BEA WebLogic® Integration

Introducing Trading Partner Integration

Version 10.2
Document Revised: March 2008

Contents

1. Introduction

2.Introducing ebXML Solutions

3.Introducing RosettaNet Solutions

4.Trading Partner Integration Security

B.Example: RosettaNet Security Configuration

CHAPTER 1

Introduction

The topic describes basic concepts, architecture, and tasks for creating WebLogic Integration(WLI) 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 more information on WebLogic Integration, see *Introducing WebLogic Integration*. For a hands-on walkthrough of building and running example ebXML and RosettaNet solutions in WebLogic Integration, see Tutorials for Trading Partner Integration, and Documentation and Example Files.

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

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

Support for Leading Industry Protocols and Standards

WLI 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 1-15, "TPM Repository Lookups Via Process and Service Broker Controls" on page 1-15, and in "Building Web Services" in the WebLogic Workshop Help.

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 1-10.

Easy Access to Run-Time Information

WLI 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

WLI 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
- Trading Partners
- Services, Service Profiles, and Protocol Bindings

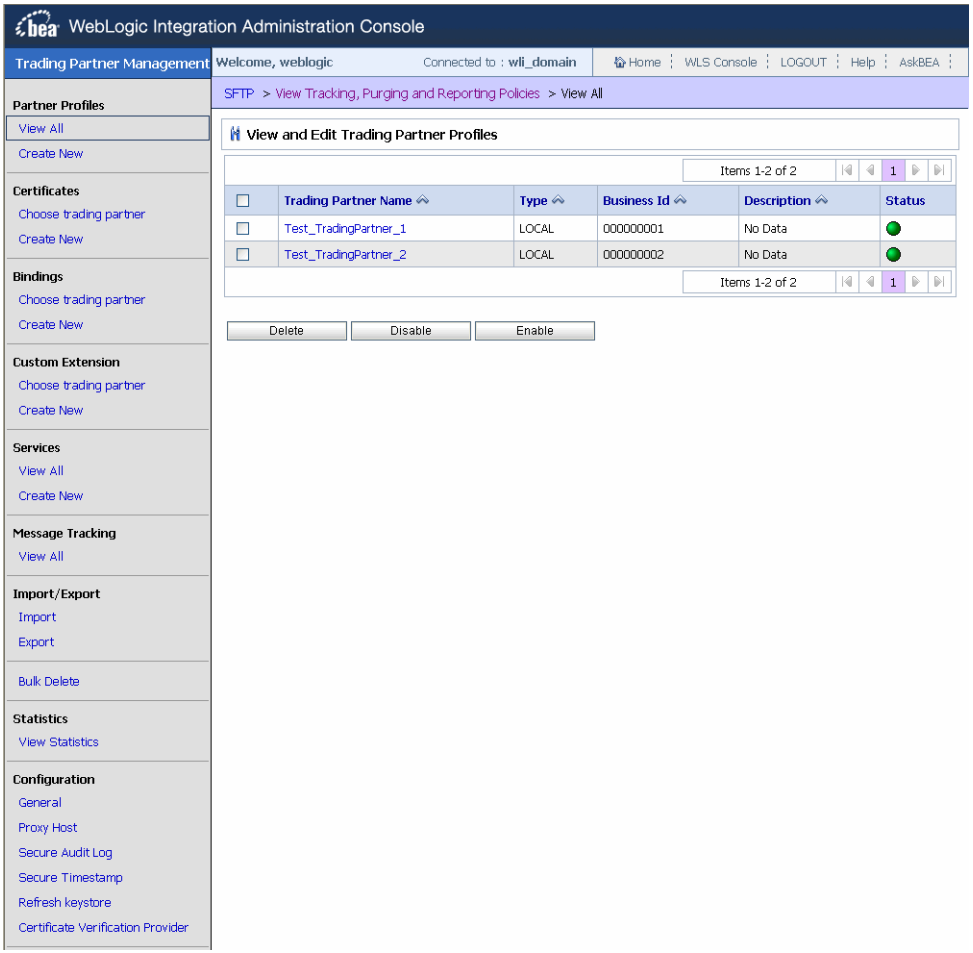
- Exchanging Data in the TPM Repository
- Default TPM Repository Settings
- MBean APIs for Third-Party Access

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.

Figure 1-1 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 *Using the WebLogic Integration Administration Help*.

Trading Partners

In WLI, 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. For example, the business purpose for such a community might be inventory management.
- Span multiple companies across firewalls and over the Internet. For example, the business purpose might be supply chain management or multistep purchasing interactions.
- Include trading partners both within a company and in other companies. For example, 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 Trading Partner Management in *Using the WebLogic Integration Administration Help.*

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 Trading Partner Management in *Using the WebLogic Integration Administration Help*
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 Trading Partner Management in *Using the WebLogic Integration Administration Help*.

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. For more information about managing certificates in the WebLogic Integration Administration Console, see Adding Certificates to a Trading Partner and Deleting Certificates, Bindings, or Custom Extensions in Trading Partner Management in *Using the WebLogic Integration Administration Help*.

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 "Adding Services" and "Deleting Services" in Trading Partner Management in *Using the WebLogic Integration Administration Help*.

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 Trading Partner Management in *Using the WebLogic Integration Administration Help*.

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 Trading Partner Management in *Using the WebLogic Integration Administration Help*.

Note: When you are using the ebXML protocol for Trading Partner messaging, the values used for **Retry Number**, **Retry Interval**, and **Persist Duration** are always the values of the *remote* trading partner, not the *local* Trading Partner.

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 using an intermediary XML data file that conforms to the `tpm.xsd` schema that ships with WebLogic Integration.

To perform export, import, or bulk delete operations in WebLogic Integration, 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 Importing and Exporting Management Data in Trading Partner Management in *Using the WebLogic Integration Administration Help*:

When you export TPM data using the console or the bulk loader utility, a file suitable for import is created. For more information about the file 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 creating a domain using Configuration Wizard, see Creating WebLogic Domains 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`	`000000001`	Default local trading partner. In the tutorials, this trading partner is usually the *initiator* of conversations.
`Test_TradingPartner_2`	`000000002`	In the tutorials, this trading partner is usually the *participant* in conversations.

For more information about the Trading Partner Integration tutorials, see Tutorials for Trading Partner Integration.

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, 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 section 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 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 Integration 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 Integration, see Tutorial: Building Your First Business Process.

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

- ebXML Business Processes
- RosettaNet Business Processes

Conversations and Roles

This section 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.

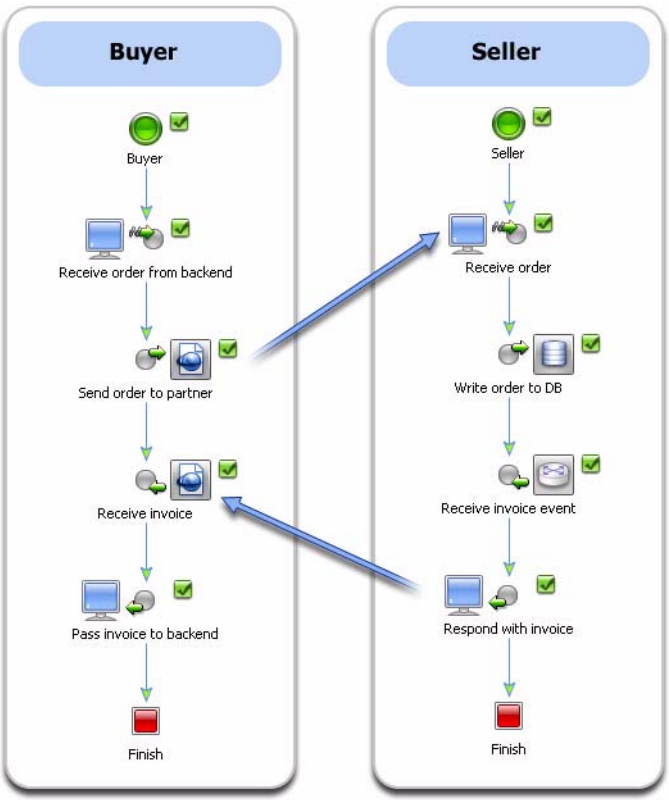
Collaborative Business Processes

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

Figure 1-2 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 Integration controls and templates that you can use for initiator and participant business processes.

Initiator Business Processes and Workshop Controls

The Table 1-3 describes the Workshop controls that you use in initiator business processes to handle communications with participants:

Table 1-3 Workshop Controls Used in Initiator Business Processes

Control Name	Description
ebXML Control	The ebXML control enables WebLogic Integration 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 Java Docs for WebLogic Integration Classes and "ebXML Initiator Business Processes" on page 2-10.
RosettaNet Control	Enables WebLogic Integration 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 Java Docs for WebLogic Integration Classes and "RosettaNet Initiator Business Processes" on page 3-16.

Participant Business Processes and Workshop Templates

The Table 1-4 describes the Workshop for WebLogic templates that you use in participant business processes to handle communications with conversation initiators:

Table 1-4 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 `@com.bea.wli.jpd.EbXML Annotation` in the Java Docs for WebLogic Integration Classes.
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 `@com.bea.wli.jpd.RosettaNet` in the Java Docs for WebLogic Integration Classes.

TPM Control For Business Processes and Web Services

The TPM (Trading Partner Management) control provides 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 Workshop 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 *Using Integration Controls* Guide.

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 the following problem scenarios:
  - 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 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 section describes 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 Protocol
- Business Message
- 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 Protocol

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 Message

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. 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 untyped XML format. The XML data is not specified at design time.
`XmlObject[]`	An array of one or more `XmlObject` elements. Only available for ebXML messages.
`RawData`	Represents any non-XML structured or unstructured data. Examples include PDF, DOC, GIF, JPG, and other binary files.
`RawData[]`	An array of one or more `RawData` elements. Only available for ebXML messages.
`MessageAttachment[]`	Array containing one or more parts of an ebXML or RosettaNet business message. Message parts can be untyped XML data (`XmlObject` data type) or non-XML data (`RawData` data type). Used when sending different kinds of payloads (XML and non-XML) in the same message. The actual number of message parts might not be known until processed. To learn about working with `MessageAttachment` objects, see Using Message Attachments.

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

For more information about these data types, see Working with Data Types in the *Guide to Building Business Processes.*

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

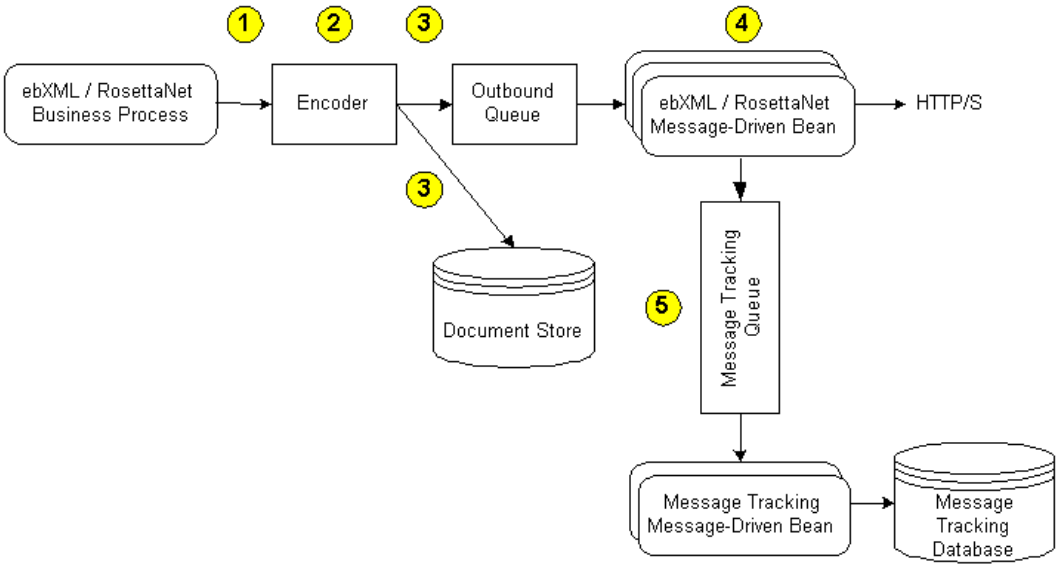
Run-Time Processing of Business Messages

This section describes how ebXML and RosettaNet business messages are processed at runtime. Inbound and outbound business messages traverse different paths. 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

Figure 1-3 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.

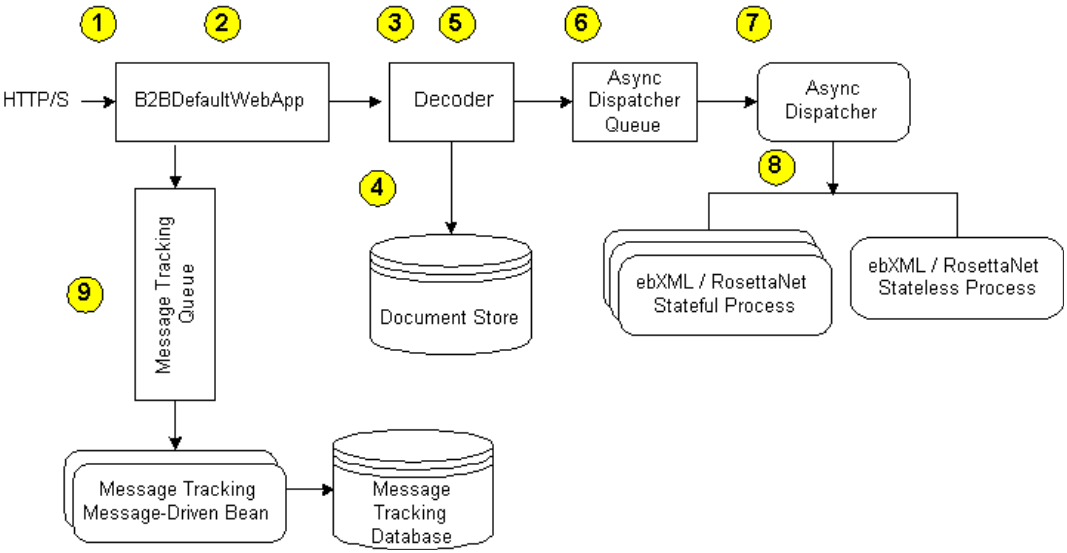
3. The message is persisted in the WebLogic Integration document store and forwarded it to the applicable JMS queue:
   - ebXML: `wli.internal.b2b.ebxmlencoder.queue`
   - RosettaNet: `wli.internal.b2b.rosettanetencoder.queue`
4. The message-driven bean associated with the queue sends the message out 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 *Using the WebLogic Integration Administration Help*.

Path for Incoming Trading Partner Messages

Figure 1-4 shows the path of an incoming business message:

Figure 1-4 Incoming Message Path



This 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.

   Note: For ebXML, the decoder does a similar thing, but it depends on which reliable message option you have selected.
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 Trading Partner Management in *Using the WebLogic Integration Administration Help*.

Run-Time Monitoring Concepts

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

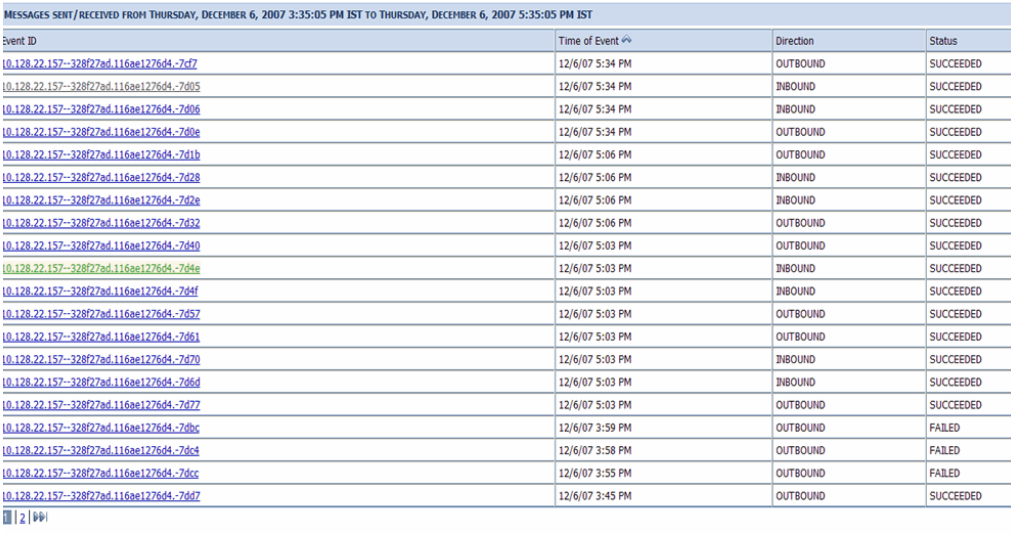
- Message Tracking
- Viewing Run-Time Statistics

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.

