BEA WebLogic Collaborate

A Component of BEA WebLogic Integration

Programming BEA WebLogic Collaborate Logic Plug-Ins

BEA WebLogic Collaborate Release 2.0
Document Edition 2.0
July 2001

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Programming BEA WebLogic Collaborate Logic Plug-Ins

Document Edition	Date	Software Version
2.0	July 2001	2.0

Contents

About This Document

This document describes how to develop applications to exchange business messages and monitor run-time activities in the BEA WebLogic Collaborate™ system.

This document is organized as follows:

- Chapter 1, “Overview,” provides an introduction to developing applications for the BEA WebLogic Collaborate environment.
- Chapter 2, “Routing and Filtering Business Messages,” describes how routing and filtering work in the BEA WebLogic Collaborate environment.
- Chapter 3, “Creating and Adding Plug-Ins,” describes how to manipulate business messages as they travel through WebLogic Collaborate.

What You Need to Know

This document is intended primarily for:

- Business process designers who use the WebLogic Process Integrator Studio to design workflows that can be integrated with the BEA WebLogic Collaborate environment.
- Application developers who write Java applications that manage the exchange of business messages or monitor run-time statistics in the BEA WebLogic Collaborate environment.
- System administrators who set up and administer BEA WebLogic Collaborate applications.

For an overview of the BEA WebLogic Collaborate architecture, see *Introducing BEA WebLogic Collaborate*.

e-docs Web Site

BEA product documentation is available at the following location:

`http://e-docs.bea.com`

How to Print this Document

You are reading the PDF version of this document, either online or a printout. You can print the entire document or any portion of the document from Adobe Acrobat Reader. If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at the following location:

`http://www.adobe.com`

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

Related Information

For more information about Java 2 Enterprise Edition (J2EE), Extended Markup Language (XML), and Java programming, see the Javasoft Web site at the following URL:

`http://java.sun.com`

Contact Us!

Your feedback about the 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 WebLogic Collaborate documentation.

In your e-mail message, please indicate that you are using the documentation for Release 2.0 of WebLogic Collaborate.

If you have any questions about this version of WebLogic Collaborate, or if you have problems installing and running WebLogic Collaborate, contact BEA Customer Support through BEA WebSupport at the following location:

`http://www.bea.com`

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
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.

Convention	Item
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 boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `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.

Convention	Item
. . .	Indicates one of the following in a command line: ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name] [-f 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 following sections provide an overview of programming logic plug-ins:

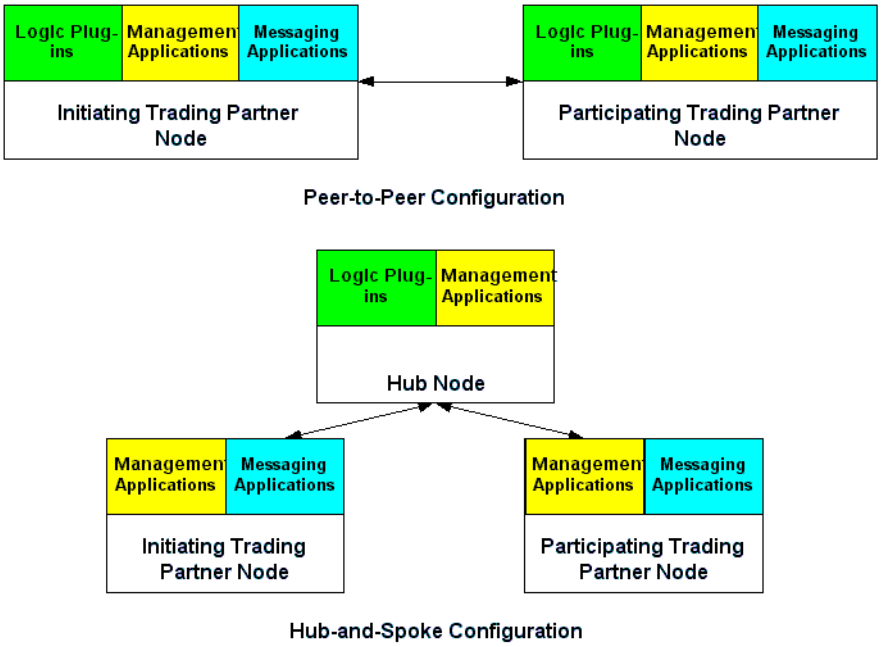
- Types of Applications
- Logic Plug-Ins

Types of Applications

This document introduces WebLogic Collaborate logic plug-ins. Plug-ins are one of three types of component applications available within WebLogic Collaborate. In addition to logic plug-ins, WebLogic Collaborate also allows you to use management applications, based on BEA-implemented MBeans, and messaging applications.

The following figure shows where these types of applications reside in the WebLogic Collaborate system. For more information about management and messaging applications, see *Programming BEA WebLogic Collaborate Management Applications* and *Programming BEA WebLogic Collaborate Messaging Applications*.

Figure 1-1 Types of WebLogic Collaborate Applications



For an introduction to the WebLogic Collaborate system, see *Installing BEA WebLogic Collaborate*.

Logic Plug-Ins

Logic plug-ins are Java classes that perform specialized processing of business messages as they pass through a node. Logic plug-ins insert rules and business logic at strategic locations along the path that business messages travel as they make their way through the node. WebLogic Collaborate provides router and filter logic plug-ins for each business protocol. A service provider or trading partner can develop and install custom logic plug-ins on a node to provide additional value in hub-and-spoke configuration for node management and for trading partners who use that node.

Logic plug-ins are stored and executed on a node and are defined in the WebLogic Collaborate repository on that node. They are transparent to users.

For more information about logic plug-ins, see Chapter 3, “Creating and Adding Plug-Ins.”

CHAPTER

2 Routing and Filtering Business Messages

The following sections describe how to use routing, filtering, and Xpath expressions to control the flow of business messages exchanged among trading partners using BEA WebLogic Collaborate:

- Run-Time Message Processing
- Working with Message-Context Documents
- Working with XPath Expressions

Run-Time Message Processing

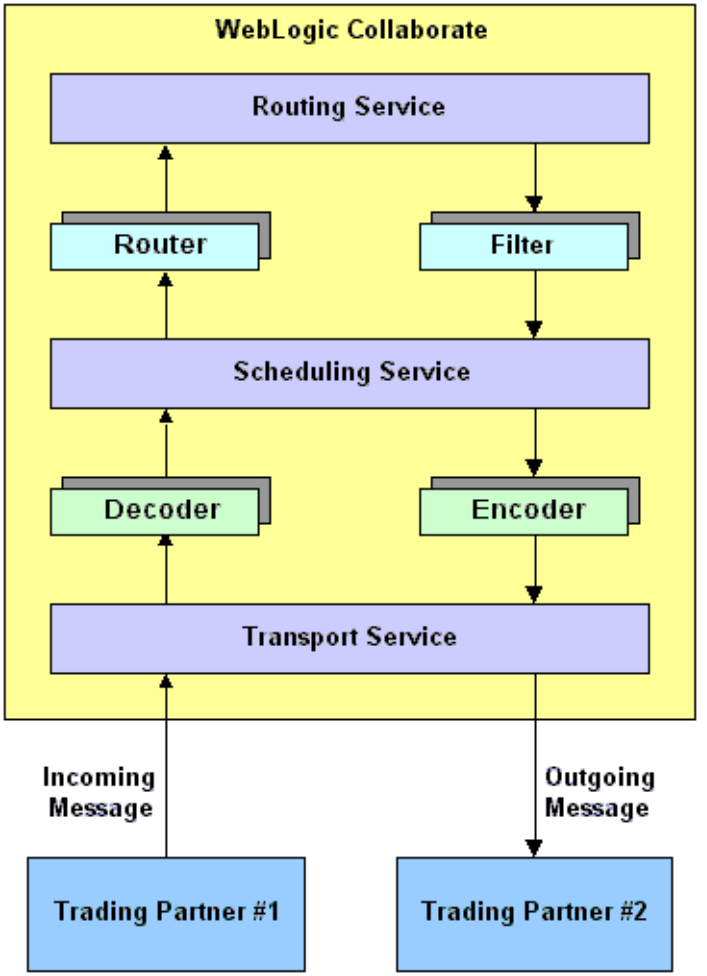
BEA WebLogic Collaborate uses logic plug-ins, acting as either routers or filters, to direct the flow of business messages to trading partners. The following example illustrates how this process is implemented for XOCP business messages.

