERT` specifies that the mode of authentication required to access the Transport Servlet Filter is SSL with mutual authentication. You can also use HTTP basic authentication.

Service Authorization

When WebLogic Integration performs service authorization, the server examines the content of the trading partner business message with respect to the service profile to which the trading partner is bound. That is, for a service profile, a trading partner may send only a specific set of business messages. WebLogic Integration validates the business message against the following information specified in the service profile for a particular service:

- Party information (trading partner)
- Service name
- Protocol binding

Once the service authorization is complete for an incoming business message, access to the B2B resources is dictated by WebLogic resource policies.

Message-Level Security

Message-level security involves digital signatures, encryption, and non-repudiation for individual business messages. This topic describes message-level security concepts and tasks for trading partner integration. It contains the following sections:

- Digital Signatures
- NonRepudiation
- Encryption—PKCS7 Enveloped Data for RosettaNet 2.0

Using digital signatures prevents tampering with business messages. Using encryption ensures message privacy. Nonrepudiation allows trading partners to prove or disprove having previously sent or received a particular business message to or from another trading partner.

Message-level security is configured in the protocol bindings that define communications between trading partners. For more information, see "Defining Protocol Bindings" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Digital Signatures

Digital signatures provide a means of preventing anyone or anything from tampering with the contents of a business message, especially when the business message is in transit between two trading partners. After verifying a signature, WebLogic Integration uses a Certificate Verification Provider (if two-way authentication is configured), as described in "Authenticating Remote Users in Two-Way Authentication" on page 4-23. Digital signatures are required for nonrepudiation, which is described at "NonRepudiation" on page 4-37.

WebLogic Integration Support for Digital Signatures

WebLogic Integration supports the following types of digital signatures:

- XMLDSig for ebXML 1.0 and ebXML 2.0
- Public Key Cryptography Standard 7 (PKCS7) Enveloped Data for RosettaNet 1.1 and 2.0

About Digital Signatures

A digital signature itself is a set of data appended to a business message consisting of an encrypted, one-way hash value of data packaged in a specific format (for example, PKCS7 SignedData or XMLDSIG signature). A digital signature:

- Validates that the contents of a digitally signed message have not been tampered with.
- Contains the identity of the sender of the business message.

The data required to create a digital signature is obtained from the trading partner configuration data in the TPM repository. The information required to create a digital signature also includes the following:

- Trading partner signature certificate and private key
- Certificate authority certificate for the trading partner signature certificate
- For ebXML, an XPath transform (optional)

You can configure whether to sign business messages or not in the protocol bindings that define communications between trading partners. For more information, see "Defining Protocol Bindings" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

After validating a signature, WebLogic Integration invokes the certificate verification provider (CVP), which is described in "Certificate Verification Process" on page 4-28.

XMLDSig for ebXML 1.0 and ebXML 2.0

WebLogic Integration supports XMLDSig for ebXML 1.0 and ebXML 2.0 message exchanges between trading partners.

Supported XMLDSig Features

WebLogic Integration supports the following XMLDSig features:

- Digital signatures for multipart messages
  - ebXML SOAP Envelope
  - One or more part(s)
- Signed acknowledgements (ebXML 2.0 only)
- Sender verification

For more information about XMLDSig, see "XML-Signature Syntax and Processing" on the W3C web site at the following URL:

`http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/Overview.html`

Supported XMLDSig Algorithms

WebLogic Integration uses the following algorithms for XMLDSig:

- Signature Algorithm
  - DSA-SHA1

    `http://www.w3.org/2000/09/xmldsig#dsa-sha1`
  - RSA-SHA1

    `http://www.w3.org/2000/09/xmldsig#rsa-sha1`
- Digest Algorithm

  `SHA1 http://www.w3.org/2000/09/xmldsig#sha1`
- Canonicalization Algorithm

  `http://www.w3.org/TR/2001/REC-xml-c14n-20010315`
- Transform Algorithms
  - Enveloped Signature

    `http://www.w3.org/2000/09/xmldsig#enveloped-signature`
  - Xpath

    `http://www.w3.org/TR/1999/REC-xpath-19991116`
  - Canonicalization

    `http://www.w3.org/TR/2001/REC-xml-c14n-20010315`

Digital Signature with PKCS7 Enveloped Data for RosettaNet 1.1 and RosettaNet 2.0

For RosettaNet 1.1 and 2.0, WebLogic Integration supports digital signature with PKCS7 Enveloped Data.