Figure 1-5 shows a summary of business messages:

Figure 1-5 Summary of sent and recieved messages



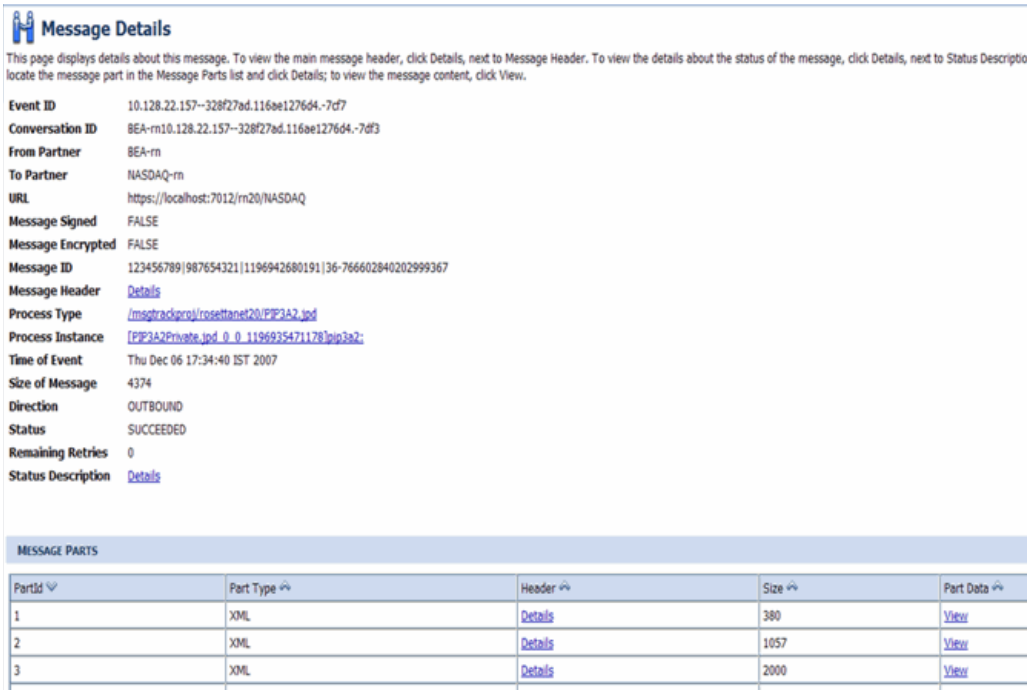
MESSAGES SENT/RECEIVED FROM THURSDAY, DECEMBER 6, 2007 3:35:05 PM IST TO THURSDAY, DECEMBER 6, 2007 5:35:05 PM IST

Event ID	Time of Event	Direction	Status
10.128.22.157--328f27ad.116ae1276d4.-7cf7	12/6/07 5:34 PM	OUTBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d05	12/6/07 5:34 PM	INBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d06	12/6/07 5:34 PM	INBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d0e	12/6/07 5:34 PM	OUTBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d1b	12/6/07 5:06 PM	OUTBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d28	12/6/07 5:06 PM	INBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d2e	12/6/07 5:06 PM	INBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d32	12/6/07 5:06 PM	OUTBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d40	12/6/07 5:03 PM	OUTBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d4e	12/6/07 5:03 PM	INBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d4f	12/6/07 5:03 PM	INBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d57	12/6/07 5:03 PM	OUTBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d61	12/6/07 5:03 PM	OUTBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d70	12/6/07 5:03 PM	INBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d6d	12/6/07 5:03 PM	INBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7d77	12/6/07 5:03 PM	OUTBOUND	SUCCEEDED
10.128.22.157--328f27ad.116ae1276d4.-7dbc	12/6/07 3:59 PM	OUTBOUND	FAILED
10.128.22.157--328f27ad.116ae1276d4.-7dc4	12/6/07 3:58 PM	OUTBOUND	FAILED
10.128.22.157--328f27ad.116ae1276d4.-7dcc	12/6/07 3:55 PM	OUTBOUND	FAILED
10.128.22.157--328f27ad.116ae1276d4.-7dd7	12/6/07 3:45 PM	OUTBOUND	SUCCEEDED

1 | 2 | ▶▶|

Figure 1-6 shows the details of a single business message:

Figure 1-6 Message Details

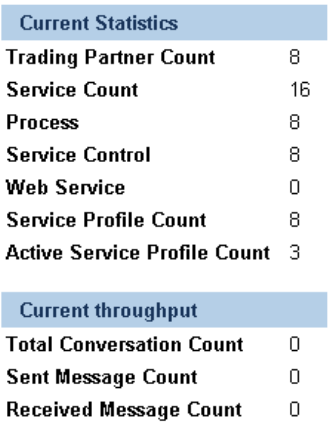


WebLogic Integration maintains message history information in the run-time repository. You configure the message tracking level for individual services profiles, as described in "Adding Service Profiles to a Service" in Trading Partner Management in *Using the WebLogic Integration Administration Help*. WLI 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 *Using the WebLogic Integration Administration Help*.

Viewing Run-Time Statistics

The Statistics module in the WebLogic Integration Administration Console displays run-time statistics for the system and for service profiles. Figure 1-7 shows system summary statistics:

Figure 1-7 System summary

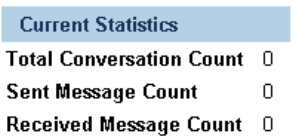


Current Statistics	
Trading Partner Count	8
Service Count	16
Process	8
Service Control	8
Web Service	0
Service Profile Count	8
Active Service Profile Count	3

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

Figure 1-8 shows service profile statistics:

Figure 1-8 Service Profile



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 *Using the WebLogic Integration Administration Help*.

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:

  - Planning the RosettaNet Solution
  - Planning the ebXML Solution

Phase 2: Design, Build, and Test the Solution

In this phase, you design, build, and test the solution using the following tools:

- Workshop for WebLogic, 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 using Workshop for WebLogic, it is recommended that you complete the following tutorials so that you know how to build business processes and create data transformations in Workshop for WebLogic:
  - Tutorial: Building Your First Business Process.
  - Tutorial: Building Your First Data Transformation.
- 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 1-9) and default security settings (described in Default Domain Security Configuration). For more information about the Configuration Wizard, see Creating WebLogic Domains 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 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*.

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

- "Building the RosettaNet Solution" on page 3-19
- "Building the ebXML Solution" on page 2-13

Phase 3: Deploy the Solution

After 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 WebLogic Integration Administration Console, see Using the WebLogic Integration Administration Help
- 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 Managing WebLogic Integration Solutions.
- Bulk Loader, which provides import / export capabilities for trading partner information stored in the TPM repository. For instructions on using the bulk loader, see Using the Trading Partner Bulk Loader in *Managing WebLogic Integration Solutions.*
- For comprehensive information about deploying WebLogic Integration, see *Deploying WebLogic Integration Solutions*
- 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
- For more information about the tasks associated with each business protocol, see the following topics:
  - Deploying the RosettaNet Solution
  - Deploying the ebXML Solution

Phase 4: Administer and Tune the Solution

After 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:

- Using the WebLogic Integration Administration Help
- Managing the RosettaNet Solution
- Managing the ebXML Solution

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 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.

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 WLI. It contains the following sections:

- About ebXML
- ebXML Support in WebLogic Integration

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, see ebXML specifications 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
- SOAP Messages with Attachments

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.
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.

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

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.

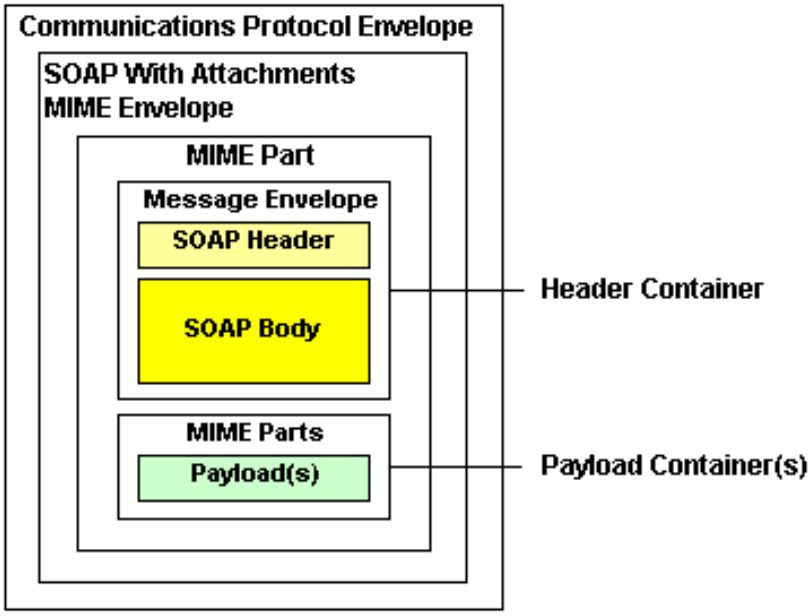
ebXML Business Messages

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 package*. 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 Workshop business processes for retrieving the ebXML message envelope that relates to the Header container from incoming business messages. For more information, see @com.bea.wli.jpd.EbXML and @EBXMLControl.EbXML in the Java Docs for WebLogic Integration Classes.

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 untyped XML data (`XmlObject` data type) or non-XML data (`RawData` data type). To learn about working with `MessageAttachment` objects, see Using Message Attachments.

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. Configure reliable messaging in the WebLogic Integration Administration Console, as described in "Defining an ebXML 1.0 or 2.0 Binding" in Trading Partner Management in *Using WebLogic Integration Administration Console Help*.

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 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 Javadoc for WebLogic Integration Classes.
- 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 file. Each ebXML control in the initiator business process and each business process JPD 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 (@EBXMLControl.EbXML on the ebXML control in initiator business processes and the @com.bea.wli.jpd.EbXML in participant business processes).
  - Dynamically—using pass-in values, such as business IDs, and calling the `setProperties` method (only applicable to the ebXML control using @EBXMLControl.EbXML).

  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 node, as the order in which acks and messages arrive is not guaranteed. For more information about the Event Choice node, see " Receiving Multiple Events" in *Guide to Building Business Processes*
  - 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 @com.bea.wli.jpd.EbXML (for participant business processes) and @EBXMLControl.EbXML (in the callback of the ebXML control for initiator business processes). For example:
  - For a participant business process:

```
@com.bea.wli.jpd.EbXML.ebxml-method message-envelope="{env}"
public void request(XmlObject payload, XmlObject env) {
}
```

  - For an initiator business process:

```
@EBXMLControl.EbxmlMethod envelope="{env}"
```

```
public void onMessage(MessageAttachment[] reply, XmlObject env);
```

- 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 an initiator business process:

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

  - For a 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 1-9.

ebXML Initiator Business Processes

In WebLogic Integration, initiator business process use an ebXML control to enable 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:

- ebXML Control
- ebXML Control Interface
- @EBXMLControl.EbxmlMethod

For an introduction to initiator business processes, see "Initiator and Participant Business Processes" on page 1-13.

ebXML Participant Business Processes

In WebLogic Integration, you can easily create a new ebXML participant business process using a 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 @com.bea.wli.jpd.EbXML in the Javadoc for WebLogic Integration Classes. For an introduction to participant business processes, see "Initiator and Participant Business Processes" on page 1-13.

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.

- "Tutorial: Building ebXML Solutions" in *Tutorials for Trading Partner Integration* to learn basic concepts that are unique to ebXML solutions, 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 1-27), 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?
  - What qualities of service are required for reliable messaging, as described in Reliable Messaging?

The *Tutorials for Trading Partner Integration* provide examples of different ebXML solutions. For more information, see Tutorial: Building ebXML Solutions.

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. For conceptual information about ebXML business processes, see ebXML Business Processes.

The "Tutorial: Building ebXML Solutions" in *Tutorials for Trading Partner Integration* (available at http://dev2dev.bea.com/code/wli.jsp) provides a detailed examples of the typical tasks 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 Domains 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 `serviceName` attribute (and others if needed) in the @EBXMLControl.EbXML
   - For participant business processes, you change the `ebxmlServiceName` attribute (and others if needed) in the @com.bea.wli.jpd.EbXML.
4. Rename the JPD and 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 Javadoc for WebLogic Integration Classes.
   - For more information about building ebXML participant business processes, see Building ebXML Participant Business Processes in the *Guide to Building Business Processes*.
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) and security settings (described in Default Domain Security Configuration).

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 1-28. For detailed information about deploying WebLogic Integration solutions, see *Deploying WebLogic Integration Solutions*.

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 Domains 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 more information, see Importing and Exporting Data in Trading Partner Management in *Using WebLogic Integration Administration Console Help*.

   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.

   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 more information, see "Defining Protocol Bindings" in Trading Partner Management.

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

   d. Define the services that you will use in your deployment. For more information, 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 more information, see Adding Services 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 4-50.

5. Configure your domain (clustering, high availability, load balancing, fail-over, and so on) as needed for your production environment. For more information, see Deploying WebLogic Integration Solutions.

6. Deploy the application and other WebLogic Integration resources associated with your ebXML solution. For more information, see Deploying WebLogic Integration Solutions

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.

For instructions on monitoring trading partner integration resources, see Viewing Statistics and Monitoring Messages in Trading Partner Management in *Using WebLogic Integration Administration Console Help*.

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, and Documentation and Example Files.

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 Support in WebLogic Integration

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) 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.

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:

- 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
• Digital signatures to protect messages from tampering	X	X
• Standard S/MIME and encryption for message privacy		X
Message Handling		
• Duplicate message removal	X	X
• Message persistence	X	X
• High-level ACK	X	X
• High-level retry	X	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 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
- Partner Interface Processes (PIPs)

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.

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`.

Asynchronous Single-Action Activity

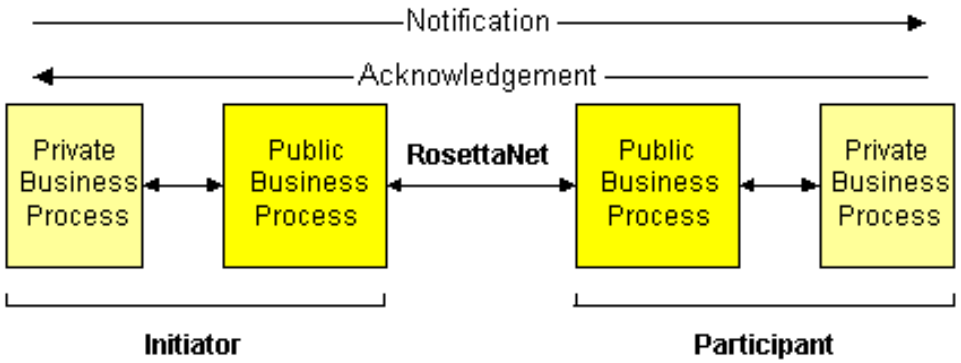
The asynchronous single-action (or one-action) activity design pattern involves a single action (business message) from the sender and a signal (a protocol response, such as an acknowledgement, ack-reject, or error) from the recipient back to the sender:

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 is PIP3B2 (Notify of Advance Shipment), which is described at `http://www.rosettanet.org`.