- After a trading partner sends an XOCP business message to WebLogic Collaborate, the logic plug-in, acting as an *XOCP router,* determines the trading partners to which the message is sent. The router logic plug-in is on the *send* side of WebLogic Collaborate message processing and determines the recipients to which the sending trading partner intends to send the message.
- Before WebLogic Collaborate sends the business message to a recipient trading partner, a second logic plug-in, acting as an *XOCP filter,* determines whether or not the trading partner should receive it. The second logic plug-in is on the

receive side of WebLogic Collaborate message processing and it can prevent a specific trading partner from receiving a specific business message.

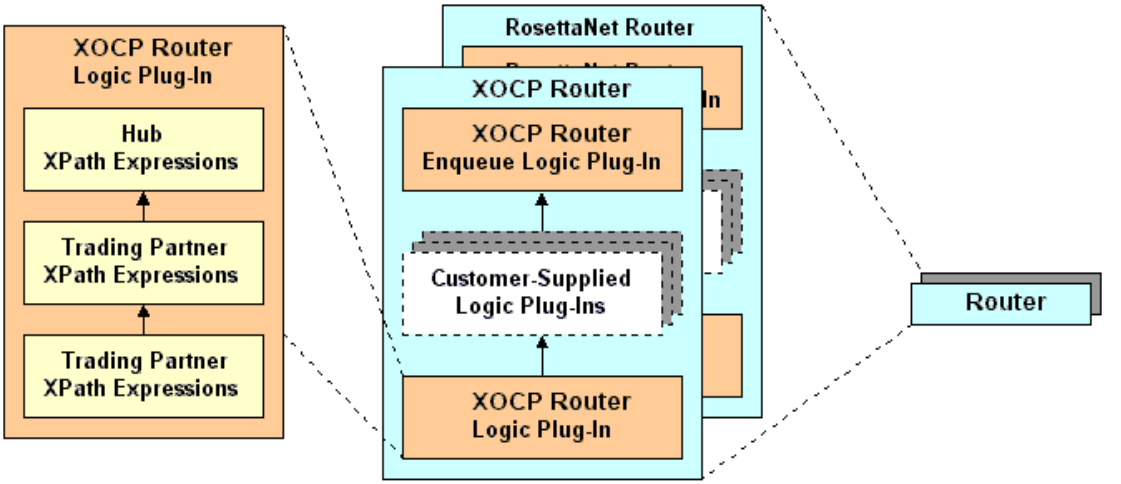
The following figure provides an overview of how WebLogic Collaborate processes a message.

Figure 2-1 Overview of Message Processing



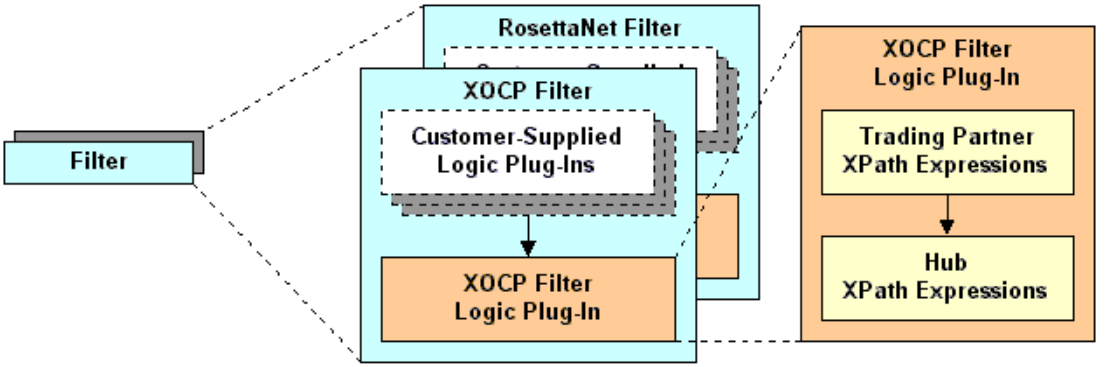
A router is provided for each business protocol that WebLogic Collaborate supports. The following figure provides a detailed look at the routers.

Figure 2-2 WebLogic Collaborate Routers



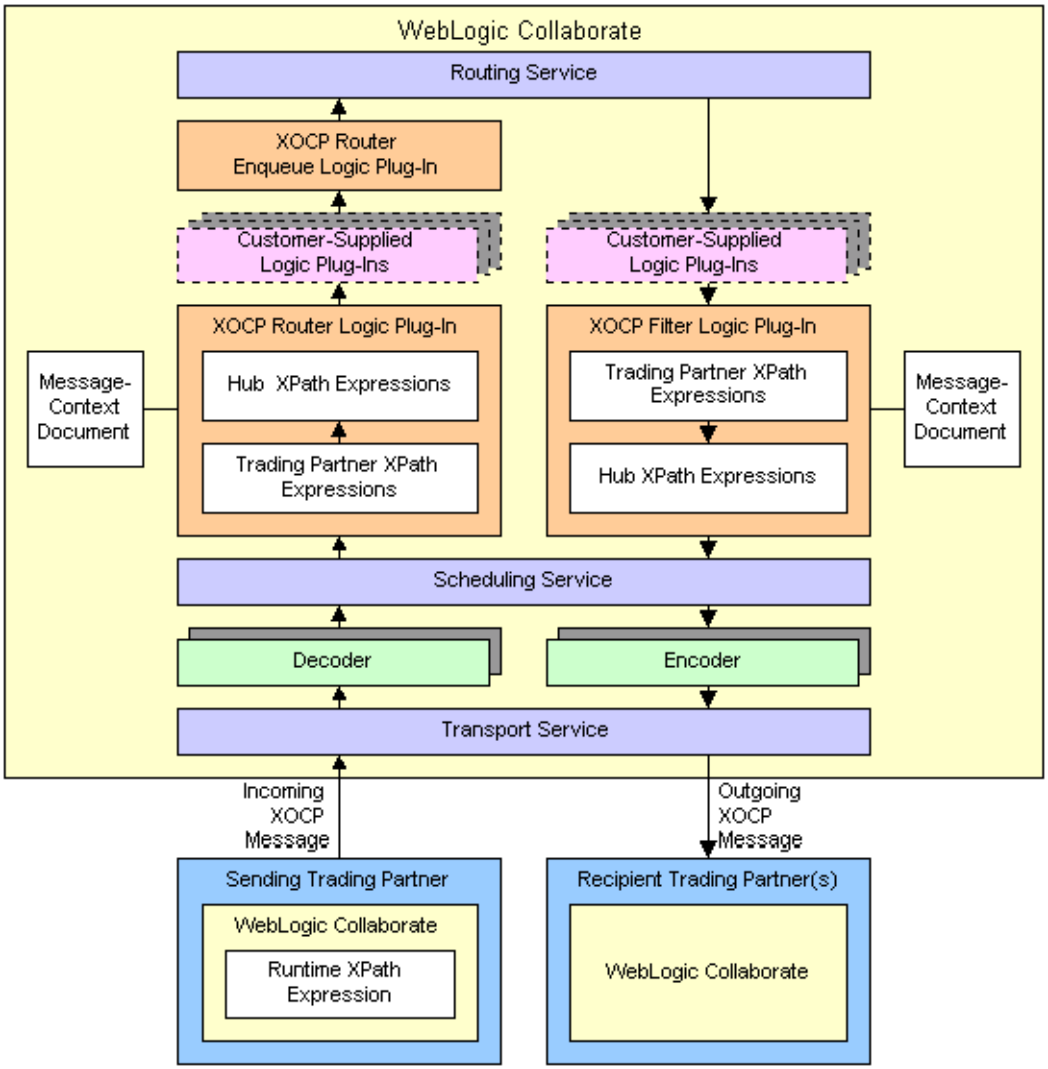
A separate filter is provided for each business protocol that WebLogic Collaborate supports. The following figure provides a detailed look at the filters.