Supported PKCS7 Enveloped Data Digital Signature Features

WebLogic Integration supports PKC7 enveloped data for digital signatures for RosettaNet 1.1 and 2.0. Digital signatures for multipart messages apply to:

- Service Header (optional)
- Service Content
- One or more attachments

Supported PKCS7 Enveloped Data Digital Signature Algorithms

WebLogic Integration supports the following PKC7 enveloped data algorithms:

- Hash algorithm name: SHA1
- Signature algorithm name: RSA

NonRepudiation

Nonrepudiation, or the ability to provide legal evidence of the involvement of a denying party, is a legal requirement for critical business messages. Nonrepudiation is the ability of a trading partner to prove or disprove having previously sent or received a particular business message to or from another trading partner. WebLogic Integration supports:

- **Nonrepudiation of origin**—Links the message received and the sender of the message. It provides legal evidence that you have sent a business message.
- **Nonrepudiation of receipt**—Links the message processed and the recipient of the message. It provides legal evidence that you have received a business message.

Nonrepudiation is configured in the protocol bindings that define communications between trading partners. For more information, see "Defining Protocol Bindings" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Nonrepudiation Example

Trading Partner A has agreed to purchase 1000 ergonomic chairs from Trading Partner B. In the course of this agreement, Trading Partner A has sent a business message to Trading Partner B agreeing to buy the chairs at a set price. Later, though, Trading Partner A disputes the original price and denies having sent a message in which they agreed to pay that price.

If a reliable nonrepudiation system has been in place, Trading Partner B can disprove Trading Partner A's claim by producing a document from Trading Partner A specifying the amount Trading Partner A agreed to pay. Further, if this original document is digitally signed, timestamped, recorded, and secured by a trusted third-party source, the validity of this document has full legal recourse.

Nonrepudiation Services

To support nonrepudiation, WebLogic Integration provides the following services:

- **Digital signatures**—Used to digitally sign a business document before it is sent to the recipient.

- **Secure Audit log SPI**—Used to store digitally signed business messages with a secure timestamp. Audit logging is necessary for nonrepudiation.
- **Secure timestamp SPI**—Used to sequence the occurrence of events in the business transaction.

Digital Signatures

Digital signatures are required for nonrepudation because they provide a means of preventing anyone or anything from tampering with the contents of a business message, especially when the business message is in transit between two trading partners. For more information, see "Digital Signatures" on page 4-34.

Secure Audit Log

A *secure audit log* is required for nonrepudation because it typically stores each business message with its digital signature and secure timestamp, allowing a trading partner to reconstruct the sequence of messages and other system events that have occurred during the exchange of business messages with other trading partners, along with the data exchanged.

The default audit log provider (`com.bea.wli.security.audit.DefaultAuditLogProvider`) logs to a file named `secureaudit.log`. This file is based on the logging subsystem and is protected by only the underlying operating system's file permissions system. This file is not digitally signed or encrypted. It should be used only for demo or development purposes, not in a production environment.

You enable and disable the audit log and specify the audit log class in the WebLogic Integration Administration Console. For more information, see "Configuring Secure Audit Logging" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Audit Log Messages

All log messages correspond to the DTD `log-message.dtd`, which defines the contents for each message type.

All audit log messages have the following three identifiers:

- Location—the location, in WebLogic Integration, in which the message is stored
- Type—the message type
- Data—the actual information that is being logged

The following table describes the contents of the data for each of the message types. All the log messages contain the timestamp obtained from the timestamp provider that is configured in WebLogic Integration.

