IBM WebSphere Partner Gateway Enterprise and Advanced Editions

Enterprise Integration Guide

Version 6.0.0.1

IBM WebSphere Partner Gateway Enterprise and Advanced Editions

Enterprise Integration Guide

Version 6.0.0.1

Note!

Before using this information and the product it supports, read the information in “Notices” on page 217.

13September2005

This edition applies to WebSphere Partner Gateway Enterprise Edition (5724-L69), version 6.0.0.1, and Advanced Edition (5724-L68), version 6.0.0.1, and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this documentation, e-mail doc-comments@us.ibm.com. We look forward to hearing from you.

Contents

About this book

This guide describes the Backend Integration interface, which is the mechanism that back-end systems and IBM(R) WebSphere(R) Partner Gateway use to communicate. The guide then describes how to integrate WebSphere Process Server, WebSphere InterChange Server, WebSphere Business Integration Message Broker, and WebSphere Data Interchange with WebSphere Partner Gateway using the Backend Integration interface.

The information in this guide pertains to WebSphere Partner Gateway Enterprise and Advanced Editions only.

Audience

This book is intended for the person responsible for integrating WebSphere Partner Gateway with back-end systems.

Typographic conventions

This document uses the following conventions.

Table 1. Typographic conventions

Convention	**Description**
`Monospace font`	Text in this font indicates text that you type, values for arguments or command options, examples and code examples, or information that the system prints on the screen (message text or prompts).
bold	Boldface text indicates graphical user interface controls (for example, online button names, menu names, or menu options) and column headings in tables and text.
italics	Text in italics indicates emphasis, book titles, new terms and terms that are defined in the text, variable names, or letters of the alphabet used as letters.
`Italic monospace font`	Text in italic monospace font indicates variable names within monospace-font text.
ProductDir	*ProductDir* represents the directory where the product is installed. All IBM WebSphere Partner Gateway product path names are relative to the directory where the IBM WebSphere Partner Gateway product is installed on your system.
`%text%` and *`$text`*	Text within percent signs (%) indicates the value of the Windows(R) *text* system variable or user variable. The equivalent notation in a UNIX(R) environment is $*text*, indicating the value of the *text* UNIX environment variable.
Underlined colored text	Underlined colored text indicates a cross-reference. Click the text to go to the object of the reference.
Text in a blue outline	(In PDF files only) A blue outline around text indicates a cross-reference. Click the outlined text to go to the object of the reference. This convention is the equivalent for PDF files of the “Underlined colored text” convention included in this table.

Table 1. Typographic conventions (continued)

Convention	Description
“ “(quotation marks)	(In PDF files only) Quotation marks surround cross-references to other sections of the document.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
`[ ]`	In a syntax line, square brackets surround optional parameters.
`< >`	Angle brackets surround variable elements of a name to distinguish them from one another. For example, <*server_name*><*connector_name*>tmp.log.
`/, \`	Backslashes (\) are used as separators in directory paths in Windows installations. For UNIX installations, substitute slashes (/) for backslashes.

Related documents

The complete set of documentation available with this product includes comprehensive information about installing, configuring, administering, and using WebSphere Partner Gateway Enterprise and Advanced Editions.

You can download this documentation or read it directly online at the following site:
http://www.ibm.com/software/integration/wspartnergateway/library/infocenter

Note: Important information about this product may be available in Technical Support Technotes and Flashes issued after this document was published. These can be found on the WebSphere Business Integration Support Web site, http://www.ibm.com/software/integration/wspartnergateway/support. Select the component area of interest and browse the Technotes and Flashes section.

New in this release

This section describes the new features of IBM WebSphere Partner Gateway that are covered in this version of the *Enterprise Integration Guide*.

New in release 6.0.0.1

In version 6.0.0.1 of Partner Gateway, support has been added for using WebSphere Process Server as the backend integration system. The document includes the following sections:

- General information about integration with WebSphere Process Server
- The WebSphere Partner Gateway-supplied data binding for Backend Integration packaging
- Transport-specific information about integrating WebSphere Process Server over the JMS, HTTP, and file-system transports as well as SOAP requests over HTTP (Web services)

New in release 6.0

The product was renamed IBM WebSphere Partner Gateway, version 6.0.

With this update, the following changes have been made to the document:

- Information about support for versions of WebSphere Business Integration InterChange Server before 4.2.2 has been removed from the document. Specifically, information about the WebSphere Business Integration Connect Servlet and the Wrapper data handler has been deleted.
- The chapter on integration with WebSphere Data Interchange has been updated to reflect the new EDI document flow configuration for this version.

New in release 4.2.2

With this update to the IBM WebSphere Business Integration Connect 4.2.2 release, the following changes have been made to this document:

- The version 4.2.1 *Integration Guide* has been renamed as the *Enterprise Integration Guide*.
- The document has been extensively reorganized to provide better usability. In particular:
  - Information on how to configure WebSphere Business Integration Connect has been separated from information on how to configure a back-end system based on the assumption that different people or roles usually perform these two tasks.
  - Information on how to integrate with WebSphere InterChange Server has been expanded and broken into chapters, which are now contained in Part 2 of the guide. For an introduction to integration with InterChange Server, see Chapter 3, “Introduction to WebSphere Process Server integration,” on page 49.
- WebSphere Business Integration Connect can now use WebSphere Business Integration Adapter for HTTP to provide support for integration with version 4.2.2 WebSphere InterChange Server over the HTTP transport protocol. For more information, see “Using HTTP transport protocol with ICS” on page 141.

- A new chapter, Chapter 11, “Integrating with WebSphere Business Integration Message Broker,” on page 173, provides information on how to integrate WebSphere Business Integration Connect with WebSphere Business Integration Message Broker.

Part 1. Introduction to back-end integration

Chapter 1. Overview

This chapter provides an overview of integration of WebSphere Partner Gateway with a back-end system.

This chapter provides the following general information about back-end integration:

- "Overview of document processing"
- "Overview of back-end integration" on page 7

Overview of document processing

With WebSphere Partner Gateway, you exchange business documents with your community participants. The purpose of exchanging these documents is to communicate information, which typically involves processing data and returning a result. When you receive data from a community participant, processing of that data generally occurs in the back-end system of your enterprise. WebSphere Partner Gateway is the point within the hub community through which messages to and from the enterprise are routed.

The enterprise is accessed through a back-end system to which WebSphere Partner Gateway connects.

