BEA WebLogic Collaborate

Getting Started

BEA WebLogic Collaborate 1.0
Document Edition 1.0
December 2000

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA WebLogic Collaborate Getting Started

Document Edition	Date	Software Version
1.0	December 2000	1.0

Contents

4. Using Logic Plug-Ins for Billing

Index

About This Document

This document explains how to get started with the BEA WebLogic Collaborate™ software.

This document is organized as follows:

- Chapter 1, “Overview,” provides an architectural overview of the BEA WebLogic Collaborate software and explains some basic concepts.
- Chapter 2, “Setting Up the WebLogic Process Integrator Environment,” provides a set of procedures you need to complete to set up your BEA WebLogic Collaborate environment after you have installed it.
- Chapter 3, “Running the WebLogic Process Integrator Verifier Example,” describes how to configure your WebLogic Collaborate environment for the WebLogic Process Integrator-based wlpiverifier application provided with the BEA WebLogic Collaborate package and describes how to run that application.
- Chapter 4, “Using Logic Plug-Ins for Billing,” describes how to build and run the logic plug-in sample applications provided with the WebLogic Collaborate software.

What You Need to Know

This document is for managers, system administrators, and programmers who are interested in understanding the architectural requirements for implementing and administering an e-market based on WebLogic Collaborate. This document assumes you have a working knowledge the BEA WebLogic Server™ system, XML, Enterprise JavaBeans, and Java programming.

e-docs Web Site

The BEA WebLogic Collaborate product documentation will be available on the BEA Systems, Inc. corporate Web site. From the BEA Home page, click Product Documentation or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the BEA WebLogic Collaborate documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Enterprise documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have Adobe Acrobat Reader installed, you can download it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

For more information about WebLogic Server, XML, or Java 2 Enterprise Edition (J2EE), see the *Bibliography* in the BEA WebLogic Collaborate online documentation.

Contact Us!

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

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Collaborate 1.0 release.

If you have any questions about this version of BEA WebLogic Collaborate, or if you have problems installing and running BEA WebLogic Collaborate, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

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

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

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

Convention	Item

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Overview

The BEA WebLogic Collaborate™ product is an XML- and Java-based open market electronic commerce platform that enables you to implement business-to-business (B2B) e-commerce systems on the Web. It helps you quickly deploy B2B e-commerce systems that link existing back-end applications, databases, and customers into automatic and flexible electronic collaborations.

WebLogic Collaborate is a software framework and a set of services built on top of the BEA WebLogic Server™ system. WebLogic Collaborate is implemented entirely in Java and leverages the J2EE blueprint and standard APIs. Central to WebLogic Collaborate is XML, which provides an open data interchange format between loosely coupled participants. WebLogic Collaborate supports HTTP because the World Wide Web is the ubiquitous communication medium where the majority of e-business gets conducted.

WebLogic Collaborate simplifies and enables the building of portals, Application Service Provider (ASP) businesses, and integrated businesses on the Web.

The following sections provide a product overview:

- Feature Support for Your E-Business
- Understanding the Architectural Requirements of an E-Market
- Using WebLogic Collaborate — the End-to-End View
- Documentation Roadmap

Feature Support for Your E-Business

The goal of WebLogic Collaborate is to provide B2B features for building mission-critical, scalable, real-world e-market collaboration and trading exchanges, including:

- Robust support for secure, high-volume business transaction levels based on BEA's award winning WebLogic Application Server™ technology. BEA's proven, high availability, 24x7-managed server technology, with dynamic load-balancing, multithreading, and fail-over without processing interruption, has delivered industry-leading results in thousands of the world's most demanding application environments.
- Reliable, role-based XML messaging that supports enhanced send and receive capabilities, including support for large messages, based on any combination of sender, receiver, and content filtering.
- Dynamic market management, allowing you to configure e-markets, trading partners, and collaborations on-the-fly.
- An SSL-based secure platform for conducting collaborations that includes mutual (two-way) authentication using digital certificates among trading partners.
- A pluggable architecture that supports multiple, standards-based business protocols, business rules, and market applications.
- Conversation coordination to manage the execution and interaction of two or more trading partners within the conversation and to manage conversation life cycles.
- An open, nonproprietary architecture that leverages Java, J2EE XML, HTTP, HTTPS, and other emerging industry standards to allow rapid system and cross-platform integration, with low barriers to entry and "plug and play" with XML-ready, best-of-breed application software.
- Support for user-defined business protocols and vocabularies, allowing you to meet custom e-market requirements and obtain competitive advantage.
- A data repository and a set of design and configuration tools to define and manage the metadata and conversation definitions of WebLogic Collaborate.

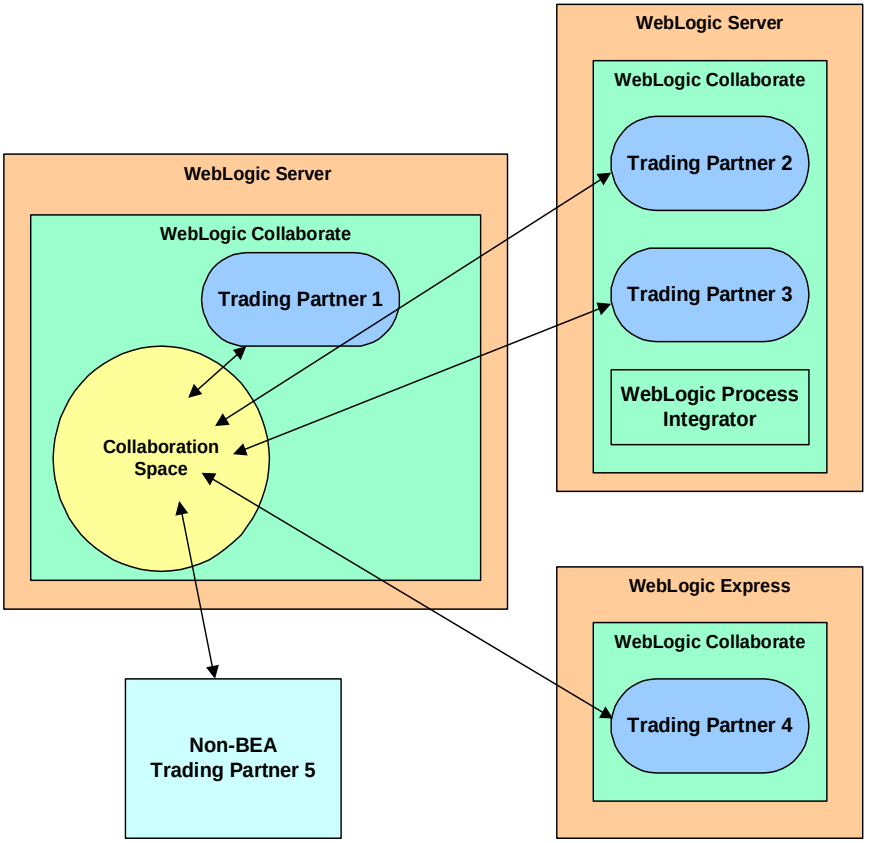
- Integration with BEA WebLogic Process Integrator, the BEA workflow automation tool, which is packaged with WebLogic Collaborate.

WebLogic Collaborate also has a number of features inherited directly from WebLogic Server, including:

- The ability to build custom portals to e-markets by leveraging BEA WebLogic Personalization Server™ components
- A portal backbone
- Controlled and secure Web access to existing business data and applications
- Support for existing applications based on CORBA, EJB, Tuxedo®, and COM+

Figure 1-1 shows a high-level topology of an example e-market based on WebLogic Collaborate.

Figure 1-1 Topology of an Example WebLogic Collaborate-based E-Market



In the preceding figure, note the following

- Trading partners connect to a WebLogic Collaborate entity called the collaboration space, or c-space, which organizes the business exchanges among trading partners. C-spaces are described in “Collaboration Spaces” on page 1-13.
- A trading partner may be colocated on the machine that hosts the c-space, or a trading partner may be remote at a site behind its own firewall.
- Multiple trading partners may be colocated on the same WebLogic Server instance.

- Trading partners may be set up on top of a WebLogic Server-based platform, on a WebLogic Express-based platform, or on a platform that has no BEA-provided software.
- Trading partners that are based on WebLogic Server can also use the WebLogic Process Integrator software to integrate workflows into their exchanges with other trading partners.

Understanding the Architectural Requirements of an E-Market

The following sections describe the architectural requirements of an e-market:

- Trading Partners
- Business Processes and Vocabularies
- Conversations and Roles
- Business Messages
- Collaboration Spaces
- Conversation Subscriptions
- Software for Implementing the C-Space

Trading Partners

One of the basic building blocks of an e-market is the notion of a trading partner. A trading partner is an e-market participant, such as a company, that joins other trading partners to form a community with a specific business purpose. For example, a typical e-market could be an automobile parts exchange: one trading partner supplies exhaust pipes and another is an auto parts retailer.

In an e-market, a trading partner must have a special identity that defines where it fits with the business purpose of the e-market. In WebLogic Collaborate, the term *trading partner* refers specifically to an entity that is authorized to participate in one or more specific business exchanges, or *conversations*, in a specific *role* that is defined for the e-market. These concepts are explained in later sections.

Business Processes and Vocabularies

One of the first things you do when you create an e-market is to define the vocabulary and business processes of the e-market. This is a key step because when you define the vocabulary and business processes for exchanging information, you establish:

- The kind of information that is exchanged
- The processes that trading partners will follow when exchanging information

For example, in an automobile parts exchange, a trading partner sending a purchase order must include a specific set of data in that purchase request so that the recipient can understand the contents of that purchase order. The identity of each piece of information, which is exchanged between trading partners in XML messages, makes up the vocabulary.

Business processes govern how trading partners behave when they receive certain types of messages or when error conditions arise. Business processes also define what certain kinds of documents, such as a bid request or a purchase order response document, mean. For example, a business process may state what kind of responses a trading partner can send when the trading partner receivs a purchase order.

Once you have defined the e-market vocabulary and business processes, you must choose:

- A business protocol appropriate for the e-market — for example, the eXtensible Open Collaboration Protocol (XOCP) or RosettaNet. While RosettaNet specifies both the business process and the protocol, XOCP refines the business process that governs the exchange of business information between trading partners. The business protocol specifies 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.
- A set of Document Type Definitions (DTDs) that define the grammar and syntax of the XML messages sent among trading partners

Conversations and Roles

A preliminary outcome of having defined the vocabulary, business processes, business protocol, and DTDs for the e-market is the outline of the *conversations* that take place between trading partners.

In WebLogic Collaborate, a conversation:

- Consists of a predefined set of message exchanges. Each message may cause any number of back-end system transactions to occur when it arrives at a trading partner's site.
- May be complex and long-running, or short-lived.
- Has an ID, which is used by the system. The conversation ID can also be used by the trading partner application to retrieve conversation-specific data.

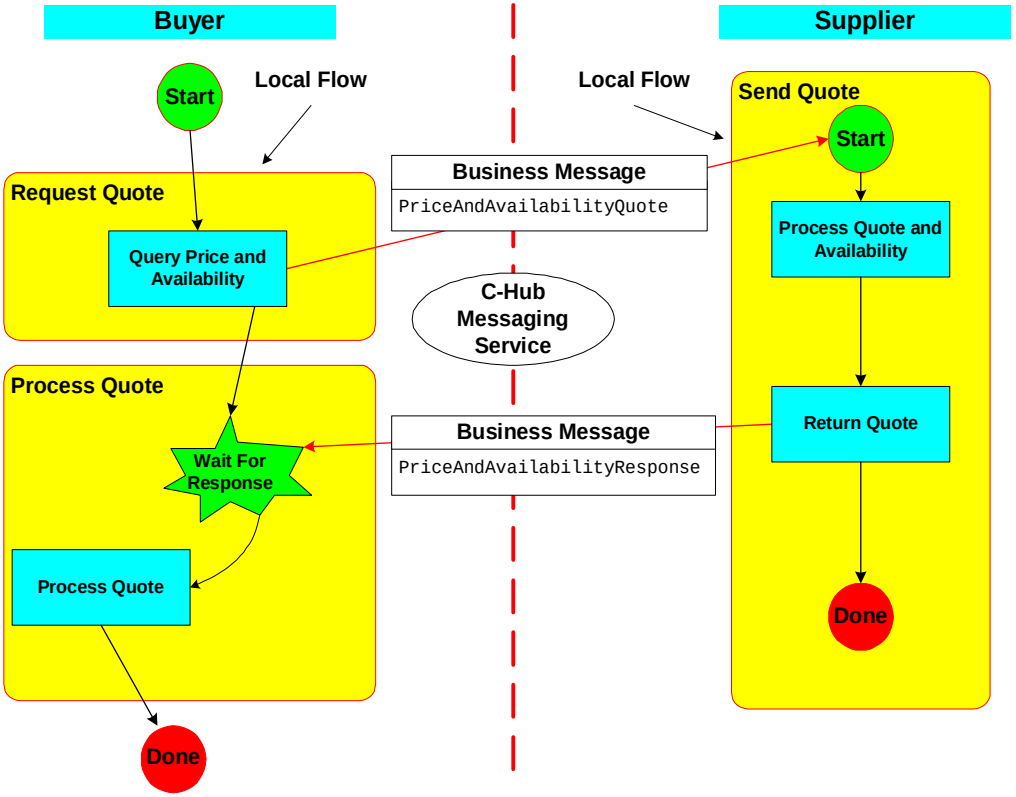
A given conversation specifies the types of messages that can be changed between two or more trading partners for a specific business process. The kinds of messages a *specific* trading partner may send or receive is a *conversation role*.

The final outcome of specifying the details of each conversation in an e-market is the *conversation definition*. A conversation definition specifies:

- A unique conversation definition name
- A conversation definition version
- The DTDs, or schema, for the business messages exchanged in the conversation
- Roles in the conversation

Each trading partner who participates in a conversation in a given role — for example, supplier, as shown in the following figure — must implement its local view of the conversation. Although this local view is partner-specific, to enable the trading partner to participate in the conversation as specified by the conversation definition, the view must encapsulate the processes required to handle the right business messages at the right time.

The following figure shows a graphical representation of a simple conversation and possible local flows for the two roles.



In this figure, note the following:

- The business messages `PriceAndAvailabilityQuote` and `PriceAndAvailabilityResponse`.
- The roles Buyer and Supplier. The implication of being in a role in a given conversation is that you send and receive only the business messages defined for your role. For example, the Buyer:
  - Starts the conversation
  - Sends the business message `PriceAndAvailabilityQuote`
  - Receives the business message `PriceAndAvailabilityResponse` and processes it

By contrast, the Supplier:

- Receives and processes the business message `PriceAndAvailabilityQuote`
- Sends the business message `PriceAndAvailabilityResponse`

- The local flows. Each role has its own set of local processes required to send and receive the right business messages at the right times.

Business Messages

A business message is the basic unit of communication among trading partners. A business message contains one or more XML business documents, one or more attachments, or a combination of both. Business messages, which are expressed in XML and are communicated in a WebLogic Collaborate-based c-space, also have other information depending on the business protocol — for example, XOCP or RosettaNet — chosen for the conversation. Business messages are exchanged as part of a conversation. “Business Protocols” on page 1-9 provides more detail about this additional, protocol-specific information.

Business Protocols

As mentioned earlier, the business protocol is a further refinement of the business process that governs the exchange of business information between trading partners. WebLogic Collaborate Version 1.0 supports two protocols:

- RosettaNet 1.1
- eXtensible Open Collaboration Protocol (XOCP), a BEA-specific protocol

WebLogic Collaborate provides a pluggable architecture to support multiple protocols, providing message routing and filtering capabilities that are customized especially for each protocol. A c-space can support multiple protocols, giving you a great deal of flexibility in organizing an e-market by reducing the need for trading partners to standardize on any single protocol. Support for additional protocols will be available in future releases of WebLogic Collaborate.

RosettaNet

From the RosettaNet Web site (http://www.rosettanet.org): RosettaNet is an independent, self-funded, non-profit consortium of major information technology, electronic components, and semiconductor manufacturing companies working to create and implement industry-wide e-business process standards. These standards form a common e-business language, aligning processes between supply chain partners on a global basis.

RosettaNet provides e-business Partner Interface Process (PIP) specifications, which define business processes between supply-chain partners, providing the models and documents for the implementation of standards.

PIPs fit into six clusters, or groups of core business processes, that represent the backbone of the supply chain. Each cluster is broken down into segments, which are cross-enterprise processes involving more than one type of supply chain partner. Within each segment are individual PIPs.

PIPs are specialized system-to-system XML-based dialogs that define business processes between supply chain partners. 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 an XML Message Guideline document.

RosettaNet is a point-to-point messaging protocol: business messages are sent between only two trading partners.

XOCP

The XOCP protocol, like RosettaNet, is also designed for deploying peer-to-peer and supply-chain market applications among suppliers, manufacturers, and distributors. However, XOCP also provides the following messaging characteristics:

- Message multicasting

  Message multicasting is the ability of the messaging service to broadcast a message from one trading partner to all conversation participants who are registered as recipients of that message. For example, if 15 trading partners are registered in the role of “supplier” in the `ProcessPurchaseOrder` conversation, then all messages received by the c-space messaging service that are meant to be sent to the “supplier” will be sent to all those trading partners (within the constraints of message routing and filtering, described later).

- Message payload definition independence

  XOCP provides the flexibility for you to specify both the vocabulary and business processes for your messages for a given conversation so that they are an exact fit for your business requirements.

- Conversation life cycle management

  XOCP is designed to manage long-lived conversations. When a conversation terminates, all trading partners who are enlisted in that conversation receive an end-of-conversation message.

- Qualities of Service (QoS) capabilities

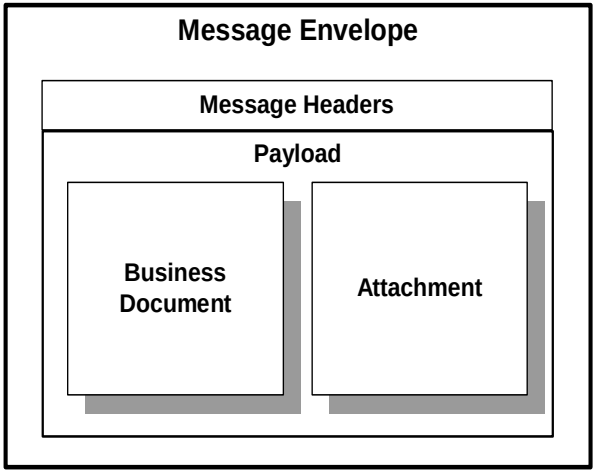
  The WebLogic Collaborate software offers a variety of settings related to QoS that allow you to set and control characteristics of XOCP messages sent in conversations, such as:

  - Message durability — Specify whether a durable message store is to be used to guarantee the delivery of messages in the case of network, machine, or software failures.
  - Timeout — Specify how long a trading partner application will wait before terminating all processing related to the business message.
  - Retry attempts — Control how many times a message should be resent in the presence of specific situations, such as timeouts, network failures, and so on.
  - Correlation ID — Set an additional business message property that can be used to correlate messages in a conversation.