Figure 2-3 WebLogic Collaborate Filters



The following figure provides a detailed look at how WebLogic Collaborate processes an XOCP business message. Processing for RosettaNet business messages is handled in a similar manner, as discussed in *Implementing RosettaNet for BEA WebLogic Collaborate*.

Figure 2-4 XOCP Message Processing



The following sections explain how the *send* and *receive* sides of WebLogic Collaborate process an XOCP business message:

- Send Side
- Receive Side

Send Side

The following sections describe the components on the *send* side of WebLogic Collaborate and explain how they process an XOCP business message:

- WebLogic Collaborate XPath Expression
- Transport Service
- Decoder
- Scheduling Service
- XOCP Router
- Routing Service

WebLogic Collaborate XPath Expression

When sending an XOCP business message, the WebLogic Collaborate instance for the sending trading partner can specify a local XPath expression that defines the intended recipients for the business message. The local XPath expression is defined in a WebLogic Process Integrator workflow or in a locally-run WebLogic Collaborate application. For more information about XPath expressions, see “Creating WebLogic Collaborate XPath Expressions” on page 2-18.

Transport Service

The transport service reads the incoming XOCP business message and does the following:

1. Wraps the message in a message envelope to expedite processing as it travels through the WebLogic Collaborate instance.
2. Forwards the message to the appropriate decoder based on the business protocol, such as XOCP, RosettaNet, or cXML. The URL at which the transport service receives the message identifies the protocol and the delivery channel. Each delivery channel/business protocol combination has a unique URL. A trading partner uses this URL to access a particular delivery channel using a particular business protocol.

   Warning: A URL for a delivery channel/business protocol combination can be used only by the WebLogic Collaborate instance. If customer-supplied software uses one of these URLs, messages are not processed correctly.

   For information about configuring business protocols, see *Administering BEA WebLogic Collaborate*.

Decoder

The decoder does the following:

1. Processes the protocol-specific headers.
2. Identifies the sending trading partner.
3. Enlists the sending trading partner in a conversation.
4. Prepares a reply to return to the sender.
5. Forwards the message to the scheduling service.

Scheduling Service

The scheduling service enqueues the message to store it for subsequent retrieval by the XOCP router.

XOCP Router

The XOCP router is a chain of logic plug-ins that specifies the recipients for the XOCP business message. Each logic plug-in can add trading partners to or remove trading partners from the set of recipient trading partners.

The logic plug-ins in the XOCP router are arranged in the following order:

1. XOCP router logic plug-in—provided by WebLogic Collaborate
2. Customer-supplied logic plug-ins—optional logic plug-ins that you can create
3. XOCP router enqueue logic plug-in—provided by WebLogic Collaborate

The following sections describe these logic plug-ins.

XOCP Router Logic Plug-In

The XOCP router logic plug-in does the following:

1. Creates a message-context document.

   A message-context document is an XML document that the XOCP router logic plug-in generates from the XOCP business message and associated information in the repository. The message-context document describes header and content information about the XOCP business message, such as the hub, business protocol, conversation, sending trading partner, and receiving trading partners. The XOCP router logic plug-in uses XPath expressions to evaluate the message-context document. For more information about message-context documents, see “Working with Message-Context Documents” on page 2-13.

2. Evaluates the message-context document against the XPath routing expressions, which can refer to values in the message-context document. This evaluation results in a set of trading partners that are targeted to receive the XOCP business message.

   The XOCP router logic plug-in uses the XPath routing expressions in the following order:

   a. Sender WebLogic Collaborate instance XPath expression

      For information about WebLogic Collaborate XPath expressions, see “Working with XPath Expressions” on page 2-14 and “Creating WebLogic Collaborate XPath Expressions” on page 2-18.

b. Sequence of sending trading partner XPath routing expressions

These XPath routing expressions are defined in the repository and are defined for the sending trading partner. Each trading partner XPath routing expression applies to all XOCP business messages that WebLogic Collaborate receives from a particular sending trading partner. Each sending trading partner can have multiple trading partner XPath routing expressions.

Each trading partner XPath routing expression can examine different parts of the message-context document and select a different set of recipient trading partners. The trading partners produced by each expression can either replace the previously generated set of recipient trading partners or add to the current set.

c. Sequence of WebLogic Collaborate XPath routing expressions

These XPath routing expressions are defined in the repository and are defined for WebLogic Collaborate. Each WebLogic Collaborate XPath routing expression applies to all XOCP business messages that WebLogic Collaborate receives for all sending trading partners. WebLogic Collaborate can have multiple XPath routing expressions.

As with trading partner XPath routing expressions, each WebLogic Collaborate XPath routing expression can examine different parts of the message-context document and return a Boolean result that indicates acceptance or rejection of the message.

You can add XPath expressions to the repository for use by the XOCP router logic plug-in. For information about XPath expressions, see "Working with XPath Expressions" on page 2-14.

3. Discards the message-context document.

4. If the set of recipient trading partners is empty, then the XOCP router logic plug-in does not forward the message to the next component in WebLogic Collaborate. Otherwise, the WebLogic Collaborate instance continues to process the message.

Customer-Supplied Logic Plug-Ins

You can create logic plug-ins and add them to the XOCP router. If you create a new logic plug-in, you must add it to the chain after the XOCP router logic plug-in and before the XOCP router enqueue logic plug-in. The order of the logic plug-ins in the XOCP router chain is specified in the XOCP business protocol definition.

A customer-supplied logic plug-in does not have to provide router functionality to be part of the XOCP router. For example, a customer-supplied logic plug-in can provide billing functionality by keeping track of the number of messages sent by a particular sending trading partner and then billing the trading partner for those messages. Even when a customer-supplied logic plug-in does not provide routing or filtering functionality, it can be added only to the XOCP router or the XOCP filter. For more information about logic plug-ins, see Chapter 3, "Creating and Adding Plug-Ins."

If the set of recipient trading partners is empty after the processing performed by a customer-supplied router logic plug-in, then the customer-supplied router logic plug-in does not forward the message to the next component in WebLogic Collaborate. Otherwise, WebLogic Collaborate continues to process the message.

XOCP Router Enqueue Logic Plug-In

The XOCP router enqueue logic plug-in does the following:

1. Enqueues the XOCP business message along with the intended recipients.
2. Forwards the message to the routing service.

Routing Service

The routing service does the following:

1. Performs the final validation of the message recipients.
2. Creates a separate message envelope for each validated recipient trading partner.
3. Forwards each copy of the message envelope to the XOCP filter.

Receive Side

The following sections describe the components on the *receive* side of WebLogic Collaborate and explain how they process an XOCP business message:

- XOCP Filter
- Scheduling Service

- Encoder
- Transport Service

XOCP Filter

The XOCP filter is a chain of logic plug-ins that determines whether or not to send an XOCP business message to an intended recipient. These logic plug-ins are evaluated after the XOCP router logic plug-ins; they can modify or override the XOCP router results. Each logic plug-in can determine not to send the message.

The logic plug-ins in the XOCP filter are arranged in the following order:

1. Customer-supplied logic plug-ins—optional logic plug-ins that you can create
2. XOCP filter logic plug-in—provided by WebLogic Collaborate

The following sections describe these logic plug-ins.

XOCP Filter Logic Plug-In

The XOCP filter logic plug-in does the following:

1. Creates a message-context document.

   A message-context document is an XML document that the XOCP filter logic plug-in generates from the XOCP business message and associated information in the repository. The message-context document describes header and content information about the XOCP business message, such as the hub, business protocol, conversation, sending trading partner, and receiving trading partners. The XOCP filter logic plug-in uses XPath expressions to evaluate the message-context document. For more information about message-context documents, see “Working with Message-Context Documents” on page 2-13.

2. Evaluates the message-context document against the XPath filtering expressions, which can refer to values in the message-context document. This evaluation determines whether or not to send the message to the intended recipient.

The XOCP filter logic plug-in uses the XPath filtering expressions in the following order:

a. Sequence of trading partner XPath filtering expressions.

   These XPath filtering expressions are defined in the repository and are defined for the recipient trading partner. Each trading partner XPath filtering expression applies to all XOCP business messages that WebLogic Collaborate receives for a particular recipient trading partner. Each recipient trading partner can have multiple trading partner XPath filtering expressions.