Figure 1 shows how documents flow through the WebSphere Partner Gateway Enterprise and Advanced Editions. A participant sends a document to WebSphere Partner Gateway (the hub). WebSphere Partner Gateway receives the document and performs any actions that have been predefined (such as validating or transforming the document). WebSphere Partner Gateway then sends the document to a back-end application, where the document is processed.

Note: As indicated in the illustration, communication flows in the opposite direction too. The back-end application can generate a document and send it to the hub, which processes it and sends it on to the participant.

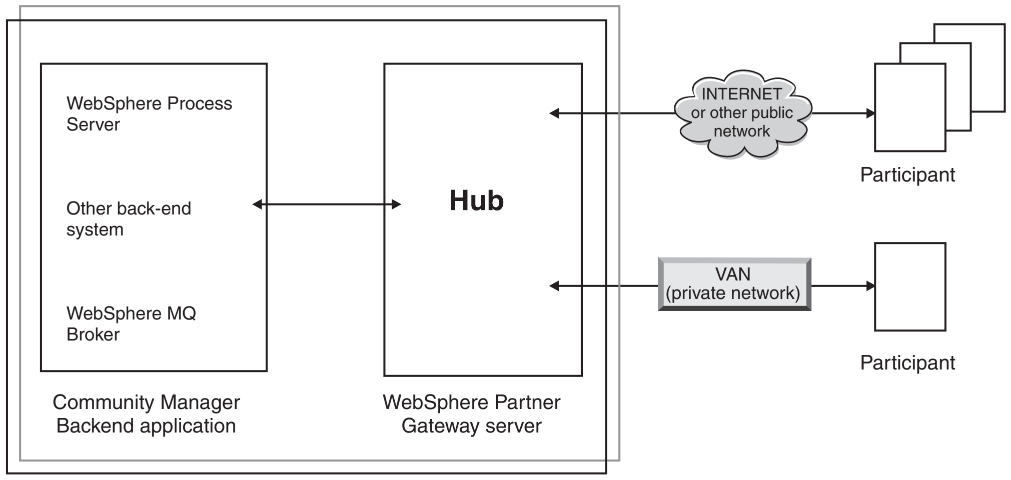

Figure 1. End-to-end document flow

This guide focuses on the integration between the hub and the back-end application (the shaded part of the illustration).

Note: The information in this document applies to the WebSphere Partner Gateway Enterprise and Advanced Editions. WebSphere Partner Gateway - Express, a light-weight, easy-to-use B2B connectivity tool, differs from WebSphere Partner Gateway Enterprise and Advanced Editions. It provides a community integration solution (versus a gateway hub solution that WebSphere Partner Gateway Enterprise and Advanced Editions provide for a Community Manager). For information about WebSphere Partner Gateway - Express, refer to its *User Guide*.

The roles in the hub community

WebSphere Partner Gateway Enterprise and Advanced Editions have three types of participants--the Community Operator, Community Manager, and participants. A Community Operator is created automatically when WebSphere Partner Gateway is installed. The Community Operator is in charge of setting up the hub and creating the participants that will interact with the hub.

The Community Manager, which is typically the owner of the hub, is actually considered to be one of the participants of the hub. The Community Operator creates a profile for the Community Manager, providing the information necessary to allow the Community Manager to send documents to and receive documents from participants. (Note that only one Community Manager can be created.) When the hub sends documents to the back-end system, it uses the information (URL or JMS queue, for example) set up for the Community Manager. The Community Operator also creates profiles for participants, of which there can be many.

The hub configuration process

The hub administrator is the Community Operator user responsible for administering the hub. The hub administrator sets up the hub to send and receive business documents from the Community Manager and participants. To receive business documents from the Community Manager, the hub administrator creates the targets for the transports that the Community Manager will use to send documents. For example, if the Community Manager uses the file-directory and JMS transports, the Community Operator sets up a file-directory target and a JMS target for the Community Manager. Similarly, if participants will use the HTTP transport and the FTP transport, the Community Operator sets up an HTTP target and an FTP target for them.

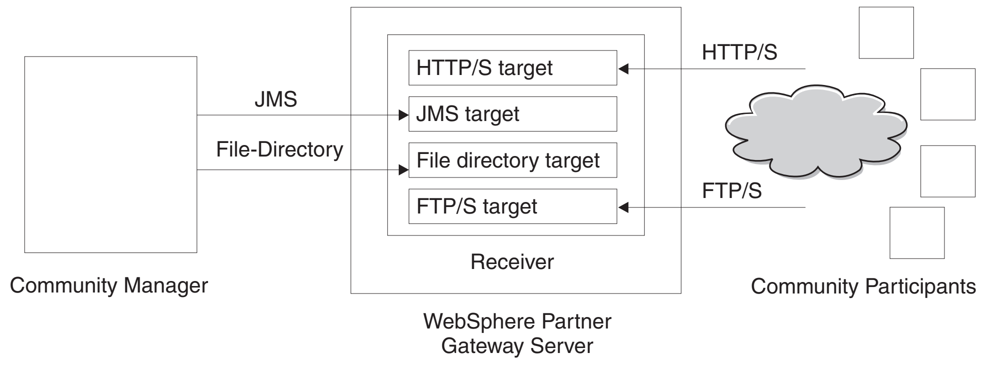

Figure 2. Targets for the Community Manager and participants

Gateways are created for the Community Manager and participants for each of the transports that they will use to receive business documents sent by the hub.

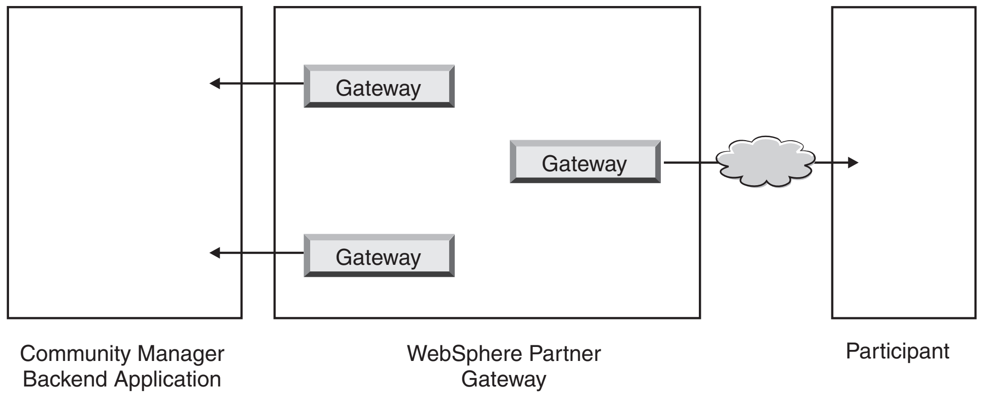
Gateway
Gateway
Gateway
Community Manager Backend Application
WebSphere Partner Gateway
Participant

Figure 3. Gateways to the Community Manager and participants

As part of the hub configuration, the Community Operator establishes document flow definitions, which define characteristics of a document flow, such as:

- Packaging, which provides information about the routing of the document
- Protocol, which is the business protocol to which the document adheres
- Document flow, which represents the document itself

When WebSphere Partner Gateway is installed, a set of document flow definitions is available for use. You can also add to the document flow definitions by creating your own definitions or by uploading definitions. For example, document flow definitions for a variety of RosettaNet PIPs are included as ZIP files on the installation medium. You can upload these files to make them available for use. If you are exchanging EDI files, you can import document flow definitions and associated maps from the Data Interchange Services client.