  WebLogic Collaborate allows you to establish QoS settings on a per-conversation and per-message basis.

XOCP Business Messages

Business messages exchanged in a conversation based on the XOCP protocol have the structure shown in the following figure. Understanding the structure of XOCP business messages is especially important for writing WebLogic Collaborate-based XOCP applications, such as trading partner applications, or special c-hub-located applications called logic plug-ins, described later.



Note the following parts:

- Message envelope — This is a logical container for the XOCP business message that is added while the business message is in transit through the WebLogic Collaborate messaging service software. The message envelope typically contains data related to the sender of the message and the recipients, and can contain other metadata. The envelope is visible in the messaging service only. The envelope is not part of the transport protocol.
- Message headers that include:
  - Transport and Qualities of Service (QoS) data, such as sender and recipient identities, content length and type, and status code. This metadata is typically used and modified while the XOCP business message is in transit through the c-hub.
  - Conversation and routing headers, such as sender information, list of message recipients, conversation information, message ID, creation timestamp, and other data. Message attributes can be set by the originating trading partner, or by logic plug-ins in the c-hub, such as the XOCP router or XOCP filter.
- Payload, which includes one or more business documents, one or more attachments, or a combination of both. Note that a business document is expressed in XML, and an attachment is a binary (non-XML) part of the

message. The business documents, with their attachments, constitute the kernel containing the information being exchanged among trading partners.

When a trading partner application or a WebLogic Process Integrator workflow prepares a message payload to be sent to one or more trading partners, that application or workflow includes a process that adds the XOCP headers to the XOCP business message.

An XOCP message is a MIME multipart message. All its components are expressed in XML, except for the non-XML attachments.

Collaboration Spaces

In its simplest sense, a collaboration space, or *c-space*, is the implementation of an e-market using WebLogic Collaborate. A c-space is an abstraction that provides the structure for organizing and administering conversations among trading partners.

A c-space supports:

- A single business model
- A business message vocabulary
- A set of business processes
- One or more business protocols
- A registered set of trading partners

In summary, a c-space provides the administration capability, conversation coordination, and underlying messaging services that are required to create a dynamic business-to-business integration environment. A WebLogic Collaborate-based e-market can support any number of c-spaces on the Web concurrently and any number of trading partners.

Conversation Subscriptions

When a trading partner wants to participate in a c-space, that trading partner subscribes to specific roles in specific conversation definitions. This establishes the relationship of the trading partner to the c-space. For example, an auto parts manufacturer can participate in a c-space by subscribing to the role of "supplier" in the `ProcessPurchaseRequest` conversation.

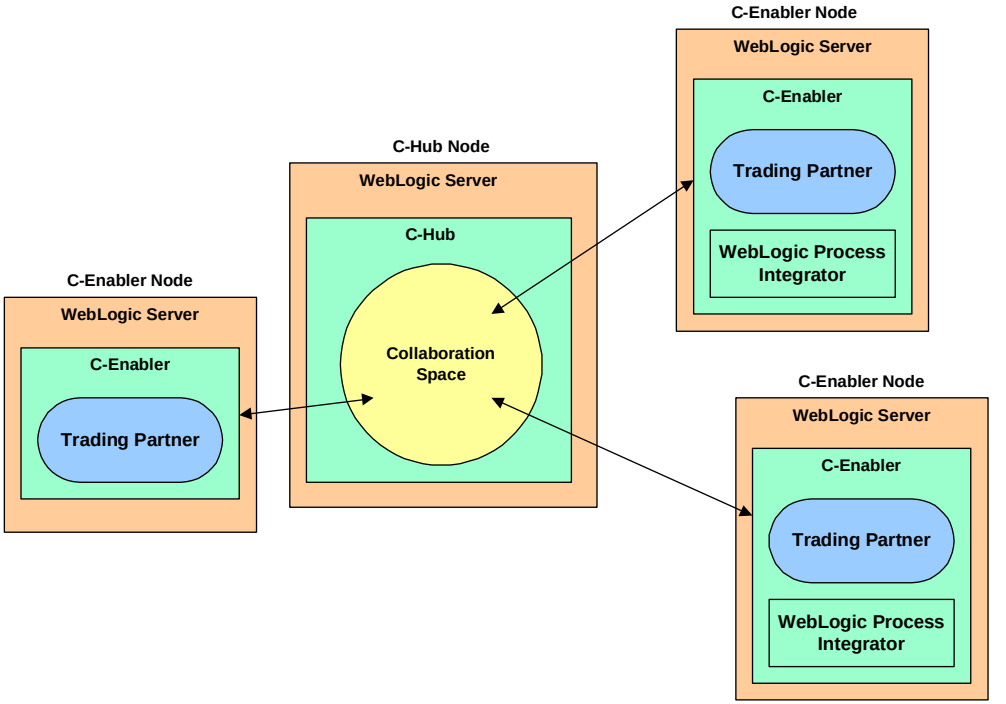
Software for Implementing the C-Space

WebLogic Collaborate provides two primary pieces of software for implementing a c-space:

- The collaboration hub, or *c-hub*, which hosts the c-space and thus serves as the transportation utility for sending messages among trading partners
- The collaboration enabler, or *c-enabler*, which exists at each trading partner's site and which allows a trading partner to participate in predefined c-spaces and possibly with multiple c-hubs

Optionally, a c-enabler can be integrated with WebLogic Process Integrator to provide better conversation flow handling. As mentioned earlier, WebLogic Process Integrator is bundled with WebLogic Collaborate.

The following figure shows the WebLogic Collaborate software deployed in an e-market. Each machine that hosts a c-enabler is referred to as a *c-enabler node*. The machine hosting the c-hub is a *c-hub node*.



The c-hub and c-enabler work together to enable the exchange of business messages among trading partners by providing the following services:

- Messaging Service
- Conversation Coordination Service
- Administration Services
- Security Services
- XML Services
- Logging Service

In addition to these services, the c-hub also includes a repository service, and the c-enabler can be integrated with WebLogic Process Integrator.

The sections that follow introduce each of these services.

Messaging Service

WebLogic Collaborate provides a flexible messaging service to facilitate information transfer between trading partners. The messaging component relies on decoupled, deferred synchronous messaging capabilities to allow communication flexibility between trading partners.

The WebLogic Collaborate messaging service has the following features and characteristics:

- Supports multiple XML-based business protocols natively; for example, the eXtensible Open Collaboration Protocol (XOCP) and RosettaNet
- Supports blocking and non-blocking message delivery
- On the c-hub:
  - Provides routing and filtering capabilities, controlling who receives messages
  - Accommodates the insertion of user-written code at key points to customize the handling of incoming or outgoing messages or to perform messaging-related operations
- For XOCP-based messages, provides Quality of Service (QoS) capabilities to establish durability, confirmation of receipt, message tracking, message timeout, retry, and other settings for messages sent among trading partners

Conversation Coordination Service

Conversations can be complex and long running; in fact, some business conversations may last years. Conversations always have life cycles, and they are explicitly demarcated by a beginning and an end. The WebLogic Collaborate software provides a service, called the conversation coordination service, that coordinates trading partners who are participating in conversations around these life cycle events.

The WebLogic Collaborate conversation coordination service is protocol-specific. In the case of XOCP-based conversations, the conversation coordinator in the c-hub does the following:

- Creates the conversation on behalf of the trading partner that initiates the conversation

- Enlists new trading partners into the conversation
- Delists trading partners that choose to leave the conversation
- Sends conversation termination notices to all conversation participants when the conversation is terminated

Also in the case of XOCP-based conversations, the conversation coordinator in the c-enabler keeps track of the following:

- Conversation handlers for the different conversation types
- All the conversations currently active in the c-enabler

Repository Service

The c-hub includes a repository service to store data required to run the c-space and manage trading partners, including:

- Conversation definitions
- Trading partner role subscriptions
- Document definition (DTD) names
- Document and configuration information
- Extended properties of trading partners
- Data used for message routing and filtering operations

In addition, the repository:

- Is accessible through the C-Hub Administration Console for administration and monitoring functions. The repository allows you to plug in a variety of JDBC RDBMs.
- Provides Oracle, Microsoft SQL Server, or Cloudscape as a primary store
- Can be loaded with the Bulk Loader utility

Administration Services

The WebLogic Collaborate administration services perform multiple configuration and system management functions, including:

- Providing the entry point for trading partners to join a c-space
- Providing a means to configure, administer, and monitor the c-space, conversations, trading partners, and more

Through the administration service, the c-space administrator can configure and manage trading partners (create IDs, grant and revoke roles), and both c-hub administrators and trading partners can browse system status (conversations, message delivery, and so on). In addition, trading partners can use the administration service to start and end c-enabler sessions and end conversations.

Access to Administration Services

WebLogic Collaborate provides two means of access to the administration services:

- For configuring and monitoring, WebLogic Collaborate provides two Web-based administration consoles:
  - C-Hub Administration Console, used by the c-hub administrator to configure and monitor the c-hub
  - C-Enabler Administration Console, used primarily by the trading partner as a monitoring tool, although it is also used to start and stop c-enabler sessions
- For monitoring use only, WebLogic Collaborate also supports user-written applications that use the Java Management Extensions (JMX) Management Beans (or MBeans) to view various data and statistics maintained by the administration service, particularly data about the c-space, conversations, and trading partners.

  Trading partners can use the JMX Mbeans through an Mbean server, which is included in the administration services package, to send and receive messages to and from the administration service.

  Note: The version of the MBean server provided with WebLogic Collaborate 1.0 supports only local access.

Security Services

WebLogic Collaborate security is built on top of the security features provided by WebLogic Server. WebLogic Collaborate implements a role-based authorization scheme in the c-hub in which trading partners apply for roles in conversation types.

The key security features are:

- Mutual, two-way authentication using digital certificates

  WebLogic Collaborate authenticates the identities of trading partners, before the trading partners can access services or participate in dynamic collaboration and the trading partner's c-enabler and the c-hub server authenticate each other.

- Authorization control for WebLogic Server and WebLogic Collaborate resources

  WebLogic Collaborate conversations and business messages delivered by the messaging service are available only to an authorized set of trading partners. The authorized set of trading partners is specified per conversation type by the c-hub administrator, which is done by defining roles and specifying which trading partners are in the role.

- SSL communication among the WebLogic Collaborate trading partners, human users, and the c-hub

- Privacy

  Messages sent and received by and through WebLogic Collaborate are only seen by the appropriate set of trading partners.

Trading partner information is captured and stored by the c-hub repository. The basic information captured includes conversations, roles, URLs, and so on.

The c-hub security mechanism leverages from WebLogic Server to perform the following tasks:

- Map digital certificates to valid trading partners, using either the email addresses contained in the certificate or the fingerprint. (WebLogic Collaborate provides a default authenticator to perform the mapping.)

- Specify root Certification Authorities (CAs) and other WebLogic Server properties.

You can read more information about the tasks associated with setting up and administering the WebLogic Collaborate security:

- For a comprehensive discussion about WebLogic Collaborate security, and specifically c-hub security, see Configuring Security in the *BEA WebLogic Collaborate C-Hub Administration Guide*.

- For c-enabler security, see the topic Configuring C-Enabler Security in the *BEA WebLogic Collaborate C-Enabler Administration Guide*.

XML Services

The WebLogic Collaborate c-hub provides the following bundled XML services from the Apache Software Foundation:

- Xerces — an XML parser
- Xalan — an XSLT engine

For more information about these services, go to the Apache Software Foundation Web site at http://www.apache.org.

Logging Service

In both the c-hub and c-enabler, WebLogic Collaborate provides a local logging capability for error and information messages, which allows for data stores (for example, files). In the WebLogic Collaborate environment, all messages are timestamped. Logging is maintained as set of log messages, which can be sent to the WebLogic Server log, or to a separate log file.

Workflow Process Engine

A workflow process engine is not a mandatory architectural requirement for an e-market, but it can provide an extremely useful service when integrated with a trading partner's platform to control the execution of local business processes. The WebLogic Collaborate software includes the WebLogic Process Integrator product.

WebLogic Process Integrator is a process engine tool suite that trading partners can integrate with their market and back-end enterprise applications to manage complex e-business processes with multiple trading partners. The overall management of these distributed processes are part of WebLogic Collaborate conversation coordination.

Key features of the WebLogic Process Integrator software make it a powerful addition to WebLogic Collaborate:

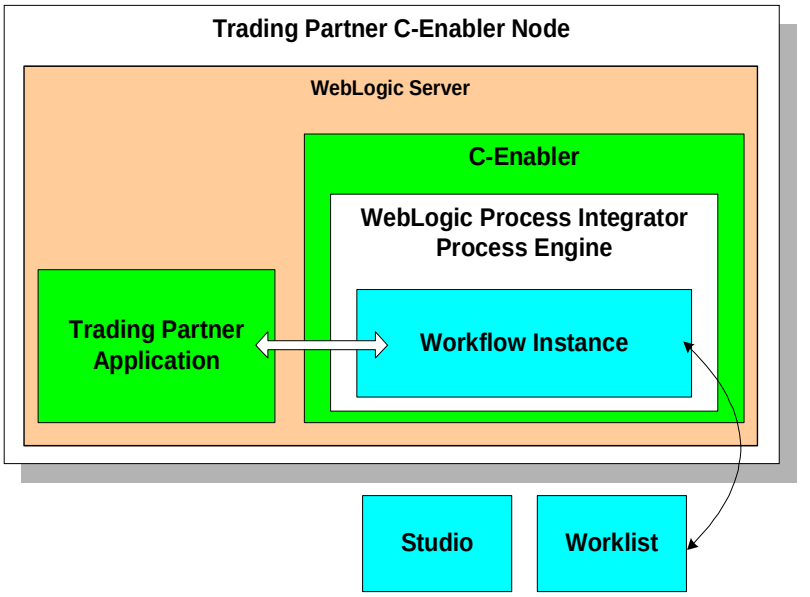
- A design tool, called Studio, that a trading partner can use to graphically model business processes, including participation in the conversation roles in which the trading partner is registered. The WebLogic Process Integrator software then provides a means to link an instance of that model, called a workflow, to the trading partner's c-enabler configuration and execute the workflow. You can also use Studio to configure, administer, debug, and monitor workflow instances.

- Integration with back-end application by means of business operations defined by the trading partner.
- You use Worklist, a convenient user interface that defines and directs e-business process exceptions to human users for resolution.

Note: WebLogic Process Integrator is available in two versions: one that ships separately (version 1.2), and one that is bundled with WebLogic Collaborate (version 1.2C). WebLogic Process Integrator 1.2C provides all the functionality of 1.2, but also provides additional integration features with WebLogic Collaborate, including:

- Specialized workflow properties for manipulating business messages, specifying the message delivery Quality of Service, handling message tokens, and so on
- A Java application programming interface (API) for use in WebLogic Collaborate applications

The following figure shows a typical trading partner configuration that integrates the WebLogic Process Integrator software with WebLogic Collaborate.



For more information about WebLogic Process Integrator integration with WebLogic Collaborate, see Using Workflows to Exchange Business Messages in the *BEA WebLogic Collaborate Developer Guide*.