   Each trading partner XPath filtering expression can examine different parts of the message-context document and return a Boolean result that indicates acceptance or rejection of the message.

b. Sequence of WebLogic Collaborate XPath filtering expressions

   These XPath filtering expressions are defined in the repository and are defined for WebLogic Collaborate. Each WebLogic Collaborate XPath filtering expression applies to all XOCP business messages that WebLogic Collaborate receives for all recipient trading partners. WebLogic Collaborate can have multiple XPath filtering expressions.

   As with trading partner XPath filtering expressions, each WebLogic Collaborate XPath filtering expression can examine different parts of the message-context document and return a Boolean result that indicates acceptance or rejection of the message.

You can add XPath expressions to the repository for use by the XOCP filter logic plug-in. For information about XPath expressions, see "Working with XPath Expressions" on page 2-14.

3. Discards the message-context document.

4. If the XOCP filter logic plug-in cancels delivery of the XOCP business message to the intended recipient, then the XOCP filter logic plug-in does not forward the message to the next component in the WebLogic Collaborate instance. Otherwise, WebLogic Collaborate continues to process the message.

Customer-Supplied Logic Plug-Ins

You can create logic plug-ins and add them to the XOCP filter. If you create a new logic plug-in, you must add it to the chain before the XOCP filter logic plug-in. The order of the logic plug-ins in the XOCP filter chain is specified in the XOCP business protocol definition.

A customer-supplied logic plug-in does not have to provide filter functionality to be part of the XOCP filter. For example, a customer-supplied logic plug-in can provide sampling functionality by keeping track of the types of messages sent to a particular recipient trading partner. Even when a customer-supplied logic plug-in does not provide routing or filtering functionality, it can be added only to the XOCP router or the XOCP filter. For more information about logic plug-ins, see Chapter 3, "Creating and Adding Plug-Ins."

If the customer-supplied logic plug-ins cancel delivery of the XOCP business message to the intended recipient, then the customer-supplied filter logic plug-in does not forward the message to the next component in the WebLogic Collaborate instance. Otherwise, WebLogic Collaborate continues to process the message.

Scheduling Service

The scheduling service does the following:

1. Performs additional internal operations related to quality of service issues and conversation management. For information about quality of service, see *Programming BEA WebLogic Collaborate Messaging Applications*.
2. Forwards the message to the encoder.

Encoder

The encoder transforms the message as necessary to support the business protocol and forwards the message to the transport service.

Transport Service

The transport service sends the message to the recipient.

Working with Message-Context Documents

For information about how message-context documents are created and used, see “XOCP Router Logic Plug-In” on page 2-7 and “XOCP Filter Logic Plug-In” on page 2-10. This section describes the DTD for the message-context document.

The following listing is the Document Type Definition (DTD) for the message-context document.

Listing 2-1 Document Type Definition for Message-Context Document

```
<!--Copyright (c) 2001 BEA Systems, Inc. -->
<!--All rights reserved                  -->

<!-- This DTD describes the message context document for XPATH routers and filters
-->

<!ELEMENT wlc (business-protocol, conversation, sender, trading-partner+) >
<!ATTLIST wlc context ( message-router | trading-partner-router | hub-router |
trading-partner-filter | hub-filter ) #REQUIRED >

<!ELEMENT business-protocol EMPTY >
<!ATTLIST business-protocol name CDATA #REQUIRED >
<!ATTLIST business-protocol version  CDATA #REQUIRED >

<!ELEMENT conversation EMPTY >
<!ATTLIST conversation name CDATA #REQUIRED >
<!ATTLIST conversation version CDATA #REQUIRED >
<!ATTLIST conversation sender-role CDATA #REQUIRED >
<!ATTLIST conversation receiver-role CDATA #REQUIRED >

<!-- A sender is a trading-partner that has sent a message from a role in a
conversation. -->
<!ELEMENT sender ( trading-partner ) >

<!-- A Trading Partner represents an entity such as a company that sends or
receives messages. -->
<!ELEMENT trading-partner ( address, extended-property-set* ) >
<!ATTLIST trading-partner email CDATA #IMPLIED >
<!ATTLIST trading-partner fax CDATA #IMPLIED >
<!ATTLIST trading-partner name ID #REQUIRED >
<!ATTLIST trading-partner phone CDATA #IMPLIED >
```

```
<!ELEMENT address ANY >

<!ELEMENT extended-property-set ANY >
<!ATTLIST extended-property-set name CDATA #REQUIRED >
```

Working with XPath Expressions

This section describes XPath expressions and how to create them:

- About XPath Expressions
- Creating WebLogic Collaborate XPath Expressions
- Creating Trading Partner XPath Expressions
- Creating WebLogic Collaborate XPath Expressions

About XPath Expressions

XPath is the XML path language that is defined by the World Wide Web Consortium. The XOCP router logic plug-in and the XOCP filter logic plug-in use XPath expressions to evaluate message-context documents. You can add XPath expressions to the repository for use by the XOCP router logic plug-in and the XOCP filter logic plug-in.

XPath expressions in the XOCP router logic plug-in and XOCP filter logic plug-in perform the following functions:

- An XPath routing expression uses the XPath syntax to select a set of trading partners from the message-context document. These trading partners are the intended recipients of the XOCP business message. Each XPath routing expression must evaluate to a set of trading partners.

  In the XOCP router logic plug-in, XPath expressions specify the business criteria for message distribution. For example, a buyer can use an XPath routing expression to send bid requests to all sellers in a particular area code or to sellers that can handle large orders.

- An XPath filtering expression uses the XPath syntax to return a Boolean result that indicates acceptance or rejection of the message. Each XPath filtering expression must evaluate to a Boolean `true` or `false` result.

  In the XOCP filter logic plug-in, XPath expressions determine whether or not WebLogic Collaborate sends a particular business message to a particular trading partner. An XPath filtering expression in the XOCP filter logic plug-in acts as a gatekeeper that filters out unwanted business messages for a receiving trading partner.

The following table provides an overview of the various types of XPath expressions.

Table 2-1 Overview of Types of XPath Expressions

Type of XPath Expression	XOCP Router Logic Plug-In	XOCP Filter Logic Plug-In
Run-Time	Evaluated: *first* # of XPath expressions: one Defined by: WebLogic Collaborate instance (in workflow or application) Purpose: defines recipients Applies to: XOCP business messages from the sending WebLogic Collaborate instance	Not applicable
Trading partner	Evaluated: *second* # of XPath expressions: one or more Defined in: repository (via Administration Console or Bulk Loader) Purpose: adds and removes recipients Applies to: all XOCP business messages from the sending trading partner	Evaluated: *fourth* # of XPath expressions: one or more Defined in: repository (via Administration Console or Bulk Loader) Purpose: determines whether or not to send the message to the recipient Applies to: all XOCP business messages to the recipient trading partner
XOCP hub	Evaluated: *third* # of XPath expressions: one or more Defined in: repository (via Administration Console or Bulk Loader) Purpose: adds and removes recipients Applies to: all XOCP business messages from all sending trading partners	Evaluated: *fifth* # of XPath expressions: one or more Defined in: repository (via Administration Console or Bulk Loader) Purpose: determines whether or not to send the message to the recipient Applies to: all XOCP business messages to all recipient trading partners

In the XOCP router logic plug-in, each XPath routing expression can examine different parts of the message-context document and select a different set of recipient trading partners. The trading partners produced by each expression can either replace the previously generated set of recipient trading partners or add to the current set.

The following table steps through an example that shows how XPath routing expressions can be used.

Table 2-2 Example for XPath Routing Expressions

XPath Expression	Resulting Set of Recipient Trading Partners
1. The XOCP spoke XPath expression selects trading partners A and B.	A, B
2. A trading partner XPath routing expression adds trading partner C.	A, B, C
3. A subsequent trading partner XPath routing expression replaces all previously selected trading partners with trading partner D.	D
4. An XOCP hub XPath routing expression adds trading partners B and F.	D, B, F
5. A subsequent XOCP hub XPath routing expression removes trading partner F.	D, B

In the XOCP filter logic plug-in, each XPath filtering expression can examine different parts of the message-context document to determine whether or not to forward the message to the recipient trading partner. Each XPath filtering expression can return `true` or `false` using different selection criteria. After an XPath filtering expression returns `false`, the message is blocked from further evaluation and is not sent to the intended recipient.