Figure 3-1 is 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 the 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

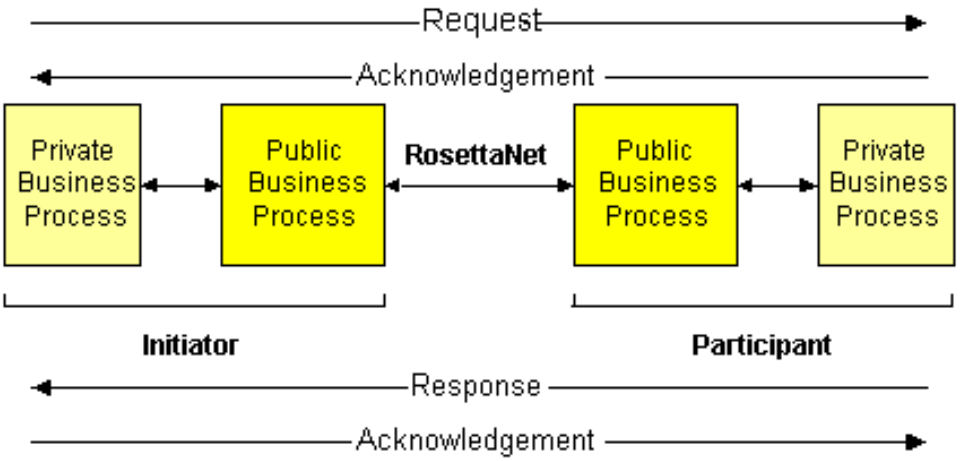
The asynchronous two-action activity design pattern involves two actions—a single action (business message) from the sender to the recipient, and a single action (business message) from the recipient back to the sender, and their corresponding signals (protocol responses, such as acknowledgements, ack-rejects, 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 PIP3A4 (Request Purchase Order), which is described at `http://www.rosettanet.org`.

Figure 3-2 is 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 the 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

Figure 3-3 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

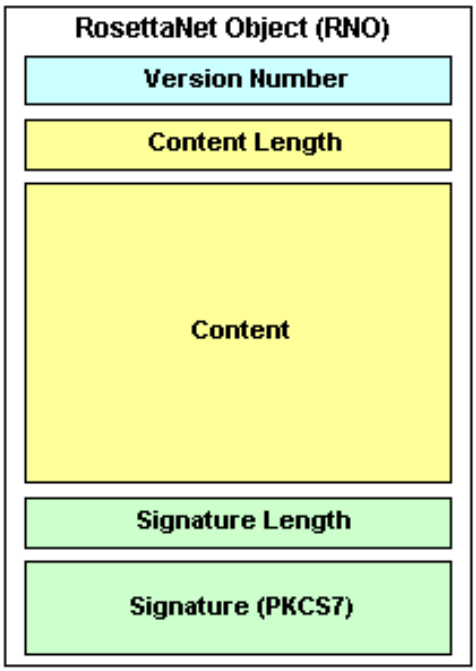


Table 3-3 describes the components of an RNO:

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.
Headers	Includes the following headers: • Preamble Header • Service Header

Table 3-3 Components of an RNO (Continued)

Component	Description
Content (Payload)	Includes the following components: • Service Content—contains either an action or a signal message. • Attachments—Optional. Can contain zero or 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 4-45.

Table 3-4 shows the components of an RBM:

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

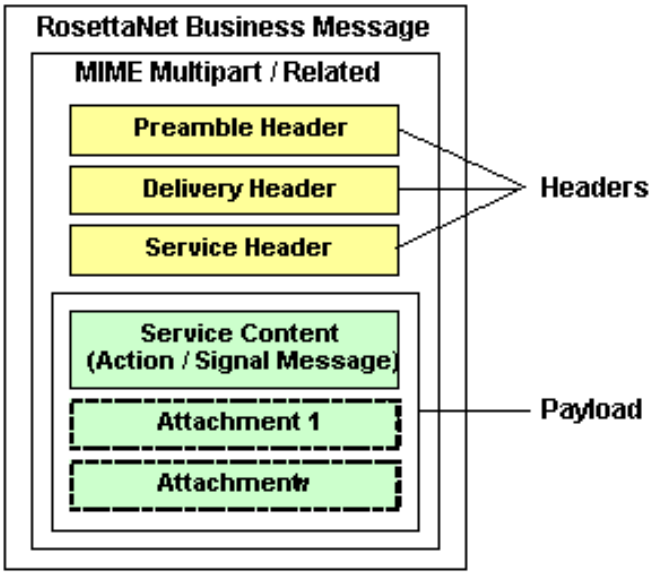


Table 3-4 describes the components of a RBM:

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 zero or 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 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 depends on the following factors:

- The `validateServiceHeader` variable settings. If this is 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 validation settings of the business process.
- The validation options specified in the business process.
- Schema validation according to the XSD schemas you have included in your project.

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`

- 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 Workshop business processes that implement RosettaNet PIPs. It contains the following sections:

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`.

- 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 Java Doc for WebLogic Integration Classes.

- Determine how you want to specify the business process properties:

  – Statically—using hard-coded annotations (@RosettaNetControl.RosettaNet on the RosettaNet control in initiator business processes and the @com.bea.wli.jpd.RosettaNet 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 1-9.

RosettaNet Initiator Business Processes

In WebLogic Integration, initiator business processes use a RosettaNet control to enable 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 Using Integration Controls Guide:

- Overview: RosettaNet Control
- Creating a New RosettaNet Control
- Using a RosettaNet Control
- RosettaNet Control Interface
- @RosettaNetControl.RosettaNet

For an introduction to initiator business processes, see Initiator and Participant Business Processes.

RosettaNet Participant Business Processes

In WebLogic Integration, you can easily create a new RosettaNet participant business process using a 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 @com.bea.wli.jpd.RosettaNet in the WebLogic Workshop 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 1-13.

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), 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.

- 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?

The *Tutorials for Trading Partner Integration* provide examples of different RosettaNet solutions. To learn more, see Tutorial: Building RosettaNet Solutions.

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. For conceptual information about RosettaNet business processes, see RosettaNet Business Processes.

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.
5. In Workshop for WebLogic, 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 `pipVersion` attributes (and others if needed) in the @RosettaNetControl.RosettaNetAnnotation.
   - For *public* participant business processes, you change the `pipName` and `pipVersion` attributes (and others if needed) in the @com.bea.wli.jpd.RosettaNet
7. Rename the JPD 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 Java Doc for WebLogic Integration Classes.
   - For more information about building RosettaNet participant business processes, see Building RosettaNet Participant Business Processes in the Java Doc for WebLogic Integration Classes.
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) and security settings (described in Default Domain Security Configuration).

Note: When you first set up connections to RosettaNet trading partners, it is a good idea to run your configuration in Test mode to take advantage of the additional debugging features provided by this mode. To run your Web Logic Integration RosettaNet configurations in Test mode, you specify two annotations in the `setProperties` method:

-Set `global-usage-code` to `Test`.

-Set `debug-mode` to `true`.

For more information about the `setProperties` method, see RosettaNet Control Interface.

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. For detailed information about deploying WebLogic Integration solutions, see Deploying WebLogic Integration Solutions.

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 Domains 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 more information, see Importing and Exporting Data in Trading Partner Management in *Using WebLogic Integration Administration Console Help*.

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

   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 Adding Protocol Bindings to a Trading partner 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 Adding Services 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.
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.
6. Deploy the application and other WebLogic Integration resources associated with your RosettaNet solution. For instructions, see Deploying WebLogic Integration Solutions

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.

For instructions on monitoring trading partner integration resources, see the following topics in in Trading Partner Management in *Using WebLogic Integration Administration Console Help*:

- 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*.
- Introduction to WebLogic Security.
- Security summary page, which provides a compendium of WebLogic security topics.
- 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 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.*

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.

Figure 4-1shows 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

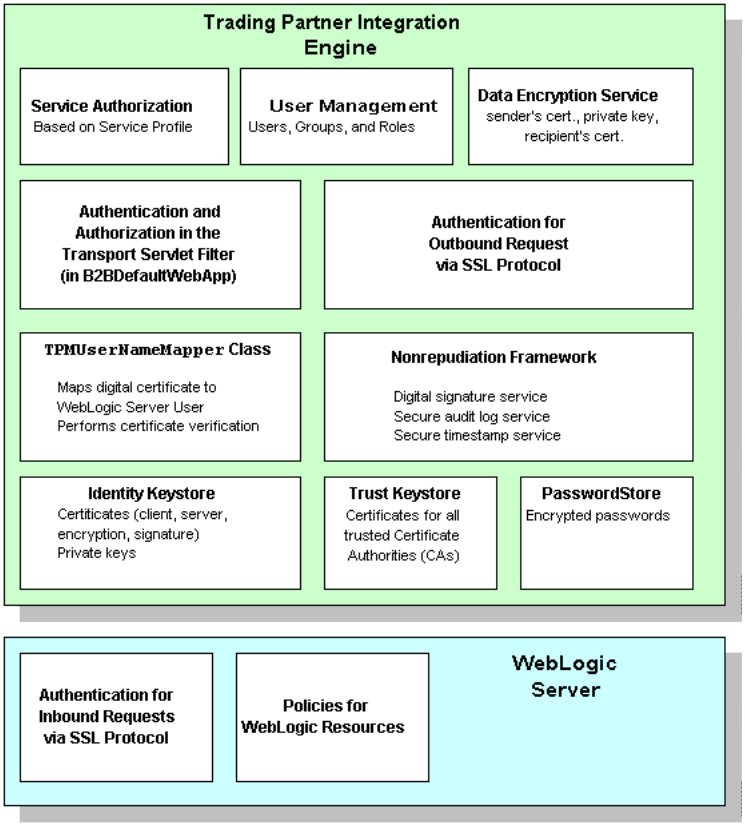


Table 4-1 describes the features in this security model.

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

Type of Certificate	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.
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.
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 *Using Worklist Console*.

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

Type of Certificate	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: • Workshop for WebLogic business processes • JDBC dataSources that are used to access the TPM repository • JMS destinations For more information about these resources, see Trading Partner Integration Resources Requiring Security Policies. 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 • "URL (Web) and EJB (Enterprise JavaBean) Resources" and "Application Resources" in "Types of WebLogic Resources" in *Securing WebLogic Resources.*
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.
`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 and reference information for the UserNameMapper interface.

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

Type of Certificate	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 approve or disapprove 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.
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. For more information about the identity keystore, see Keystore for Private Keys and Certificates.
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.
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.

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

Type of Certificate	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.
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.

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 "WebLogic Integration Users, Groups, and Roles" in User Management in *Using Worklist Console*.
- B2BDefaultWebApp, on which you can configure policies for access control in trading partner authorization, which is described in Authenticating Trading Partner Messages. For configuration instructions, see "URL (Web) and EJB (Enterprise JavaBean) Resources" and "Application Resources" in "Types of WebLogic Resources" in *Securing WebLogic Resources*.
- PasswordStore, which is described in WebLogic Integration PasswordStore for Encrypted Passwords.
- Default identity and trust keystores, which are described in Keystore for Private Keys and Certificates.
- Default trading partners and services profiles, described in Default TPM Repository Settings, 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.

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 *Using WebLogic Integration Administration Console Help* :

- "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.
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.*
2. Configure the keystores using the WebLogic Server Administration Console according to the instructions in "Configuring KeyStores" in "Configuring SSL" in *Managing WebLogic Security.*
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 *Using WebLogic Integration Administration Console Help* .

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
- Workshop for WebLogic business processes
- JDBC dataSources 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.

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.
- **Digital certificates**—Trading partners use digital certificates to prove their identity to WebLogic Integration. For more information, see Digital Certificates.

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 *Using WebLogic Integration Administration Console Help* .

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 from a 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.*

- Example: ebXML Security Configuration

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.

- Authentication of the incoming trading partner message, which is described in Authenticating Trading Partner Messages.

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, Trading Partner Integration engine 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.

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.

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)

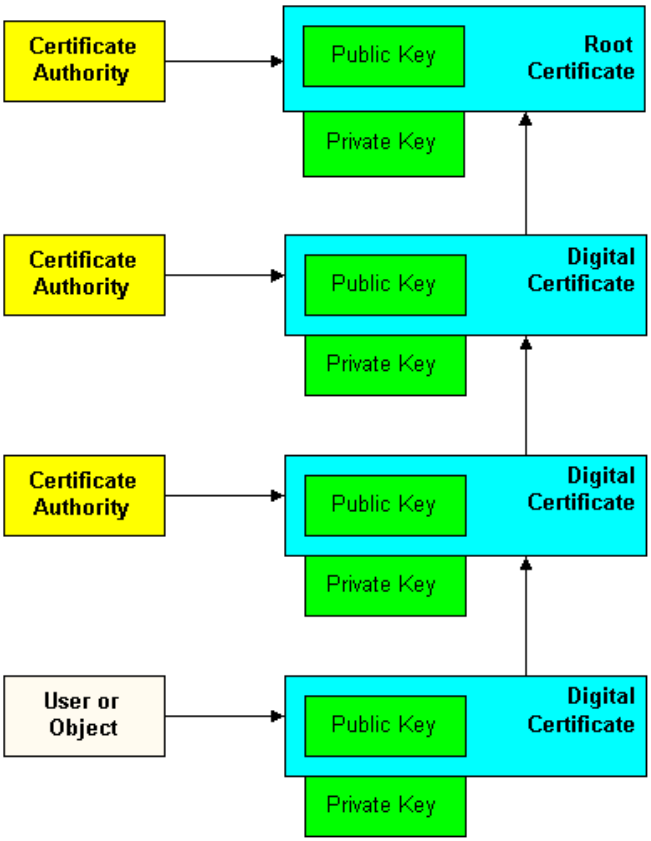
The most widely accepted format for digital certificates is defined by the ITU-T X.509 international standard. Thus, digital certificates can be read or written by any application complying with the X.509 standard. The public key infrastructure (PKI) in WebLogic Server recognizes digital certificates 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. Figure 4-2 describes how the hierarchy of certificate authorities is terminated by a self-signed digital certificate known as the root certificate.

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. **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. **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 module 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. 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 Importing and Exporting Data in Trading Partner Management in *Using WebLogic Integration Administration Console Help*). 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.

For configuration instructions, see Adding Certificates to a Trading Partner in Trading Partner Management in *Using WebLogic Integration Administration Console Help* .

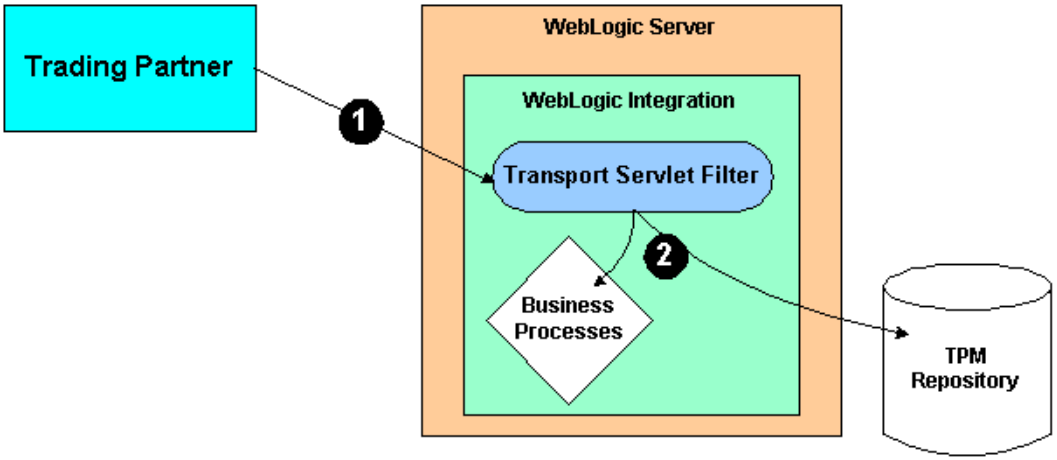
Note: WebLogic Integration does not validate any of the trading partner certificates against a trusted Certificate Authority as you load them into the keystore.

Authenticating Trading Partner Messages

As described in Authentication Levels, after a trading partner's certificate has been validated by WebLogic Server, Trading Partner Integration engine 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.

Figure 4-3 shows the process of authenticating a trading partner message.

Figure 4-3 Authenticating a Trading Partner Message



In Figure 4-3 note the following:

- The Transport Servlet Filter is the entry point into Trading Partner Integration engine. 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.