For more information about WebLogic Process Integrator, see the following documents:

- *BEA WebLogic Process Integrator Studio User Guide*
- *BEA WebLogic Process Integrator Worklist Guide*
- *BEA WebLogic Process Integrator Tutorial*

Using WebLogic Collaborate — the End-to-End View

This section provides more details about the WebLogic Collaborate architecture and features by showing a step-by-step, end-to-end view of:

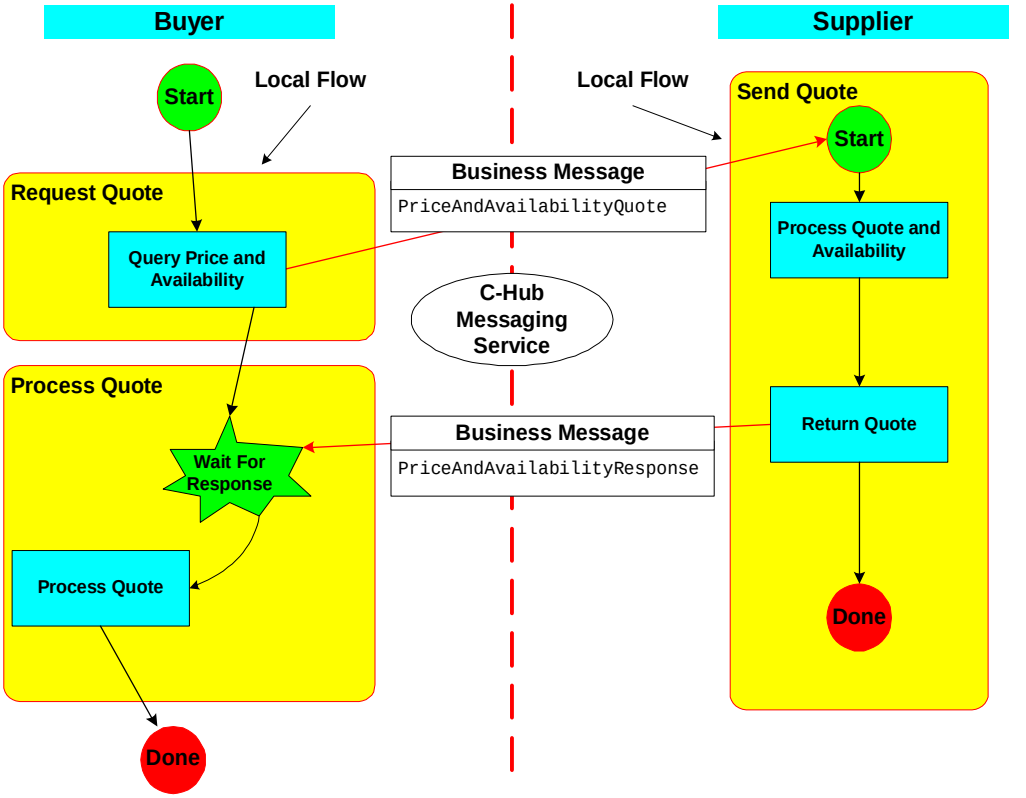
- The procedure for setting up the components that make a c-space work, which shows what you need to do
- An exchange of business messages among trading partners registered for a conversation in a c-space, which shows how WebLogic Collaborate works

The procedures described in this section include:

- Registering Trading Partners for the C-Space
- Creating the Trading Partner Application
- Configuring the C-Hub
- Configuring the C-Enabler
- Starting the Conversation
- Broadcasting a Message to Trading Partners

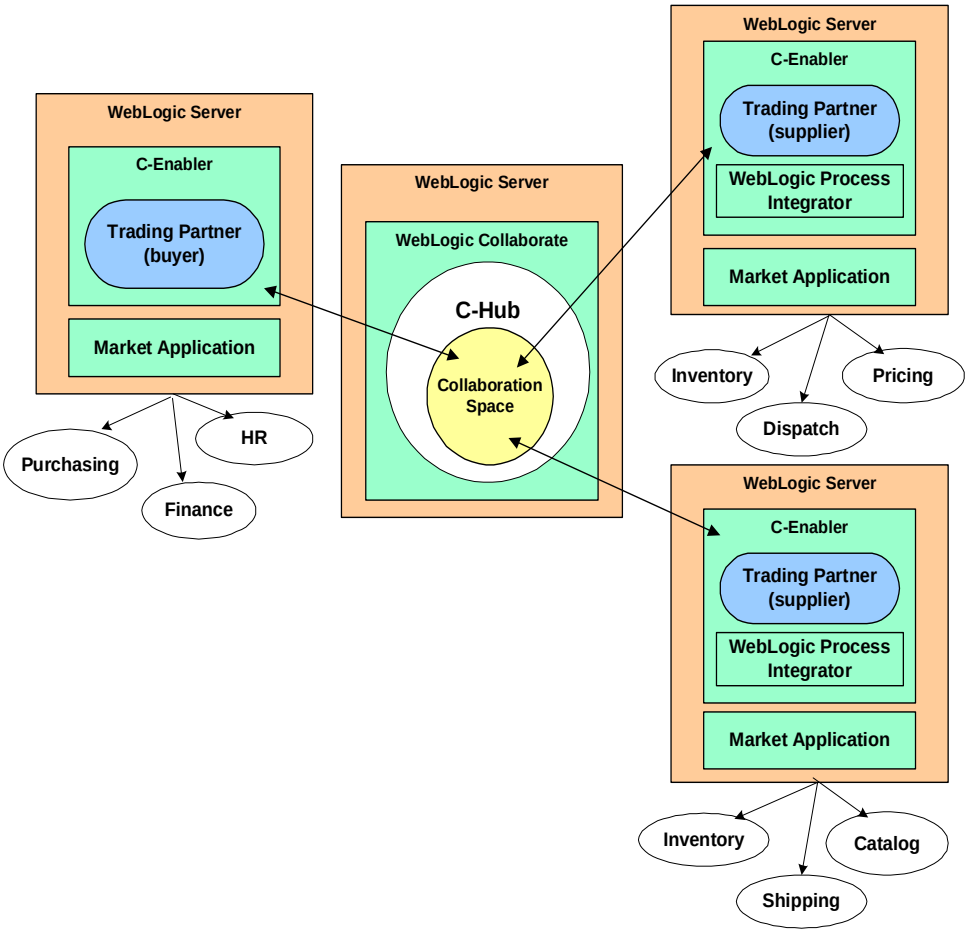
- Receiving a Business Message and Responding Through WebLogic Process Integrator
- The C-Hub Sending a Response to the Buyer

The procedures described in this section focus on three trading partners: one has an application that invokes the c-enabler directly by way of the C-Enabler API to send and receive XOCP business messages. The other trading partners' configurations use WebLogic Process Integrator to process business messages received and sent through their c-enabler. The conversation used in this example is described in "Conversations and Roles" on page 1-7, and a figure showing a representation of this conversation is repeated as follows.



The following figure shows a static, high-level view of the WebLogic Collaborate environment in which the conversation takes place. Note the following:

- The buyer is shown on the left. The buyer's IT environment includes its Finance, HR, and Purchasing enterprise applications that are connected to the underlying WebLogic Server platform. (Note that the buyer's configuration does not include WebLogic Process Integrator. The use of WebLogic Process Integrator is optional. The configuration shown in this figure is hypothetical and could potentially have a number of different types of platform configurations.)
- The suppliers are shown on the right. Each supplier is configured with its own set of enterprise applications and WebLogic Process Integrator.



Registering Trading Partners for the C-Space

After the conversation definitions for a c-space have been created, the c-hub administrator can register trading partners in the c-space. Registering a trading partner in a c-space requires the following:

1. The c-hub administrator and the trading partner need a means to communicate, whether via email, a solicitation on a Web page, or whatever is appropriate for the business purpose of the c-space, to match a trading partner to a role in a conversation definition.
2. The c-hub administrator uses the C-Hub Administration Console to *subscribe* the trading partner to one or more specific roles in one or more conversation definitions.
3. The trading partner needs the c-enabler software. The trading partner can download the c-enabler from the BEA Web site, or from the c-hub administrator. If the trading partner platform includes WebLogic Process Integrator, the software sent by the c-hub administrator typically includes the local WebLogic Process Integrator workflow template definitions for each role to which the trading partner is subscribed. The trading partner also needs to have the software and business processes required to implement its roles.

In addition to completing the preceding steps, each trading partner also needs to have an application that:

- Interacts with the c-enabler
- Implements the processes defined in their local flows

The application may be provided by the c-hub administrator and configured by the trading partner, or may be created and configured entirely by the trading partner.

Creating the Trading Partner Application

The trading partner application is the implementation of the local business processes required to participate in the c-space conversations. The requirements for the application vary depending on:

- Whether the trading partner applications invokes the c-enabler directly, via the Enabler API, to participate in a conversation, or is integrated with WebLogic Process Integrator.
- Whether the trading partner's role is to initiate a conversation, or participate in an existing conversation

The trading partner application must implement the following functionality:

- Populating the outgoing business messages with the information required for the conversation
- Extracting information from the incoming business messages that have been received by the c-enabler
- The business process that is needed for the conversation, which typically includes access to back-end applications

About Trading Partner Applications That Invoke the C-Enabler Directly

The c-enabler has an API that provides trading partner applications with access to the following basic capabilities:

- Joining the c-space
- Registering conversation handlers
- Creating conversations
- Sending and receiving messages
- Terminating conversations

The trading partner application can use these APIs to implement the business logic for performing these tasks directly in Java.

For complete details about implementing applications that invoke the c-enabler directly, see Using XOCP C-Enabler Applications to Exchange Business Messages in the *BEA WebLogic Collaborate Developer Guide*.

About Trading Partner Applications Integrated with WebLogic Process Integrator

A trading partner whose c-enabler software is integrated with WebLogic Process Integrator needs to create an application that does the following:

- Invoke the workflow template to start the workflow instance
- Provide data for outgoing messages
- Process incoming messages
- Link to back-end applications, as appropriate

As mentioned earlier, the logic flow of the trading partner application depends on whether the trading partner's role is to *initiate* a conversation or to *participate* in a conversation initiated by someone else. A powerful way to implement a c-space is for the c-hub owner to create workflows that trading partners, who are subscribed to roles as participants in a conversation, can download from the c-hub administrator. These trading partners can then implement the business logic required to connect the workflows to their back-end applications, including other local workflows.

For complete details about using WebLogic Process Integrator to exchange business messages in the WebLogic Collaborate environment, see Using Workflows to Exchange Business Messages in the *BEA WebLogic Collaborate Developer Guide*.

Configuring the C-Hub

Before any trading partner activity can take place, the c-space administrator needs to configure the c-hub. Configuring the c-hub, which runs in a WebLogic Server instance, is a set of tasks in which you configure the following:

- WebLogic Server, to set values specific to the environment in which the c-hub runs
- The c-hub start-up classes
- The c-hub repository, which primarily involves setting up the JDBC connection to the database in which the repository exists
- The Java Message Service (JMS) queue

- The C-Hub Administration Console
- C-space, which includes associating trading partners with conversation definitions
- Security
- Persistence and recovery of business messages
- Business protocols used in the c-spaces

The c-space is the key entity that links trading partners with specific conversation definitions and roles. Each c-space has a URL that trading partners use to access the c-space. The URL also indicates the business protocol for the conversation in which the trading partner's role is subscribed.

For complete information about configuring the c-hub and c-spaces, see the *BEA WebLogic Collaborate C-Hub Administration Guide*.

Configuring the C-Enabler

To participate in conversations coordinated by the c-hub, each trading partner needs to create a c-enabler session between their c-enabler and the c-hub. Each c-enabler session allows the trading partner to exchange business messages with other trading partners in a c-space.

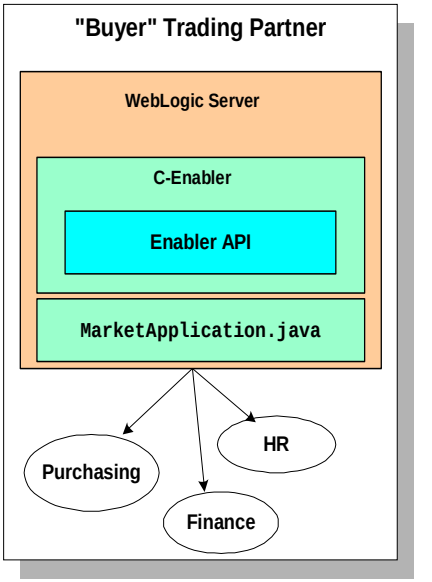
Each c-enabler requires an XML configuration file, which specifies:

- C-enabler name, URL, and session name
- Trading partner name, as registered in the c-space, and security information
- URL of the c-hub with which the c-enabler interacts

For complete information about creating the c-enabler XML configuration file, see the *BEA WebLogic Collaborate C-Enabler Administration Guide*.

Starting the Conversation

The trading partner in the role of buyer begins a conversation. The buyer's configuration is shown in the following figure.



When the buyer's application starts, the following events occur:

1. The market application, shown in this figure as `MarketApplication.java`, invokes the C-Enabler API to start a c-enabler session and a conversation. Because of the buyer's role in the conversation definition, the buyer is a *conversation initiator*. (This has an impact on the code that needs to be implemented in the market application, as opposed to a trading partner in the role of *conversation participant*, described in "Receiving a Business Message and Responding Through WebLogic Process Integrator" on page 1-33.)

   When the trading partner creates an XOCP conversation, the trading partner becomes enlisted in the conversation.

2. The market application constructs an XML business document containing a quote request to be sent to trading partners in the role of "supplier." The business document uses the appropriate DTDs specified in the conversation definition to which the trading partner is subscribed.

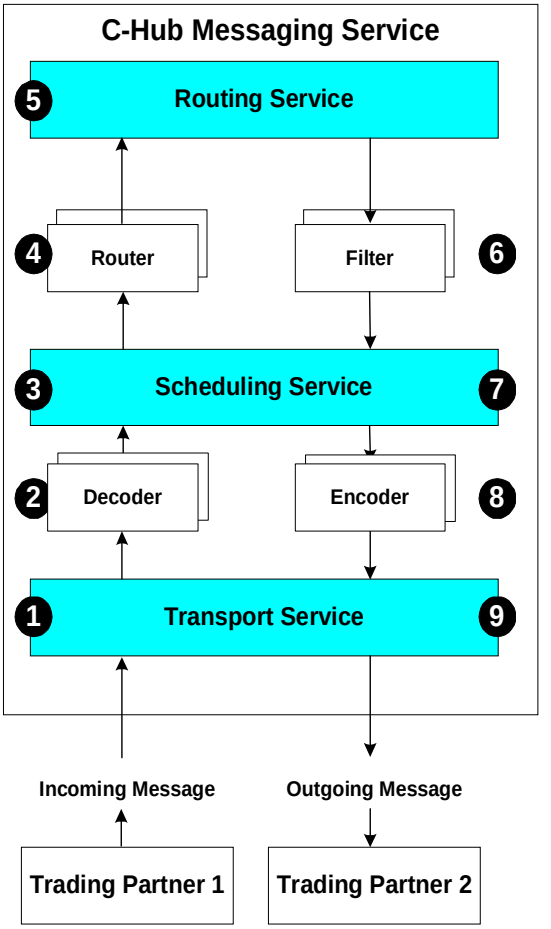
3. The market application invokes the C-Enabler API to create an XOCP business message containing the XML business document created in step 2.
4. The market application sends the XML business message through the transport service to the c-hub, and then waits for a system reply from the c-hub.

   The specific replies that the application may receive depends on the message Qualities of Service (QoS) settings that have been established. For example, with the lowest QoS, there is no reply and the return is immediate. With the highest QoS, the reply contains the list of trading partners to whom the message has been delivered. In the latter case, the system reply arrives only when one of the following events has occurred:

   - All trading partners have received the business message.
   - The message has expired.

Broadcasting a Message to Trading Partners

When the buyer's XOCP business message arrives at the c-hub, the c-hub messaging service executes the sequence shown in the following figure:



The preceding figure shows that the messaging service in the c-hub is made up of three components: the transport service, the scheduling service, the routing service. Among these services are decoder, encoder, router, and filter modules that do specialized processing of the business messages, described in the following sequence:

1. The transport service receives and passes the message to the appropriate decoder module.
2. The decoder:
   - Processes the protocol-specific headers, creating the message envelope

- Identifies the sending trading partner (in this case, the buyer)
- Prepares a system reply to be returned to the buyer, which is returned immediately if this QoS is requested
- Passes the business message to the scheduling service

3. The scheduling service queues the business message for the router.
4. The router processes the router logic plug-ins that are configured for the current conversation definition. In the case of an XOCP business message, at a minimum the BEA-provided XOCP router logic plug-ins are processed. The XOCP router logic plug-in typically consists of a sequence of Xpath expressions that are maintained in the repository. The purpose of an Xpath expression in the XOCP router logic plug-in is to add or remove recipients to the list of trading partners who will receive the message.

   If the incoming business message uses the RosettaNet protocol, BEA provides a RosettaNet router logic plug-in to process routing operations, such as matching the DUNS number of the incoming business message and matching that number to a trading partner identity in the repository and updating the message header.

   Note that the c-hub administrator can add one or more user-written logic plug-ins to the router — for example, to implement a billing function. A series of such plug-ins in the router is called a *router chain*. For information about creating logic plug-ins, see Developing Logic Plug-Ins in the *BEA WebLogic Collaborate Developer Guide*. For information about configuring the router chain, including adding or removing logic plug-ins and specifying Xpath expressions, see Routing and Filtering XOCP Business Messages in the BEA WebLogic Collaborate *C-Hub Administration Guide*.
5. The routing service performs final validation of the message recipients, and queues the message for the filter chain of the selected recipients.
6. The filter processes the filter logic plug-ins that are configured for the current conversation definition. In the case of an XOCP business message, at a minimum the BEA-provided XOCP filter logic plug-ins are processed. As with the router, the XOCP filter logic plug-in typically consists of a sequence of Xpath expressions that are maintained in the repository. The purpose of an Xpath expression in the XOCP filter logic plug-in is to accept or reject a business message intended for a particular recipient.

If the outgoing business message uses the RosettaNet protocol, BEA provides a RosettaNet filter logic plug-in for completeness, although it does not perform any operations.

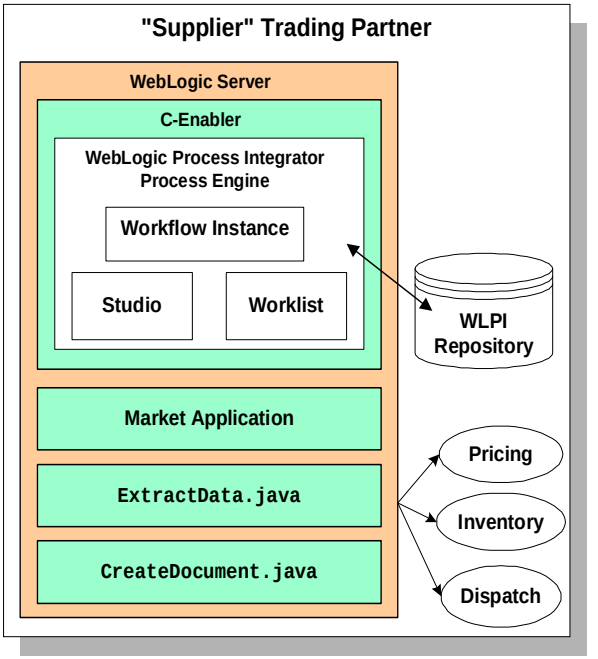
Note that the c-hub administrator can add one or more user-written logic plug-ins to the filter. A series of such plug-ins in the filter is called a *filter chain*. For information about creating logic plug-ins, see Developing Logic Plug-Ins in the *BEA WebLogic Collaborate Developer Guide*. For information about configuring the filter chain, including adding or removing logic plug-ins and specifying Xpath expressions, see Routing and Filtering XOCP Business Messages in the *BEA WebLogic Collaborate C-Hub Administration Guide*.

7. The scheduling service performs any Quality of Service functions on the business message, as required, and passes the business message to the encoder.
8. The encoder prepares the business message for the transport service.
9. The transport service sends the business message onto the recipient trading partners. This causes the following two events to occur:
   a. The recipient trading partners are enlisted in the conversation.
   b. The system reply is sent back to the originating trading partner if the QoS requires it.

Receiving a Business Message and Responding Through WebLogic Process Integrator

In this scenario, the suppliers each have a configuration that includes the WebLogic Process Integrator software, as shown in the following figure. Note that the WebLogic Process Integrator software is not a mandatory component of a trading partner platform, but it is shown in this figure as an example.

Regardless of whether a trading partner is configured with WebLogic Process Integrator, in order to be invoked in conversations, the c-enabler for a trading partner must be running, have joined the c-space, and be registered for the conversation. (Note that “The C-Hub Sending a Response to the Buyer” on page 1-35 describes a scenario in which a c-enabler that does *not* use WebLogic Process Integrator receives a message.)