An XPath expression can refer to the following kinds of information:

- Trading partner attributes, including:
  - Standard attributes, such as the trading partner name or a postal code
  - Extended attributes, which are custom attributes defined by the WebLogic Collaborate administrator
- Message information, such as the type of business document, a purchase order number, or an invoice amount

For more information on XPath Expressions, see *Administering BEA WebLogic Collaborate*.

Creating WebLogic Collaborate XPath Expressions

When sending an XOCP business message, the WebLogic Collaborate instance for the sending trading partner can specify a WebLogic Collaborate XPath expression that defines the intended recipients for the business message. The WebLogic Collaborate XPath expression is defined in a WebLogic Process Integrator workflow or in a WebLogic Collaborate application. This XPath expression selects a subset of `<trading-partner>` nodes from the message-context XML document that the XOCP router logic plug-in generates.

The sending WebLogic Collaborate instance defines this XPath expression and sends it to the XOCP hub along with the message. WebLogic Collaborate defines an XPath expression as follows:

- If WebLogic Collaborate uses a workflow to exchange business messages, the XPath expression is defined in the workflow definition template and applied when the WebLogic Collaborate instance sends the message to another WebLogic Collaborate instance. Use the Send Business Message dialog to define the XPath expression. For more information, see *Creating Workflows for BEA WebLogic Collaborate*.
- If WebLogic Collaborate uses a WebLogic Collaborate application to exchange business messages, the XPath expression is defined in the WebLogic Collaborate application when it sends the message to another WebLogic Collaborate instance. Call the `setExpression` method on the `com.bea.b2b.protocol.messaging.Message` instance, passing the XPath expression as the parameter. For more information, see *Creating Workflows for BEA WebLogic Collaborate*.

  Note: In many cases, a WebLogic Collaborate application sends a business message to a single, known trading partner; for example, when replying to a request from that trading partner. In this case, a WebLogic Collaborate application can bypass the evaluation of XPath expressions in the XOCP router logic plug-in by specifying a trading partner name instead of an XPath expression. To specify a trading partner name, call the `setRecipient` method instead of `setExpression` on the `com.bea.b2b.protocol.messaging.Message` instance.

Creating Trading Partner XPath Expressions

A trading partner XPath expression is an XPath expression that is defined for a trading partner. For routing, a trading partner XPath expression is used by the XOCP router logic plug-in and is defined for the sending trading partner. For filtering, a trading partner XPath expression is used by the XOCP filter logic plug-in and is defined for the receiving trading partner.

Trading partner XPath expressions are defined in the repository. You can use the following tools to create trading partner XPath expressions for the XOCP router logic plug-in and the XOCP filter logic plug-in:

- Bulk Loader as described in "Working with the Bulk Loader" in *Administering BEA WebLogic Collaborate*. The format for an XPath expression in a repository data file is:

  ```
  <xpath-expression expression="//TradingPartner1"
  location="ROUTER" type="APPEND"/>
  ```

  For more information about XPath syntax and usage, see the "XML Path Language Specification," published by the World Wide Web Consortium, at the following URL:

  `http://www.w3.org/TR/xpath.html`

- WebLogic Collaborate Administration Console as described in the *BEA WebLogic Collaborate Administration Console Online Help*.

  The following table describes the properties that you set when using the WebLogic Collaborate Administration Console to define XPath expressions for trading partners and the WebLogic Collaborate instance.

Table 2-3 Properties for XPath Expressions

Component	Description
XPath Expression	XPath routing or filtering expression as previously described.
Type	Flag that specifies whether the results of evaluating the XPath expression append or replace the results of the evaluations of the previous XPath expressions.

As another example, a trading partner might want to route requests to trading partners that are located in California. To do this, the trading partner can use the detail window on the Trading Partners tab in the WebLogic Collaborate Administration Console to create the following XPath expression for the XOCP router logic plug-in:

```
/hub/trading-partner[extended-property-set/state='California']
```

Creating WebLogic Collaborate XPath Expressions

A WebLogic Collaborate XPath expression is an XPath expression that is defined for a business protocol. For routing, a WebLogic Collaborate XPath expression is used by the XOCP router logic plug-in and is defined for all the XOCP business messages that pass through the WebLogic Collaborate instance. For filtering, a WebLogic Collaborate XPath expression is used by the XOCP filter logic plug-in and is defined for all the XOCP business messages that pass through the WebLogic Collaborate instance.

WebLogic Collaborate XPath expressions are defined in the repository. You can use the following tools to create WebLogic Collaborate XPath expressions for the XOCP router logic plug-in and the XOCP filter logic plug-in:

- Bulk Loader as described in "Working with the Bulk Loader" in *Administering BEA WebLogic Collaborate*. The format for an XPath expression in a repository data file is:

  ```
  <xpath-expression expression="//TradingPartner1"
  location="ROUTER" type="APPEND"/>
  ```

  For more information about XPath syntax and usage, see the "XML Path Language Specification," published by the World Wide Web Consortium, at the following URL:

  `http://www.w3.org/TR/xpath.html`

- WebLogic Collaborate Administration Console as described in the *BEA WebLogic Collaborate Administration Console Online Help*.

  Table 2-3 describes the properties that you set when using the WebLogic Collaborate Administration Console to define an XPath expression.

For example, a WebLogic Collaborate administrator might want to filter messages for trading partners that are shippers so that they receive only shipping requests, while all other types of trading partners receive all messages. To do this, the administrator can use the Business Protocol Definitions tab in the WebLogic Collaborate Administration Console to create the following XPath expression for the XOCP filter logic plug-in:

```
/hub/trading-partner/extended-property-set/business='shipper') OR
(/hub/trading-partner/extended-property-set/business!='shipper')
```

3 Creating and Adding Plug-Ins

The following sections describe how to develop logic plug-ins in WebLogic Collaborate:

- About Logic Plug-Ins
- Logic Plug-In API
- Rules and Guidelines for Logic Plug-Ins
- Creating and Adding Logic Plug-Ins

About Logic Plug-Ins

The following sections describe logic plug-ins and related concepts:

- What Are Logic Plug-Ins?
- Logic Plug-In Architecture
- Chains
- Business Messages and Message Envelopes
- System and Custom Logic Plug-Ins

What Are Logic Plug-Ins?

Logic plug-ins are individual components that perform specialized processing of business messages that pass through WebLogic Collaborate. A logic plug-in is a custom service that a WebLogic Collaborate provider or trading partner can develop and install on a WebLogic Collaborate instance to provide additional value for WebLogic Collaborate management and for trading partners who use that WebLogic Collaborate instance.

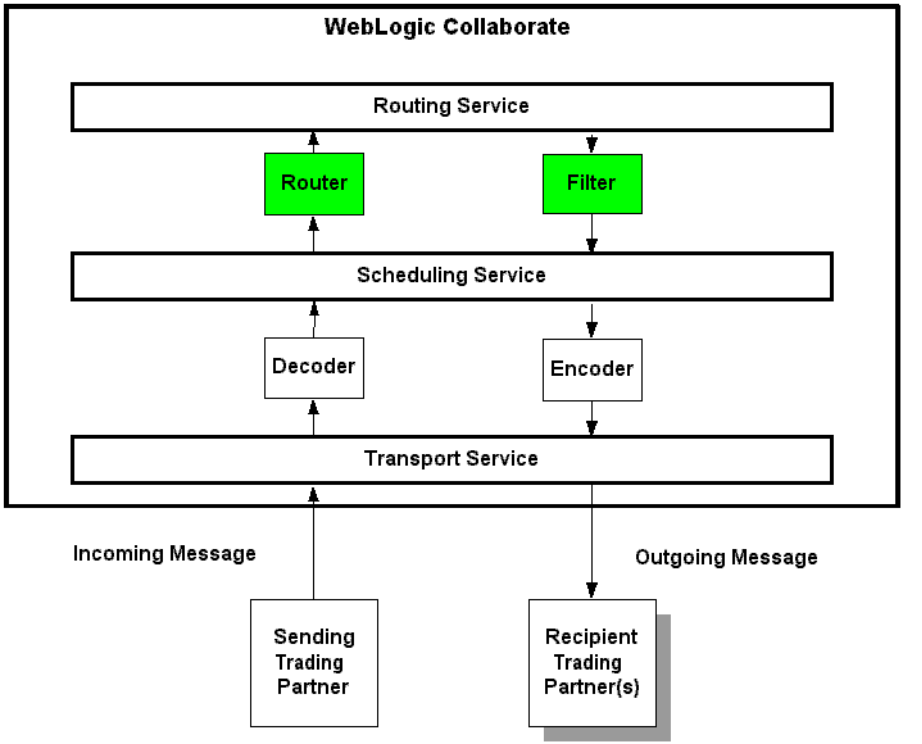
Logic plug-ins insert rules and business logic at strategic locations along the path that business messages travel as they make their way through WebLogic Collaborate. Logic plug-ins are instances of Java classes that are created when business protocols are created in WebLogic Collaborate, and are activated when a delivery channel is started. They are invoked when a message passes through.