Authenticating Remote Users in Two-Way Authentication

This section describes 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.

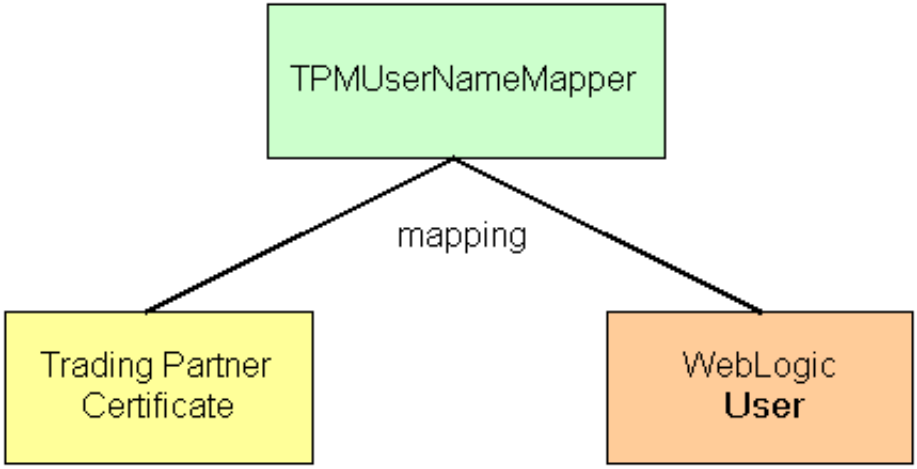
About the TPMUserNameMapper Class

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 Figure 4-4.

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.

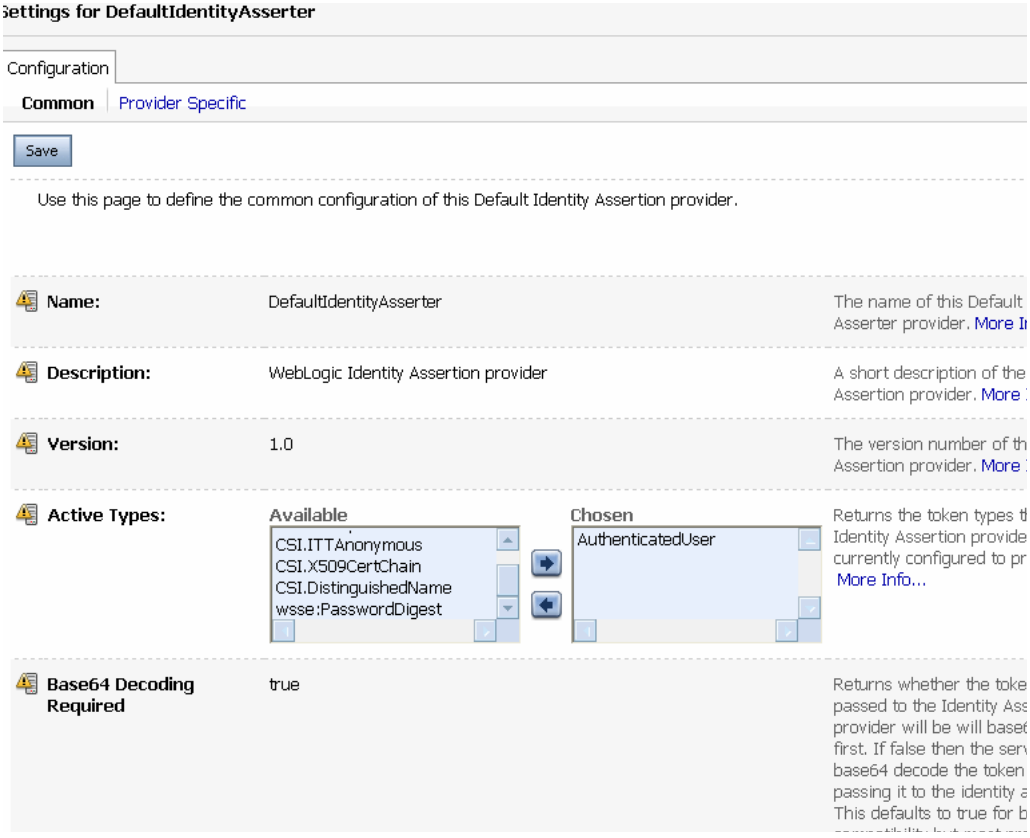
To configure the `TPMUserNameMapper`:

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

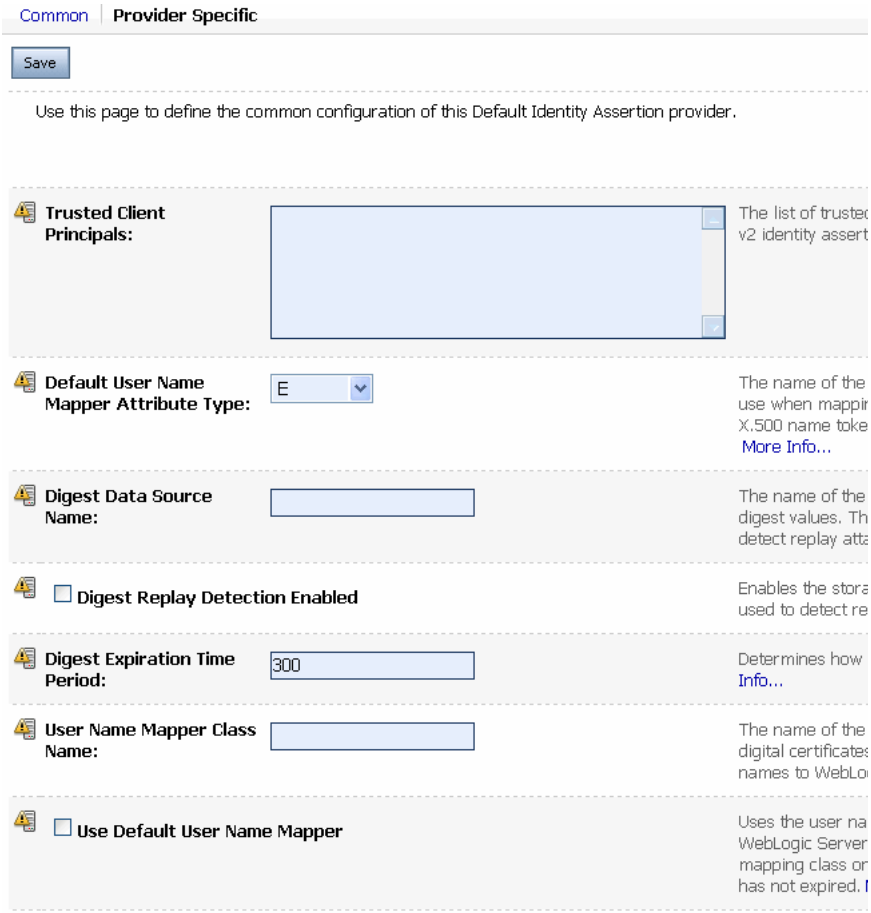
   The WebLogic Server Administration Console displays the configuration tabs for the `DefaultIdentityAsserter`.

Figure 4-5 Configuration of the Default Identity Asserter



4. In the Name Mapper Class Name field, type `com.bea.b2b.security.TPMUserNameMapper`.
5. From the list of available Active types, select `X.509` and click the right arrow.
6. You can also 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 **Provider Specific** tab.

Figure 4-6 Mapping Client Certificate Attribute



b. 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.Ensure that the **Use Default User Name Mapper** checkbox not checked.

c. 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 **Save**.

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".

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 *Using WebLogic Integration Administration Console Help* .

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

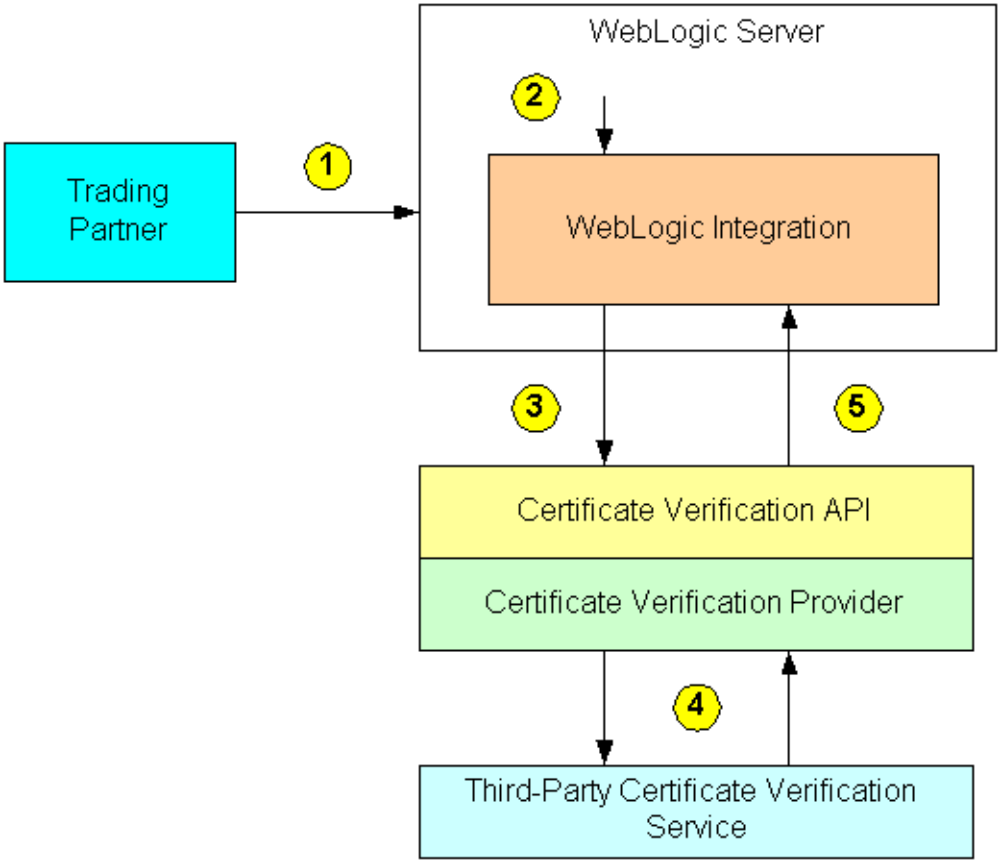
The CVP is used by WebLogic Integration in the following cases:

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

Certificate Verification Process

Figure 4-7 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 4-7 Trading Partner Certificate Verification in WebLogic Integration



Note the following callouts in Figure 4-7.

Table 4-5 Certificate verification callouts

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 which verifies the following: • The validity of the certificate (lifetime). • That the keyusage is a valid and specified in 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.
- 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 Trading Partner Integration engine 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.

- `public String verify(X509Certificate[] 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 Trading Partner Integration engine 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 Trading Partner Integration engine 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 *Using WebLogic Integration Administration Console Help* .

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

Note: When you are running WebLogic Integration in iterative development mode, no security verification settings are active. The CVP is only active in development mode.

Authorization

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
- Workshop for WebLogic business processes
- JDBC dataSources that are used to access the TPM repository
- JMS destination (for message tracking, asynchronous dispatcher queues for trading partner integration business processes)

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 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*.

In addition to configuring B2BDefaultWebApp, you can also configure other trading partner integration resources (such as the JDBC dataSources used to access the TPM repository, Workshop for WebLogic business processes, and JMS destinations) that need to be configured as well. For more information, see Authorization. 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.

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 Trading Partner Integration engine 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. Trading Partner Integration engine 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 *Using WebLogic Integration Administration Console Help* .

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. Digital signatures are required for nonrepudiation, which is described at NonRepudiation.

WebLogic Integration Support for Digital Signatures

WebLogic Integration supports the following types of digital signatures:

- XML 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 *Using WebLogic Integration Administration Console Help* .

After validating a signature, WebLogic Integration invokes the certificate verification provider (CVP), which is described in Certificate Verification Process.

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 and MD5
- 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 *Using WebLogic Integration Administration Console Help* .

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.

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 *Using WebLogic Integration Administration Console Help* .

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.

Table 4-6 Message type data contents

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 4-1 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

Trading Partner Integration engine 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 4-2 Example of Writing to the Audit Log Directly

```
package orderprocessing;
import com.bea.jpd.JpdContext;
import org.apache.xmlbeans.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>::
*
*/
@com.bea.wli.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
*/
@Control()
    @EBXMLControl.EbXML(serviceName = "SecureOrderService",
                         from = "BEA-IT-id",
                         to = "SUN-id",
                        ebXMLActionMode = EBXMLControl.EbXML.ActionMode.DEFAULT)
private SecureOrderServiceControl orderService;
/**
*@common:context
*/
@com.bea.wli.jpd.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 *Using WebLogic Integration Administration Console Help* .

Exclusive and Default Timestamps

Trading Partner Integration engine 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

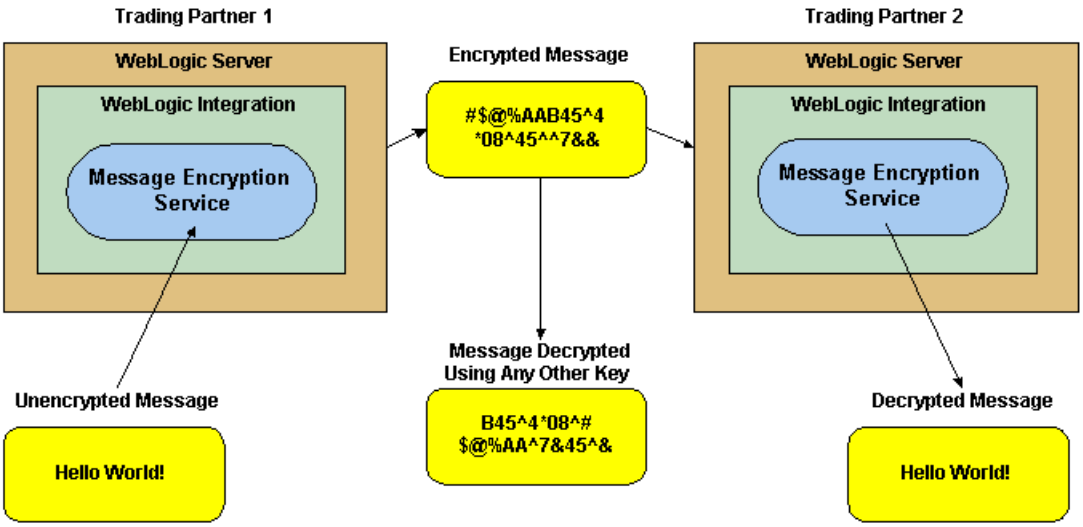
WebLogic Integration encrypts business messages for the business protocols that require it. In this WebLogic Integration release, message encryption is supported only for RosettaNet 2.0.

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

How WebLogic Integration Handles Data Encryption

Figure 4-8 shows how data encryption is performed using the public and private keys.

Figure 4-8 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, Trading Partner Integration engine disables the encryption service. However, Trading Partner Integration engine 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 *Using WebLogic Integration Administration Console Help* .

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

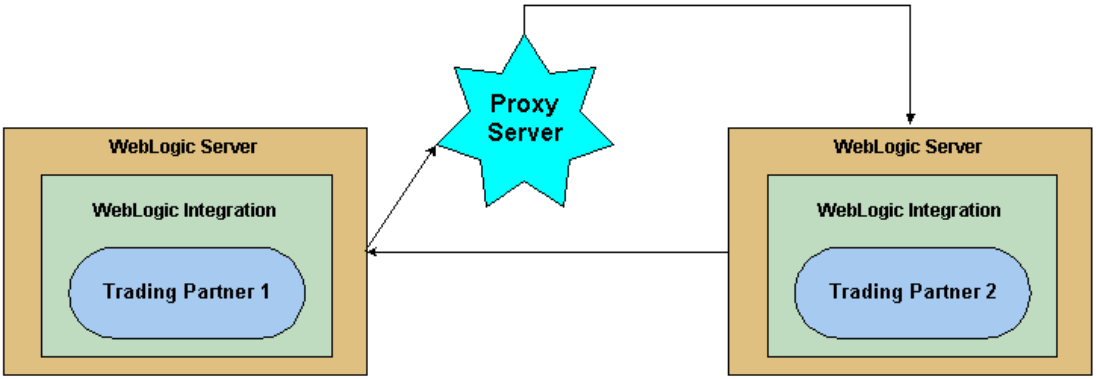
Configuring Trading Partner Integration to Use an Outbound HTTP Proxy Server

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 is tunneled through, or delegated to, the proxy server and then to the external network. Figure 4-9 illustrates how a proxy server might be used in the WebLogic Integration environment.

Figure 4-9 Proxy Server



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

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";
```