When the business document arrives at each supplier's c-enabler, the following events occur:

1. The WebLogic Process Integrator process engine, which is a subscriber to WebLogic Collaborate events, receives the business document and instantiates the workflow instance that matches the conversation definition identified in the business message.
2. The workflow instance starts and processes the incoming business document by passing it to a user-written Java message manipulator class, `ExtractData.java`, which extracts the information from the incoming XML business document and passes that information back to the workflow instance.
3. Because approving the quote request requires human intervention, the workflow instance adds a task to the Worklist. The use of the Worklist is not mandatory. A trading partner has the flexibility to create whatever user interface applications are appropriate for its business processes that require human intervention.
4. The human user at the supplier site starts the worklist, double-clicks the task, and approves the request.

5. The workflow instance invokes another user-written Java message manipulator class, `CreateDocument.java`, to create an XML business document containing the response.
6. The message manipulator class returns the business document to the workflow instance.
7. The workflow instance passes the business document to the supplier's c-enabler.
8. The c-enabler prepares the XOCP protocol metadata for the business message containing the XML document, and sends it to the c-hub.

The C-Hub Sending a Response to the Buyer

The c-hub processes the supplier's response similar to how the c-hub processed the buyer's business message, applying Xpath routing and filtering expressions as necessary.

When the buyer's c-enabler receives the XOCP business message, the following events occur:

1. The c-enabler passes the business message to the market application via the conversation handler. The application is invoked asynchronously.
2. The conversation handler passes the business message to the local Java class, `ProcessPriceResponse.java`.
3. The Java class extracts the data from the business message, and returns the data to the market application.
4. The market application may invoke the C-Enabler API to terminate the conversation or send another message. If the market application terminates the conversation, the following events occur:
   a. The c-enabler sends the termination to the c-hub.
   b. The c-hub propagates the termination notice to all the trading partners enlisted in the conversation.
   c. Each trading partner receives the termination notice from the c-hub.

This concludes the end-to-end view of WebLogic Collaborate. The next section describes the other topics covered in this document and the rest of the WebLogic Collaborate documentation set.

Documentation Roadmap

The following topics are described in this document and throughout the entire WebLogic Collaborate documentation set:

- Installing the WebLogic Collaborate software — See the *BEA WebLogic Collaborate Installation Guide* for complete details about the installation prerequisites, installation steps, and installation verification instructions.
- WebLogic Collaborate Tour — See the tour to get a high-level description of the WebLogic Collaborate architecture and how to build and run an example furniture exchange.
- Setting up the WebLogic Collaborate environment — See Chapter 2, "Setting Up the WebLogic Process Integrator Environment," in this document for instructions on setting up WebLogic Collaborate to work with WebLogic Process Integrator. You need to complete this step to run the WebLogic Process Integrator-based example.
- Example applications — See Chapter 3, "Running the WebLogic Process Integrator Verifier Example," and Chapter 4, "Using Logic Plug-Ins for Billing," in this document for instructions on building and running two WebLogic Collaborate example applications.
- Configuration and monitoring tools — See the *BEA WebLogic Collaborate C-Hub Administration Guide* and the *BEA WebLogic Collaborate C-Enabler Administration Guide* for complete documentation on using the administration service to perform configuration, administration, and monitoring tasks associated with running, respectively, a c-space and a trading partner platform.
- Online help — When you use the C-Hub Administration Console or the C-Enabler Administration Console, click on the Help menu to obtain full, context-sensitive online help.

- Developing applications — See the *BEA WebLogic Collaborate Developer Guide* for a details about building and deploying trading partner, logic plug-in, and administration applications for the WebLogic Collaborate environment.
- Programmer reference documentation — See the *BEA WebLogic Collaborate Javadoc* for reference information on all the API packages provided with both WebLogic Collaborate and WebLogic Process Integrator.
- WebLogic Process Integrator documentation:
  - *BEA WebLogic Process Integrator Studio User Guide* explains how to use Studio to design workflows.
  - *BEA WebLogic Process Integrator Worklist Guide* explains how to use Worklist to view, perform, and work with tasks that are currently the roles to which you are subscribed.
  - *BEA WebLogic Process Integrator Tutorial* provides a tutorial on using the WebLogic Process Integrator software.

2 Setting Up the WebLogic Process Integrator Environment

The following sections explain how to set up the WebLogic Process Integrator environment, so that you can run WebLogic Process Integrator Studio:

- Configuring WebLogic Collaborate for WebLogic Process Integrator
- Troubleshooting Your WebLogic Process Integrator Configuration
- Starting WebLogic Process Integrator Studio

Before beginning the procedures in this chapter, run the WebLogic Collaborate installation verification application, `verifier`, described in the *BEA WebLogic Collaborate Installation Guide*.

Configuring WebLogic Collaborate for WebLogic Process Integrator

To configure your WebLogic Collaborate environment so that it operates with WebLogic Process Integrator, you need to verify WebLogic Process Integrator Studio and Weblogic Server run-time requirements, as described in this section.

1. Open a command window, and change to the following directory:

   Windows

   `%WLC_HOME%\enabler`

   UNIX

   `$WLC_HOME/enabler`

2. Open the `weblogic.properties` file. This file is configured during the installation process to match your environment. This step describes how to confirm that the settings are correct for your environment.

   The proper repository pool should be uncommented and configured to the database you selected at install time. The `weblogic.ejb.deploy` property and the `weblogic.jdbc.connectionPool.wlpiPool` property configurations should match your environment. Verify the configurations as follows:

   - Find the text in the `weblogic.properties` file that specifies the `weblogic.ejb.deploy` property. The following extract from the file displays the `weblogic.ejb.deploy` property:

     ```
     weblogic.ejb.deploy=\
     D:/bea/wlcollaborate1.0/lib/wlpi-ejb.jar,\
     D:/bea/wlcollaborate1.0/lib/wlpi-wlc-ejb.jar,\
     D:/bea/wlcollaborate1.0/lib/wlpi-cc-ejb.jar
     ```

     Ensure that the pathname is correct for the WebLogic Collaborate `lib` directory on your system. For this example, the WebLogic Collaborate software is installed in `D:/bea/wlcollaborate1.0`.

   Note: The `weblogic.properties` file uses forward slash format in pathnames, on UNIX and NT systems.

   - Find the text in the `weblogic.preperties` file that specifies the `weblogic.jdbc.connectionPool.wlpiPool` property.

The following extract from the file shows the property settings for the supported databases:

```
##Oracle##
##Oracle##weblogic.jdbc.connectionPool.wlpiPool=\
##Oracle##url=jdbc:weblogic:oracle,\
##Oracle##driver=weblogic.jdbc.oci.Driver,\
##Oracle##loginDelaySecs=1,\
##Oracle##initialCapacity=1,\
##Oracle##maxCapacity=10,\
##Oracle##capacityIncrement=1,\
##Oracle##allowShrinking=true,\
##Oracle##shrinkPeriodMins=15,\
##Oracle##refreshTestMinutes=10,\
##Oracle##props=user=<ORACLE_USER>;password=<ORACLE_PASSWORD>;
server=<ORACLE_SERVICENAME>

### Cloudscape settings ###

weblogic.jdbc.connectionPool.wlpiPool=\

    url=jdbc:cloudscape:wlpidb,\
    driver=COM.cloudscape.core.JDBCDriver,\
    loginDelaySecs=1,\
    initialCapacity=1,\
    maxCapacity=10,\
    capacityIncrement=1,\
    allowShrinking=true,\
    shrinkPeriodMins=15,\
    refreshTestMinutes=10

### SQL Server settings (MSSQL) ###

##MSSQL##weblogic.jdbc.connectionPool.wlpiPool=\
##MSSQL##url=jdbc:weblogic:mssqlserver4,\
##MSSQL##driver=weblogic.jdbc.mssqlserver4.Driver,\
##MSSQL##loginDelaySecs=1,\
##MSSQL##initialCapacity=1,\
##MSSQL##maxCapacity=10,\
##MSSQL##capacityIncrement=1,\
##MSSQL##allowShrinking=true,\
##MSSQL##shrinkPeriodMins=15,\
##MSSQL##refreshTestMinutes=10, \
##MSSQL## testTable=dual,\
##MSSQL##props=user=<MSSQL_USER>;password=<MSSQL_PASSWORD>;
server=<MSSQL_HOSTNAME>
```

The preceding extract shows that during installation this `weblogic.properties` file was configured for the Cloudscape database (the Cloudscape

`weblogic.jdbc.connectionPool.wlpiPool` property is uncommented). Confirm that the property corresponding to the database you are using is uncommented, and that the other database properties are commented.

Within the `weblogic.jdbc.connectionPool.wlpiPool` property, confirm that the configuration is correct for the database you are using:

- If you are using an Oracle database, confirm that *<ORACLE_USER>*, *<ORACLE_PASSWORD>*, and *<ORACLE_SERVICENAME>* (shown in bold in the extract above) are replaced by the user ID, password, and *ORACLE_SERVICENAME* values for your system. *ORACLE_SERVICENAME* is defined in the Oracle client `tnsnames.ora` file and is usually the Oracle system ID.
- If you are using a Microsoft SQL Server database, confirm that *<MSSQL_USER>*, *<MSSQL_PASSWORD>*, and *<MSSQL_HOSTNAME>* (shown in bold in the extract above) are replaced by the appropriate user ID, password, and Microsoft SQL Server host name values, respectively. (*MSSQL_HOSTNAME* may take one of the following forms: *DATABASE_NAME* or *DATABASE_NAME@HOST_NAME*.)

Note: The Cloudscape database is a single-user database and requires no user ID and password configurations in the `weblogic.jdbc.connectionPool.wlpiPool` property.

3. Close the `weblogic.properties` file.
4. Change to the directory where the WebLogic Process Integrator scripts are installed:

   Windows

   ```
   %WLC_HOME%\wlpi
   ```

   UNIX

   ```
   $WLC_HOME/wlpi
   ```

5. Create the WebLogic Process Integrator repository database tables by executing one of the following commands:

   - **Oracle**

     Windows

     ```
     prompt> createOracle.cmd
     ```

     UNIX

     ```
     prompt> . ./createOracle.sh
     ```

   - **Cloudscape**

     Windows

     ```
     prompt> createCloud.cmd
     ```

     UNIX

     ```
     prompt> . ./createcloud.sh
     ```

   - **Microsoft SQL Server (Windows)**

     ```
     prompt> createMSSQL.cmd
     ```

   Note: These scripts use DDL files (`wlpi_oracle.ddl`, `wlpi_cloud.ddl`, or `wlpi_mssql.ddl`). By default, the `create`*`database`*`.xx` scripts do not drop tables. If you have existing tables that you wish to drop, edit the appropriate DDL file.

   After you execute these scripts, you may see the SQL prompt. Exit from SQL before proceeding to the next step.

6. Change to the directory where you installed WebLogic Collaborate:

   Windows

   ```
   %WLC_HOME%
   ```

   UNIX

   ```
   $WLC_HOME
   ```

7. Ensure that the `setenv.sh` script defines the following environment variables.

Variable	Definition
JAVA_HOME	The directory path where you installed the JDK software. For example: *JAVA_HOME*=/usr/local/JDK1.2.2
WL_HOME	The directory path where you installed the WebLogic Server software. For example: *WL_HOME*=/usr/local/weblogic
WLC_HOME	The directory path where you installed the WebLogic Collaborate software. For example: *WLC_HOME*=/usr/local/WLCollaborate

8. Execute the `setenv` script to set the WebLogic Collaborate environment variables:

   Windows

   ```
   prompt> setEnv.cmd
   ```

   UNIX

   ```
   prompt> . ./setenv.sh
   ```

9. Change directory to the `enabler` subdirectory and start WebLogic Server:

   Windows

   ```
   prompt> cd enabler
   prompt> startweblogic.cmd
   ```

   UNIX

   ```
   prompt> cd enabler
   prompt> ./startwebLogic.sh
   ```

Note: WebLogic Server must be running when you start WebLogic Process Integrator Studio (refer to “Starting WebLogic Process Integrator Studio” on page 2-7).

Troubleshooting Your WebLogic Process Integrator Configuration

If you observe binding errors on the listening port, make sure another instance of WebLogic Server is not running using the same port (that is, another command window with the `startwebLogic` script running).

If WebLogic Server is unable to create a connection pool, as indicated by exceptions displayed in the command window, check that the connection pool property, `weblogic.jdbc.connectionPool.wlpiPool`, is specified correctly.

If you continue to have a problem, make sure that the database server is running and accessible, and that you have the appropriate permissions.

Starting WebLogic Process Integrator Studio

WebLogic Process Integrator Studio is the workflow editing component of the WebLogic Process Integrator software, and can be used to create the workflows for conversations and business processes in the WebLogic Collaborate environment.

Before using WebLogic Process Integrator in your WebLogic Collaborate environment, complete the following procedure to verify that you can start WebLogic Process Integrator Studio and connect to WebLogic Server.

1. Complete the procedure described in “Configuring WebLogic Collaborate for WebLogic Process Integrator” on page 2-2, and ensure that WebLogic Server is still running before you proceed to step 2.
2. Open a new command window, change to the directory where WebLogic Collaborate is installed, and execute the script to set the WebLogic Collaborate environment variables:

   Windows

   ```
   prompt> cd %WLC_HOME%
   prompt> setEnv.cmd
   ```

UNIX

```
prompt> cd $WLC_HOME
prompt> . ./setenv.sh
```

3. Change to the directory where the WebLogic Process Integrator scripts are installed:

   Windows

   ```
   %WLC_HOME%\wlpi
   ```

   UNIX

   ```
   $WLC_HOME/wlpi
   ```

4. Execute the following script, which starts WebLogic Process Integrator Studio and displays the WebLogic Process Integrator Studio login dialog box:

   Windows

   ```
   prompt> Studio.cmd
   ```

   UNIX

   ```
   prompt> . ./studio.sh
   ```

   The WebLogic Process Integrator Studio login dialog box is displayed.

5. Enter the BEA user name (`bea`) and password (`12345678`) in the dialog box.
6. In the Server text field, enter the following:

   `t3://localhost:7501`

   where *`localhost`* represents the name of your machine.
7. Click OK.

If you are unable to log in:

- Check for errors in the command window in which you started WebLogic Process Integrator Studio.
- See the section "Troubleshooting Your WebLogic Process Integrator Configuration" on page 2-7.
- Make sure that the WebLogic Server instance is still running.
- Make sure you ran the script (`createOracle`, `createCloud`, or `createMSSQL`) to create the WebLogic Process Integrator database tables. Refer to "Configuring WebLogic Collaborate for WebLogic Process Integrator." (See step 5.)

3 Running the WebLogic Process Integrator Verifier Example

The WebLogic Process Integrator Verifier example application (`wlpiverifier`), uses WebLogic Process Integrator workflows to exchange business messages in the WebLogic Collaborate environment. This example application demonstrates how WebLogic Process Integrator workflows are integrated into the WebLogic Collaborate environment, and how data is passed between two WebLogic Process Integrator partner workflows (trading partners) in a conversation.

The following sections describe how to run the Verifier example, and how WebLogic Process Integrator workflows are integrated in the WebLogic Collaborate environment:

- Setting Up and Running the WebLogic Process Integrator Verifier Example
- Understanding the WebLogic Process Integrator Verifier Example

Note: An installation Verifier example is also provided with WebLogic Collaborate to verify that the product is correctly installed. Before running the WebLogic Process Integrator Verifier example, run the installation Verifier example to ensure that your WebLogic Collaborate environment is configured so that the procedures described in the following sections will work. For information on how to run the installation Verifier example, refer to the *BEA WebLogic Collaborate Installation Guide*.

Setting Up and Running the WebLogic Process Integrator Verifier Example

You can configure the WebLogic Process Integrator Verifier example to run on one or two instances of WebLogic Server. Execute it on two instances of WebLogic Server before attempting to run it on a single instance.

Running the Example on Two WebLogic Server Instances

The WebLogic Process Integrator Verifier example is in the following location:

Windows

```
%WLC_HOME%\examples\wlpiverifier
```

UNIX

```
$WLC_HOME/examples/wlpiverifier
```

This example is configured to run on two instances of WebLogic Server. In this example, you start the c-hub on one instance of WebLogic Server, and two c-enablers on the second instance.

Complete the following procedures to set up and run the Verifier example:

- Step 1: Configuring WebLogic Collaborate for WebLogic Process Integrator
- Step 2: Building the Example
- Step 3: Starting the C-Hub
- Step 4: Starting the C-Enabler
- Step 5: Importing and Deploying the Workflow Templates
- Step 6: Running the Example

Step 1: Configuring WebLogic Collaborate for WebLogic Process Integrator

1. Configure and bring up WebLogic Server with WebLogic Process Integrator, according to the procedure in "Configuring WebLogic Collaborate for WebLogic Process Integrator" on page 2-2.

   This ensures that your environment is configured to run WebLogic Process Integrator and WebLogic Server.

2. Shut down WebLogic Server before proceeding to "Step 2: Building the Example." (Type Ctrl+C in the command window where WebLogic Server is running.)

Step 2: Building the Example

To build the WebLogic Process Integrator Verifier example application, complete the following procedure:

1. Open a new command window, change directory to the location where WebLogic Collaborate is installed, and execute the script to set the WebLogic Collaborate environment variables:

   Windows

   ```
   prompt> cd %WLC_HOME%
   prompt> setEnv.cmd
   ```

   UNIX

   ```
   prompt> cd $WLC_HOME
   prompt> . ./setenv.sh
   ```

2. Build the example to populate the compiled classes and required scripts:

   Windows

   ```
   prompt> cd %WLC_HOME%\examples\wlpiverifier
   prompt> build.cmd
   ```

   UNIX

   ```
   prompt> cd $WLC_HOME/examples/wlpiverifier
   prompt> . ./build.sh
   ```

Step 3: Starting the C-Hub

To start WebLogic Server and the c-hub, execute the following commands:

Windows

```
prompt> cd %WLC_HOME%\hub
prompt> startweblogic.cmd
```

UNIX

```
prompt> cd $WLC_HOME/hub
prompt> . ./startweblogic.sh
```

Note: Make sure WebLogic Server is still running before proceeding to "Step 5: Importing and Deploying the Workflow Templates."