Each logic plug-in is associated with a business protocol: logic plug-ins process only the messages that are exchanged using that protocol. For example, if a particular plug-in is associated with the XOCP protocol, then it processes only XOCP business messages.

Logic Plug-In Architecture

Logic plug-ins can be installed at two processing locations in the WebLogic Collaborate instance: the router and the filter, as shown in the following figure.

Figure 3-1 Logic Plug-In Locations in WebLogic Collaborate: Router and Filter



Logic Plug-In Processing Tasks

WebLogic Collaborate-provided router and filter plug-ins for all supported business protocols, including XOCP and RosettaNet, are directly involved in the processing of message recipient information in messages based on Xpath expressions in the repository. However, custom logic plug-ins can perform a wide range of services that

are entirely unrelated to routing or filtering, as well as routing and filtering operations. For example, a custom logic plug-in might track the number of messages sent from each trading partner for billing purposes.

Logic plug-ins perform the types of tasks described in the following table.

Table 3-1 Tasks Performed by Logic Plug-Ins

Process	Description	Examples
Route Modification	Changes the list of target recipients for a business message. Subject to conversation and collaboration agreement validation of the recipient. (WebLogic Collaborate plug-ins and custom plug-ins.)	■ "If a computer chip order over $1M is placed, make sure that NewChipCo is one of the recipients." ■ "After January 1, 2000, no orders should be sent to OldChipCo."
Examination	Examines the contents of a business message and takes certain actions based on the results of the examination. (Custom plug-ins.) **Note:** The business messages that are usually examined are those *without* encrypted contents.	■ "Log all senders of messages for billing purposes." ■ "For messages of type X, how many are conversation version 1 versus conversation version 2?"

Chains

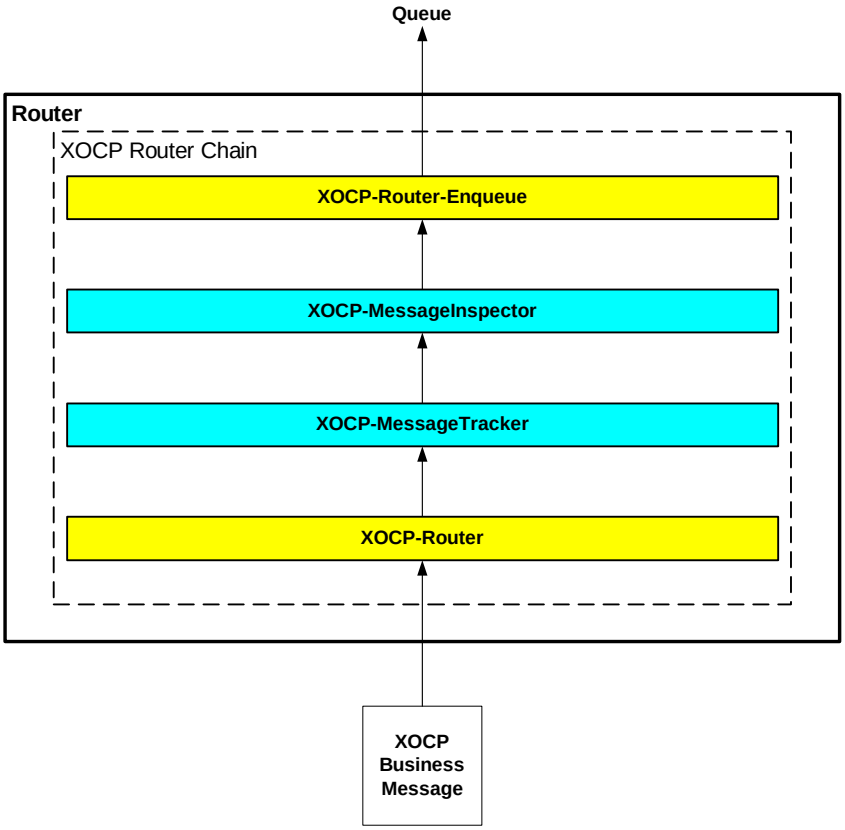
Both the router and filter modules can have multiple plug-ins that are executed when a business message passes through those modules in WebLogic Collaborate. Multiple logic plug-ins that share the same protocol are sequenced as a logic plug-in chain.

In a chain, logic plug-ins are processed sequentially at run time. After one plug-in has finished executing, the next plug-in in the chain is normally activated. Each successive plug-in can access any changes made previously to the shared message information as the business message passes throughout WebLogic Collaborate.

Note: The position of a logic plug-in in a chain is configured in the repository using the WebLogic Collaborate Administration Console, as described in *Administering BEA WebLogic Collaborate*.

The following figure shows an example of a chain of XOCP logic plug-ins in the router location in WebLogic Collaborate.

Figure 3-2 Sample XOCP Router Chain



Note that even when custom logic plug-ins do not provide routing or filtering capability, they must still be part of a router or filter logic plug-in chain. In this example, the chain contains four logic plug-ins that are processed in the order described in the following table.

Table 3-2 Logic Plug-Ins in the Sample XOCP Router Chain

Logic Plug-In	Description
XOCP router	WebLogic Collaborate provides this logic plug-in, which might modify the list of recipients for an XOCP business message based on XPath router expressions configured in the repository. This should be the first logic plug-in in the XOCP router chain.
XOCP-MessageTracker	Hypothetical logic plug-in. A WebLogic Collaborate owner or trading partner might provide such a custom logic plug-in to track the number of business messages sent from each trading partner for billing purposes.
XOCP-MessageInspector	Hypothetical logic plug-in. A WebLogic Collaborate owner or trading partner might provide such a custom logic plug-in to examine and maintain statistics for the types of business documents being exchanged through WebLogic Collaborate (for example, purchase orders, invoices, and so on).
XOCP router enqueue	WebLogic Collaborate provides this logic plug-in, which enqueues the XOCP business message in an internal WebLogic Collaborate router message queue. This should be the last logic plug-in in the XOCP router chain.

In this example, only XOCP business messages trigger the logic plug-ins in the XOCP router chain. Non-XOCP business messages (such as RosettaNet or cXML messages) are processed separately by the router chain associated with those business protocols.

Business Messages and Message Envelopes

A *business message* is the basic unit of communication exchanged between trading partners in a conversation. The business message contains the list of message recipients. A business message is represented in the WebLogic Collaborate API by the

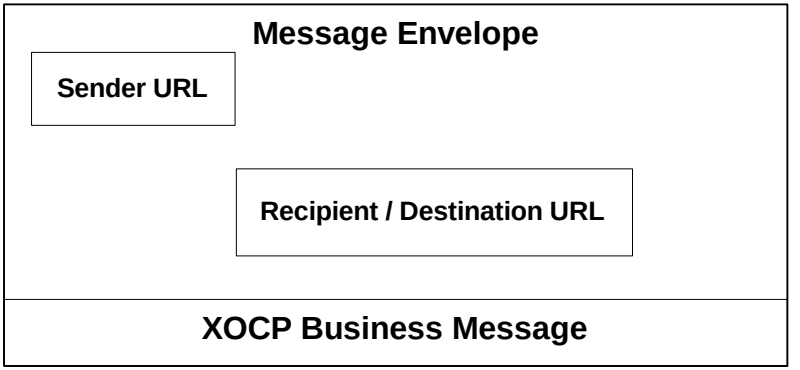
com.bea.b2b.protocol.messaging.Message interface. In addition, the following classes implement this interface and represent protocol-specific business messages:

- com.bea.b2b.protocol.xocp.messaging.XOCPMessage
- com.bea.b2b.protocol.rosettanet.messaging.RNMessage

When a business message enters the WebLogic Collaborate instance, WebLogic Collaborate creates a *message envelope* that acts as a container for the business message as it travels through WebLogic Collaborate. Message envelopes are instances of the com.bea.b2b.protocol.messaging.MessageEnvelope class.