In addition, you need to specify the following WebLogic Web Server system properties:

`-http.proxyHost`
`-http.proxyPort`
`-https.proxyHost` (if proxy server is configured for SSL tunneling)
`-https.proxyPort` (if proxy server is configured for SSL tunneling)

The `weblogic.common.ProxyAuthenticator` interface is used to obtain authentication information to communicate with the proxy. For more information, see `Interface ProxyAuthenticator`

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

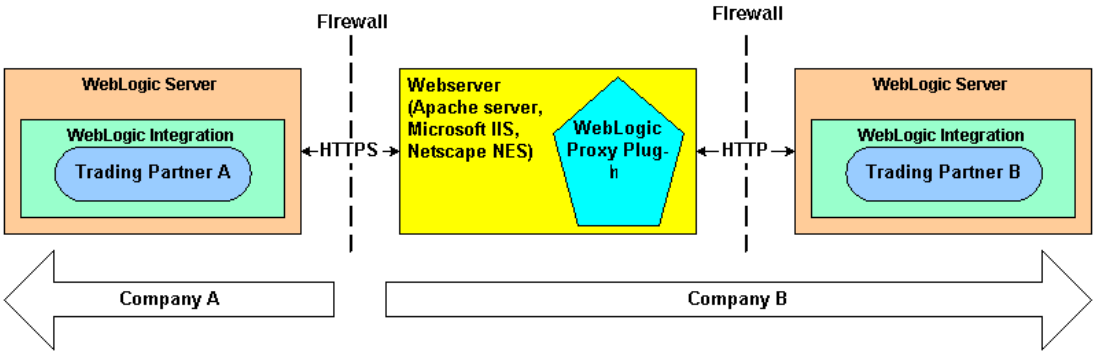
The Web server uses the WebLogic proxy plug-in, which you can configure to provide the following services:

- 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 topology of an environment that uses a Web server, the WebLogic proxy plug-in, and WebLogic Integration is illustrated in Figure 4-10.

Figure 4-10 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 Web Server Functionality for WebLogic Server.

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.extension

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

Here, *`extension`* is the file extension used by your Unix installation.

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.

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 Use

in *Using Worklist Console*.

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. For instructions on configuring trading partner profiles in the WebLogic Integration Administration Console, see the following topics in Trading Partner Management in *Using WebLogic Integration Administration Console Help* :

- Adding 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. 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.*
- Configure the keystores using the WebLogic Server Administration Console according to the instructions in “Configuring KeyStores” in “Configuring SSL” in *Managing WebLogic Security.*

Configure Certificates

You need to add digital certificates to trading partners. For an introduction to certificates, see Digital Certificates. For instructions on configuring certificates in the WebLogic Integration Administration Console, see Adding Certificates to a Trading Partner in Trading Partner Management in *Using WebLogic Integration Administration Console Help* :

- "Adding Certificates to a Trading Partner

Configure SSL

You need to configure SSL for transport-level security using the WebLogic Server Administration Console. For an introduction, see SSL Protocol. For configuration instructions, see "Configuring SSL" in *Managing WebLogic Security*.

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 *Using WebLogic Integration Administration Console Help* .

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

To Learn More

Security Topics in the WebLogic Platform Documentation Set

The WebLogic Platform documentation set contains several additional security topics.

BEA Security Advisories

To keep you informed of the latest security advisories and to make sure you have access to security-related patches as soon as they become available, BEA maintains the BEA Advisories & Notifications page at:

http://dev2dev.bea.com/advisories

Reporting Security Issues

If you discover a possible security issue with WebLogic Platform 10.0 or any of its components, BEA recommends that you report it to secalert@bea.com.

dev2dev Security Resources

The BEA dev2dev Web site includes a Web page that provides links to several security-specific resources, including:

- Security code samples
- Training
- Newsgroups

The dev2dev site also hosts a Web page that provides answers to frequently asked questions (FAQ) about BEA WebLogic Platform and other BEA products. The site is updated monthly. To visit the dev2dev FAQ page, see:

http://dev2dev.bea.com/topitems/topsolutions/index.jsp

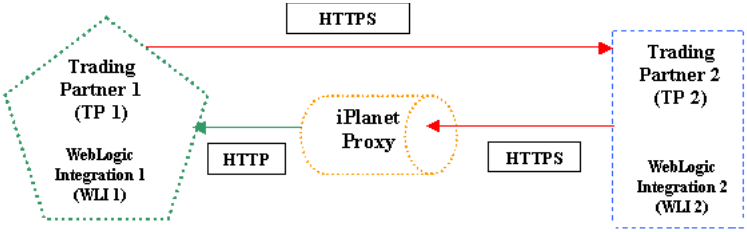
APPENDIX A

Example: ebXML Security Configuration

This example demonstrates how to configure the security settings for ebXML message exchange between trading partners over HTTPS protocol through a proxy server. Although any proxy server can be used in this configuration, the example demonstrates how to configure the iPlanet Web Server 6.0 (Sun ONE 6.0) as the proxy server. A demonstration version of this server is available for download at http://wwws.sun.com/software/download/products/3f186391.html.

In the Figure A-1, the example involves two trading partners. Both the trading partners, Trading Partner 1 and Trading Partner 2 are configured in WebLogic Integration. The WLI instance that hosts Trading Partner 1 is termed as WLI 1 and the WLI instance that hosts Trading Partner 2 is termed WLI 2.

Figure A-1 Trading Partner Configuration



In the preceding figure:

- Messages are sent from Trading Partner 1 to Trading Partner 2 using HTTPS protocol through a secure connection.

- Acknowledgements and responses are sent from the participant to the initiator through an iPlanet proxy server.

Note: In production scenarios, firewalls are usually configured between WebLogic Integration 1 and the proxy server and between the proxy server and WebLogic Integration 2. To keep the IP addresses simple in this sample, the firewalls are left out of the examples.

The following topics are discussed in this section:

trading partner bindings and add the appropriate certificates to the trading partner and keystore.

"Step 7: Configuring the Remote Trading Partner in WebLogic Integration" on page A-16
In this section, you import the file which you exported from WebLogic Integration 2 in the preceding section and configure the information imported to be used as the remote trading partner profile.

"Step 8: Creating Services and Service Profiles in WebLogic Integration" on page A-17
In this step, you configure the Services and the Service profiles for the local and the remote trading partner profiles in WebLogic Integration.

"Step 9: Configuring the iPlanet Server" on page A-20
In this procedure, you complete the iPlanet proxy server configuration install the appropriate server and trusted certificates needed for the message exchange between your two trading partners.

Related Topics

Managing WebLogic Security

Trading Partner Integration Security

Guide to Building Business Processes

Before You Begin

The instructions in this sample is geared towards users that are already familiar with WebLogic Integration tasks and procedures.

- If you are new to WebLogic Integration, consider completing Tutorial: Building Your First Business Process before using this sample.
- If you are also new to ebXML business processes, consider completing Tutorial: Building ebXML Solutions before proceeding.

Step 1: Generating a Test Certificate

Before you can configure and run this example, you need to generate a certificate which you will later on import into your WebLogic Integration keystore. Once the certificate is imported into the keystore, you can use it as an encryption, a signature, or a client certificate for WebLogic Integration and also as a WebLogic Server certificate. In a production environment, you would

most likely have several certificates, but since this example is for testing only, you use one certificate for both client and server purposes.

You can generate the test certificate using any tool, however, the procedures in this section describe how to generate the certificates using OpenSSL. This is an open source tool which can be downloaded from www.openssl.org.

Before you create the certificate, you need to create the a Public/Private key pair that you then use to create the test certificate. If you are running OpenSSL in a Windows environment, you must first complete the "Configuring Windows to Run OpenSSL" below, before you can create the key pair. stand

This step includes the following procedures:

Configuring Windows to Run OpenSSL

The following steps should be completed to configure your Windows environment to run OpenSSL:

1. In a DOS command window, type the following to set the `OPENSSL_CONF` environment variable to point to the OpenSSL configuration file:

   `$ set OPENSSL_CONF=c:\openssl-`*`[X.X.X]`*`-src\apps\gencert.conf`

   Where *`[X.X.X]`* is the version of your OpenSSL installation, for example 0.9.7.

   Note: Due to the fact that Internet Explorer uses files of the type `.cnf` for Speed Dial configuration files, the OpenSSL configuration file might appear without the `.cnf` suffix and may have a shortcut icon.

2. Randomly select any five large files on your hard drive and then copy them to a folder where you intend to create the keys and certificates.

3. Rename the files to `file1`, `file2`, `file3`, `file4`, and `file5`. These files will be used by the OpenSSL facility to create the public/private key pair.

4. Verify that your path includes `c:\openssl\bin`.

You are now ready to create a public/private key pair using OpenSSL.

Creating a Public/Private Key Pair

The following section describes how to create 1024-bit RSA public/private key pair using OpenSSL:

In a DOS command window, type the following:

```
$ openssl genrsa rand file1:file2:file3:file4:file5 out WLCert.key 1024
```

where *file1:file2:file3:file4:file5* represents the five large files you created in Configuring Windows to Run OpenSSL.

You are now ready to create any type of X.509 certificate using OpenSSL.

Generating the Test Certificate

To generate the self-signed test certificate, complete the following procedure:

1. In a DOS command window, type the following:

   ```
   $ openssl req new key WLCert.key out WLCert.csr
   ```

2. In a DOS command window, type the following:

   ```
   $ openssl x509 req days 30 -in WLCert.csr signkey WLCert.key
   -outWLCert.crt
   ```

   You should now have two new files, `WLCert.key` and `WLCert`.crt, in your directory.

Step 2: Configuring Keystores for WebLogic Integration

Digital certificates are stored in two types of keystores in WebLogic Integration:

- Identity keystore—the keystore which stores private keys for local trading partners and certificates for both the local and remote trading partners.
- Trust keystore—the keystore which stores the trusted certificate authority certificates associated with any certificates used in WebLogic Integration.

This example assumes that you have created your own keystores and trusts. However, you can also complete this example by using the demonstration keystore file (`DemoIdentity.jks`) and the demonstration trust (`DemoTrust.jks`) that are part of your WebLogic Server installation and therefore WebLogic Integration installation. These Java Key Store files are located in the following location:

BEA_HOME`\weblogic102\server\lib\`

where *BEA_HOME* is the directory in which you installed your product.

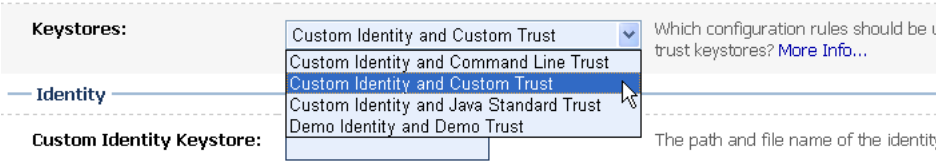
Since the underlying server used by WebLogic Integration is the WebLogic Server application, this section demonstrates how to use the WebLogic Server Administration Console to configure the keystores.

To configure the keystores:

1. Start your WebLogic Server:
2. Open the WebLogic Server Console.

   From WebLogic Integration, you do this by selecting **Tools** > **WebLogic Server** > **WebLogic Console**.
3. Login using the username and password specified when you created the WebLogic Integration domain. (The default username and password for the default domains is weblogic/weblogic.)
4. In the left pane, navigate to Servers > server_name
   Where *server_name* is the name of your WebLogic Server.
5. Select the **Keystores** tab**.**
6. From the **Keystores** drop-down menu, select **Custom Identity And Custom Trust,** as shown in Figure A-2.

Figure A-2 Setting KeyStore Tab



7. Click **Continue**.

   The Configure Keystore Properties screen appears.
8. In the fields described, enter the following information:

   Custom Identity

   – **Custom Identity Key Store File Name:** The fully qualified path to your identity keystore.
   If you are using the demonstration keystores, enter
   BEA_HOME\weblogic102\server\lib\DemoIdentity.jks

Where *BEA_HOME* is the directory in which you installed WebLogic Server.

- **Custom Identity Key Store Type:** The type of the keystore. Generally, this attribute is JKS. If this attribute is not specified, the default keystore type defined in the security policy file for the JDK is used.
  If you are using the demonstration keystores, enter `JKS`.
- **Custom Identity Store Pass Phrase:** The password defined when creating the keystore. Confirm the password.
  If you are using the demonstration keystores, enter `DemoIdentityKeyStorePassPhrase`.

  Note: This attribute is optional or required depending on the type of keystore. All keystores require the passphrase in order to write to the keystore. Some keystores do not require the passphrase to read from the keystore. Whether or not you define this property depends on the requirements of the keystore. For example, WebLogic Server only reads from the keystore so a passphrase is not required, however, WebLogic Integration writes to keystores and therefore requires a passphrase.

Custom Trust

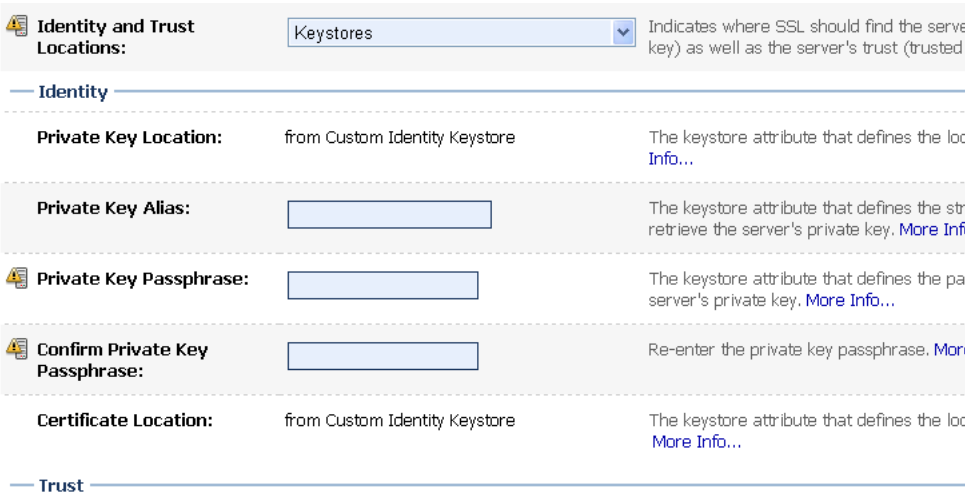
- **Custom Trust Store File Name:** The fully qualified path to your trust keystore.
  If you are using the demonstration keystores, enter *BEA_HOME*`\weblogic102\server\lib\DemoTrust.jks`.

  Where *BEA_HOME* is the directory in which you installed WebLogic Server.
- **Custom Trust Key Store Type:** The type of the keystore. Generally, this attribute is JKS. If this attribute is not specified, the default keystore type defined in the security policy file for the JDK is used.
  If you are using the demonstration keystores, enter `JKS`.
- **Custom Trust Key Store Pass Phrase:** The password defined when creating the keystore. Confirm the password.
  If you are using the demonstration keystores, enter `DemoTrustKeyStorePassPhrase`.

  Note: This attribute is optional or required depending on the type of keystore. All keystores require the passphrase in order to write to the keystore. Some keystores do not require the passphrase to read from the keystore. Whether or not you define this property depends on the requirements of the keystore. For example, WebLogic Server only reads from the keystore so a passphrase is not required, however, WebLogic Integration writes to keystores and therefore requires a passphrase.

9. Click **Save**.

10. Click **SSL** tab. You use this screen to configure the SSL configuration for your WebLogic Server.



Note: However, since you need to load the private key you created in Creating a Public/Private Key Pair into the keystore before you can configure the SSL settings, you can minimize this window for now. Instead restart your WebLogic Server and continue to the next section, "Step 3: Configuring the Local Trading Partner in WebLogic Integration 1," which includes loading the private key into the keystore.