Message Type	Description
NRR	Nonrepudiation of receipt. Contains that name of the trading partner receiving the business message and the application data.
NRO	Nonrepudiation of origin. Contains the name of the trading partner sender, the business message, and the application data.
APP	Is logged from any trading partner Java class via the `Audit.log(byte[] data)` method.The data format for this message type is any stringified XML document. Because the application is logging the message, the contents of the data are controlled by the application itself.

Audit Log DTD

The following code example shows the `log-message.dtd` file:

Listing 4-2 Sample log-message.dtd file

```
<!ELEMENT LOG (non-repudiation-origin| non-repudiation-receipt |
application)>
<!ATTLIST LOG  time-stamp CDATA  #REQUIRED >
<!ATTLIST LOG location CDATA  #IMPLIED >
<!ATTLIST LOG Principal CDATA  #IMPLIED >
<!ELEMENT non-repudiation-origin (#PCDATA)>
<!ELEMENT non-repudiation-receipt (#PCDATA)>
<!ELEMENT application (#PCDATA)>
```

Using the SPI for the Secure Audit Log

WebLogic Integration provides a Service Provider Interface (SPI) for you to configure a trusted, third-party provider of the secure audit log. If you incorporate a secure audit log service from a trusted third-party provider, you need to create a class that implements the

`com.bea.wli.security.audit.AuditLogProvider` interface. In the methods of your class (for example, `log`), you call out to the third party audit log provider.

Note: If you implement an audit log service using the SPI described in this section, you must configure this service later in the WebLogic Integration Administration Console so that the service is invoked properly during run time.

The `com.bea.wli.security.audit.AuditLogProvider` interface has the following methods, which a secure audit log application must implement:

- `void init()`

  This method initializes the audit log.

- ```
  void log (java.lang.String component,
            java.lang.String type,
            byte[] data,
             java.lang.String principal)
  ```

  This method is invoked to log a message in the secure audit log. It has the following parameters:

  - `java.lang.String component`

    Contains the component that is logging the message

  - `java.lang.String type`

    Specifies the type of the nonrepudiation message

  - `byte[] data`

    Contains the data to be logged

  - `java.lang.String principal`

    Contains the name of the trading partner who is logging this message

Your implementation of the secure audit interface must include a default public constructor with no arguments. Neither the constructor nor any methods in the class that implements the `AuditLogProvider` interface should throw any exceptions.

Writing to the Audit Log Directly

As an alternative to writing a Java implementation of the `com.bea.wli.security.audit.AuditLogProvider` interface to call out to an application that writes to the audit log, you can write an application that writes to the audit log directly via an invocation to the `com.bea.wli.security.audit.Audit.log(byte[] data)` method, as
  ```

shown in the following code example from a business process. In this example, the bolded code shows the statements that have been added to show writing to the audit log.

Listing 4-3 Example of Writing to the Audit Log Directly

```
package orderprocessing;
import com.bea.jpd.JpdContext;
import com.bea.xml.XmlObject;
import com.bea.data.MessageAttachment;
// Import the Audit class from the WLI security package
import com.bea.wli.security.audit.Audit;
/**
 * @jpd:process process::
 * <process name="ServerBuyer">
 *   <clientRequest name="Receive order request from client" method="start"/>
 *   <controlSend name="Send PO to enterprise server seller"
method="sendOrder"/>
 *   <controlReceive name="Receive PO receipt from enterprise server seller"
method="orderService_onMessage"/>
 *   <clientCallback name="sendAck" method="sendAck"/>
 * </process>::
 *
 */
public class EnterpriseServerBuyer implements com.bea.jpd.ProcessDefinition
{
    public com.bea.tutorial.b2B.order.OrderDocument pcOrder;
  /**
   * @jc:ebxml ebxml-service-name="SecureOrderService" from="BEA-IT-id"
to="SUN-id" ebxml-action-mode="default"
   * @common:control
   */
  private SecureOrderServiceControl orderService;
  /**
   *@common:context
   */
  JpdContext context;
  public void start( String str )
  {
    //create an order
    pcOrder = ...
  }
  public void sendOrder()
  {
        //#START: CODE GENERATED - PROTECTED SECTION - you can safely add code
above this comment in this method. #//
        // input transform
```