Step 4: Starting the C-Enabler

1. Open a new command window, change directory to the location where WebLogic Collaborate is installed, and execute the script to set the WebLogic Collaborate environment variables:

   Windows

   ```
   prompt> cd %WLC_HOME%
   prompt> setEnv.cmd
   ```

   UNIX

   ```
   prompt> cd $WLC_HOME
   prompt> . ./setenv.sh
   ```

2. Change directory to where the c-enabler is located:

   Windows

   ```
   %WLC_HOME%\enabler
   ```

   UNIX

   ```
   $WLC_HOME/enabler
   ```

3. Open the `weblogic.properties` file and uncomment the lines indicated in bold in the following listing:

```
# The following are used for the WLPIVerifier example
#
#weblogic.system.startupClass.WlpiVerifierCaller=com.bea.b2b.wlpi.Start
#weblogic.system.startupArgs.WlpiVerifierCaller=ConfigFile=xml/enablers.xml,Ses
sionName=caller-session,\
#User=bea,Password=12345678,OrgName=BEA

#weblogic.system.startupClass.WlpiVerifierCallee=com.bea.b2b.wlpi.Start
#weblogic.system.startupArgs.WlpiVerifierCallee=ConfigFile=xml/enablers.xml,Ses
sionName=callee-session,\
#User=bea,Password=12345678,OrgName=BEA

#weblogic.httpd.register.WlpiVerifier=examples.wlpiverifier.WlpiVerifierServlet
# end of WLPIVerifier example
```

4. Start WebLogic Server:

Windows

```
prompt> startweblogic.cmd
```

UNIX

```
prompt> . ./startweblogic.sh
```

Note: You may get warning messages in the WebLogic Server log indicating that there are no active WebLogic Process Integrator templates for session caller-session in organization BEA. You can ignore these messages and continue to “Step 5: Importing and Deploying the Workflow Templates” on page 3-5.

For more information about configuring the c-enabler, see the *BEA WebLogic Collaborate C-Enabler Administration Guide*.

Step 5: Importing and Deploying the Workflow Templates

A WebLogic Process Integrator workflow template is a folder or a container for workflow template definitions. Each workflow template can hold one or more WebLogic Process Integrator workflow template definitions.

To import the workflow templates from the `templates` directory into WebLogic Process Integrator Studio, and deploy them in WebLogic Process Integrator Studio, complete the following procedure:

1. Open a new command window, change to the directory where WebLogic Collaborate is installed, and execute the script to set the WebLogic Collaborate environment variables:

   Windows

   ```
   prompt> cd %WLC_HOME%
   prompt> setEnv.cmd
   ```

   UNIX

   ```
   prompt> cd $WLC_HOME
   prompt> . ./setenv.sh
   ```

2. Change to the directory where WebLogic Process Integrator scripts are installed:

   Windows

   ```
   %WLC_HOME%\wlpi
   ```

   UNIX

   ```
   $WLC_HOME/wlpi
   ```

3. Execute the following script to start WebLogic Process Integrator Studio, and display the WebLogic Process Integrator Studio login dialog box:

   Windows

   ```
   prompt> Studio.cmd
   ```

   UNIX

   ```
   prompt> . ./studio.sh
   ```

The WebLogic Process Integrator Studio login dialog box is displayed.

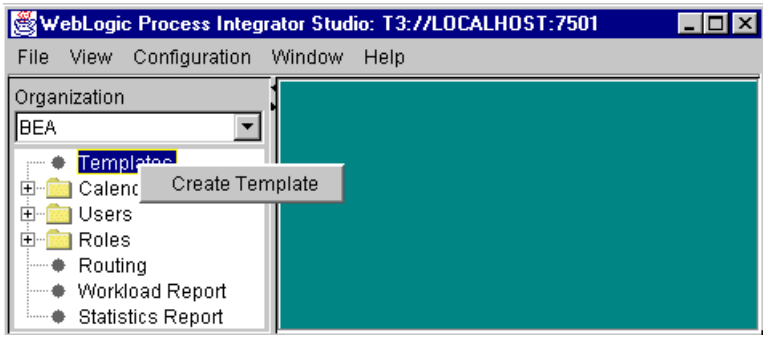
Figure 3-1 Logon to WebLogic Process Integrator Dialog Box

4. Enter the BEA user name (`bea`) and password (`12345678`) in the dialog box.
5. In the Server text field, enter the following:

   `t3://`*`localhost`*`:7501`

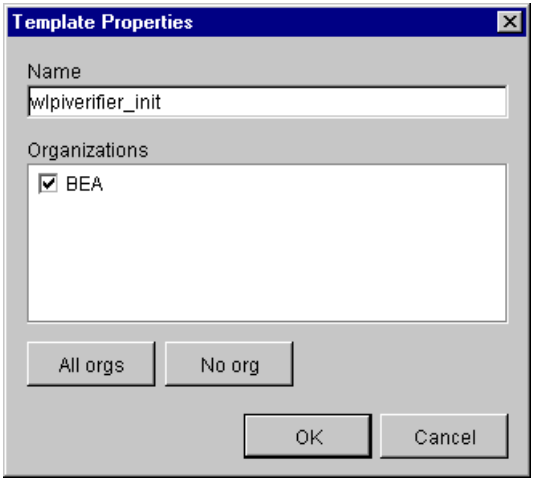
   where *`localhost`* represents the name of your machine.
6. Click OK to log in to WebLogic Process Integrator Studio and display the WebLogic Process Integrator Studio main window.

Figure 3-2 WebLogic Process Integrator Studio Main Window



7. Right-click on Templates, and click Create Template to display the Template Properties dialog box.

Figure 3-3 Template Properties Dialog Box

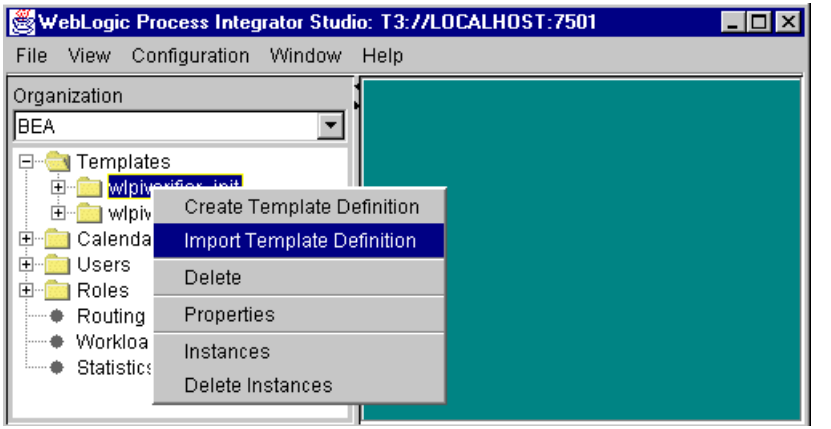


8. Enter the workflow template name (`wlpiverifier_init`) and click OK.
9. Repeat steps 8 and 9 to generate a second workflow template named `wlpiverifier_partner`.

   The WebLogic Process Integrator Studio screen displays the two workflow template folders you created.

10. Right-click on the `wlpiverifier_init` workflow template folder and choose Import Template Definition.

Figure 3-4 WebLogic Process Integrator Import Template Definitions Menu

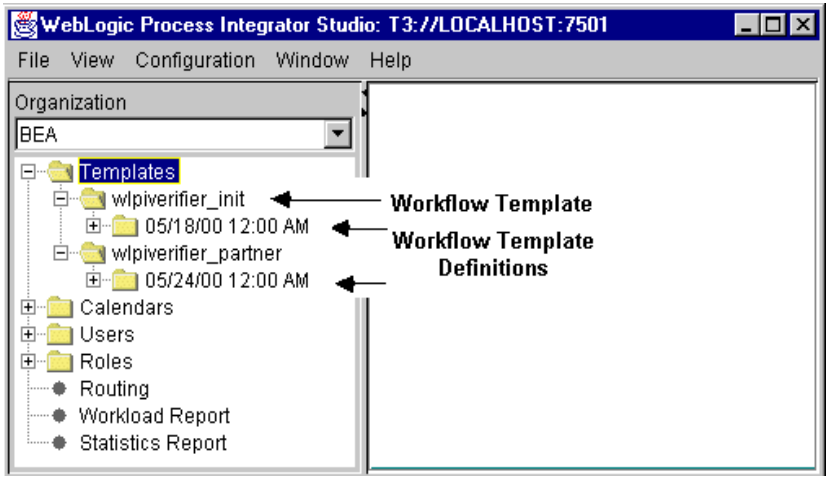


11. Navigate to the `%WLC_HOME%\examples\wlpiverifier\templates` directory and select the appropriate workflow template:

    - `wlpiverifier_init.xml`—for the `wlpiverifier_init` workflow
    - `wlpiverifier_partner.xml`—for the `wlpiverifier_partner` workflow

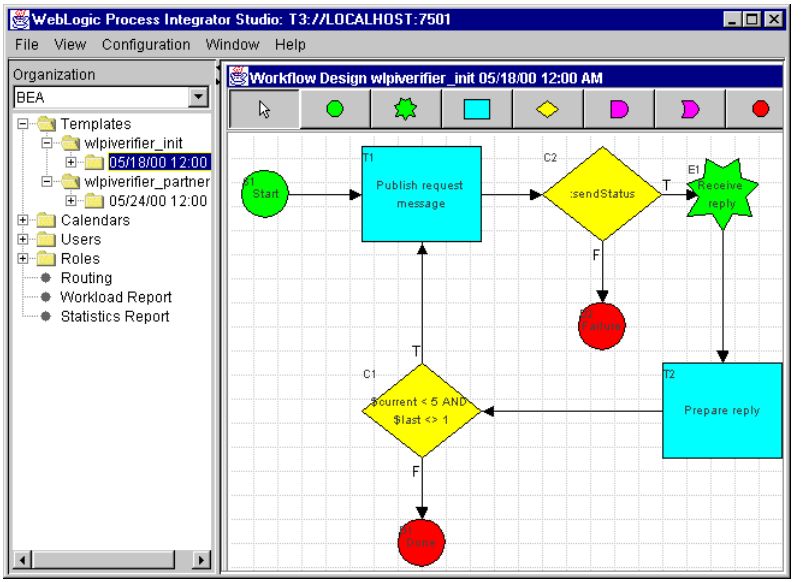
    The templates are imported from the `templates` directory into WebLogic Process Integrator Studio, and displayed as subfolders under the `wlpiverifier_init` and `wlpiverifier_partner` folders. The workflow template definitions subfolders are identified by an effective date (see Figure 3-5).

Figure 3-5 WebLogic Process Integrator Workflow Template Folders



12. Right-click on the `wlpiverifier_init` subfolder (the folder identified by the effective date (Figure 3-5)) and choose Open to display the workflow template definition in WebLogic Process Integrator Studio.

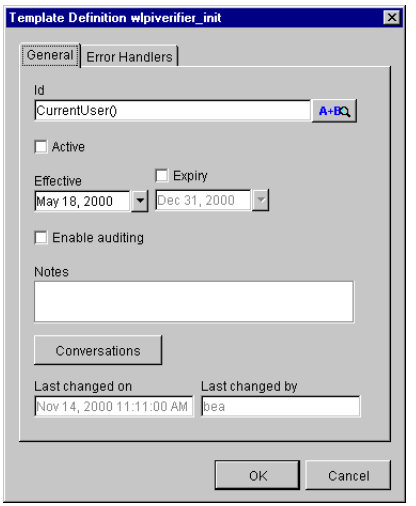
Figure 3-6 WebLogic Process Integrator Workflow Template Definition



13. Right-click on the `wlpiverifier_init` subfolder (the folder identified by the effective date (Figure 3-5)) and choose Properties from the drop-down menu.

    The Template Definition dialog box is displayed.

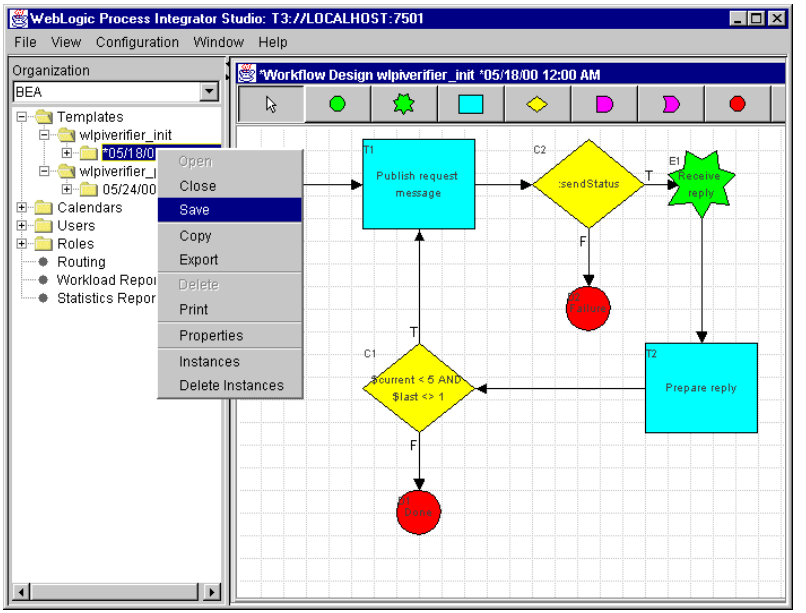
Figure 3-7 Template Definition Dialog Box



14. Select the Active check box and click OK.

15. Right-click on the `wlpiverifier_init` subfolder and choose Save.

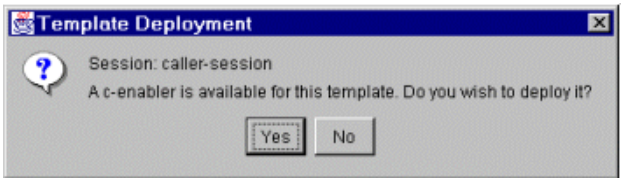
Figure 3-8 Save Workflow Definition Menu



The Template Deployment dialog box is displayed.

Note: This dialog box is displayed only if matching templates were not found by the c-enabler when it started. In that case, you would have seen error messages when you started the c-enabler (see "Step 4: Starting the C-Enabler" on page 3-4).

Figure 3-9 Template Deployment Dialog Box



16. Click Yes.

    The steps to import and deploy the `wlpiverifier_init` workflow template are complete.

17. Repeat steps 11 through 17 to import and deploy the `wlpiverifier_partner` workflow template.
18. Exit the WebLogic Process Integrator Studio.

Step 6: Running the Example

To run the WebLogic Process Integrator Verifier example application, complete the following procedure:

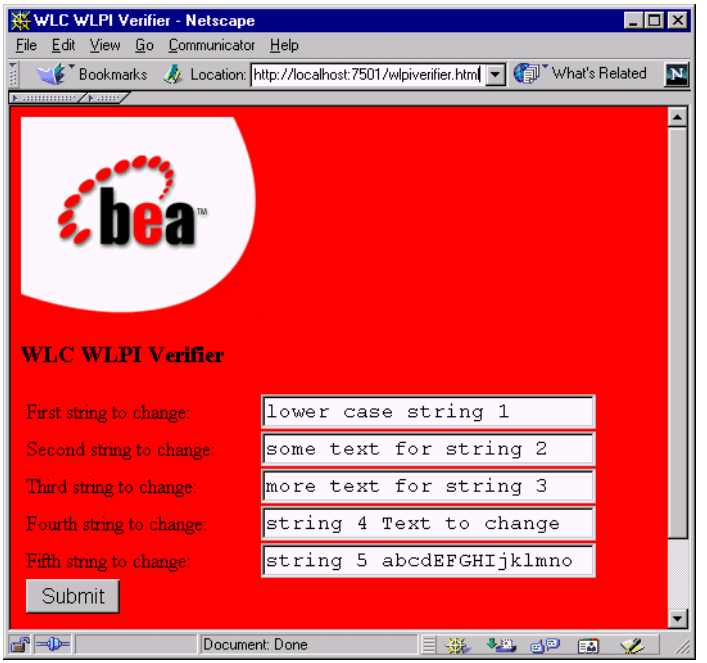
1. Make sure the c-hub and c-enabler are still running (see step 3 and step 4 above).
2. Invoke a Web browser with the following URL:

   `http://localhost:7501/wlpiVerifier.html`

   where *`localhost`* represents the name of your machine.

   The HTML page used to start the example application is displayed in your browser.

Figure 3-10 WebLogic Process Integrator Verifier Example Start Page



3. Click Submit to start the `wlpiverifier_init` workflow.

   The application starts. If the application executes successfully, a report containing the results is displayed in the browser.

Figure 3-11 WebLogic Process Integrator Verifier Example Report Page

4. If errors occur, check the WebLogic Server log for more information.

Running the Example on a Single WebLogic Server Instance

When you configure your system to execute the example application on a single instance of WebLogic Server, the c-hub and two c-enablers are colocated on that same instance.

Make sure that the following conditions are met before running the colocated example:

- You completed the execution of the WebLogic Process Integrator Verifier example application on two instances of WebLogic Server successfully (as described in "Setting Up and Running the WebLogic Process Integrator Verifier Example" on page 3-2).
- If you ran a logic plug-ins example application before running the WebLogic Process Integrator Verifier example, use one of the following procedures to prepare your system for the WebLogic Process Integrator Verifier example application:
  - Use the WebLogic Collaborate C-Hub Administration Console to remove the MessageCounter and CheckAccount plug-ins from the XOCP protocol. For details about using the C-Hub Administration Console, refer to Working With Logic Plug-Ins in the *BEA WebLogic Collaborate C-Hub Administration Guide*.
  - Initialize the WebLogic Collaborate c-hub database, and reload the c-hub WebLogic Process Integrator Verifier example application configuration.

  Warning: This procedure deletes your current c-hub configuration. Before proceeding, you can save the existing c-hub configuration by using the C-Hub Administration Console to export it. For more information, see Working With the Bulk Loader in the *BEA WebLogic Collaborate C-Hub Administration Guide*.

  To initialize and reload the c-hub database, complete the following procedure:

  a. Create the database schema:

     Windows

     ```
     prompt> createDB.cmd [oracle|cloudscape|mssql]
     ```

     UNIX

```
prompt> createDB.sh [oracle|cloudscape|mssql]
```

where the database name you use corresponds to your database.

b. Change to the examples directory, and reload the WebLogic Process Integrator Verifier example configurations:

Windows

```
prompt> cd %WLC_HOME%\examples\verifier
```

```
prompt> bulkloader.cmd [BulkloadOracleconfig.xml |
BulkloadCloudscapeconfig.xml |
BulkloadMSSQLconfig.xml]
```

UNIX

```
prompt> cd $WLC_HOME/examples/verifier
```

```
prompt> bulkloader.sh [BulkloadOracleconfig.xml |
BulkloadCloudscapeconfig.xml]
```

where the XML file you use corresponds to your database.