Consider the following example--a community participant sends an RNIF 2.0 message containing a RosettaNet PIP 3A4 purchase order document to the HTTP target of WebSphere Partner Gateway. The message is intended for the Community Manager. The Community Manager has a back-end system that processes purchase orders and expects to receive the purchase order, which essentially is the payload of the RNIF message sent by the participant. Before the participant connections in WebSphere Partner Gateway are set up, it is agreed that:

- The participant will send an RNIF message containing the RosettaNet PIP 3A4 purchase order document over HTTP.
- WebSphere Partner Gateway will extract the business payload or the RosettaNet Service Content from the incoming message.
- The document will be routed to the back-end system over JMS. The Backend Integration packaging will be used.
- The back-end application will then process the received document.

When Backend Integration packaging is used, WebSphere Partner Gateway-defined transport headers are added to the document to convey information helpful for the document exchange.

For the previous example, the Community Operator would upload the appropriate PIP package, which would set up the following document flow definitions for the exchange of RosettaNet PIP 3A4:

- A flow that consists of RNIF packaging, the RosettaNet protocol, and the 3A4 PIP
- A flow that consists of Backend Integration packaging, RNSC protocol, and the 3A4 PIP

After the Community Operator establishes the document flow definitions, the Community Operator creates interactions for the document flow definitions. For example, the Community Operator might indicate that the RNIF/RosettaNet/3A4 document flow definition can come into the hub from a source.

The Community Operator (or the participants) select the appropriate B2B capabilities for the document exchange. In this example, the Community Manager would have the following B2B capability enabled:

- Package: Backend Integration
- Protocol: RNSC
- Document Flow: 3A4

The participant would have the following B2B capability enabled:

- Package: RNIF
- Protocol: RosettaNet
- Document Flow: 3A4

The Community Operator then creates connections between participants.

In the following illustration, the Community Operator has created profiles for the Community Manager and participant, has created targets for receiving documents and gateways for sending documents, has created the document flow definitions listed above, has set the B2B capabilities of the participant and Community Manager, and has created a connection between the two.

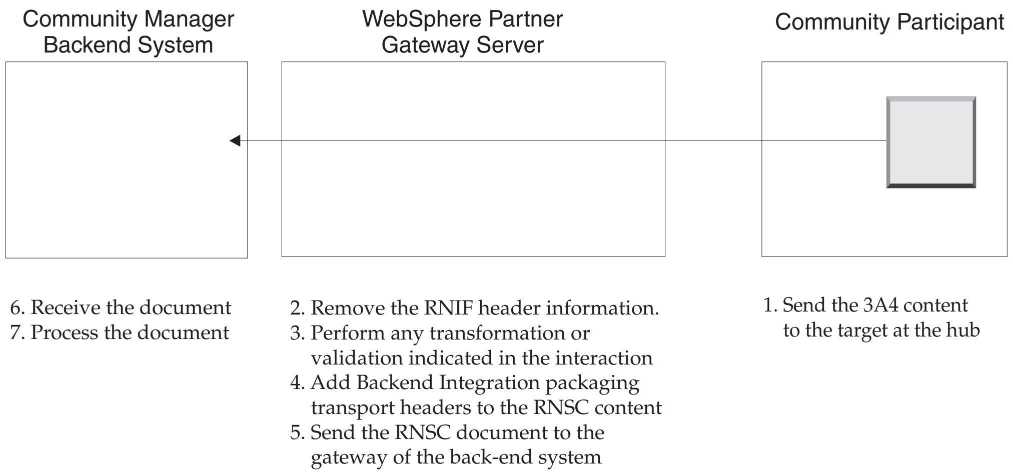
Community Manager Backend System
WebSphere Partner Gateway Server
Community Participant
6. Receive the document
7. Process the document
2. Remove the RNIF header information.
3. Perform any transformation or validation indicated in the interaction
4. Add Backend Integration packaging transport headers to the RNSC content
5. Send the RNSC document to the gateway of the back-end system
1. Send the 3A4 content to the target at the hub

Figure 4. How a document flows to the back-end system

For information on setting up the hub, refer to the *Hub Configuration Guide*.

Overview of back-end integration

All editions of WebSphere Partner Gateway provide the ability to connect to back-end systems. These editions differ in the transport protocols they can support, as follows:

- WebSphere Partner Gateway - Express provides file-based integration.
- WebSphere Partner Gateway Enterprise and Advanced Editions provide file-based integration. In addition, they provide integration over the HTTP, HTTPS, and JMS protocols.

Documents exchanged between the community participant and WebSphere Partner Gateway can be in a variety of formats. Documents can be in the SOAP, cXML, XML, EDI, record-oriented data (ROD), or binary formats or in any custom format mutually agreed upon by the participants. The *Administrator Guide* has a complete list of the document types supported as well as the transport protocols (for example, HTTP) that can be used to send the documents.

Documents that can be exchanged between WebSphere Partner Gateway and the back-end system of the Community Manager as well as the transport types associated with the documents are shown in Table 12 on page 29, Table 13 on page 29, and Table 14 on page 30.

Figure 5 illustrates how WebSphere Partner Gateway uses the back-end integration interface to communicate with the back-end system at the Community Manager. Note that the arrows go in both directions; that is, the document can originate from the back-end system of the Community Manager.