```
        // method call
        orderService.sendOrder(pcOrder);
        // output transform
        // output assignments
        //#END  : CODE GENERATED - PROTECTED SECTION - you can safely add code
below this comment in this method. #//

    }
  public void orderService_onMessage(MessageAttachment[] reply)
  {
    //assume only one object of type XmlObject in reply
    XmlObject xo = reply[reply.length - 1].getXmlObject();
    if(Audit.isEnabled()) {
      Audit.log(xo.toString().getBytes());
    }
  }
  public Callback callback;
  public interface Callback {
    public void onAck(String reply);
  }
  void sendAck() {
    callback.onAck("This is an ACK from ServerBuyer.jpd.");
  }
}
```

Timestamp Provider

A *timestamp provider* is required for nonrepudation because a secure timestamp service attaches a Coordinated Universal Time (UTC) timestamp to the secure audit log when business messages are also logged to the secure audit log, providing precise time and date information.

For example, when a trading partner receives a business message, a timestamp is entered as a nonrepudiation of receipt (NRR) message in the audit log. When a trading partner sends a business message, a timestamp is entered as a nonrepudiation of origin (NRO) message in the audit log.

You configure the timestamp provider in the WebLogic Integration Administration Console. For more information, see “Configuring Secure Audit Logging” in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Exclusive and Default Timestamps

WebLogic Integration prohibits more than one secure timestamp provider from being registered in WebLogic Integration. This restriction ensures that all timestamps created in WebLogic Integration are ordered chronologically.

Note: If you do not configure a secure timestamp service provider in WebLogic Integration, system time is used for timestamping system events and signatures if the default log provider is used.

Using the SPI for the Secure Timestamp Service

Trading Partner Integration includes a Service Provider Interface (SPI) so that you can incorporate a secure timestamp service from a trusted third-party provider.

If you incorporate a secure timestamp service from a trusted third-party provider, you need to create a Java class that implements the `com.bea.wli.security.time.TimestampProvider` interface. In the methods (for example, `getTimestamp`) of your class implementing the `com.bea.wli.security.time.TimestampProvider` interface, you call out to the third party timestamp provider.

Trading Partner Integration allows you to create a customized secure timestamp service by implementing the `com.bea.wli.security.time.TimestampProvider` interface. If you implement a timestamp using the SPI described in this section, you must configure this service later in the WebLogic Integration Administration Console so that the service is invoked properly during run time.

The `com.bea.wli.security.time.TimestampProvider` interface has the following methods, which a timestamp application must implement:

- `String getTimestamp()`

  This method returns a string specifying the time in Coordinate Universal Time (UTC) format.

- `long getTimestampInMillis()`

  This method returns a string specifying the UTC time in milliseconds.

Your implementation of the timestamp interface must include a default public constructor with no arguments. Neither the constructor nor any methods in the class that implements the `TimeStampProvider` interface should throw any exceptions.

Encryption—PKCS7 Enveloped Data for RosettaNet 2.0