To learn more about the setting you just entered, see "Configuring Keystores" in Configuring SSL.

Step 3: Configuring the Local Trading Partner in WebLogic Integration 1

WebLogic Integration contains two default trading partners named Test_TradingPartner_1(TP 1)and Test_TradingPartner_2(TP 2). In this section, you configure Test_TradingPartner_1 to be your local trading partner in WebLogic Integration 1(WLI 1).

Note: Before you start any of the procedures in this section, you must have configured your keystores as described in Step 2: Configuring Keystores for WebLogic Integration and restarted your WebLogic Server after completing the keystore configuration.

This section contains the following procedures:

Configuring the Local Trading Partner

The following procedure describes how to configure the default trading partner Test_TradingPartner_1 to act as the local trading partner in your WebLogic Integration application:

1. If it is not already running, start your WebLogic Server.
2. Open the WebLogic Integration Administration Console.
3. Navigate to **Trading Partner Management** > **Profile Management**

   The View and Edit Trading Partner Profiles screen appears with the two trading partners Test_TradingPartner_1 and Test_TradingPartner_2 listed, as shown in Figure A-3.

Figure A-3 Editing Trading partner Profiles

   Since you are going to import the configuration for the remote trading partner from WebLogic Integration 2, you can delete Test_TradingPartner_2 from the list.

4. Select **Test_TradingPartner_2** by clicking on the option box next to it.
5. Click **Delete**.

You now need to add the appropriate certificates to your local trading partner so that they will be imported into the keystore.

Adding the Test Certificate to the Keystore

The following procedure describes how to add the certificate, which you created in "Generating the Test Certificate" on page A-5, to your local trading partner configuration:

1. Click **Test_TradingPartner_1**.

   The details of your trading partner, including general information, bindings, and certificates are displayed. Note that there are no certificates configured for this trading partner.

2. Click **Add Certificate**

   The Add Certificate (Step 1 of 2) screen appears.

3. Select the **Import certificate from file** option.

4. Click **Next**

   The Add Certificate (Step 2 of 2) screen appears. You use this screen to import a client certificate file to be stored in the key store and used by the local trading partner. However, before you can create the client certificate, you have to create a password alias.

5. Click **Add alias...**,

   The **Add New Password Alias** screen appears.

6. In the **Password Alias Name field**, enter **TP1-client**.

7. Enter **TP1Client** as password to use for this alias and confirm it.

8. Click **Submit**.

   The Add Certificate (Step 2 of 2) screen appears again, with the alias values you just entered.

9. In the **Name** field, enter **TP1ClientCert**.

10. From the **Type** drop-down list, select **CLIENT**.

11. Next to the **Import Certificate Location,** click **Browse**.

12. Navigate to the `WLCert.crt` file which you created in "Generating the Test Certificate" on page A-5.

13. Next to the **Private Key Location**, click **Browse**.
14. Navigate to the `WLCert.key` file which you created in "Generating the Test Certificate" on page A-5.
15. Make sure that the **Import Certificate in Keystore** option is selected.
16. Click **Create certificate**.

By selecting **CLIENT** from the Type drop-down list, you specified the certificate to be a client certificate. You can add a signature certificate by using this same procedure, but instead selecting **SIGNATURE** from the Type drop-down list.

You can review all your configurations of the Test_TradingPartner_1 trading partner by navigating to **Trading Partner Management** > **Profile Management** and clicking on the **Test_TradingPartner_1** trading partner.

The next step is to edit the protocol bindings for your trading partner.

Editing the Trading Partner Binding

The default trading partner you just configured to be the local trading partner for WebLogic Integration, contains two ebXML default bindings. The following steps describes how to edit the ebXML 2.0 binding with the correct transport protocol and signature settings:

1. In the left pane, click **Bindings**.
2. From the **Name** drop-down list, select **Test_TradingPartner_1**.
3. Click **Go**.

   The Edit Binding screen appears.
4. In the list of bindings, click **TP1-ebxml20-binding**.

   The View Binding Details screen appears
5. Click **Edit Binding**.
6. Make the following edits:

   Transport Configuration

   – **Transport Protocol**: HTTPS

   – **End Point**: specify the URL to use *https* instead of *http* protocol and change the port number to the SSL port number to the port number of your WebLogic Server

domain.This is usually the even number immediately following your local port number. For example, for local port number 7001, the SSL port number is 7002.

7. Click **Submit**.

Your new binding settings are saved. To learn more about how to configure ebXML bindings including how to configure signatures and signature transforms, see "Adding Protocol Bindings to a Trading Partner" in Trading Partner Management.

Since you have completed the configuration of the local trading partner and have imported the test certificate into the keystore, you can now return to the WebLogic Server console and configure the SSL settings.

Step 4: Configuring the SSL Settings in WebLogic Server

Although you specified the certificate you loaded into the keystore as a client certificate when you configured the WebLogic Integration, for testing purposes you can also use this certificate as the server certificate for WebLogic Server. You just have to configure the server with the correct alias in the SSL settings.

You configure the SSL settings on the WebLogic Server in the WebLogic Console:

1. If the console window you opened in Step 2: Configuring Keystores for WebLogic Integration is still opened, return to it. If not, complete the following procedure:
   a. If not already started, start your WebLogic Server:
   b. Open the WebLogic Server Console.

      From WebLogic Integration, you do this by selecting **Tools** > **WebLogic Server** > **WebLogic Console**.
   c. Login using the username and password specified when you created the WebLogic Integration domain. (The default username and password for the default domains is weblogic/weblogic.)
   d. In the left pane, navigate to **Servers** > *server name*
      Where *server name* is the name of your WebLogic Server.
   e. Select the **Keystores & SSL** tab**.**
   f. Scroll down to the **SSL Configuration** part of the screen and click **Change**.
   g. From the **Identity and Trust Locations** drop-down menu, select **Key Stores**.

h. Click **Continue**.

2. On the Review SSL Private Key Settings screen, enter the following information:
   - **Private Key Alias**: `TP1-client`

     This is the alias you specified when loading the private key for WebLogic Server from the keystore in Adding the Test Certificate to the Keystore.
   - **Passphrase**: `TP1Client`

     This is the password specified when loading the private key for WebLogic Server into the keystore in Adding the Test Certificate to the Keystore.
3. Click **Continue**.

   An alert screen appears, which informs you that you need to restart your server. You can ignore this for now, instead restart your server after you have completed all the SSL configuration steps.
4. Click **Finish**.

   The Keystore Configuration screen appears.
5. Scroll to the end of the screen and click **Show** to display the Advanced Options. The Advanced options is where you configure mutual authentication.
6. From the **Two Way Client Cert Behavior**, select **Client Certs Requested And Enforced**. This option assures mutual authentication behavior.
7. Click **Apply**.

   To learn more about the settings you just entered, see "Configuring Two-Way SSL" in Configuring SSL.
8. If you have not already done so, restart the WebLogic Server.
9. If the keystores are configured correctly, you should see details similar to the following in the WebLogic Sever Log:

```
<Feb 1, 2007 4:11:45 PM IST> <Notice> <Security> <achepuri02>
<examplesServer> <[ACTIVE] ExecuteThread: '0' for queue:
'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1170326505038> <BEA-090082> <Security initializing using security
realm myrealm.>

<Feb 1, 2007 4:11:49 PM IST> <Notice> <WebLogicServer> <achepuri02>
<examplesServer> <Main Thread> <<WLS Kernel>> <> <> <1170326509604>
<BEA-000365> <Server state changed to STANDBY>
```

```
<Feb 1, 2007 4:11:49 PM IST> <Notice> <WebLogicServer> <achepuri02>
<examplesServer> <Main Thread> <<WLS Kernel>> <> <> <1170326509604>
<BEA-000365> <Server state changed to STARTING>

<Feb 1, 2007 4:12:04 PM IST> <Warning> <HTTP> <achepuri02>
<examplesServer> <[ACTIVE] ExecuteThread: '0' for queue:
'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1170326524145> <BEA-101369>
<weblogic.servlet.internal.WebAppServletContext@1dab0f0 - appName:
'BEA_WLS_DBMS_ADK', name: 'BEA_WLS_DBMS_ADK_Web', context-path:
'/BEA_WLS_DBMS_ADK_Web': The encoding jsp-descriptor param has been
deprecated. Consider declaring the encoding in the jsp-config element
(web.xml) or as a page directive (pageEncoding) instead.>

<Feb 1, 2007 4:12:18 PM IST> <Notice> <Log Management> <achepuri02>
<examplesServer> <[STANDBY] ExecuteThread: '5' for queue:
'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1170326538816> <BEA-170027> <The server initialized the domain log
broadcaster successfully. Log messages will now be broadcasted to the
domain log.>

<Feb 1, 2007 4:12:19 PM IST> <Notice> <WebLogicServer> <achepuri02>
<examplesServer> <Main Thread> <<WLS Kernel>> <> <> <1170326539377>
<BEA-000365> <Server state changed to ADMIN>

<Feb 1, 2007 4:12:19 PM IST> <Notice> <WebLogicServer> <achepuri02>
<examplesServer> <Main Thread> <<WLS Kernel>> <> <> <1170326539407>
<BEA-000365> <Server state changed to RESUMING>

<Feb 1, 2007 4:12:21 PM IST> <Notice> <Security> <achepuri02>
<examplesServer> <[ACTIVE] ExecuteThread: '2' for queue:
'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1170326541089> <BEA-090171> <Loading the identity certificate and
private key stored under the alias DemoIdentity from the jks keystore
file C:\bea_GA\WEBLOG~1\server\lib\DemoIdentity.jks.>

<Feb 1, 2007 4:12:21 PM IST> <Notice> <Security> <achepuri02>
<examplesServer> <[ACTIVE] ExecuteThread: '2' for queue:
'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1170326541360> <BEA-090169> <Loading trusted certificates from the
jks keystore file C:\bea_GA\WEBLOG~1\server\lib\DemoTrust.jks.>

<Feb 1, 2007 4:12:21 PM IST> <Notice> <Security> <achepuri02>
<examplesServer> <[ACTIVE] ExecuteThread: '2' for queue:
'weblogic.kernel.Default (self-tuning)'> <<WLS Kernel>> <> <>
<1170326541370> <BEA-090169> <Loading trusted certificates from the
jks keystore file C:\bea_GA\JROCKI~1\jre\lib\security\cacerts.>

<Feb 1, 2007 4:12:21 PM IST> <Error> <Server> <achepuri02>
<examplesServer> <DynamicListenThread[Default[2]]> <<WLS Kernel>> <>
```

```
<> <1170326541620> <BEA-002606> <Unable to create a server socket for
listening on channel "Default[2]". The address 127.0.0.1 might be
incorrect or another process is using port 7001:
java.net.BindException: Address already in use: JVM_Bind.>
.
.
.
```

You have now completed the WebLogic Server configuration. To learn more about WebLogic Server SSL configuration, see Configuring SSL. The next step is to export the Test_TradingPartner_1 data so that you can import this data later on when you configure the remote trading partner in WebLogic Integration 2.

Step 5: Exporting the WebLogic Integration Trading Partner Data

Instead of configuring both the company profile and partner profile by going through the configuration screens in WLI 2, you can import data that has been exported from WLI 1directly into WLI 2 and have the partner profile automatically configured.

Complete the following procedure to export the WebLogic Integration trading partner data:

1. If it is not already running, start your WebLogic Server.
2. Open the WebLogic Integration Administration Console.
3. Navigate to **Trading Partner Management** > **Profile Management**
4. In the left pane, click **Import/Export**.
5. In the **Import/Export** pane, select **Export**.
6. Select the **Trading Partner** option.
7. Click **Browse** next to **Trading Partner**.
8. Deselect all but the **Test_TradingPartner_1** trading partner.
9. Click **Done**.
10. For the **Format** option, retain **WLI Standard** option that is selected by default.
11. Click **Export**.
12. If a **File Download** dialogue opens, click **Save**.

13. In the **Save As** window navigate to a location in which you want to save the exported file to.
14. Enter **`TP1.xml`** as the filename and click **Save**.

    Note: Remember the navigation path to the file. You need this when you import your trading partner information.

You have completed the WebLogic Integration local trading partner configuration. To learn more about creating, configuring, and managing trading partners in WebLogic Integration, see Trading Partner Management.

You can create your remote trading partner in WLI 1 using the procedures you just completed for Test_TradingPartner_1. However, in this example you take a short cut by importing the company profile settings from WLI 2 and use that as the remote trading partner.

The next step shows you how to configure a trading partner named Test_TradingPartner_2 as the company profile partner in the WLI 2 application and how to export the company profile information into a file that you can then import into WLI 1.

Step 6: Configuring the Local Trading Partner in WebLogic Integration 2

In this section, you configure the default trading partner Test_TradingPartner_2 to be your local trading partner with WLI 1 at the other end. You then edit the trading partner bindings and add the appropriate certificates to the trading partner and keystore. Then export TPM data.

To configure the Test_Trading Partner_2 in WebLogic Integration 2, follow the same steps through 2 to 5.

Step 7: Configuring the Remote Trading Partner in WebLogic Integration

In this section, you create a remote trading partner in the WLI 1 application by importing the company profile information which you exported from WLI 2.

In WLI 1, import the TPM file you that you exported from WLI 2 in step 5(TP2.xml). Open the TP2.xml file and change type = **Local** to type = **Remote** before importing it in WLI 1**.** Repeat the same to import the TPM file that you exported from WLI 1.

The following steps describes the importing procedure:

1. If it is not already running, start your WebLogic Server.

2. Open the WebLogic Integration Administration Console.
3. Navigate to **Trading Partner Management** > **Profile Management**
4. In the **Import/Export** pane, click **Import**.
5. In the **File Name** field, enter the path to `TP2.xml` location to the file you exported from WebLogic Integration.
6. Select **WLI 2** as the **Import Format**.
7. Click **Import**.

After successfully importing the trading partner information, remember to review the new trading partner profile and make sure that the end point URL is correct. You do this by navigating to Profile Management, clicking on Test_TradingPartner_2, and clicking on its binding. When you click on Test_TradingPartner_2, note that three certificates (client, server, signature) were automatically created in the Company Profile in WLI 2 and imported into WLI 1.

Now that you have configured both the local and the remote trading partner for WLI, the next step is to add services and service profiles to those trading partners.

Step 8: Creating Services and Service Profiles in WebLogic Integration

Once the Test_TradingPartner_1 and Test_TradingPartner_2 configurations are completed, you have to create services and corresponding service profiles for those trading partners.

For WLI 2, create a process service with the name as the URL of the participant jpd process. Configure the service profiles for the local and remote trading partner profiles in WebLogic Integration.

In WebLogic Integration:

- 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.
- *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.

To be able to configure the services correctly, the business process which initiates the ebXML message exchange must be currently deployed. This section contains the following procedures:

Creating the Trading Partner Service

Complete the following steps to add a service to your trading partner profile in WLI 1.

1. Deploy your ebXML initiator business process.

   If you are not familiar with how to build and deploy ebXML business processes, consider completing one of the exercises in Tutorials: Building ebXML Solutions.

2. In the WebLogic Integration Administration Console, navigate to **Trading Partner Management** > **Service Management**.

3. In the left pane, click **Create New**.

   The **Add Service** screen appears.

4. Click Browse and navigate to the appropriate service control.

5. From the **Type** drop-down menu, select **Service Control**.

6. From the **Business Protocol** drop-down menu, select **EBXML**.

7. Click **Add Service**.

Your service is created and the View And Edit Service Details screen appears on which you add the service profile.

Creating the Process Service

Complete the following steps to add a service to your trading partner profile in WLI 2.

1. Deploy your ebXML initiator business process.

   If you are not familiar with how to build and deploy ebXML business processes, consider completing one of the exercises in Tutorials: Building ebXML Solutions.

2. In the WebLogic Integration Administration Console, navigate to **Trading Partner Management** > **Service Management**.

3. In the left pane, click **Create New**.

The **Add Service** screen appears.