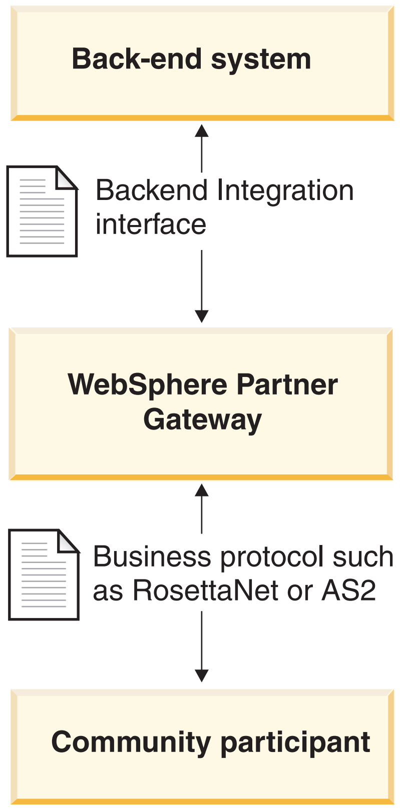
Back-end system
Backend Integration interface
WebSphere Partner Gateway
Business protocol such as RosettaNet or AS2
Community participant

Figure 5. The role of the business protocol and packaging in the flow of documents

Chapter 2. Planning for back-end integration

This chapter describes how to plan for integration of WebSphere Partner Gateway with a back-end system. It describes the types of decisions you will make when planning for back-end integration:

- "Which business protocol are you using?"
- "Which packaging will you use?" on page 19
- "Which message transport will you use?" on page 28
- "How do you access your back-end application?" on page 36

It also describes the following information:

- "Message handling" on page 36
- "Configuring WebSphere Partner Gateway" on page 37

Which business protocol are you using?

The business protocol of your message determines the format of the document. The business protocol affects many of the decisions you must make as you plan for integration to a back-end system. The choice of business protocol determines the packaging method you must use, which, in turn, affects which message-transport protocols you can use.

For a complete description of business protocols, see the *Hub Configuration Guide*. This section describes integration information that is specific to the following business protocols:

- "Web services (SOAP)"
- "cXML" on page 10
- "EDI" on page 10

  Note: The section on EDI also describes how XML and record-oriented-data (ROD) documents are processed.
- "RosettaNet" on page 15

Web services (SOAP)

WebSphere Partner Gateway can make the following Web services available to members of the hub community:

- Web services provided by the Community Manager can be available to community participants.

  You will have to provide your community participant with the public WSDL that WebSphere Partner Gateway generates. It is important to note that the URL on which the community participant invokes the Web service is the Web service public URL specified while uploading the Web service. WebSphere Partner Gateway acts as a proxy. It receives a SOAP message from the participant and figures out the corresponding private Web service. It then invokes the private Web service (provided by the Community Manager) using the same SOAP message. The response returned by the Community Manager is then returned to the participant.
- Web services provided by community participants can be available to the Community Manager.

It is important to note that the same Web Service Interface can be provided by multiple partners. WebSphere Partner Gateway makes the Web service available to the Community Manager at the Web service URL specified in the console while uploading the Web service. Additionally the Community Manager will have to provide the URL parameter to identify "To Partner". Refer to the *Hub Configuration Guide* for more details. WebSphere Partner Gateway acts as a proxy. It receives a SOAP message from the Community Manager and figures out the corresponding Web service and the "To Partner". It then invokes the Web service provided by the partner using the same SOAP message. The response message returned by the partner is then returned to the Community Manager.

Refer to the *Hub Configuration Guide* for more information, including how to set up your document flow definitions for Web services.

cXML

You can send or receive cXML documents to or from your community participants. When WebSphere Partner Gateway receives a cXML document from a community participant, it validates the document and translates it (if specified) before sending it to the back-end system at the Community Manager. Note that translation should not be used for synchronous cXML messages. In a synchronous exchange, the back-end system generates a response, which WebSphere Partner Gateway returns to the community participant (if appropriate for the message).

A back-end system at the Community Manager that needs to send a cXML document can do one of two things:
- Generate and send a cXML document, which WebSphere Partner Gateway passes through to the community participant
- Generate and send an XML document, which WebSphere Partner Gateway converts to cXML before sending to the community participant

Note: If XML document translation is used, for synchronous request/response transactions with the community participant, the response is returned asynchronously to the back-end system.

Refer to the *Hub Configuration Guide* for more information, including how to set up your document flow definitions for cXML.

EDI

WebSphere Partner Gateway accepts EDI documents from participants from value added networks (VANs) as well as from the Internet. EDI documents sent to or received from a VAN use the FTP Scripting transport. The FTP Scripting transport can also be used to send documents to or receive documents from the Internet. See the *Hub Configuration Guide* for information on the FTP Scripting transport.

An EDI document enters the hub and leaves the hub in an EDI envelope, known as an *interchange*. The interchange contains individual EDI transactions or groups of transactions.

If the EDI interchange will be passed through the hub (without being de-enveloped), you create one connection between the hub and Community Manager.

However, if the EDI interchange will be de-enveloped, the process for creating interactions and connections is different from other business protocols. The interchange must be de-enveloped, and the individual transactions processed. The transactions are typically translated to another form, according to a transformation map that is imported from the Data Interchange Services client. If the EDI transactions are translated to XML or record-oriented-data (ROD) documents, those documents are sent to the Community Manager or participant. If the transactions are translated to other EDI formats, the transactions are first enveloped before being sent to the Community Manager or participant.

Back-end application to participant flows

A back-end application can send the following types of documents:

- A single EDI interchange that contains one or more transactions

  WebSphere Partner Gateway de-envelopes the individual EDI transactions and translates these individual transactions. If the transactions are translated into EDI, they are enveloped and then routed to the participant. The back-end application can use None or Backend packaging and send the interchange over a variety of transports, as defined in Table 13 on page 29.

  Figure 6 shows an X12 interchange consisting of three transactions being de-enveloped. The transactions are transformed into EDIFACT format and are then enveloped and sent to the participant.