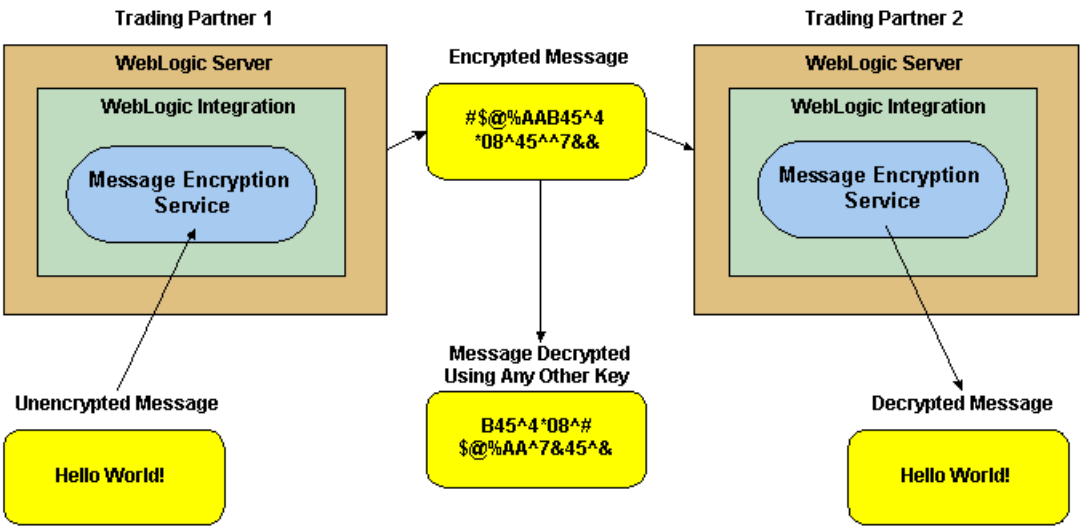
WebLogic Integration encrypts business messages for the business protocols that require it. In this WebLogic Integration release, message encryption is supported only for RosettaNet 2.0.

Note: To use message encryption, you must have a valid license for using the encryption service.

How WebLogic Integration Handles Data Encryption

The following figure shows how data encryption is performed using the public and private keys.

Figure 4-6 Message Encryption in Trading Partner Integration



Data encryption works by using a combination of the sender's certificate, private key, and the recipient's certificate to encode a business message. The message can then be decrypted only by the recipient using the recipient's private key.

Note: WebLogic Integration encryption is controlled by licensing (Encryption/Domestic or Encryption/Export), but the decryption of a business message is not. If WebLogic Integration does not have a valid encryption license, WebLogic Integration disables the encryption service. However, WebLogic Integration can always decrypt business messages that are received.

Supported Encryption Algorithms

The WebLogic Integration message encryption service supports the following algorithms:

- RC5

  For more information, see the RSA web site at: `http://www.rsasecurity.com/`

- Data Encryption Standard (DES)
- Triple Data Encryption Standard (3DES)

You use the WebLogic Integration Administration Console to enable or disable business messages encryption in the protocol bindings that define communications between trading partners. For more information, see "Defining Protocol Bindings" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Using Proxy Servers with Trading Partner Integration

This topic describes how to use proxy servers with trading partner integration. It includes the following sections:

- Configuring Trading Partner Integration to Use an Outbound HTTP Proxy Server
- Configuring WebLogic Integration with a Web Server and a WebLogic Proxy Plug-In

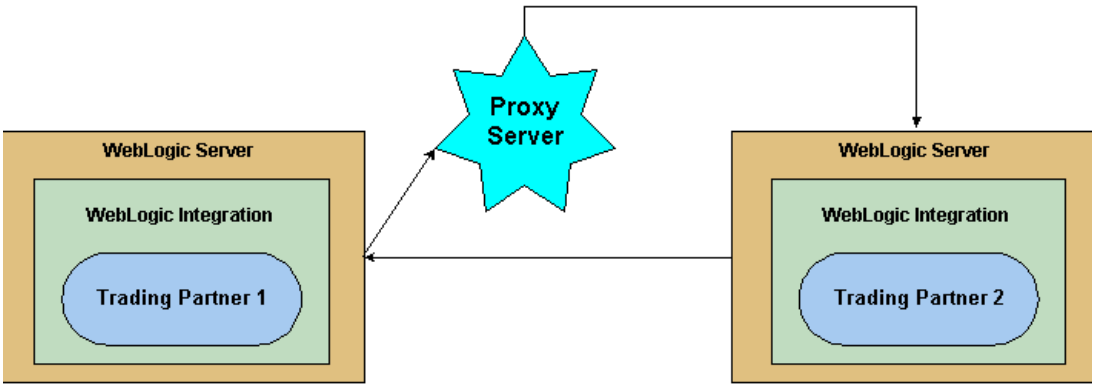
Configuring Trading Partner Integration to Use an Outbound HTTP Proxy Server

If you are using WebLogic Integration in a security-sensitive environment, you may want to use WebLogic Integration behind a proxy server. A proxy server allows trading partners to communicate across intranets or the Internet without compromising security. A proxy server is used to:

- Hide, from external hackers, the local network addresses of the WebLogic Servers that host WebLogic Integration
- Restrict access to the external network
- Monitor external network access to the WebLogic Servers that host WebLogic Integration

When proxy servers are configured on the local network, network traffic (SSL and HTTP) is tunneled through the proxy server to the external network. The following figure illustrates how a proxy server might be used in the WebLogic Integration environment.

Figure 4-7 Proxy Server



To configure a proxy server in the WebLogic Integration Administration Console, see "Configuring a Proxy Host" in Trading Partner Management in *Managing WebLogic Integration Solutions*.

Note: If you configure a proxy server, you also need to add permissions to read and write the `ssl.proxyHost` and `ssl.proxyPort` system properties for the WebLogic Server. These system properties are stored in the `weblogic.policy` file, which is located in the directory where you installed WebLogic Server. Add the following lines to the *`grant`* section of the `weblogic.policy` file:

```
permission java.util.PropertyPermission "ssl.proxyHost", "read, write";
permission java.util.PropertyPermission "ssl.proxyPort", "read, write";
```

Configuring WebLogic Integration with a Web Server and a WebLogic Proxy Plug-In

You can configure WebLogic Integration with a Web server, such as an Apache server, that is programmed to service business messages from a remote trading partner. A Web server can provide the following services:

- Receive business messages from a remote trading partner
- Authenticate a trading partner digital certificate

Services Provided by WebLogic Proxy Plug-In

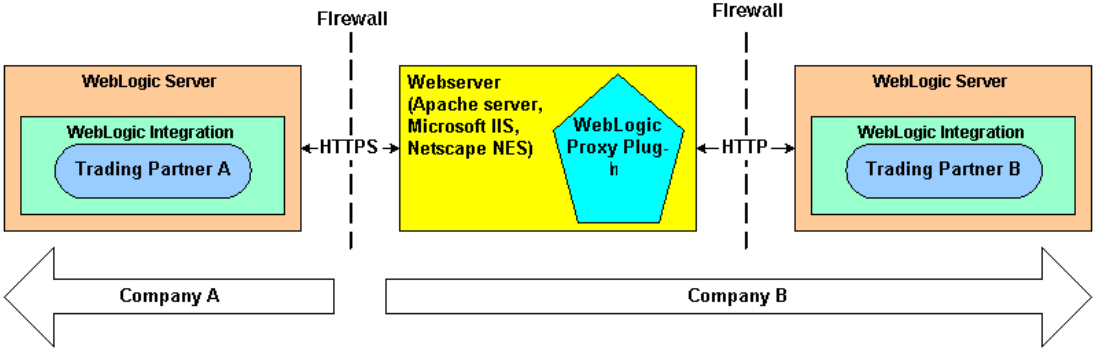
The Web server uses the WebLogic proxy plug-in, which you can configure to provide the following services:

- Forward business messages received by the Web server to WebLogic Integration, which is running inside a secure internal network.
- Extract the remote trading partner certificate from the Web server and forward it to WebLogic Server for authentication. WebLogic Integration can then authenticate the trading partner certificate and business message.

Topology Using WebLogic Proxy Plug-In

The following figure shows the topology of an environment that uses a Web server, the WebLogic proxy plug-in, and WebLogic Integration.

Figure 4-8 Using a Web Server and the WebLogic Proxy Plug-In



When using the WebLogic Proxy Plug-In, note that:

- Even though the proxy plug-in uses HTTP, you must configure WebLogic Integration to use the HTTPS protocol when using the proxy plug-in to forward business messages.
- If a trading partner in a conversation uses Microsoft IIS as a proxy server, all the certificates used in the conversation must be trusted by a well-known Certificate Authority, such as VeriSign or Entrust. The use of self-signed certificates will cause a request passed through the IIS proxy server to fail. This is a restriction in IIS, not WebLogic Integration.

Configuring the Web Server

To configure the Web server, see "Configuring WebLogic Server Web Components" in the *BEA WebLogic Server Administration Guide* at the following URL:

`http://edocs.bea.com/wls/docs81/adminguide/web_server.html`

The following code example provides the segment of `httpd.conf` (for an Apache server) needed to configure the proxy plug-in:

```
# LoadModule foo_module libexec/mod_foo.so
LoadModule weblogic_module    libexec/mod_wl_ssl.<suffix>

<Location /weblogic>
 SetHandler weblogic-handler
 PathTrim /weblogic
 WebLogicHost myhost
 WebLogicPort 80
</Location>
```

Implementing Security for Trading Partner Integration

For development and testing purposes, you use the default security configuration that is generated when you create a new WebLogic Integration domain using the WebLogic Platform Configuration Wizard. For more information, see "Default Domain Security Configuration" on page 4-8.

For a production environment, you need to configure security as part of your deployment. This topic provides a summary of the tasks that you need to complete. It contains the following sections:

- Configure Users, Groups, and Roles
- Configure Trading Partner Profiles
- Configure the Keystores
- Configure Certificates
- Configure SSL
- Configure Transport-Level and Message-Level Options in Service Profiles

Configure Users, Groups, and Roles

You use the *User Management* module of the WebLogic Integration Administration Console to manage the users, groups, and roles defined in the default security realm. For instructions on configuring users, groups, and roles in the WebLogic Integration Administration Console, see User Management in *Managing WebLogic Integration Solutions*.

Configure Trading Partner Profiles

You need to configure the profiles for trading partners with whom you will exchange business messages. For an introduction to trading partner profiles, see "Trading Partner Profiles" on page 1-6. For instructions on configuring trading partner profiles in the WebLogic Integration Administration Console, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Adding Trading Partner Profiles"
- "Defining Trading Partner Profiles"
- "Deleting Trading Partner Profiles"

Configure the Keystores

You need to create and configure the identity and trust keystore for certificates and private keys. For an introduction to the keystore, see "Keystore for Private Keys and Certificates" on page 4-9. For instructions on creating and configuring the keystore, see the following topics:

- If the identity and trust keystores do not already exist, create them according to the instructions in "Storing Private Keys, Digital Certificates, and Trusted Certificate Authorities" in "Configuring SSL" in *Managing WebLogic Security* at the following URL:

  `http://edocs.bea.com/wls/docs81/secmanage/ssl.html`

- Configure the keystores using the WebLogic Server Administration Console according to the instructions in "Configuring KeyStores" in "Configuring SSL" in *Managing WebLogic Security* at the following URL:

  `http://edocs.bea.com/wls/docs81/secmanage/ssl.html`

Configure Certificates

You need to add digital certificates to trading partners. For an introduction to certificates, see "Digital Certificates" on page 4-15. For instructions on configuring certificates in the WebLogic

Integration Administration Console, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*:

- "Adding Certificates to a Trading Partner"
- "Viewing and Changing Certificates"
- "Deleting Certificates, Bindings, or Custom Extensions"

Configure SSL

You need to configure SSL for transport-level security using the WebLogic Server Administration Console. For an introduction, see "SSL Protocol" on page 4-12. For configuration instructions, see "Configuring SSL" in *Managing WebLogic Security* at the following URL:

`http://edocs.bea.com/wls/docs81/secmanage/ssl.html`

Configure Transport-Level and Message-Level Options in Service Profiles

You need to decide the transport-level security and message-level security options that you want to use in your message exchanges, and then configure those options in the service profile for each trading partner. For example, you might use mutual authentication with one trading partner and basic authentication with another. Similarly, you might implement nonrepudiation with a customer or vendor, but not with a trading partner that is within your organization.

For instructions on how to managing service profiles in the WebLogic Integration Administration Console, see the following topics in Trading Partner Management in *Managing WebLogic Integration Solutions*.

- "Adding Service Profiles to a Service"
- "Viewing and Changing Service Profiles"
- "Deleting Service Profiles from a Service"
- "Enabling and Disabling Trading Partner and Service Profiles"

Index

E

F

I

J

K

L

M

N

P

R

S

T

U

V

W

X