The message envelope is used for routing purposes and is analogous to a paper envelope for a letter: the message envelope contains the business message plus addressing information, such as the identity of the sender (return address) and one recipient of the business message (destination address), as shown in the following figure.

Figure 3-3 Message Envelope Containing an XOCP Business Message



Message envelopes also contain other information about the business message. For detailed information about the MessageEnvelope class, see the *BEA WebLogic Collaborate Javadoc*.

For XOCP business messages, after the system XOCP router processes an XOCP business message and finalizes the list of intended message recipients, WebLogic Collaborate validates the recipients and creates a separate message envelope (and a logical copy of the XOCP business message) for each recipient in the list. These message envelopes are then forwarded to the XOCP filter for processing.

System and Custom Logic Plug-Ins

WebLogic Collaborate provides the following logic plug-ins to provide standard services for processing business messages.

Table 3-3 System Logic Plug-Ins

Logic Plug-In	Description
XOCP router	Modifies the list of recipients for an XOCP business message based on XPATH router expressions configured in the repository. In general, this system logic plug-in should be first in the router logic plug-in chain so that custom logic plug-ins can subsequently process a business message after its list of intended recipients is known.
XOCP router enqueue	Enqueues the XOCP business message in the WebLogic Collaborate router message queue. In general, this system logic plug-in should be last in the XOCP router logic plug-in chain.
XOCP filter	Determines whether an XOCP business message is sent to a trading partner based on XPATH filter expressions configured in the repository. In general, this system logic plug-in should be first in the XOCP filter logic plug-in chain so that custom logic plug-ins can subsequently process a business message after rejected business messages have been filtered out.
RosettaNet router enqueue	Enqueues the RosettaNet business message in the WebLogic Collaborate router message queue. In general, this system logic plug-in should be last in the RosettaNet router logic plug-in chain.
RosettaNet filter	Determines whether a RosettaNet business message is sent to a trading partner. In general, this system logic plug-in should be first in the RosettaNet filter logic plug-in chain.

In addition to using the system logic plug-ins, delivery channel owners and trading partners can develop their own custom logic plug-ins to provide specialized services through WebLogic Collaborate. Each logic plug-in is a Java class that implements the logic plug-in API, as described in “Programming Steps for Logic Plug-Ins” on page 3-13.

Logic Plug-In API

WebLogic Collaborate provides a logic plug-in API that allows WebLogic Collaborate applications to:

- Add or remove target trading partners from the message recipient list when using XOCP multicast. WebLogic Collaborate validates the list of recipients before sending the business message.
- Retrieve, examine, and process parts of business messages. To ensure that the contents of business messages are not altered or misrepresented programmatically, the logic plug-in API provides methods for examining business messages, but *not* for changing their contents.

The following table lists the components of the logic plug-in API. For more information, see the *BEA WebLogic Collaborate Javadoc*.

Table 3-4 Logic Plug-In API

Class/Interface	Description
`com.bea.b2b.protocol.PlugIn`	Tagging interface that describes a generic logic plug-in, that is, code that can be inserted, for execution, at various places in WebLogic Collaborate.
`com.bea.b2b.protocol.PlugInException`	Exception class that is thrown if an error occurs while a logic plug-in is being executed.
`com.bea.b2b.protocol.messaging.MessageEnvelope`	Represents the container (*envelope*) for a business message passing through WebLogic Collaborate. The `MessageEnvelope` contains the actual business message plus high-level routing and processing information associated with the business message, such as the sender URL and the URL for one recipient (There is a single message envelope for each recipient). A Java `InputStream` is available in case access to the native message is needed (because message content modification is not allowed, however, no `OutputStream` is provided).

Table 3-4 Logic Plug-In API (Continued)

Class/Interface	Description
`com.bea.b2b.protocol.messaging.Message`	Represents a business message passing through WebLogic Collaborate. It provides additional information to be used to properly route a message between trading partners. It also contains information specific to the particular business protocol being used for this business message. Depending on the protocol used, the `Message` class usually includes subclasses to provide additional protocol-specific information about the message.
`com.bea.b2b.protocol.messaging.PayloadPart`	Represents a component of the message payload. Specific classes that implement this information are provided for some of the different types of parts of a business message, such as XML or non XML parts, or to assist in accessing business protocol-specific information.
`com.bea.b2b.protocol.conversation.ConversationType`	Represents a single role in a specific conversation definition. It contains information such as the conversation name, conversation version, and trading partner role.
`com.bea.b2b.tpa.CPAInstance`	Represents a collaboration agreement instance. The available methods allow you to retrieve a variety of information about the collaboration agreement. Because no modification of the collaboration agreement is allowed from this API, only retrieval and verification methods are provided.
`com.bea.b2b.tpa.PartyInstance`	Represents the holder of a party to a collaboration agreement. The available methods allow you to verify or retrieve assorted information about the collaboration agreement party.
`com.bea.b2b.tpa.TradingPartnerInstance`	Represents a trading partner instance. This is used in conjunction with `PartyInstance` or in a stand-alone mode with a router or filter.

Rules and Guidelines for Logic Plug-Ins

Logic plug-ins should conform to the following rules and guidelines:

- Logic plug-ins must be thread-safe and, therefore, stateless. At run time, logic plug-in instances are cached and shared by multiple threads. Using instance variables is not recommended.
- If access to shared resources is required, then use the `synchronized` Java keyword to restrict access to the shared resource. Certain resources, such as instance variables within the class, shared objects, or external system resources (such as files) might need shared access. Using the `synchronized` keyword can affect overall application performance, so use it only when necessary.
- Logic plug-ins can modify the message envelope and the list of recipients in the business message, but they *cannot* modify the message contents. Changing the business message invalidates the digital signature, if present. The logic plug-in API provides mutator methods for modifying the message envelope only.
- Logic plug-ins must be self-contained: they are not interdependent with other logic plug-ins; they cannot exchange variables; and they do not return a variable. The message envelope is the only input and the only output. If the logic plug-in makes a change to the message envelope, it outputs the message envelope as modified.
- The main logic plug-in class must implement the `com.bea.b2b.protocol.PlugIn` interface.
- To ensure secure messaging, logic plug-ins are generally *not* able to inspect encrypted business messages. The business messages that are examined are usually those that do not have encrypted contents. To examine the encrypted contents of a business message, the logic plug-in must decrypt the message, inspect its contents, and then encrypt it again. Users must have their own public key infrastructure.
- It is the responsibility of the plug-in provider to ensure that any custom plug-ins that are installed on WebLogic Collaborate are properly debugged and designed from a security perspective.
- A logic plug-in is always associated with at least one particular protocol in the repository. The logic plug-in is triggered only when a business message that uses

that protocol passes through WebLogic Collaborate. For example, a RosettaNet business message does not trigger an XOCP-defined logic plug-in, and vice versa.

- A single logic plug-in can be associated with multiple protocols in the repository. For example, the same logic plug-in class named `SentMessages` can be associated with the XOCP and RosettaNet protocols. In the WebLogic Collaborate Administration Console, you can define separate logic plug-ins for each business protocol (such as `XOCP-SentMessages`, `RN-SentMessages`, and `cXML-SentMessages`), although each points to the same `SentMessages` class. Alternatively, the same logic plug-in can be used in two different protocol chains; such chains share initialization parameters, but they are separate instances.
- An efficient logic plug-in quickly determines whether a business message qualifies for processing and, if not, exits immediately.
- Logic plug-ins can call other modules, including shared methods in a utility library (for example, a module that accesses a database).
- Logic plug-ins are initialized one time, when the delivery channel is activated.
  - If the delivery channel is shut down (that is, if the `shutdown` method is called on the associated `com.bea.b2b.management.hub.runtime.DeliveryChannelMBean`), then all protocol-specific logic plug-ins associated with that delivery channel are shut down as well. The delivery channel *must* be restarted for the logic plug-ins to be active.
  - If WebLogic Collaborate is shut down (that is, if the `shutdown` method is called on the associated `com.bea.b2b.management.runtime.WLCMBean`), then all logic plug-ins running on that WebLogic Collaborate instance are shut down as well. WebLogic Collaborate and the delivery channel must be restarted.
  - If logic plug-in definitions change in the WebLogic Collaborate repository, such as when the chain is resequenced or when logic plug-in definitions are added, changed, or removed, then the delivery channel must be shut down and restarted to reflect the repository changes.
- The WebLogic Server instance *must* be restarted (and the Java Virtual Machine, or JVM, reloaded) if an upgraded version of logic plug-in source code is installed on WebLogic Collaborate.