To configure the Verifier example application to execute on a single instance of WebLogic Server, complete the following procedure:

1. Open a command window, change to the directory where WebLogic Collaborate is installed, and execute the script to set the WebLogic Collaborate environment variables:

   Windows

   ```
   prompt> cd %WLC_HOME%
   ```

   ```
   prompt> setEnv.cmd
   ```

   UNIX

   ```
   prompt> cd $WLC_HOME
   ```

   ```
   prompt> . ./setenv.sh
   ```

2. Change to the following directory:

   Windows

   ```
   %WLC_HOME%\hub
   ```

   UNIX

   ```
   $WLC_HOME/hub
   ```

3. It is recommended that you make a copy of the `weblogic.properties` file. You will edit this file in step 4 below and then use the edited `weblogic.properties` file when you run the WebLogic Process Integrator Verifier example application on a single instance of WebLogic Server.

4. Open the `weblogic.properties` file and locate the section in the file that applies to running the WebLogic Process Integrator Verifier example application on a single instance (colocated c-enablers) of WebLogic Server.

   The steps in this procedure describe how to uncomment and configure this section of the `weblogic.properties` file to match your environment:

   a. Search for the following line (near the beginning of the file) and comment it out:

   ```
   weblogic.system.startupClass.WLCStartup=com.bea.b2b.hub.Startup
   ```

   b. Search for the following comment to identify the section of the `weblogic.properties` file that applies to the WebLogic Process Integrator example application running on colocated c-enablers:

   ```
   # The rest of this properties file is associated with
   # colocated enablers that use WLPI.
   ```

   Uncomment all the code in this section of the `weblogic.properties` file. You should uncomment only one of the `weblogic.jdbc.connectionPool.wlpiPool` properties (the one for the database you are using). See (c) below.

   c. Uncomment the `weblogic.jdbc.connectionPool.wlpiPool` property that corresponds to the database you are using. The following extract from the file shows the property settings for the supported databases.

   ```
   ### Oracle settings

   #####Oracle##weblogic.jdbc.connectionPool.wlpiPool=\
   ##Oracle##url=jdbc:weblogic:oracle,\
   ##Oracle##driver=weblogic.jdbc.oci.Driver,\
   ##Oracle##loginDelaySecs=1,\
   ##Oracle##initialCapacity=1,\
   ##Oracle##maxCapacity=10,\
   ##Oracle##capacityIncrement=1,\
   ##Oracle##allowShrinking=true,\
   ##Oracle##shrinkPeriodMins=15,\
   ##Oracle##refreshTestMinutes=10,\
   ##Oracle##props=user=<ORACLE_USER>;password=
   ##Oracle##<ORACLE_PASSWORD>;server=<ORACLE_SERVICENAME>
   ```

```
### Cloudscape settings###

weblogic.jdbc.connectionPool.wlpiPool=\

       url=jdbc:cloudscape:wlpidb,\
       driver=COM.cloudscape.core.JDBCDriver,\
       loginDelaySecs=1,\
       initialCapacity=1,\
       maxCapacity=10,\
       capacityIncrement=1,\
       allowShrinking=true,\
       shrinkPeriodMins=15,\
       refreshTestMinutes=10

### SQL Server settings (MSSQL) ###

##MSSQL##weblogic.jdbc.connectionPool.wlpiPool=\
##MSSQL##url=jdbc:weblogic:mssqlserver4,\
##MSSQL##driver=weblogic.jdbc.mssqlserver4.Driver,\
##MSSQL##loginDelaySecs=1,\
##MSSQL##initialCapacity=1,\
##MSSQL##maxCapacity=10,\
##MSSQL##capacityIncrement=1,\
##MSSQL##allowShrinking=true,\
##MSSQL##shrinkPeriodMins=15,\
##MSSQL##refreshTestMinutes=10, \
##MSSQL## testTable=dual,\
##MSSQL##props=user=<MSSQL_USER>;password=<MSSQL_PASSWORD>;
##MSSQL##server=<MSSQL_HOSTNAME>
```

d. Within the `weblogic.jdbc.connectionPool.wlpiPool` property, confirm that the configuration is correct for the database you are using:

- If you are using an Oracle database, confirm that *<ORACLE_USER>*, *<ORACLE_PASSWORD>*, and *<ORACLE_SERVICENAME>* (shown in bold in the extract above) are replaced by the user ID, password, and *ORACLE_SERVICENAME* values for your system. *ORACLE_SERVICENAME* is defined in the Oracle client `tnsnames.ora` file and is usually the Oracle system ID.
- If you are using a Microsoft SQL Server database, confirm that *<MSSQL_USER>*, *<MSSQL_PASSWORD>*, and *<MSSQL_HOSTNAME>* (shown in bold in the extract above) are replaced by the appropriate user ID, password, and Microsoft SQL Server host name values, respectively. (*MSSQL_HOSTNAME* may take one of the following forms: *DATABASE_NAME* or *DATABASE_NAME@HOST_NAME*.

e. The Cloudscape database is a single-user database and requires no user ID and password configurations in the `weblogic.jdbc.connectionPool.wlpiPool` property.

f. Find the text in the `weblogic.properties` file that specifies the `weblogic.ejb.deploy` property.

The following extract from the file displays the `weblogic.ejb.deploy` property:

```
weblogic.ejb.deploy=\
D:/bea/wlcollaborate1.0/lib/wlpi-ejb.jar,\
D:/bea/wlcollaborate1.0/lib/wlpi-wlc-ejb.jar,\
D:/bea/wlcollaborate1.0/lib/wlpi-cc-ejb.jar
```

Ensure that the pathname is correct for the WebLogic Collaborate `lib` directory on your system. For this example, the WebLogic Collaborate software is installed in `D:/bea/wlcollaborate1.0`.

Note: The `weblogic.properties` file uses forward slash format in pathnames on UNIX and NT systems.

5. Close the `weblogic.properties` file.

6. Open the `wlcstartup.properties` file and uncomment the lines indicated in bold in the following listing:

```
# Start of WLPI Verifier example
#WLCStartup.WLPIVerifierCaller=com.bea.b2b.wlpi.Start
#ConfigFile=xml/enablers.xml
#SessionName=caller-session
#User=bea
#Password=12345678
#OrgName=BEA

#WLCStartup.WLPIVerifierCallee=com.bea.b2b.wlpi.Start
#ConfigFile=xml/enablers.xml
#SessionName=callee-session
#User=bea
#Password=12345678
#OrgName=BEA# end of WLPIVerifier example
# End of WLPI Verifier example
```

7. Close the `wlcstartup.properties` file.

8. Start WebLogic Server, which starts the c-hub and two c-enablers:

Windows

```
prompt> startweblogic.cmd
```

UNIX

```
prompt> . ./startweblogic.sh
```

9. Invoke a Web browser with the following URL:

   `http://localhost:7001/wlpiverifier.html`

   where

   - *`localhost`* represents the name of your machine
   - `7001` is the port number for the c-hub instance of WebLogic Server

10. Refer to "Step 6: Running the Example" on page 3-13 to see the HTML pages used to invoke the example. From this point in the process, the application runs on a single instance of WebLogic Server in the same way as it does on two instances.

    Note: The port number is 7001 for the c-hub instance of WebLogic Server; the port number for the c-enabler instance is 7501 ("Step 6: Running the Example" on page 3-13).

Understanding the WebLogic Process Integrator Verifier Example

This section describes workflow definitions for the WebLogic Process Integrator Verifier example application. For complete details about integrating workflow templates with WebLogic Collaborate, see Using Workflows to Exchange Business Messages in the *BEA WebLogic Collaborate Developer Guide*.

Data is passed between WebLogic Process Integrator partner workflows (trading partners) in a WebLogic Collaborate conversation. During execution of the WebLogic Process Integrator Verifier example, the following events occur:

- Trading partner `caller` sends a message to Trading partner `callee`. Trading partner `callee` converts the message to lowercase and returns it as a reply.

- The caller workflow (`wlpiverifier_init`) is a sequence of preparing the request message, sending the message, waiting for a reply, and storing the reply for processing by the calling application. Within the workflow, the sequence is repeated five times.
- The callee workflow (`wlpiverifier_partner`) is a sequence of getting a request, processing it, and returning the result. This sequence is repeated until the last request (identified by the `last` variable).

The application code manipulates the business messages that a workflow sends and receives using a class that implements the `com.bea.b2b.wlpi.MessageManipulator` interface.

Note: When you are using WebLogic Process Integrator Studio, you can view details for actions in any workflow task by double-clicking on the task in the workflow diagram. In the Actions box, click Activated. Select the action and click Update. A dialog box containing Class name, Input variable, and Output variable fields is displayed; the fields contain the name of the Java class to call, the name of a workflow variable to pass to the class, and name of the workflow variable that will be assigned the result of the class execution, respectively.

The wlpiverifier_init Workflow

This section describes the tasks in the `wlpiverifier_init` workflow diagram, and the variables associated with the `wlpiverifier_init` workflow.

Note: The variables associated with a workflow are displayed in a subfolder under the workflow template folder. To view the properties for a variable, right-click the variable name in the folder.

The following variables are associated with the `wlpiverifier_init` workflow:

`requestMsg`
: A Java object that holds the message to be used in the Send Business Message action.

`replyMsg`
: A Java object that holds the reply message received from a Receive Business Message event.

`requestString[0-4]`
: Variables that hold the request data passed in by the client application. These variables are used to fill in elements when the request message is being

constructed. (The parameter properties for these variables are marked as input.)

`replyString[0-4]`
: Variables that hold the reply data. When a reply message is received, the relevant data is extracted from the message and assigned to these variables. The WebLogic Process Integrator Verifier example application code accesses the `replyString` variables to obtain the data that it returns. (The parameter properties for these variables are marked as output.)

`noOfRecipients[0-4]`
: Variables that hold the number of recipients who received each of the requests. One variable is assigned for each message send.

`noOfRecipientsHolder`
: A variable that holds the number of recipients who received the request. The value from the variable is then used to assign the variables *noOfRecipients*[0-4] described above.

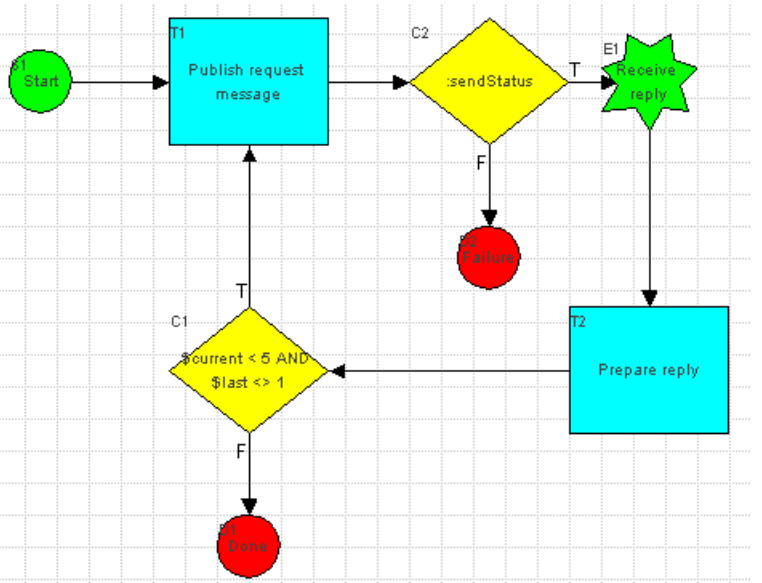
`current`
: The counter for the current message. For each `current` count, the workflow takes a `requestString[current]` and constructs a request message.

`last`
: A variable used to indicate the final request. When `last` is set, the workflow moves to the end stage.

`sendStatus`
: A Boolean value representing the status of the WebLogic Collaborate send.

The following figure shows the `wlpiverifier_init` workflow.

Figure 3-12 The wlpiverifier_init Workflow



Start Node (Green Circle)

This is the beginning of the workflow. It is started programmatically by `WlpiVerifierServlet.java`. To see how this is done, refer to the `startworkflow` method in the servlet class.

Publish Request Message (Blue Box)

This is the first task in the workflow. It consists of the following three actions:

- Manipulate business message using class `examples.wlpiverifier.PrepareQuery`

  This action makes a request to the specified class. The class is a user implemented class that implements the `MessageManipulator` interface. (See `<WLC_HOME>/examples/wlpiverifier/PrepareQuery.java`.) This class implements the `manipulate` method. The method takes the appropriate workflow variables (`requestString`) and constructs an XOCP business message. The constructed message is the return value of the method.

- Send Business Message

  This action sends the request message. Associated with this action are the Source variable that holds the request message, the Router Expression that holds the trading partner's name for the message, and the Target role that is the role of the receiver of the message; their values are `requestMsg`, `callee`, and `string-changer`, respectively.

  The partner workflow is linked with the conversation and role. A generic conversation handler is registered when the c-enabler session is created. When the first message comes in for the subordinate handler, the handler looks for an entry in the workflow table. Since it does not find one, it starts the workflow and passes it the first message.

- Mark task "Publish request message done"

  This ends the task and moves to the next stage of the workflow.

Decision Box:Send Status (Yellow Diamond)

This decision box determines whether the workflow should end or continue. The value placed in Send Status is evaluated to determine whether the message was sent successfully. If the evaluation is true, the workflow proceeds to the Receive Reply task, if the evaluation is false, the workflow fails and moves to the failure node. This failure node issues a conversation terminate failure to all participants in the conversation.

Receive Reply (Green Star)

This task waits for a business message to arrive, that is, it waits for a response message from the callee (`wlpiverifier_partner`). Associated with this action is a Target variable (*`replyMsg`*). The received business message is assigned to this variable.

Prepare Reply (Blue Box)

This task contains the following actions:

- Manipulate business message using class `examples.wlpiverifier.PrepareReply`

  This action is similar to the Manipulate business message using class `examples.wlpiverifier.PrepareQuery` action in the Publish Request Message task, except that this class processes the returned message and assigns the results from the message to the *`replyMsg`* variable. (The *`replyMsg`*

workflow variable is extracted by the WebLogic Process Integrator Verifier example application to get the data.)

- Set workflow variable *current* expression: current+1

  Increments the workflow variable *current* by 1.

- Mark task Prepare reply done

  This ends the task and moves to the next stage of the workflow.

Decision Box (Yellow Diamond)

This decision box determines if the *count* is less than five, and if the *last* variable is set to true or false. If the action is true, another iteration is performed, sending another message. If the action is false, the workflow terminates.

Done Node (Red Circle)

Terminates the workflow. A conversation terminate success is issued to all participants of the conversation.

The wlpiverifier_partner Workflow

This section describes the tasks in the `wlpiverifier_partner` workflow diagram, and the variables associated with the `wlpiverifier_partner` workflow.

Note: The variables associated with a workflow are displayed in a subfolder under the workflow template folder. To view the properties for a variable, right-click the variable name in the folder.

The following variables are associated with the `wlpiverifier_partner` workflow:

`requestMsg`
: A Java object that holds the message that is sent by the `wlpiverifier_init` workflow.

`replyMsg`
: A Java object that holds the message to be sent as a reply to the `wlpiverifier_init` workflow.

`last`
: A variable used to indicate the final request. When *`last`* is set to true, the workflow moves to the end stage.

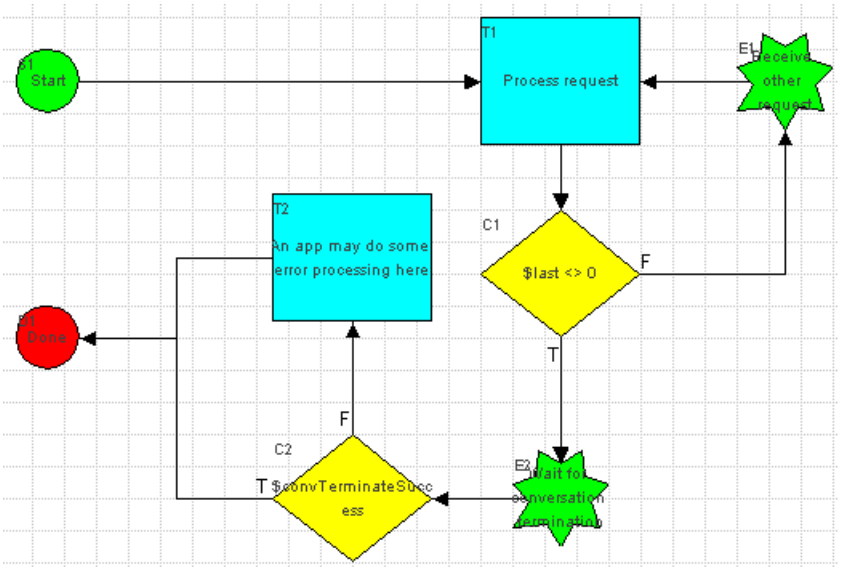
`convTerminateSuccess`
: A Boolean value representing conversation termination status.

`senderName`
: A string containing the name of the message sender.

`sendStatus`
: A Boolean value representing the status of the WebLogic Collaborate send business message.

The following figure shows the `wlpiverifier_partner` workflow.

Figure 3-13 The wlpiverifier_partner Workflow



Start Node (Green Circle)

This is the beginning of the `wlpiverifier_partner` workflow. When a message intended for this workflow arrives, the workflow is started by the c-enabler code. The message is then ready for processing. The format for the business message is specified in the *`requestMsg`* variable.

Process Request (Blue Box)

This task consists of the following actions:

- Manipulate business message using class `examples.wlpiverifier.ProcessRequest`

  This action makes a request to the specified class. The class is a user implemented class that implements the `MessageManipulator` interface. (See *`<WLC_HOME>`*`/examples/wlpiverifier/ProcessRequest.java`.) This class implements the `manipulate` method. This method takes the *`requestMsg`* workflow input variable, processes the data, and creates a reply message. The reply message is returned in the *`replyMsg`* output variable.

- Send Business Message

  This action sends the business message. Associated with this action are the Source variable that holds the request message, the router expression that holds the trading partner's name for the message, and the target role that is the role of the receiver of the message; their values are *replyMsg*, *senderName*, and *initiator*, respectively.