4. Click Browse and navigate to the appropriate process.
5. From the **Type** drop-down menu, select **Process**.
6. Type an appropriate JPD URL. For example, `/testWeb/processes/Process.jpd`
7. From the **Business Protocol** drop-down menu, select **EBXML**.
8. Click **Add Service**.

Your process is created and the View And Edit Service Details screen appears on which you add the process service.

Creating the Service Profile

After you have created the trading partner service, you create a service profile which specify the protocol binding and URL endpoints for the local and remote trading partners that offer and call the service. The following procedure describes how to add a service profile:

1. On the **View And Edit Service Details** screen, click **Add Service Profile**.

   The Add Service Profile screen is displayed.
2. From the **Name** drop-down menus, select your **LOCAL** and **REMOTE** trading partners.
3. Specify your LOCAL and REMOTE trading partners according to the following table:

Table A-1 Local and Remote Trading Partner for WLI 1

	LOCAL	REMOTE
Name	`Test_TradingPartner_1`	`Test_TradingPartner_2`
Binding	`wli-ebxml20-secure-binding`	`wli-ebxml20-secure-binding`

Note: Similarly, for WLI 2, the local trading partner will be `Test_TradingPartner_2` and remote trading partner will be `Test_TradingPartner_1`.

Also, make sure you change the endpoint URLs to use *https*, not *http*. If they are set to the wrong protocol, follow the directions in Editing the Trading Partner Binding to select the correct one.

4. Click **Submit**.

5. On the next screen, click **Yes** to begin configuring authentication.
6. From the **Choose type of Authentication Mode** options, select **Mutual** for both the **LOCAL** and **REMOTE** trading partners.

   Note: Although it is not enforced, typically the same type of authentication is selected for both the local and remote trading partner.
7. Click **Next**.
8. On the next screen, select:

   a. **TP1-clt** as the client certificate for the LOCAL trading partner.

   b. xxxx-client as the client certificate for the REMOTE trading partner.

   c. xxxx-server as the server certificate for the REMOTE trading partner.

   Where xxxx is a number which was randomly generated when you imported the WLI 2 self-signed certificate file.
9. To preview to the configuration, click **Preview config**.
10. Click **Add**.

    Authentication is added and the View and Edit Service Details page is displayed.

    Note: If there is an error, the Add Authentication page is redisplayed. A message indicating the problem is displayed above the input requiring correction.

Note: Repeat the same steps for configuring Service Profile in WLI 2.

You have now created a service and its service profile. To learn more about services and service profiles, see “Adding Services” and “Adding Service Profiles to a Service” in Trading Partner Management. The next step is to configure the iPlanet SSL settings and then proceed to complete the configuration of WLI 2.

Step 9: Configuring the iPlanet Server

This section describes how to configure your iPlanet Web Server as the proxy server for this sample.

If you do not already have iPlanet Web Server installed, you can download it from Sun’s website the following location:
`http://www.sun.com/software/download/products/3f186391.html`

Refer to the product documentation to install and start the iplanet admin and managed server.

This section contains the following procedures:

Creating the Trust Database

Before you can configure your iPlanet server certificates, you have to create a trust database in which to store the certificates. To do so, complete the following procedure:

1. Open the iPlanet administration console.
2. Navigate to **Servers** > **Mange Servers**.
3. Select a managed server and click **Manage**.

   The Managed Server Configuration screen appears.
4. Select the **Security** tab.
5. Click **Create Database**.
6. Enter and confirm a password for the database.
7. Click **OK**.

A dialog window confirming the successful initialization appears. The next step is to request a trial digital certificate from Verisign which you use as the server certificate for iPlanet and also imported later on into the Partner Profile in WebLogic Integration 2.

Requesting a Trial Digital Certificate from Verisign

You can request a trial digital certificate from Verisign to use for testing purposes. The certificate is valid for a limited number of days. To request a certificate, complete the following steps:

1. Navigate to **Managed Server Console** > **Security** > **Request a Certificate**.
2. Select the **New certificate** option.
3. From the **Submit to Certificate Authority via** option, select **CA Email Address** and enter your email address.
4. From the **Cryptographic Module** drop-down list, select **internal**.
5. In the **Key Pair File Password** field, enter the password you want to use as the iPlanet server private key password.
6. Enter your name and contact information details in the remaining fields.
7. Click **OK**.

   A confirmation message is displayed in the Managed Server Console.
8. Copy all the text between **`-----BEGIN NEW CERTIFICATE REQUEST-----`** and **`-----END NEW CERTIFICATE REQUEST-----`** and paste it into a text file. This is your certificate request which you send to a certificate authority such as Verisign.
9. Using a web browser, navigate to `http://www.verisign.com/products/srv/trial/step1.html`.
10. Follow the directions on the Verisign site.

    After you complete the request, Verisign will send you an email with the digital certificate. Copy the content and save it in a file named `iPlanetServer.pem`. This file is your digital certificate for the iPlanet Server.

You have completed the trial digital certificate request process. The next step is to install the digital trial certificate as the iPlanet server certificate.

Installing the iPlanet Server Certificate

You are now ready to install the server certificate for iPlanet. The following procedure describes the steps to complete:

1. Navigate to **Managed Server Console** > **Security** > **Install Certificate**

   The Install a Server Certificate screen appears.
2. Enter the following information:
   - **Certificate For**: This Server

- **Cryptographic Module**: internal
- **Key Pair File Password**: the same password as in Requesting a Trial Digital Certificate from Verisign.
- **Message text (with headers):** enter the contents of the `iPlanetServer.pem` file that you created in Requesting a Trial Digital Certificate from Verisign.

3. Click **OK**.

   The Add Server Certificate screen appears with the details of the certificate you are adding.

4. Click **Add Server Certificate**.

You have successfully added the server certificate. The next step is to download and convert the trusted CA server certificate.

Requesting a Trusted CA Certificate from Verisign

In addition to the server certificate you just installed, you also need a trusted certificate from an Certificate Authority such as Verisign. To request a trusted certificate from Verisign, complete the following procedure:

1. Using a web browser, navigate to `http://www.verisign.com/server/trial/faq/index.html` to retrieve a CA certificate for the iPlanet server from Verisign.
2. Click **Accept**.
3. Save the certificate on your local drive as a file named `iPlanetCA.der`.

The server certificate is in binary format. Before you can use it with iPlanet, you must convert it to PEM format. The `der2pem` command line utility included with WebLogic Server can be used to convert the certificate. To learn how to use the utility, see “der2pem” in Using the WebLogic Server Java Utilities in the *WebLogic Server Command Reference*.

Follow the procedure described to convert the `iPlanetCA.der` file to `iPlanetCA.pem`.

Note: A DER format file contains binary data and can only be used for a single certificate. A PEM format file supports multiple digital certificates. For example, a certificate chain can be included. The order of the files is important, they should be in the order of trust. The server digital certificate should be the first digital certificate in the file, the issuer of the digital certificate should be next, and so on, until you get to the self-signed root certificate authority certificate.

After you have completed the conversion, the next step is to install the trusted CA certificate.

Installing the Trusted CA Certificate

To install the trusted certificate you just requested, do the following:

1. Navigate to **Managed Server Console** > **Security** > **Install Certificate**
2. Enter the following information:
   - **Certificate For**: Trusted Certificate Authority (CA)
   - **Cryptographic Module**: internal
   - **Certificate Name**: Verisign CA.
   - **Message is in this file**: enter the location of the `iPlanetCA.pem` file you created in Requesting a Trusted CA Certificate from Verisign
3. Click **OK**.

   The Add Server Certificate screen appears with the details of the certificate you are adding.
4. Click **Add Server Certificate**.

You have successfully installed the trusted CA certificate.

The next step is to install the WebLogic Integration 2 certificate.

Installing the WebLogic Integration 2 certificate

Use the procedures described in "Installing the Trusted CA Certificate" on page A-24 to import the TP2.xml file which you exported to create WLI 2 CA certificate for your iPlanet server.

You have now successfully installed the necessary iPlanet certificates. There is just one final step required to get the iPlanet configuration to work with SSL.

Configuring iPlanet for SSL

1. Open the iPlanet administration console.
2. Navigate to **Servers** > **Mange Servers**.
3. Select **Preference** > **Edit Listen Sockets**.
4. In the Security column, select On.

5. Click **OK**.
6. Click **Attributes**.
7. In the **Client Authorization** column, click **Off** to change it to **On**. This assures mutual authentication.
8. As the final configuration step, you need to modify two of the installed iPlanet configuration files:
   - To your `obj.conf` file add the following lines of code:

     ```
     <Object name="myProxy" ppath="*">
     PathCheck fn="get-client-cert" method="(GET|POST)" dorequest="1"
     Service fn=wl_proxy FileCaching="OFF" Debug="ALL"
     KeepAliveEnabled=false DebugConfigInfo="ON"
     WebLogicHost=172.16.17.183\
     WebLogicPort=7001 WLLogFile="C:/depot/newlog.txt" SecureProxy="OFF"
     WLProxySSL="ON" RequireSSLHostMatch="False"
     </Object>
     ```

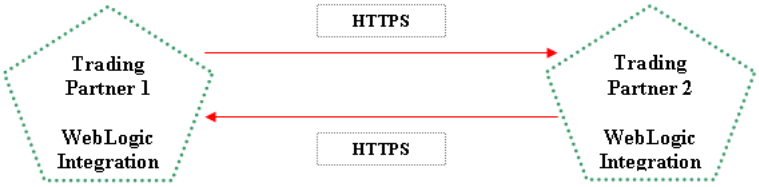
   - To your `magnus.conf` file, add the following lines of code:

     ```
     Init fn="load-modules"
     shlib="D:/iPlanet/Servers/bin/https/bin/proxy36.dll"
     funcs="wl_proxy,wl_init"
     Init fn="wl_init"
     ```

This concludes the iPlanet configuration step. To learn more about the settings you just configured, see Installing and Configuring the Netscape Enterprise Server Plug-In in *Using Web Server Plug-Ins With WebLogic Server*.

Example: RosettaNet Security Configuration

This example demonstrates how to configure the security settings for RosettaNet message exchange between trading partners over HTTPS protocol, using mutual authentication. In this example, both trading partners are configured on WebLogic Integration as shown in the figure below:



In the preceding figure:

- Messages (and acknowledgements) are sent from Trading Partner 1 to Trading Partner 2 through mutual authentication HTTPS.
- Acknowledgements and responses are sent from Trading Partner 2 process through mutual authentication HTTPS.

The following topics are discussed in this section:

"Keystores Used in the Example" on page B-3

This section discusses the use of keystores in WebLogic Integration and describes the demonstration keystores used in this example.

Related Topics

Managing WebLogic Security

Trading Partner Integration Security

Tutorial: Building RosettaNet Solutions

Building Your First Business Processes

Keystores Used in the Example

The procedures in this example uses the demonstration keystore which is included in your WebLogic Integration installation. This keystore file can only to be used for testing purposes. This Java Key Store files is located in the following location:

`BEA_HOME\wlserver_10.0\server\lib\DemoIdentity.jks`

where *BEA_HOME* is the directory in which you installed your product.

You can use any other keystore to complete the example, however, the demonstration keystore already has some of the certificates necessary loaded into it. If you need details about how to create and load certificates into your custom keystore, see Step 1: Generating a Test Certificate and Step 2: Configuring Keystores for WebLogic Integration in Example: ebXML Security Configuration.

In this sample, the following terminology is used:

- Client certificate—the digital certificate used in mutual authentication.
- Encryption certificate—the certificate used for encryption.
- Signature certificate—the certificate used to sign messages for authenticity of the sender and integrity of messages.
- Server certificate—the digital certificate of the server on which the remote application is running.

Note: WebLogic Integration requires remote trading partners to have both a client and server certificate configured while local trading partners only need a client certificate.

- Identity keystore—the keystore which stores private keys for local trading partners and certificates for both the local and remote trading partners.
- Trust keystore—the keystore which stores the trusted certificate authority certificates associated with any certificates used in WebLogic Integration.

Before You Begin

The instructions in this sample is geared towards users that are already familiar with the basic WebLogic Server and WebLogic Integration tasks and procedures. If you are new to these WebLogic applications, consider completing Tutorial: Building Your First Business Process before using this sample. It is further assumed that you are familiar with RosettaNet concepts and how to configure RosettaNet business processes in WebLogic Integration. If you are new to RosettaNet business processes, consider completing Tutorial: Building RosettaNet Solutions before proceeding.

This sample configuration involves a participant process and a initiator process created and deployed in WebLogic Integration, either on two different computers or on two different domains on the same machine. Before you can start the security configuration part of this sample, you must complete the following:

1. If you are going to run the Trading Partner 1 and the Trading Partner 2 processes on the same computer, create two domains with different port numbers. Be sure to make the domains SSL enabled.

   For instructions of how to create a domain, see Creating WebLogic Domains Using the Configuration Wizard.

   When you create a new WebLogic Integration domain using the Configuration Wizard, the Configuration Wizard automatically populates the Trading Partner Management (TPM) repository with default trading partners and bindings. You will use these default trading partners in this sample.

   If you are using a point based database, you will have to configure your second domain to point to a different database instance, i.e. use a different port number, since local and remote trading partners are not allowed to use the same database in WebLogic Integration. You do this by changing the default port number from 9093 to, for example, 9090 in the following files:

   – `config.xml` (2 instances)

   – `URLS.dat`

   – If you are using a Windows operating system: `startWeblogic.cmd` and `stopWeblogic.cmd`

   – If you are using an Unix operating system: `startWeblogic.sh` and `stopWeblogic.sh`.

These files are all located in: *BEA_HOME*\user_projects\domains*domainName* where *BEA_HOME* is the directory in which you installed WebLogic Integration (such as c:\bea) and *domainName* is the name of your second domain (such as tptutorial2).

2. Create the initiator process.
3. Create the participant process.

   For instructions on how to create business processes, see Tutorial: Building Your First Business Process. To learn more about RosettaNet processes, see Tutorial: Building RosettaNet Solutions.

   If you are running both processes on one computer in different domains, both domains use the same keystores. It is important that you only have the current process and corresponding domain running when you are configuring trading partner information, or simultaneous updates to the keystore may occur and overwrite each other.

Step 1: Configuring the Local Trading Partner for the Trading Partner 1 Setup

In this section, you configure the default trading partner Test_TradingPartner_1 to be the local trading partner for the initiator business process. You then add certificates to the trading partner and configure the binding for that trading partner. Lastly, you export the trading partner data into an XML file which, you later import as the remote trading partner for the participant. This section contains the following procedures:

- Configuring the Local Trading Partner
- Adding the Certificates
- Editing the Trading Partner Binding
- Enabling the Trading Partner Profile
- Exporting the Trading Partner Data
- Exporting the Server Certificate

Configuring the Local Trading Partner

To configure Test_TradingPartner_1 to be your local trading partner, complete the following procedure:

1. Start WebLogic Workshop.

2. Open the application which contains the initiator process that you created in Before You Begin

3. If you are running both processes on the same computer in different domains, navigate to **Tools** > **Application Properties...**.

   The Application Properties window opens.

4. Select the domain that you want to use from the **Server Home Directory** drop-down menu or by using the **Browse...** button.

5. Click **OK**.

6. If it is not already running, start your WebLogic Server.

7. Open the WebLogic Integration Administration Console.

8. Navigate to **Trading Partner Management** > **Profile Management.**

   The View and Edit Trading Partner Profiles screen appears with the two trading partners Test_TradingPartner_1 and Test_TradingPartner_2 listed, as shown in Figure B-1

Figure B-1 Trading Partner Profiles

   Since you are going to import the configuration for the remote trading partner later on, you delete Test_TradingPartner_2 from the list at this point.

9. Select **Test_TradingPartner_2** by clicking on the option box next to it.

10. Click **Delete**.

You now need to add the appropriate certificates—client, encryption, and digital signature—to your local trading partner.

Adding the Certificates

To add the appropriate certificates to your local trading partner:

1. Click **Test_TradingPartner_1**.

   The details of your trading partner, including general information, bindings, and certificates are displayed. Note that there are no certificates configured for this trading partner.

2. Click **Add Certificate**

   The Add Certificate (Step 1 of 2) screen appears.