Creating and Adding Logic Plug-Ins

Implementing a custom logic plug-in requires a combination of development and administrative tasks. The following steps describe the requires procedures:

- Programming Steps for Logic Plug-Ins
- Administrative Tasks

Programming Steps for Logic Plug-Ins

This section describes the programming steps that you must perform in the logic plug-in code. Although each logic plug-in processes business messages in its own way, all logic plug-ins must perform certain tasks.

To implement a logic plug-in, complete the following steps:

- Step 1: Import the Necessary Packages
- Step 2: Implement the PlugIn Interface
- Step 3: Specify the Exception Processing Model
- Step 4: Implement the Process Method
- Step 5: Get the Business Message from the Message Envelope
- Step 6: Validate the Business Message
- Step 7: Get Business Message Properties
- Step 8: Process the Business Message as Needed

This section uses code excerpts from a logic plug-in that:

- Intercepts a business message en route through WebLogic Collaborate
- Obtains the names of the message sender, its target recipient, and its associated conversation definition
- Inserts a row with this information in the billing database

Step 1: Import the Necessary Packages

At a minimum, a logic plug-in needs to import the following packages:

- `com.bea.b2b.protocol.*`
- `com.bea.b2b.protocol.messaging.*`

The following listing from the `SentMsgCounter.java` file shows how to import the necessary packages.

Listing 3-1 Importing the Necessary Packages

```
import java.util.Hashtable;
import com.bea.b2b.protocol.*;
import com.bea.b2b.protocol.messaging.*;
import com.bea.eci.logging.*;
import javax.naming.*;
import javax.sql.DataSource;

// This package is needed to access the DB pool
import java.sql.*;
```

Step 2: Implement the PlugIn Interface

A logic plug-in needs to implement the `com.bea.b2b.protocol.PlugIn` interface, as shown in the following listing.

Listing 3-2 Implementing the PlugIn Interface

```
public class SentMsgCounter implements PlugIn
{
    ...
}
```

Step 3: Specify the Exception Processing Model

A `PlugInException` is thrown if:

- A run-time exception (such as a `NullPointerException`) is thrown by a logic plug-in and caught by WebLogic Collaborate processing code.
- The logic plug-in throws an exception to indicate problems encountered during logic plug-in processing. The logic plug-in might handle the exception directly or it might notify the WebLogic Collaborate processing code.

The exception processing model specified in a logic plug-in determines what happens if an exception is thrown. logic plug-ins must implement the `exceptionProcessingModel` method and specify one of the return values described in the following table.

Table 3-5 Options for the Exception Processing Model

Class/Interface	Description
`EXCEPTION_CONTINUE`	Indicates that processing should continue to the next logic plug-in in the chain if a `PlugInException` is thrown. Use this option to allow a business message to continue through WebLogic Collaborate even if an error occurs during the execution of this logic plug-in.
`EXCEPTION_STOP`	Indicates that processing should stop at this logic plug-in if a `PlugInException` is thrown. The business message does not continue to the next logic plug-in in the chain. Use this option to cancel message processing and prevent a message from progressing further through WebLogic Collaborate. For example, a logic plug-in that is validating business documents can reject any that contain insufficient or incorrect data.

Table 3-5 Options for the Exception Processing Model (Continued)

Class/Interface	Description
EXCEPTION_UNWIND	Indicates that processing should unwind if a `PlugInException` is thrown. The business message does not continue to the next logic plug-in in the chain. Use this option to reject a message; to prevent its further progress through WebLogic Collaborate; and to undo any changes made by this plug-in, along with any changes made by previous plug-ins in the chain. If an exception is thrown and this is the exception processing model, then the `unwind` methods in all *previous* plug-ins in the chain (but not the current logic plug-in), are invoked in reverse order. In effect, unwinding cancels all changes made by the chain. For example, if a logic plug-in inserts a row in a database table, its `unwind` method should delete that row. **Note:** To use this exception processing model, all logic plug-ins in the chain must implement the `unwind` method, even if the method does nothing.

If a business message is rejected, what happens next depends on the business protocol as well as on the specified Quality of Service associated with the message. For example, the sending WebLogic Collaborate application might be notified that message delivery failed and it might then attempt to send the business message again.

The following listing shows how the `SentMsgCounter` plug-in implements the `exceptionProcessingModel` method.

Listing 3-3 Specifying the Exception Processing Model

```
public int exceptionProcessingModel()
{
    return EXCEPTION_CONTINUE;
}
```

Step 4: Implement the Process Method

To process a business message, a logic plug-in must implement the `process` method, which accepts the message envelope of the business message as its only parameter. In the following listing, the `SentMsgCounter` class begins its implementation of the `process` method by defining the variables that it later uses to store message properties.

Listing 3-4 Implementing the Process Method

```
public void process(MessageEnvelope mEnv) throws PlugInException
{

    String sender, conversation;
    String tRecipient;
    Connection conn = null;
    Statement stmt = null;
    Message bMsg = null;
    ...
}
```

Note: When processing a business message, a logic plug-in is allowed to modify only the message envelope, not the business message.

Step 5: Get the Business Message from the Message Envelope

If a logic plug-in needs to inspect the contents of a business message, it must call the `getMessage` method on the `MessageEnvelope` instance, which retrieves the business message as a `Message` object.

In the following listing, the `SentMsgCounter` class gets the business message from the message envelope by calling the `getMessage` method.

Listing 3-5 Retrieving the Business Message from the Message Envelope

```
if((bMsg = mEnv.getMessage())== null)
{
    throw new PlugInException("message is NULL");
}
```

Step 6: Validate the Business Message

Optionally, a logic plug-in can determine whether a message is a valid business message that should be processed, or a system message that should be ignored by the logic plug-in. To check a business message, the logic plug-in can call the `isBusi-`

nessMessage method on the Message instance. In the following listing, the SentMsgCounter class uses the isBusinessMessage method.

Listing 3-6 Validating the Business Message

```
if (bMsg.isBusinessMessage())
{
    ...
}
```

Step 7: Get Business Message Properties

Optionally, a logic plug-in can retrieve certain properties of the business message by calling methods on the MessageEnvelope or Message instance. In the following listing, the SentMsgCounter class gets the name of the conversation definition associated with the conversation in which this message was sent, the name of the sender of the business message, and the name of the recipient trading partner.

Listing 3-7 Retrieving Business Message Properties

```
conversation= bMsg.getConversationType().getName();
sender = mEnv.getSender();
tRecipient = mEnv.getRecipient();
```

Step 8: Process the Business Message as Needed

After a logic plug-in has obtained the necessary information from the business message, it processes this information as needed. For example, the SentMsgCounter plug-in updates the billing database with the message statistics it has collected.

Administrative Tasks

An administrator adds the logic plug-in definition to the repository by performing the following tasks from the Logic Plug-Ins tab of the WebLogic Collaborate Administration Console:

1. Specify the following logic plug-in properties:
   - Name of the logic plug-in.
   - Java class that implements the `PlugIn` interface. This class can call auxiliary classes in the class library, but it must be the main point of entry for the logic plug-in. In addition, the Java class file must reside in a location specified by the WebLogic `CLASSPATH`.
   - Parameter name/value pairs to use when initializing the Java class.
2. Assign a logic plug-in to a business protocol.
3. Specify the position of the logic plug-in in the chain.

For more information about administrative tasks, see *Administering BEA WebLogic Collaborate*.

Index

S

T

V

X