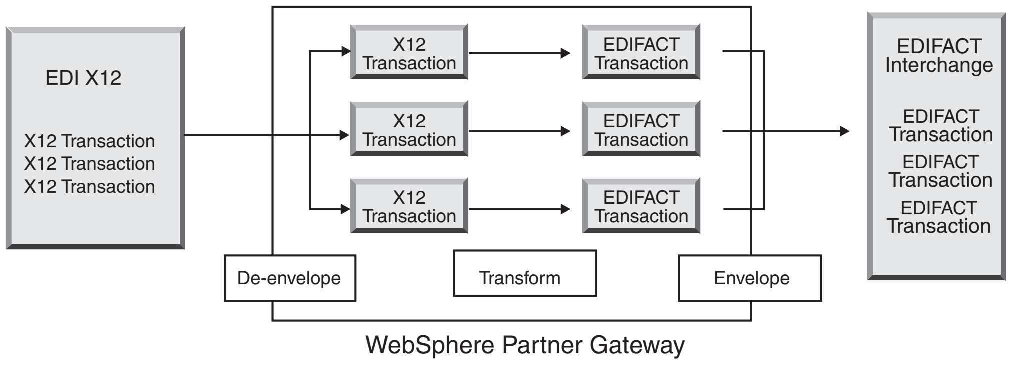

Figure 6. EDI interchange from back-application to participant

  Each of the transactions has a transformation map associated with it, which specifies how the transaction is transformed. The transaction can be transformed into a single transaction or, if map chaining was used to create the map, multiple transactions.

  If the transaction is translated into an XML or ROD document, it is routed as configured in the participant connection for that transaction.

  Figure 7 on page 12 shows an EDI X12 interchange being de-enveloped and transformed into XML documents, which are then sent to the participant.

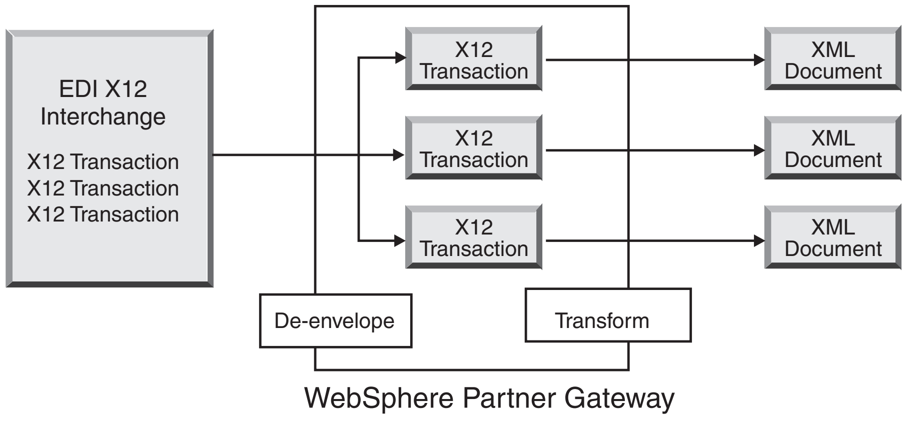

Figure 7. EDI interchange sent from backend application to participant (as XML documents)

The transaction can be transformed into a single document or, if map chaining was used to create the map, multiple documents.

- A single document, such as an XML or ROD document

  WebSphere Partner Gateway translates the document into an EDI transaction, envelopes the transaction, and sends it to the participant. The back-end application can use None or Backend Integration packaging and can send the document over a variety of transports, as defined in Table 13 on page 29.

  Figure 8 shows an XML document that is transformed into X12 transactions and then enveloped.

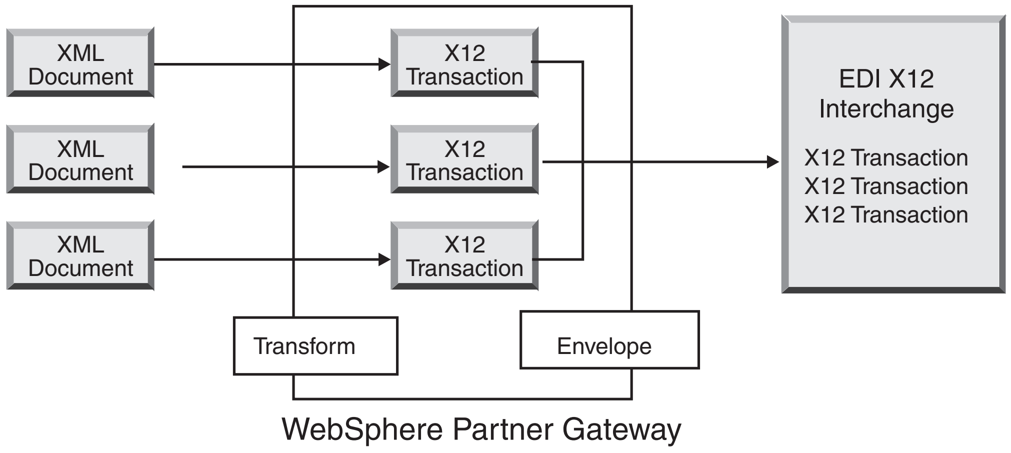

Figure 8. XML documents sent from backend application to participant (as an EDI interchange)

One document can be transformed into multiple transactions (if map chaining was used to create the map), and the transactions can be enveloped into different interchanges.

Figure 9 on page 13 shows an XML document that is transformed into three X12 transactions. Two of the transactions are enveloped together. One is put in a separate envelope.

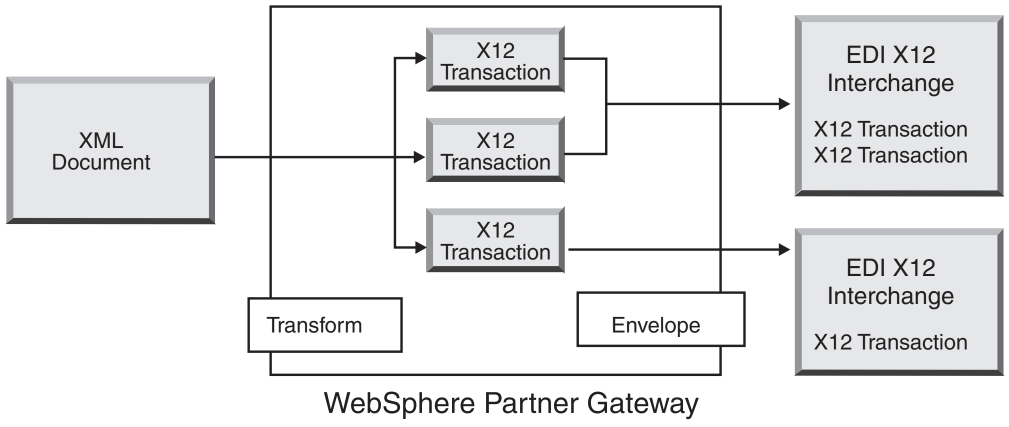

Figure 9. XML document sent from back-end application to participant (as EDI interchanges)

If the document is translated into another XML document or another ROD document, it is routed as configured in the participant connection for the document.