3. Select the **Generate a certificate for TEST USE only** option.

   Note: The certificates you create in this sample is only for demo use and should never be used in production mode.

4. Click **Next**

   The Add Certificate (Step 2 of 2) screen appears. You use this screen to create a client certificate to be stored in the keystore and used by the local trading partner. However, before you can create the client certificate, you have to create a password alias.

5. Click **Add alias...**, as shown below.

   The **Add New Password Alias** screen appears.

6. In the **Password Alias Name field**, enter **init-rn20-clt**.

7. Enter and confirm a password to use for this alias.

8. Click **Submit**.

   The Add Certificate (Step 2 of 2) screen appears again, with the alias values you just entered.

9. In the **Name** field, enter **init-clt**.

10. If not already selected, from the **Type** drop-down list, select **CLIENT**.

11. If not already selected, select the **Import Certificate in Keystore** option.

12. Click **Create certificate**.

By selecting CLIENT from the Type drop-down list, you specified the certificate to be a client certificate. Using the instructions in step 2-6, create:

- A digital **SIGNATURE** certificate named **init-sig**.
- An **ENCRYPTION** certificate named **init-enc**.

You have completed adding the three certificates. The next step is to edit the binding for the Test_TradingPartner_1 trading partner.

Editing the Trading Partner Binding

To edit the default bindings for your local trading partner:

1. In the left pane, click **Bindings**.
2. From the **Name** drop-down list, select **Test_TradingPartner_1**.
3. Click **Go**.

   The Edit Binding screen appears.
4. In the list of bindings, click **TP1-rn20-binding**.

   The View Binding Details screen appears
5. Click **Edit Binding**.
6. Make the following edits:

   Transport Configuration

   – **Transport Protocol**: HTTPS

   – **End Point**: specify the URL to use *https* instead of *http* protocol and change the port number to the SSL port number you specified in "Before You Begin" on page B-4 when you created your domain.This is usually the even number immediately following your local port number. For example, for local port number 7001, the SSL port number is 7002.

   Message Level Encryption Configuration

   – **Encryption Certificate**: init-enc

   – **Encryption Level**: Entire Payload

   – **Cipher Algorithm**: 3DES

   Digital Signature Configuration for Non Repudiation

- **Signature Certificate**: init-sig
- **Signature Required**: select this option
- **Signature Receipt Required**: select this option
- **Hash Function**: SHA1

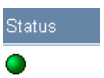
7. Click **Submit**.

You have completed editing the binding information. Before you proceed further, make sure that the trading partner profile you just configured is enabled.

Enabling the Trading Partner Profile

Complete the following procedure to enable the trading partner profile that you just created and configured:

1. Navigate to **Trading Partner Management** > **Profile Management.**
2. In the Status column, make sure that a green ball is displayed:



3. If a red ball is showing in the **Status** column, select the trading partner and click **Enable**.

Rather than entering all the Test_TradingPartner_1 configuration data again for the participant business process computer/domain, you export the data from the initiator process computer/domain and then import it as the remote trading partner configuration for the participant.

Exporting the Trading Partner Data

To import the trading partner data from the local trading partner:

1. In the left pane, click **Import/Export**.
2. In the **Import/Export** pane, select **Export**.
3. Select the **Trading Partner** option.
4. Click **Browse** next to **Trading Partner**.
5. Deselect all but the **Test_TradingPartner_1** trading partner.

6. Click **Done**.
7. For the **Format** option, select **WLI Standard**.
8. Click **Export**.
9. If a **File Download** dialog opens, click **Save**.
10. In the **Save As** window navigate to the location which you want to save the exported file to.
11. Enter **TradingPartner1** as a filename and click **Save**.

    Note: Remember the navigation path to the file. You will need this when you import your trading partner information later on.

12. Open **TradingPartner1.xml** in a text editor.
13. Change all instances of **LOCAL** to **REMOTE**.
14. In the **trading-partner** element, set the **is-default** attribute to **false**.
15. Save and close the file.

You have now completed the configuration of your the initiator local trading partner. The next section explains how to use the keytool utility to export the Test_TradingPartner_1 server certificate which, you will import later when setting up services and service profiles for the participant computer/domain.

Exporting the Server Certificate

1. At the command line prompt, navigate to *BEA_HOME*\wlserver_10.0\server\lib where *BEA_HOME* is the directory in which you installed your WebLogic Integration installation.
2. Enter the following: `keytool -export -alias demoidentity -file servercert1.crt -keystore DemoIdentity.jks -storepass DemoIdentityKeyStorePassPhrase`

   This exports your server certificate into a file named `servercert1.crt`.

   Note: If you are running both business processes on the same computer in different domains, both domains are accessing the same keystore. To avoid any problems associated with this configuration, create a copy of the `servercert1.crt` and name it `servercert2.crt` which, and use it as the second domains server certificate.

You have now completed the Test_TradingPartner_1 configuration. To learn more about creating, configuring, and managing trading partners in WebLogic Integration, see Trading Partner Management.

The next step is to configure the local trading partner information for the participant side of the sample.

Step 2: Configuring the Local Trading Partner for the Trading Partner 2 Setup

You configure the trading partner profile for the participant side of the sample in much the same way that you created the initiator side. Use the instructions in Step 1: Configuring the Local Trading Partner for the Trading Partner 1 Setup to complete the following:

1. If you are running two domains on the same computer, stop the WebLogic Server which is running on your initiator domain.
2. Open the participant business process that you created in Before You Begin.
3. If you are running both processes on the same computer, make sure that the participant process is configured to use the domain that is not the one used by the initiator process.
4. Start your WebLogic Server.
5. Deploy your participant process.
6. Open the WebLogic Administration Console.
7. Navigate to **Trading Partner Management** > **Profile Management.**
8. Delete Test_TradingPartner_1.
9. Select Test_TradingPartner_2 to be the default trading partner:
   a. Click on Test_TradingParnter_2 to view its profile settings.
   b. If **Default Trading Partner** is not already set to **true**, click **Edit Profile** and select the **Default Trading Partner** option.
10. In the Test_TradingPartner_2 profile create the following test certificates:
    – A **CLIENT** certificate named **part-clt**
    – A digital **SIGNATURE** certificate named **part-sig**.

– An **ENCRYPTION** certificate named **part-enc**

Note: This step is similar to Adding the Certificates, but you need to use the port numbers of the second computer/domain for your configuration.

11. Configure the binding for Test_TradingPartner_2.
12. Make sure that the trading partner is enabled.
13. Export the trading partner information to a file named **TradingPartner2.xml.**
14. In the TradingPartner2.xml file, change all instances of **LOCAL** to **REMOTE and** in the **trading-partner** element, set the **is-default** attribute to **false**.
15. If you are using a two computer configuration, use the JDK keytool utility to export the Test_TradingPartner_2 server certificate.

You have completed configuring Test_TradingPartner_2. The next step is to import the remote trading partner profiles for the two trading partners.

Step 3: Importing the Remote Trading Partner Information

Rather than recreating the information that you already created for each local trading partners when you create the remote trading partners, you import the XML files that you exported in Step 1: Configuring the Local Trading Partner for the Trading Partner 1 Setup and Step 2: Configuring the Local Trading Partner for the Trading Partner 2 Setup.

This set of instructions are the same for both Trading Partner 1 and Trading Partner 2. Go through this section twice, once for your Trading Partner 1 configuration and once for your Trading Partner 2 configuration.

To import the trading partner data:

1. Navigate to **Trading Partner Management** > **Profile Management.**
2. In the left pane, click **Import/Export.**

   The Import Trading Partner Management Data screen is displayed.
3. Click Browse and navigate to the file which corresponds to your trading partner according to the following table:

Table 4-7 Local and Remote Trading Partners

Current Local Trading Partner	Import File
Test_TradingPartner_1	TradingPartner2.xml
Test_TradingPartner_2	TradingPartner1.xml

4. Select **WLI Standard** as the **Import Format**.
5. Click **Import.**

After successfully importing the trading partner information, remember to review the new trading partner profile and make sure that the end point URL is correct. If they need to be changed, edit your bindings as described in Editing the Trading Partner Binding.

Step 4: Creating Services and Service Profiles in WebLogic Integration

Once the local and remote trading partner configurations are completed, you have to create services and corresponding service profiles for those trading partners. In WebLogic Integration, 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. *Service profiles* specify the protocol binding and URL endpoints for the local and remote trading partners that offer and call the service.

Note: Make sure that the process that you want to create the service and service profile for is deployed before you start this procedure.

This section is the same for both trading partner configuration. Complete the following procedure twice, once for your Trading Partner 1 configuration and once for your Trading Partner 2 configuration:

1. Deploy the process that has the trading partner you want to configure and open your administration console.

   Note: If you are running both trading partner configurations on the same machine in different domains, it is important that you only have the server that you are currently configuring running. Otherwise, you could end up with simultaneous updates to the keystores that could overwrite each, since both domains access the same keystore.

2. In the WebLogic Integration Administration Console, navigate to **Trading Partner Management** > **Service Management**.
3. In the left pane, click **Create New**.

   The **Add Service** screen appears.
4. Click **Browse** and navigate to one of the following:
   - If you are configuring **Test_TradingPartner_1**, the service control which corresponds to your initiator process,
   - If you are configuring **Test_TradingPartner_2**, your deployed participant process,
5. From the **Type** drop-down menu select one of the following:
   - **Service Control**, if you are configuring **Test_TradingPartner_1**.
   - **Process**, if you are configuring **Test_TradingPartner_2**.
6. From the **Business Protocol** drop-down menu, select **RosettaNet**.
7. Click **Add Service**.

   The View and Edit Service Details screen appears.
8. Click **Add Service Profile**.

   The Add Service Profile screen appears.
9. From the **Name** drop-down menus, select your **LOCAL** and **REMOTE** trading partners according toTable 4-8.

Table 4-8 Local and Remote Trading Partners

Process Currently Deployed	Local Trading Partner	Remote Trading Partner
`Initiator Process`	`Test_TradingPartner_1`	`Test_TradingPartner_2`
`Participant Process`	`Test_TradingPartner_2`	`Test_TradingPartner_1`

10. From the **Binding** drop-down menus, select your **LOCAL** and **REMOTE** bindings according to Table 4-9.

Table 4-9 Local and Remote Bindings

Process Currently Deployed	Local Binding	Remote Binding
Initiator Process	TP1-rn20-binding	TP2-rn20-binding
Participant Process	TP2-rn20-binding	TP1-rn20-binding

11. Click **Submit**.
12. On the next screen, click **Yes**.

    The Add Authentication (Step 1 of 2) screen appears**.**
13. From the **Choose type of Authentication Mode** options, select **Mutual** for both the **REMOTE** and **LOCAL** trading partner.

    Note: Although it is not enforced, typically the same type of authentication is selected for both the local and remote trading partner.
14. Click **Next**.

    The Add Authentication (Step 2 of 2) screen appears**.** On this screen, you import the server certificate for your remote trading partner.
15. Next to the Server Certificate drop-down list, click **Add Certificate...**.
16. Select **Import certificate from file**.
17. Click **Next**.
18. Click **Add alias...**.

    The **Add New Password Alias** screen appears.
19. In the **Password Alias Name field**, enter any name.
20. Enter and confirm a password to use for this alias.
21. Click **Submit**.
22. Next to the **Import Certificate Location** field, click **Browse**.
23. Navigate to one of the following:

- **servercert1.crt,** if you are configuring **Test_TradingPartner_2**
- **servercert2.crt,** if you are configuring **Test_TradingPartner_1**.

24. Click **Create Certificate**.

    Authentication is added and the View and Edit Service Details page is displayed.

    Note: If there is an error, the Add Authentication page is redisplayed. A message indicating the problem is displayed above the input requiring correction.

25. Make sure that all of your trading partners and your service profiles are enabled.

You have successfully created a service and its service profile. To learn more about services and service profiles, see Trading Partner Management.

This concludes this sample. To test your sample you simply run the two processes that you created in Before You Begin. Before you test your sample, you may want to review Testing Tips.

Testing Tips

Before you test your sample, you can use the keytool utility to make sure that your keystore entries are properly configured. If you want to test that your encryption works properly, you can use the Trace Raw Messages option to save the raw messages that are sent to a specific location in your file system.

Note: When you first set up connections to RosettaNet trading partners, it is a good idea to run your configuration in Test mode to take advantage of the additional debugging features provided by this mode. To run your Web Logic Integration RosettaNet configurations in Test mode, you specify two annotations in the `setProperties` method:

-Set `global-usage-code` to `Test`.

-Set `debug-mode` to `true`.

For more information about the `setProperites` method, see RosettaNet Control Interface.

This section contains the following testing tips:

- Listing the Keystore Content
- Enabling the Trace Raw Messages Option

Listing the Keystore Content

You can use the JDK keytool utility to make sure that your key store entries are properly configured. The keystore used in this sample, is the demo keystore, `DemoIdentity.jks`, which is installed automatically when you install your product.

1. At the command line prompt, navigate to *`BEA_HOME`*`\wlserver_10.0\server\lib\` where *`BEA_HOME`* is the directory in which you installed WebLogic Server
2. Enter the following keytool command: `keytool -list -keystore DemoIdentity.jks -storepass DemoIdentityKeyStorePassPhrase`

The resulting list should look similar to the following:

```
Keystore type: jks
Keystore provider: SUN

Your keystore contains 9 entries:

secdomain1-client, Sat Oct 18 15:12:15 PDT 2003, keyEntry,
Certificate fingerprint (MD5):
53:13:78:D4:D0:E1:0D:EA:F4:6F:A1:19:6F:BE:4B:AF
secdomain2-sig, Sat Oct 18 16:44:00 PDT 2003, trustedCertEntry,
Certificate fingerprint (MD5):
97:24:2D:B5:CC:F3:FF:E5:06:E0:BD:CC:B6:E2:EF:E6
secdomain1servcert, Sat Oct 18 17:07:30 PDT 2003, trustedCertEntry,
Certificate fingerprint (MD5):
0F:AB:D0:92:0E:28:20:2C:70:4B:54:3E:84:AC:7F:E7
secdomain1-sig, Sat Oct 18 15:10:11 PDT 2003, keyEntry,
Certificate fingerprint (MD5):
AD:9F:BA:80:44:F2:7D:54:65:2C:7B:86:8B:2F:AA:D7
secdomain2-enc, Sat Oct 18 16:44:00 PDT 2003, trustedCertEntry,
Certificate fingerprint (MD5):
3E:6C:A9:E8:5E:03:51:80:AD:6A:76:41:44:76:37:7B
secdomain2servcert, Sat Oct 18 16:46:15 PDT 2003, trustedCertEntry,
Certificate fingerprint (MD5):
0F:AB:D0:92:0E:28:20:2C:70:4B:54:3E:84:AC:7F:E7
secdomain1-enc, Sat Oct 18 15:10:47 PDT 2003, keyEntry,
Certificate fingerprint (MD5):
51:AB:CD:71:A2:E9:26:C8:CC:B2:A8:4C:49:DB:F1:CA
secdomain2-client, Sat Oct 18 16:43:59 PDT 2003, trustedCertEntry,
Certificate fingerprint (MD5):
F9:FA:43:6E:DE:00:FB:FB:D5:68:EF:F6:2A:77:FD:01
demoidentity, Sat Oct 18 13:25:12 PDT 2003, keyEntry,
Certificate fingerprint (MD5):
0F:AB:D0:92:0E:28:20:2C:70:4B:54:3E:84:AC:7F:E7
```

Enabling the Trace Raw Messages Option

The Trace Raw Messages option enables you to save raw messages in a specified location. You can then open these messages to review your security settings, such as encryption and signatures.

1. In the WebLogic Integration Administration Console, navigate to **Trading Partner Management** > **Configuration.**

   The General Configuration screen appears.

2. Specify the **Message Tracking Level** and **Mode** by selecting from the drop-down menu.

3. In the **Directory** field, enter the path to the folder which you want the raw messages to be written to.

4. From the **Trace Raw Messages** options, select **Yes**.

5. Click **Submit**.

After you have run your processes, check the specified directory for any raw messages.

Index

R

S

T

U

V

W

X