- Mark task Process request done

  This ends the task and moves to the next stage of the workflow.

Decision (Yellow Diamond)

This decision box determines if the workflow should finish based on the value of the workflow Boolean variable *last*. If *last* is set to `false`, the workflow proceeds to receive another request. If `true`, the workflow waits for conversation termination.

Receive Other Request (Green Star)

This task waits for a business message to arrive, that is, the Receive Other Request event waits for the next message from the caller (`wlpiverifier_init`). Associated with this action is a Target variable (*requestMsg*), and a Sender's Name (*senderName*).

Wait for Conversation Termination (Green Star)

This task blocks and waits for the initiator of the conversation (`wlpiverifier_init`) to terminate the conversation, if the last message has been received. It places the Termination Status in the defined workflow variable (`convTerminateSuccess`).

Decision (Yellow Diamond)

This decision box determines whether to end the conversation based on the value of the workflow Boolean variable *convTerminateSuccess*.

An Application May Do Some Error Processing Here (Blue Box)

Error processing tasks can happen at this node in the workflow.

Done Node (Red Circle)

End of the workflow. A conversation leave is issued.

Understanding WebLogic Process Integrator Java Classes

Business operations use WebLogic Process Integrator variables and Java code to manipulate business messages that are exchanged between trading partners. In general, the workflows sequence the flow of data, and Java classes process the data. The Java classes read and write the workflow variables using the WebLogic Process Integrator API.

The following Java classes are used in the WebLogic Process Integrator Verifier example application. For more details about business operations, see Using Workflows to Exchange Business Messages in the *BEA WebLogic Collaborate Developer Guide*.

Java Class	How This Class Works with the WebLogic Process Integrator Workflows
`WlpiVerifyServlet.java`	Starts the workflow. The `startWorkflow` method completes the following tasks using the WebLogic Process Integrator API: ■ Creates an instance of the `wlpiverifier_init` workflow ■ Copies the strings from the array into the *`requestString[0-4]`* workflow variables ■ Starts the `wlpiverifier_init` workflow ■ Waits for the workflow to end in the `waitForWorkFlowToEnd` method ■ Extracts data from the `wlpiverifier_init` workflow return ■ Passes the return values back to the HTML page The `wlpiverifier_init` workflow tasks `Publish request message` and `Prepare reply` invoke Java classes that operate on the workflow variables and thereby effect the execution of the workflow.
`PrepareQuery.java`	The Publish Request Message task invokes this class. The `PrepareQuery.java` class constructs the business messages to send. At run time, the Manipulate Business Message action is invoked to create a business message (based on the contents of WebLogic Process Integrator variables), and return the business message for storage in a variable. The `Prepare Query.java` class also checks to see if five messages were sent, and if so, sets the *`last`* variable to `true`, which causes the conditional test to fail, thereby leading to termination of the workflow.
`PrepareReply.java`	The Prepare reply task invokes this class. The `PrepareReply.java` class receives and processes the business messages. It implements the `MessageManipulator` interface.

Java Class	How This Class Works with the WebLogic Process Integrator Workflows
`ProcessRequest.java`	The Process request task in the `wlpiverifier_partner` workflow invokes this class to convert the characters in the input message to uppercase. The `ProcessRequest.java` class implements the `MessageManipulator` interface.

4 Using Logic Plug-Ins for Billing

The BEA WebLogic Collaborate billing examples illustrate how to implement and install user-written logic plug-ins. A logic plug-in is a service that a c-hub owner can develop and install on a c-hub, providing additional value for management of the c-hub, or providing additional value to customers who use that c-hub. A c-hub administrator accomplishes this by introducing custom code at well-defined plug-in points, for additional processing of the information that passes through the c-hub. This functionality is transparent to the c-enabler user.

For a comprehensive description of developing logic plug-ins, see Developing Logic Plug-Ins in the *BEA WebLogic Collaborate Developer Guide*.

The following sections describe the structure, purpose, and set-up procedures for two example plug-ins:

- Overview of the Logic Plug-In Examples
- Structure of the Logic Plug-Ins for Billing
- MessageCounter Logic Plug-In Example
- CheckAccount Logic Plug-In Example

Overview of the Logic Plug-In Examples

The MessageCounter and CheckAccount logic plug-ins are enhancements to an underlying example application. These plug-ins customize the billing process for a c-hub owner. In this example, you can add each plug-in separately or use them in combination. The plug-ins operate at different logical points in the flow of control. Applying these plug-ins and observing the results is a way to understand how and where the c-hub provider can influence the flow of information traveling between c-enablers and the c-hub, in the context of a conversation.

For complete information about setting up and running the logic plug-in examples, see the main sections:

- MessageCounter Logic Plug-In Example
- CheckAccount Logic Plug-In Example

Structure of the Logic Plug-Ins for Billing

In this scenario, a c-hub owner establishes a billing algorithm based on the number of messages an e-market trading partner sends, and on the cost for a specific conversation. The algorithm requires keeping track of all the messages a trading partner sends and all the conversations in which the partner participates.

The c-hub owner registers all messages passing through the hub, storing the registry information by trading partner and conversation name. The c-hub owner or administrator has the option of assigning a rate for messages in a conversation, depending on the characteristics of the conversation. The rate varies based on the quantity of c-hub resources a conversation consumes.

Purpose of the MessageCounter Logic Plug-In

The MessageCounter plug-in records the activities of each trading partner that uses the c-hub. The c-hub owner has the authority to review trading partner accounts and adjust an outstanding balance when a trading partner makes a payment.

Purpose of the CheckAccount Logic Plug-In

The c-hub owner can disallow access to the c-hub for specific trading partners. One way to do this is to apply selection criteria under program control, automatically excluding trading partners whose outstanding account balance is greater than an amount set by the owner. When you add the CheckAccount plug-in to an application, you can set the account balance due threshold to a value other than the default value. The CheckAccount plug-in filters trading partners, disallowing access to the c-hub, for partners who do not meet the selection criteria.

Utility Servlets for the Logic Plug-Ins

The `Inquiry` and `Reset Account` functions are tools for checking account status, and optionally setting outstanding account balances back to zero. Only the c-hub owner or an administrator has the authority to use these tools. The utility servlet source files in the Using Logic-Plug Ins for Billing example are `InquiryAccountServlet.java` and `ResetMsgServlet.java`.

MessageCounter Logic Plug-In Example

The MessageCounter logic plug-in shows how a c-hub owner can introduce a service that provides additional value for management of the c-hub. This plug-in collects data that the c-hub owner applies to a billing algorithm. This functionality is transparent to the c-enabler user.

The MessageCounter logic plug-in for billing is one of two enhancements for an example application. (The second is the CheckAccount logic plug-in.)

This logic plug-in example uses a relational database to extend features for managing a c-hub. The set of example files includes scripts for Oracle, Cloudscape, and MS SQL Server databases.

Structure of the MessageCounter Logic Plug-In Example

A c-hub owner needs to know the number of messages an e-market trading partner sends, and all the conversations in which the partner participates. This information is applied to a billing algorithm. The c-hub owner has the option of assigning a cost for each message depending on the characteristics of the conversation. The cost can vary from conversation to conversation, based on the quantity of c-hub resources a conversation consumes.

The c-hub owner or administrator creates two database tables that are external to the WebLogic Collaborate repository:

- The `MESSAGECOST` table stores the conversation name and the cost of each message sent in the context of this conversation. The order of events in this example sets conversation name and cost parameters in the `MESSAGECOST` table before messaging begins.
- The `Billing` table is the store for trading partner name, conversation name, and a count of messages sent by each trading partner for a named conversation. Custom code in the MessageCounter plug-in records the number of sent messages, storing the information in the `Billing` table. The count increases each time a trading partner sends a message in the context of an identified conversation.

The MessageCounter custom code plugs in to the c-hub as the first element in the XOCP filter logic plug-in chain.

After the c-hub applies all routing criteria and filters, the MessageCounter logic is executed once for each ultimate recipient of a message.

Required Files

Files required for setting up and running the MessageCounter logic plug-in are in directories subordinate to the location defined by the *WLC_HOME* environment variable.

Windows

```
%WLC_HOME%\examples\LogicPlugIns\MessageCounter
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/MessageCounter
```

The following sections and tables identify the name of each file that is required to set up and run the MessageCounter logic plug-in. The files are grouped by function. Where required, a table provides supplementary notes specific to a file. See “Setting Up the MessageCounter Logic Plug-In” on page 4-8 and “Running an Application with the MessageCounter Logic Plug-In” on page 4-14 for complete information on how to use these files.

Files for Loading the WebLogic Collaborate Repository

The XML file for input to the Bulk Loader utility is in the examples directory subordinate to the location defined by the *WLC_HOME* environment variable. The `RepData.xml` file and the command files are in the `MessageCounter` directory.

Filename	Purpose	Notes
`Bulkload`*`database`*`Config.xml`	An XML configuration file used with the Bulk Loader utility to populate the WebLogic Collaborate repository. Works in conjunction with the `RepData.xml` file in the `MessageCounter` directory.	As supplied, the information in the file relates to the `verifier` example application only.To run this logic plug-in with an example application other than `verifier`, for example `wlpiverifier`, you must first register the MessageCounter logic plug-in the target application using the C-Hub Administration Console. To use the MessageCounter logic plug-in in combination with the CheckAccount logic plug-in, populate the repository using the `RepData.xml` file from the `LogicPlugIns` directory.
`RepData.xml`	The XML data file containing MessageCounter data to be loaded into the repository.	Follow the directives in this file: `Bulkload`*`database`*`Config.xml`

`build.cmd`
or
`build.sh`
	A command file you invoke to compile the Java source files and to copy all required files into target directories.	**Windows**
		`build.cmd`
		UNIX
		`build.sh`

Java Source Files for Using the MessageCounter Logic Plug-In

The Java source files listed in this table are in these directories:

Windows

```
%WLC_HOME%\examples\LogicPlugIns\MessageCounter\Sources
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/MessageCounter/Sources
```

Filename	Purpose
SentMsgCounter.java	Contains the Java class code for implementing the MessageCounter logic plug-in.
InquiryAccountServlet.java	Java servlet for inquiring about the status of a trading partner account.
ResetMsgServlet.java	Java servlet for modifying the status of a trading partner account.

HTML File for Querying Trading Partner Account Status

There is a version of this file in each of these directories:

Windows

```
%WLC_HOME%\examples\LogicPlugIns\MessageCounter\HtmlFiles
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/MessageCounter/HtmlFiles
```

Filename	Purpose
LogicPlugIns.htm	Used to inquire about the status of a trading partner account.

Files for Managing Database Tables

There is a version of the `CreateDataBaseTables` command file in each of these directories:

Windows

```
%WLC_HOME%\examples\LogicPlugIns\MessageCounter\DataBase
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/MessageCounter/DataBase
```

The SQL scripts are in a database-specific directory subordinate to the `DataBase` directory.

Filename	Purpose	Notes
`CreateDataBaseTables.cmd` or `CreateDataBaseTables.sh`	Command file that directs the `createTables.sql` script and the `populateTables.sql` as input to a database utility.	**Windows** `*.cmd` **UNIX** `*.sh`
`createTables.sql`	SQL script. Input file to the `CreateDataBaseTables` command file. Creates the `Billing` table and the `MESSAGECOST` table.	SQL scripts are platform independent.
`populateTables.sql`	SQL script. Input file to the `CreateDataBaseTables` command file. Inserts data into the `MESSAGECOST` table.	SQL scripts are platform independent.

Setting Up the MessageCounter Logic Plug-In

You can apply the MessageCounter logic plug-in to an existing example application such as the `verifier` example or the `wlpiverifier` example. The procedure here applies the logic plug-in to the `verifier` example.

The next two sections summarize the set-up steps on Windows and UNIX. The "MessageCounter Setup Steps in Detail" section describes the same steps in more detail.

Setup on Windows: Main Steps

This section summarizes the steps for enabling the MessageCounter logic plug-in in an application. See "MessageCounter Logic Plug-In Setup Steps in Detail" on page 4-10 for more detailed information about each step.

1. Run the `setenv.cmd` command file.

2. Customize the `Bulkload`*`database`*`Config.xml` file, where *`database`* identifies the database you are using.
3. Change to the `%WLC_HOME%\LogicPlugIns\MessageCounter` directory.
4. From the `MessageCounter` directory, execute this command:
   `bulkloader ..\..\BulkLoad`*`database`*`Config.xml`
5. Customize the `CreateDataBaseTables.cmd` command file.
6. Run the `CreateDataBaseTables.cmd` command file.
7. Run the `build.cmd` command file.
8. Customize the `c-hub weblogic.properties` configuration file, enabling the `Java servlet` and `JDBC Connection Pool` sections.
9. Ensure that your environment is defined, and start the c-hub.

Setup on UNIX: Main Steps

See "MessageCounter Logic Plug-In Setup Steps in Detail" on page 4-10 for complete information about each step.

1. Run the `setenv.sh` shell.
2. Customize the `Bulkload`*`database`*`Config.xml` file, where *`database`* identifies the relational database you are using.
3. Change to this directory:
   `$WLC_HOME/examples/LogicPlugIns/MessageCounter`
4. From the `MessageCounter` directory, execute the following command:

   `bulkloader.sh ../../BulkLoad`*`database`*`Config.xml`
5. Customize the `CreateDataBaseTables.sh` shell.
6. Run the `CreateDataBaseTables.sh` shell.
7. Run the `build.sh` shell.
8. Customize the `c-hub weblogic.properties` configuration file, enabling the `Java servlet` and `JDBC Connection Pool` sections.
9. Ensure that your environment is set, and start the c-hub.

MessageCounter Logic Plug-In Setup Steps in Detail

The following procedure explains each main step. The numbering corresponds to the numbering in the main step sections.

1. Set the environment.

   Run the `setenv.cmd` command file (Windows), or the `setenv.sh` shell (UNIX). This step establishes the environment for your system.

2. Customize the Bulk Loader XML configuration file.

   Customize the configuration file that corresponds to the database you are using:

   - `BulkloadCloudscapeConfig.xml`
   - `BulkloadMSSQLConfig.xml`
   - `BulkloadOracleConfig.xml`

   This `XML` file is an input file to the Bulk Loader utility that you invoke in step 4 of this series of steps. You must define the database connectivity variables correctly before populating the repository, creating database tables, and populating database tables.

3. Change to the `MessageCounter` directory.

   Change to the `MessageCounter` directory, and run subsequent steps from this location.

   Windows

   ```
   %WLC_HOME%\examples\LogicPlugIns\MessageCounter
   ```

   UNIX

   ```
   $WLC_HOME/examples/LogicPlugIns/MessageCounter
   ```

4. Populate the WebLogic Collaborate repository.

   From the `MessageCounter` directory, run the Bulk Loader utility, supplying the correct path to the input XML configuration file. Additionally, this step reads the `RepData.xml` file as input.

   Windows

   Syntax:

   `bulkloader ..\..\Bulkload`*`database`*`Config.xml`

   Example:

```
bulkloader ..\..\BulkloadOracleConfig.xml
```

UNIX

Syntax:

```
bulkloader.sh ../../Bulkload<i>database</i>Config.xml
```

Example:

```
bulkloader.sh ../../BulkloadCloudscapeConfig.xml
```

5. Customize the `CreateDataBaseTables` command file in the `DataBase` directory.

   Windows

   ```
   MessageCounter\DataBase\CreateDataBaseTables.cmd
   ```

   UNIX

   ```
   MessageCounter/DataBase/CreateDataBaseTables.sh
   ```

   The values for the connection parameters must match those specified in the `JDBC examplesPool` in the `weblogic.properties` file. Modify the connection values required for your underlying database. The user name you specify must be an existing user with authority to create tables in the database.

6. Create the database tables and populate with data.

   To create the `MESSAGECOST` and `Billing` database tables, run the command file you customized in the previous step. The database tables are external to the repository. You supply one input parameter to the command file, identifying the subdirectory for your database. The command file reads the `createTables.sql` and `populateTables.sql` script files from the subordinate directory.

   As supplied, the `populateTables.sql` script inserts two rows of data into the `MESSAGECOST` table. These `insert` statements are from the script:

   ```
   insert into MESSAGECOST values ('verifierConversation',<b>20.37</b>);
   insert into MESSAGECOST values ('to-upper',3);
   ```

   To change the cost for each message in a conversation, in the `populateTables.sql` script, modify the numeric value to be inserted into the `MESSAGE_COST` column. For example, to change the cost of a message in the `verifierConversation` conversation from the supplied value of 20.37, modify the numeric value in the first `insert` statement. You can run step 6 again later, if you decide to change a message cost value.

Windows

Syntax:

```
CreateDataBaseTables { oracle | cloudscape | mssql }
```

Example:

```
CreateDataBaseTables oracle
```

UNIX

```
Syntax:
```

```
CreateDataBaseTables.sh { oracle | cloudscape }
```

Example:

```
CreateDataBaseTables.sh cloudscape
```

7. Create the executable programs for this example.

   Run the build command file for your platform. This step compiles required Java source files and copies resulting files to target locations.

   Windows

   ```
   build.cmd
   ```

   UNIX

   ```
   build.sh
   ```

8. Using a text editor, uncomment the following sections of the c-hub `weblogic.properties` configuration file specific to the logic plug-in examples:

   a. The `Logic Plug-Ins servlets` section. This section is identified by the following section header, which appears before the lines that you need to uncomment:

```
#########################################################################
# Servlets used for the WLC Logic PlugIns example
# Uncomment to use
# The Logic PlugIns require the Verifier example to be working
#########################################################################
```

   b. The section that specifies the data source to be used. This section is identified by the following section header:

```
#########################################################################
#Data Source used for the Logic PlugIns examples
```

```
#Uncomment it whatever database you use
#########################################################################
```

c. The section that specifies the JDBC connection pool specific to the database you are using. For example, if you are using Oracle, this section is identified by the following section header:

```
#########################################################################
# Oracle 8.1.5
# Note: EXAMPLE_USER, EXAMPLE_PASSWORD must match the values used in the
# CreateDatabaseTables.cmd(sh) command
#########################################################################
```

If you are using Oracle, be sure to specify values for *<EXAMPLE_USER>* and *<EXAMPLE_PASSWD>* that match the values used in the `weblogic.properties.verifier CreateDatabaseTables` script.