- A single file containing multiple XML or ROD documents

  WebSphere Partner Gateway splits the documents and translates them. If the documents are translated into EDI transactions, WebSphere Partner Gateway envelopes the transactions and sends the envelope to the participant. If batch IDs were assigned to the XML or ROD documents, WebSphere Partner Gateway attempts to send the EDI transactions in one envelope (as a batch). The back-end application can use None or Backend Integration packaging and can send the document over a variety of transports, as defined in Table 13 on page 29.

  Figure 10 shows a set of XML documents being split, resulting in individual XML documents. The XML documents are transformed into X12 transactions, and the transactions are enveloped.

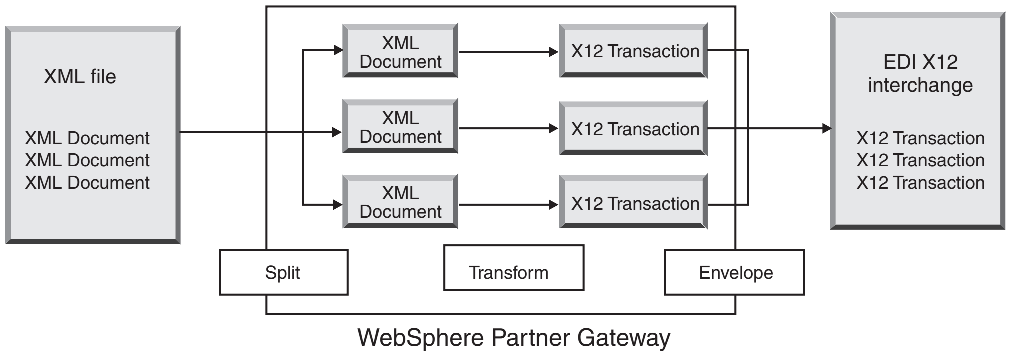

Figure 10. Multiple XML documents sent from backend-application, split, and then sent to participant (as an EDI interchange)

Figure 10 shows how the documents are split and the transformed transactions are enveloped together. To allow documents to be split, you must configure a splitter handler (in this case, the XML Splitter Handler) for the target you are using to send the documents. The XML Splitter Handler must have the BCG_BATCHDOCS option set to on (the default value) for this scenario to occur. BCG_BATCHDOCS assigns a batch ID to the XML documents so that the resulting transactions will be put into the same envelope. See the *Hub Configuration Guide* for information on the XML Splitter Handler and the BCG_BATCHDOCS attribute.

If the documents are translated into other XML documents or other ROD documents, they are routed as configured in the participant connection for the documents.

- A single file containing multiple EDI interchanges

  WebSphere Partner Gateway splits the file into individual interchanges. It then de-envelopes the interchanges into individual transactions and translates them. If the documents are translated into EDI transactions, WebSphere Partner Gateway envelopes the transactions and sends the envelope to the participant. The back-end application can use None or Backend Integration packaging and can send the document over a variety of transports, as defined in Table 13 on page 29.

  If the documents are translated into XML or ROD documents, they are routed as configured in the participant connection for the documents.

Participant to backend-application flows

A participant can send the following types of documents:

- A single EDI interchange that contains one or more transactions

  WebSphere Partner Gateway de-envelopes the individual EDI transactions and translates these transactions. If the transactions are translated into EDI, they are enveloped and routed to the back-end application. The back-end application can use None or Backend Integration packaging, and the transactions can be sent over a variety of transports, as defined in Table 14 on page 30.

  If the transactions are translated into XML or ROD documents, they are routed as configured in the participant connection for the transactions.

- A single document, such as an XML or ROD document

  WebSphere Partner Gateway translates the document into an EDI transaction, envelopes the transaction, and routes the envelope to the back-end application. Either None or Backend Integration packaging can be used.

  If the document is translated into another XML document or another ROD document, it is routed as configured in the participant connection for the document.

- A single file containing multiple XML or ROD documents

  WebSphere Partner Gateway splits the documents and translates them. If the documents are translated into EDI transactions, WebSphere Partner Gateway envelopes the transactions and sends the envelope to the back-end application. If batch IDs were assigned to the XML or ROD documents, WebSphere Partner Gateway attempts to send the EDI transactions in one envelope (as a batch). Either None or Backend Integration packaging can be used.

  If the documents are translated into other XML documents or other ROD documents, they are routed as configured in the participant connection for the documents.

- A single file containing multiple EDI interchanges

  WebSphere Partner Gateway splits the file into individual interchanges. It then de-envelopes the interchanges into individual transactions and translates them. If the documents are translated into EDI transactions, WebSphere Partner Gateway envelopes the transactions and sends the envelope to the back-end application. Either None or Backend Integration packaging can be used.

  If the documents are translated into XML or ROD documents, they are routed as configured in the participant connection for the documents.

Functional acknowledgments

A functional acknowledgment specifies that an EDI interchange was received. A functional acknowledgment is always enveloped before being delivered.

Note: Functional acknowledgments apply only to those interchanges that are de-enveloped by WebSphere Partner Gateway or generated by WebSphere Partner Gateway. They do not apply to interchanges that are passed through WebSphere Partner Gateway.

For interchanges received by WebSphere Partner Gateway

- If the interchange is received from a back-end application, WebSphere Partner Gateway can send functional acknowledgments back to the back-end application.
- If the interchange is received from a participant, WebSphere Partner Gateway can send functional acknowledgments back to the participant.

For interchanges generated by WebSphere Partner Gateway:

- If the interchange is sent to a participant, the participant can send a functional acknowledgment back to WebSphere Partner Gateway. WebSphere Partner Gateway will not send this functional acknowledgment to the backend system.
- If the interchange is sent to the back-end application, the back-end application can send a functional acknowledgment back to WebSphere Partner Gateway. WebSphere Partner Gateway will not send this functional acknowledgment to the participant.

RosettaNet

WebSphere Partner Gateway provides support to let you send and receive documents that adhere to the RosettaNet 1.1 and 2.0 standards. When a participant sends a RosettaNet message to the hub, the target side of the participant connection must have Backend Integration specified. The hub converts the payload of the message to RNSC format and sends the message to the back-end system. Because Backend Integration packaging is used, the hub adds transport-level headers to the message. The message is sent through the HTTP or JMS transport protocol. The transport-level header retains meta-information that is not part of the PIP and enables WebSphere Partner Gateway to route the message appropriately.

Similarly, when the Community Manager back-end system sends an RNSC message to the hub, the source side of the participant connection must have Backend Integration packaging specified, and the back-end system must supply the transport-level headers.