If you are using MS SQL server, be sure to specify values for *<MSSQL_USER>*, *<MSSQL_PASSWORD>*, and *<MSSQL_HOSTNAME>* that match your MS SQL server installation.

d. The section that represents ACLs for the examplesPool. This section is identified by the following section header:

```
#########################################################################
# The following three lines represent ACLs for the examplesPool and
# must be un-commented independently of the database type used
#########################################################################
```

To uncomment the appropriate lines in each of these sections, simply delete the comment character (#) at the beginning of each line in those sections. For example, the section that represents ACLs for the examplesPool should appear as:

```
weblogic.allow.reserve.weblogic.jdbc.connectionPool.examplesPool=everyone
weblogic.allow.reset.weblogic.jdbc.connectionPool.examplesPool=everyone
weblogic.allow.shrink.weblogic.jdbc.connectionPool.examplesPool=everyone
```

9. Start the example application.

Ensure that your environment is defined correctly, and start the c-hub.

Running an Application with the MessageCounter Logic Plug-In

Follow these steps to observe how the MessageCounter logic plug-in adds value to an underlying application.

1. Start the `verifier` example application, or the alternative example application you configured for the logic plug-in.
2. Examine the WebLogic Server system log for messages from the MessageCounter logic plug-in.

   The logic plug-in writes messages to the log each time the c-hub processes a business document. The first time a trading partner sends a message in the `verifier` conversation, the logic plug-in records this message:

   ```
   ** MessageCounter ##### 1 Record Inserted
   ```

   This message tells you that custom program logic in the logic plug-in has inserted a record into the `Billing` table. The count of messages sent is set to one.

   Messages from the same trading partner after the first trigger this message:

   ```
   ** MessageCounter ##### 1 Record Updated
   ```

   This message tells you that the count of sent messages has been incremented by one.
3. Use your Web browser to inquire about the status of a trading partner account. Enter the following URL in a browser:

   ```
   http://localhost:7001/examples/LogicPlugIns/LogicPlugIns.htm
   ```

   Enter the name of a trading partner and click on the `Submit` button. This Java servlet utility returns information about the trading partner.

   Note: The valid names for the two trading partners are `'Partner1'` and `'Partner2'`. These names are case-sensitive, and do include the single-quote characters.
4. Continuing with the `LogicPlugIns.htm` tool, reset the trading partner balance due back to zero by clicking the `Reset` button.

   If you have set up the CheckAccount logic plug-in, the MessageCounter logic plug-in processes database information recorded by both logic plug-ins.

CheckAccount Logic Plug-In Example

The CheckAccount logic plug-in shows how a c-hub owner can introduce a service that provides additional value for management of the c-hub. This logic plug-in implements a custom filtering mechanism that is an extension of the filtering mechanisms available through BEA WebLogic Collaborate based on an XPATH expression. This functionality is transparent to the c-enabler user.

The CheckAccount logic plug-in for billing is one in a series of enhancements for an example application.

This logic plug-in example uses a relational database to extend features for managing a c-hub. The set of example files includes scripts for Oracle, Cloudscape, or MS SQL Server databases.

Structure of the CheckAccount Logic Plug-In Example

A c-hub owner chooses to filter messages to trading partners for specific conversations. The criterion for filtering is the account balance a potential recipient owes to the c-hub owner. Trading partners can be excluded from a conversation if they have outstanding balances higher than a threshold amount the c-hub owner defines.

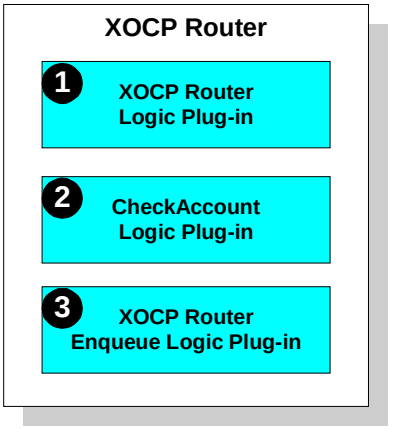
The formula for determining the account balance includes a count of messages a trading partner sends, and a rate for each conversation in which the trading partner participates. The rate for each conversation varies depending on the amount of c-hub resources the conversation consumes. For each conversation, the cost is the product of conversation rate and number of messages sent. The total account balance due is the sum of current conversation costs. In the CheckAccount logic plug-in, you can vary the threshold amount for excluding trading partners from a conversation. In the MessageCounter logic plug-in, you can vary the rate of cost for a conversation. You can implement each logic plug-in separately or in combination. Each allows you to vary parameters and observe the results.

The c-hub owner or administrator creates two database tables that are external to the WebLogic Collaborate repository:

- The `MESSAGECOST` table stores the conversation name and the cost of each message sent in the context of this conversation. This order of events in this

example sets conversation name and cost parameters in the `MESSAGECOST` table before messaging begins.

- The `Billing` table is the store for trading partner name, conversation name, and a count of messages sent by each trading partner for a named conversation. The CheckAccount custom code plugs into the c-hub after the XOCP router logic plug-in and before the XOCP router enqueue logic plug-in. The custom logic is executed once for each incoming message. The following figure shows the sequence in which the logic plug-ins in the XOCP router should be configured.



Required Files

Files required for setting up and running the CheckAccount logic plug-in are in directories subordinate to the directory defined by the *`WLC_HOME`* environment variable.

Windows

```
%WLC_HOME%\examples\LogicPlugIns\CheckAccount
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/CheckAccount
```

The following sections and tables identify the name of each file that is required to set up and run the CheckAccount logic plug-in. The files are grouped by function. Where required, a table provides supplementary notes specific to a file. See "Setting Up the CheckAccount Logic Plug-In" on page 4-20 and "Running an Application with the CheckAccount Logic Plug-In" on page 4-25 for complete information on how to use these files.

Files for Loading the WebLogic Collaborate Repository

The XML file for input to the Bulk Loader utility is in the examples directory subordinate to the location defined by the `WLC_HOME` environment variable. The `RepData.xml` file and the command files are in the `CheckAccount` directory.

Filename	Purpose	Notes
`Bulkload`*`database`*`Config.xml`	An XML configuration file used with the Bulk Loader utility to populate the WebLogic Collaborate repository. Works in conjunction with the `RepData.xml` file in the `CheckAccount` directory.	As supplied, the information in the file relates to the `verifier` example application only. To run this logic plug-in with an example application other than `verifier`, for example `wlpiverifier`, you must first register the CheckAccount logic plug-in the target application using the C-Hub Administration Console. To use the CheckAccount logic plug-in in combination with the MessageCounter logic plug-in, populate the repository using the `RepData.xml` file from the `LogicPlugIns` directory.
`RepData.xml`	The XML data file containing CheckAccount data to be loaded into the repository.	Follow the directives in this file: `Bulkload`*`database`*`Config.xml`

`build.cmd`
or
`build.sh`
	A command file you invoke to compile the Java source files and to copy all required files into target directories.	**Windows**
		`build.cmd`
		UNIX
		`build.sh`

Java Source Files for Using the CheckAccount Logic Plug-In

The Java source files listed in this table are in these directories:

Windows

```
%WLC_HOME%\examples\LogicPlugIns\CheckAccount\Sources
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/CheckAccount/Sources
```

Filename	Purpose
`CheckAccount.java`	Contains the Java class code for implementing the CheckAccount logic plug-in.
`InquiryAccountServlet.java`	Java servlet for inquiring about the status of a trading partner account.
`ResetMsgServlet.java`	Java servlet for modifying the status of a trading partner account.

HTML File for Querying Trading Partner Account Status

There is a version of this file in each of these directories:

Windows

```
%WLC_HOME%\examples\LogicPlugIns\CheckAccount\HtmlFiles
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/CheckAccount/HtmlFiles
```

Filename	Purpose
`LogicPlugIns.htm`	Used to inquire about the status of a trading partner account.

Files for Managing Database Tables

There is a version of the `CreateDataBaseTables` command file in each of these directories:

Windows

```
%WLC_HOME%\examples\LogicPlugIns\CheckAccount\DataBase
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/CheckAccount/DataBase
```

The SQL scripts are in a database-specific directory subordinate to the `DataBase` directory.

Filename	Purpose	Notes
`CreateDataBaseTables.cmd` or `CreateDataBaseTables.sh`	Command file that directs the `createTables.sql` script and the `populateTables.sql` script as input to a database utility.	**Windows** `*.cmd` **UNIX** `*.sh`
`createTables.sql`	SQL script. Input file to the `CreateDataBaseTables` command file. Creates the `Billing` table and the `MESSAGECOST` table.	SQL scripts are platform independent.
`populateTables.sql`	SQL script. Input file to the `CreateDataBaseTables` command file. Inserts data into the `MESSAGECOST` table.	SQL scripts are platform independent.

Setting Up the CheckAccount Logic Plug-In

You can apply the CheckAccount logic plug-in to an existing example application such as the `verifier` example or the `wlpiverifier` example. The procedure here applies the logic plug-in to the `verifier` example.

The next two sections summarize the setup steps on Windows and UNIX. The "CheckAccount Setup Steps in Detail" section describes the same steps in more detail.

Setup on Windows: Main Steps

This section summarizes the steps for enabling the CheckAccount logic plug-in in an application. See "CheckAccount Logic Plug-In Setup Steps in Detail" on page 4-21 for more information about each step.

1. Run the `setenv.cmd` command file.
2. Customize the `Bulkload`*`database`*`Config.xml` file, where *`database`* identifies the database you are using.
3. Change to the `%WLC_HOME%\LogicPlugIns\CheckAccount` directory.
4. From the `CheckAccount` directory, execute this command:
   `bulkloader ..\..\Bulkload`*`database`*`Config.xml`
5. Customize the `CreateDataBaseTables` command file.
6. Run the `CreateDataBaseTables.cmd` file.
7. Run the `build.cmd` command file.
8. Customize the `c-hub weblogic.properties` configuration file, enabling the `Java servlet` and `JDBC Connection Pool` sections.
9. Ensure that your environment is defined, and start the c-hub.

Setup on UNIX: Main Steps

See "CheckAccount Logic Plug-In Setup Steps in Detail" on page 4-21 for more detailed information about each step.

1. Run the `setenv.sh` shell.

2. Customize the Bulkload*database*Config.xml file, where *database* identifies the relational database you are using.
3. Change to this directory:
   $WLC_HOME/examples/LogicPlugIns/CheckAccount
4. From the CheckAccount directory, execute this command:
   bulkloader.sh ../../Bulkload*database*Config.xml
5. Customize the CreateDataBaseTables.sh shell.
6. Run the CreateDataBaseTables.sh shell.
7. Run the build.sh shell.
8. Customize the c-hub weblogic.properties configuration file, enabling the Java servlet and JDBC Connection Pool sections.
9. Ensure that your environment is set, and start the c-hub.

CheckAccount Logic Plug-In Setup Steps in Detail

The following procedure explains each step. The numbering corresponds to the numbering in the main steps sections.

1. Set the environment.

   Run the setenv.cmd command file (Windows), or the setenv.sh shell (UNIX). This step establishes the environment for your system.

2. Customize the Bulk Loader XML configuration file.

   Customize the configuration file corresponding to the database you are using:

   - BulkloadCloudscapeConfig.xml
   - BulkloadMSSQLConfig.xml
   - BulkloadOracleConfig.xml

   This XML file is an input file to the Bulk Loader utility which you invoke in step 4 of this series of steps. You must define the database connectivity variables correctly before populating the repository, creating database tables, and populating database tables.

3. Change to the CheckAccount directory.

Change to the `CheckAccount` directory, and run subsequent steps from this location.

Windows

```
%WLC_HOME%\examples\LogicPlugIns\CheckAccount
```

UNIX

```
$WLC_HOME/examples/LogicPlugIns/CheckAccount
```

4. Populate the WebLogic Collaborate repository.

   From the `CheckAccount` directory, run the Bulk Loader utility, supplying the correct path to the input XML configuration file. Additionally, this step reads the `RepData.xml` file as input.

   Windows

   Syntax:

   ```
   bulkloader ..\..\Bulkload<i>database</i>Config.xml
   ```

   Example:

   ```
   bulkloader ..\..\BulkloadOracleConfig.xml
   ```

   UNIX

   Syntax:

   ```
   bulkloader.sh ../../Bulkload<i>database</i>Config.xml
   ```

   Example:

   ```
   bulkloader.sh ../../BulkloadCloudscapeConfig.xml
   ```

5. Customize the `CreateDataBaseTables` command file in the `DataBase` directory.

   Windows

   ```
   CheckAccount\DataBase\CreateDataBaseTables.cmd
   ```

   UNIX

   ```
   CheckAccount/DataBase/CreateDataBaseTables.sh
   ```

   The values for the connection parameters must match those specified in the `JDBC examplesPool` in the `weblogic.properties` file. Modify the connection values required for your underlying database. The user name you specify must be an existing user with authority to create tables in the database.

6. Create the database tables and populate with data.

   To create the `MESSAGECOST` and `Billing` database tables, run the command file you customized in the previous step. The database tables are external to the repository. You supply one input parameter to the command file, identifying the subdirectory for your database. The command file reads the `createTables.sql` and `populateTables.sql` script files in the subordinate directory.

   As supplied, the `populateTables.sql` script inserts two rows of data into the `MESSAGECOST` table. These `insert` statements are from the script:

   ```
   insert into MESSAGECOST values ('verifierConversation',20.37);
   insert into MESSAGECOST values ('to-upper',3);
   ```

   To change the cost for each message in a conversation, in the `populateTables.sql` script, modify the numeric value to be inserted into the `MESSAGE_COST` column. For example, to change the cost of a message in the `verifierConversation` conversation from the supplied value of 20.37, modify the numeric value in the first insert statement.

   Windows

   Syntax:

   ```
   CreateDataBaseTables { oracle | cloudscape | mssql }
   ```

   Example:

   ```
   CreateDataBaseTables oracle
   ```

   UNIX

   Syntax:

   ```
   CreateDataBaseTables.sh { oracle | cloudscape }
   ```

   Example:

   ```
   CreateDataBaseTables.sh cloudscape
   ```

7. Create the executable programs for this example.

   Run the build command file for your platform. This step compiles required Java source files and copies resulting files to target locations.

   Windows

   ```
   build.cmd
   ```

   UNIX

   ```
   build.sh
   ```

You can vary the threshold for account balances due. Once the outstanding balance due for a trading partner is above the threshold, custom code in the logic plug-in denies access to the c-hub for this trading partner. To change the threshold from the default value of 100, modify the value of the `MAXLIMIT` variable in the `CheckAccount.java` source file and run the build step again. This is a code fragment from the `CheckAccount.java` source file:

```
public void process(MessageEnvelope mEnv) throws PlugInException {
   final float MAXLIMIT = 100;
   String sender, conversation ;
   String tRecipients[];
   Connection conn = null;
   float debit = 0;
   Message bMsg = null;

   // Gets the Business Message from the Message Envelope
```

8. Using a text editor, uncomment the following sections of the c-hub `weblogic.properties` configuration file specific to the logic plug-in examples:

   a. The `Logic Plug-Ins servlets` section. This section is identified by the following section header, which appears before the lines that you need to uncomment:

```
#########################################################################
# Servlets used for the WLC Logic PlugIns example
# Uncomment to use
# The Logic PlugIns require the Verifier example to be working
#########################################################################
```

   b. The section that specifies the data source to be used. This section is identified by the following section header:

```
#########################################################################
#Data Source used for the Logic PlugIns examples
#Uncomment it whatever database you use
#########################################################################
```

   c. The section that specifies the JDBC connection pool specific to the database you are using. For example, if you are using Oracle, this section is identified by the following section header:

```
#########################################################################
# Oracle 8.1.5
# Note: EXAMPLE_USER, EXAMPLE_PASSWORD must match the values used in the
# CreateDatabaseTables.cmd(sh) command
#########################################################################
```

If you are using Oracle, be sure to specify values for *<EXAMPLE_USER>* and *<EXAMPLE_PASSWD>* that match the values used in the `weblogic.properties.verifier CreateDatabaseTables` script.

If you are using MS SQL server, be sure to specify values for *<MSSQL_USER>*, *<MSSQL_PASSWORD>*, and *<MSSQL_HOSTNAME>* that match your MS SQL server installation.

d. The section that represents ACLs for the examplesPool. This section is identified by the following section header:

```
######################################################################
# The following three lines represent ACLs for the examplesPool and
# must be un-commented independently of the database type used
######################################################################
```

To uncomment the appropriate lines in each of these sections, simply delete the comment character (#) at the beginning of each line in those sections. For example, the section that represents ACLs for the examplesPool should appear as:

```
weblogic.allow.reserve.weblogic.jdbc.connectionPool.examplesPool=everyone
weblogic.allow.reset.weblogic.jdbc.connectionPool.examplesPool=everyone
weblogic.allow.shrink.weblogic.jdbc.connectionPool.examplesPool=everyone
```

9. Start the example application.

   Ensure that your environment is defined correctly, and start the c-hub.

Running an Application with the CheckAccount Logic Plug-In

Follow these steps to observe how the CheckAccount logic plug-in adds value to an underlying application.

1. Start the `verifier` example application, or the alternative example application you configured for the logic plug-in.

   Note: The CheckAccount logic plug-in requires that the target recipient list not be empty. If the list is empty when you implement this logic plug-in, the conversation will hang until it times out. You can change the timeout period for a conversation.

2. Examine the WebLogic system log for messages from the CheckAccount logic plug-in.

   The CheckAccount logic plug-in writes a message to the log each time the c-hub processes a business document. Look in the log for messages similar to these:

```
** CheckAccount ##### Recipient 1 = Partner2
** CheckAccount ##### Debit for Partner2 = 10.0
** CheckAccount ##### The recipient has not exceeded MAXLIMIT
** CheckAccount ##### Recipient has been removed because
      MAXLIMIT has been exceeded
```

3. Use your Web browser to inquire about the status of a trading partner account. Set your browser to this address:

   `http://localhost:7001/examples/LogicPlugIns/LogicPlugIns.htm`

   Enter the name of a trading partner and click on the `Submit` button. This Java servlet utility returns information about the trading partner.

   Note: The valid names for the two trading partners are `'Partner1'` and `'Partner2'`. These names are case-sensitive, and do include the single-quote characters.

4. Continuing with the `LogicPlugIns.htm` tool, reset the trading partner balance due back to zero by clicking the `Reset` button.

   If you have set up the MessageCounter logic plug-in, the CheckAccount logic plug-in processes database information recorded by both plug-ins.

Index

D

E

F

J

L

W

X