For example, suppose an application wants to send a message to a community participant using RosettaNet sent on HTTP. The application provides the RosettaNet service content and adds the transport-level header. The header identifies which community participant will handle the request, which PIP will be sent, and the version of the PIP along with other information. This information enables WebSphere Partner Gateway to send the correct PIP to the community participant.

You can find information about setting up RosettaNet support and configuring PIPs in the *Hub Configuration Guide*.

Event notification

WebSphere Partner Gateway executes RNIF PIP processes with community participants on behalf of the Community Manager back-end applications. Therefore, WebSphere Partner Gateway provides *event notification* as the mechanism to notify the back-end application about various aspects of the RNIF PIP process execution. Event notification enables WebSphere Partner Gateway to, for example, notify the application if WebSphere Partner Gateway is unable to send a PIP to the participant. The application can then handle the failure.

An event notification message is an XML document that carries information about events that occurred within WebSphere Partner Gateway or an application. These messages have the same structure as any other message that enters or leaves WebSphere Partner Gateway; that is, they have a transport-level header and payload. WebSphere Partner Gateway can be configured to send or not send event notification messages, because they are optional.

Table 2 summarizes the event notification messages that WebSphere Partner Gateway can send to back-end systems.

Table 2. Event notification messages sent to back-end system

Event condition	Event notification message
WebSphere Partner Gateway delivers a RosettaNet document to a community participant and receives a Receipt Acknowledgment.	Event 100
WebSphere Partner Gateway cancels a PIP by generating an 0A1 message and delivering it to the community participant.	Event 800
WebSphere Partner Gateway receives a Receipt Acknowledgment exception or a general exception from a community participant.	Event 900

WebSphere Partner Gateway can send 0A1 messages to the destination application as it would do with any other PIP, if it has been configured to send these messages using Exclusion List Management. See “Managing Exclusion Lists” in the *Administrator Guide*.

An application can send an event notification message to WebSphere Partner Gateway to cancel a RosettaNet PIP.

Event message structure

An event notification message has the standard transport-level header with the x-aux-process-type field set to `XMLEvent`. However, the payload of the message has a specific structure, as shown in the XML schema in Figure 11.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   targetNamespace=
 "http://www.ibm.com/websphere/bcg/2003/v1.0/xmleventnotification"
   xmlns:evntf=
 "http://www.ibm.com/websphere/bcg/2003/v1.0/xmleventnotification"
   elementFormDefault="qualified">

 <!-- EventNotification version 1.0 document element -->
    <xsd:element name="EventNotification">
       <xsd:complexType>
         <xsd:all>
             <xsd:element ref="evntf:StatusCode"/>
             <xsd:element ref="evntf:StatusMessage"/>
             <xsd:element ref="evntf:EventMessageID"/>
             <xsd:element ref="evntf:BusinessObjectID"/>
             <xsd:element ref="evntf:GlobalMessageID"/>
             <xsd:element ref="evntf:Timestamp"/>
          </xsd:all>
       </xsd:complexType>
    </xsd:element>

<!-- StatusCode element -->
    <xsd:element name="StatusCode">
       <xsd:simpleType>
          <xsd:restriction base="xsd:string">
             <xsd:enumeration value="100"/>
             <xsd:enumeration value="800"/>
             <xsd:enumeration value="900"/>
             <xsd:enumeration value="901"/>
             <xsd:enumeration value="902"/>
             <xsd:enumeration value="903"/>
             <xsd:enumeration value="904"/>
          </xsd:restriction>
       </xsd:simpleType>
    </xsd:element>
```

Figure 11. XML schema for an event notification message (Part 1 of 2)

```
<!-- StatusMessage element -->
    <xsd:element name="StatusMessage">
       <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
       </xsd:simpleType>
    </xsd:element>
<!-- EventMessageID element -->
    <xsd:element name="EventMessageID">
       <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
       </xsd:simpleType>
    </xsd:element>
<!-- BusinessObjectID element -->
    <xsd:element name="BusinessObjectID">
       <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
       </xsd:simpleType>
    </xsd:element>
 <!-- GlobalMessageID element -->
    <xsd:element name="GlobalMessageID">
       <xsd:simpleType>
          <xsd:restriction base="xsd:string"/>
       </xsd:simpleType>
    </xsd:element>
  <!-- Timestamp element -->
    <xsd:element name="Timestamp">
       <xsd:simpleType>
          <xsd:restriction base="xsd:dateTime"/>
       </xsd:simpleType>
    </xsd:element>
 </xsd:schema>
```

Figure 11. XML schema for an event notification message (Part 2 of 2)

Table 3 describes each field within the event payload.

Table 3. Event notification XML fields

Field	**Description**
StatusCode	Type of message. The valid values are: • **100** - WebSphere Partner Gateway has delivered the document and received a receipt acknowledgment. • **800** - The application canceled the PIP. • **900** - WebSphere Partner Gateway received a Receipt Acknowledgment exception, a general exception, or a 0A1Failure PIP from the community participant.
StatusMessage	Alphanumeric description of this event notification message.
EventMessageID	Alphanumeric identifier of this particular event notification message.
BusinessObjectID	The x-aux-msg-id in the transport-level header of the message affected by this message notification event. This links the payload of the original message to this event.
GlobalMessageID	The x-aux-system-msg-id in the transport-level header of the message that caused this message notification event.
Timestamp	When the event occurred using the UTC time stamp format: `CCYY-MM-DDThh:mm:ssZ` including fractional precision of seconds (...ss.ssssZ). The date stamp must conform to the XML schema data type for dateTime (w3.org/TR/2001/REC-xmlschema-2-20010502#dateTime).

Event notification example

Figure 12 shows an example of an event notification message sent using the HTTP protocol.

```
POST /builderURL HTTP/1.1
Content-Type: application/xml
Content-length: 250
x-aux-sender-id: 000000001
x-aux-receiver-id: 000000002
x-aux-third-party-bus-id: 000000003
x-aux-create-datetime: 2002-10-28T23:05:02Z
x-aux-protocol: XMLEvent
x-aux-protocol-version: 1.0
x-aux-process-type: XMLEvent
x-aux-process-version: 1.0
x-aux-payload-root-tag: evntf:EventNotification
x-aux-msg-id: 98732
x-aux-system-msg-id: 12345
x-aux-production: Production
x-aux-process-instance-id: 3456
x-aux-event-status-code: 100
x-aux-transport-retry-count: 0

<?xml version="1.0" encoding="UTF-8"?>
<evntf:EventNotification xmlns:evntf=
   "http://www.ibm.com/websphere/bcg/2003/v1.0/xmleventnotification">
   <evntf:StatusCode>100</evntf:StatusCode>
   <evntf:StatusMessage>The message was delivered</evntf:StatusMessage>
   <evntf:EventMessageID>12345</evntf:EventMessageID>
   <evntf:BusinessObjectID>34234</evntf:BusinessObjectID>
   <evntf:GlobalMessageID>98732</evntf:GlobalMessageID>
   <evntf:Timestamp>2001-01-31T13:20:00Z</evntf:Timestamp>
</evntf:EventNotification>
```

Figure 12. Example of an event notification message using HTTP

Which packaging will you use?

The packaging type determines the format in which WebSphere Partner Gateway sends the message to the back-end system and the format in which the back-end system sends the message to WebSphere Partner Gateway.

You use the Community Console to establish the connection with your community participants and to specify the packaging that is used between WebSphere Partner Gateway and the back-end system. To determine which packaging to use, you must understand the following issues:

- Which packaging types are valid for use with a back-end system?
- Which packaging types are valid with a message in a particular business protocol?

For more information on how to set up partner connections, see the *Hub Configuration Guide*.

Not all packaging types are valid when you use WebSphere Partner Gateway for integration. Table 4 lists the packaging types that are relevant when WebSphere Partner Gateway is exchanging documents or messages with a back-end application of the Community Manager.

Table 4. Packaging types relevant for back-end integration

Packaging type	Description
None packaging	Causes the message to be sent to the back-end system or to the hub *without* any header data
Backend Integration packaging	Adds additional attributes to the message header and, optionally, wraps the message contents in an XML transport envelope

Note: Other packaging types (such as AS) are available with WebSphere Partner Gateway. However, for integration with back-end systems, only the None and Backend Integration packaging types are recommended.

None packaging

When packaging is set to None, WebSphere Partner Gateway neither adds a transport-level header when it sends a message to a back-end system nor expects one when it receives a message from a back-end system. Instead, WebSphere Partner Gateway sends only the message to the back-end system. Information in the document controls routing.

Backend integration packaging

When packaging is set to Backend Integration, messages sent to or received from a back-end system have the following components:

- A transport-level header, which contains meta-information about the message (required)
- A payload, which contains the content of the message (required)
- An attachment (optional)

The header and payload are mandatory while attachments are optional. The following sections describe each of the components of a document that uses Backend Integration packaging.

Transport-level header content

The transport-level header contains information that WebSphere Partner Gateway uses to process and route the message to the correct destination. The transport-level header is bi-directional so that all messages entering and leaving WebSphere Partner Gateway have the mandatory fields and any of the optional fields that apply.

Table 5 lists the fields of the transport-level header.

Table 5. Transport-level header fields

Header field	Description	Required?
x-aux-sender-id	Identifier of the message sender, such as a DUNS number.	Yes
x-aux-receiver-id	Identifier of the message receiver, such as a DUNS number.	Yes
x-aux-protocol	Protocol of the message content. Valid values include RNSC for RosettaNet service content, XMLEvent, and Binary. For WebSphere Partner Gateway, the value in this field has priority over any protocol field in the payload.	Yes
x-aux-protocol-version	Version of the message content protocol.	Yes

Table 5. Transport-level header fields (continued)

Header field	Description	Required?
x-aux-process-type	Process to be performed or what type of message is being sent. For RosettaNet messages, this is the PIP code (for example, 3A4). For event messages, it is XMLEvent, and for Binary messages, it is Binary. For WebSphere Partner Gateway, the value in this field has priority over any process field in the payload.	Yes
x-aux-process-version	Version of the process. For RosettaNet messages, this is the version number of the PIP.	Yes
x-aux-create-datetime	When the message was successfully posted using the UTC time stamp format (CCYY-MM-DDThh:mm:ssZ).	
x-aux-msg-id	Identifier of the payload content. For example, it could be the identifier of the RNPIPServiceContent instance for a RosettaNet message or it could be a proprietary document identifier. This links the message payload with something in the message sender's system for tracing purposes.	
x-aux-production	Routing of the message. Valid values are: Production and Test. This value is populated for requests in both directions. Note that when the message is a response to a two-way PIP initiated by a community participant, WebSphere Partner Gateway uses the GlobalUsageCode in the request and ignores the value in the transport level header.	
x-aux-system-msg-id	Global Unique Identifier (GUID) for the message, which is used for duplicate checking.	Yes
x-aux-payload-root-tag	Root tag element of the payload. For example, for 3A4 RosettaNet service content, the value of this field would be Pip3A4PurchaseOrderRequest. For event notification messages, the value for this field would be EventNotification.	
x-aux-process-instance-id	Identifier that links documents in a multiple-message business process to a unique process instance. For RosettaNet, it must be unique for RosettaNet processes within the last 30 days. All messages exchanged as part of a RosettaNet process instance, including retries, use the same process instance ID.	
x-aux-event-status-code	Status code for the event notification. See the StatusCode field in "Event message structure" on page 16.	
x-aux-third-party-bus-id	Identifier such as a DUNS number of the party that delivered the message. This can be different from both the x-aux-sender-id and the x-aux-receiver-id if a third party is hosting WebSphere Partner Gateway on behalf of the community owner.	
x-aux-transport-retry-count	Number of unsuccessful attempts at posting this message prior to this attempt. If a message posts successfully on the first attempt, the value of this field will be 0.	
x-out-file-name	The original file name for messages being sent over JMS with Backend Integration packaging. (See note 2 on page 22.)	
content-type	The content type of the message.	
content-length	The length of the message (in bytes).	

Notes:

1. For compatibility with IBM WebSphere MQ (a JMS provider), the fields of a JMS protocol message use underscores instead of hyphens. For example, in a JMS message, there is an x_aux_sender_id field instead of an x-aux-sender-id field.

2. If the target is specified as HTTP and the package is specified as None, the original file name is set in the HTTP headers as “Content-Disposition: attachment;po.xml”.

   If the target is specified as JMS and the package is specified as Backend Integration, the original file name is written into x-out-file-name along with other x-aux-* headers.

Table 5 provides an overview of the transport-level header information. The following sections provide transport-level header information specific to certain business protocols:
- “Transport-level header and a RosettaNet message”
- “Transport-level header and an AS2 message” on page 23
- “Transport-level header and an AS1 message” on page 24

Transport-level header and a RosettaNet message: Table 6 describes where WebSphere Partner Gateway obtains values for the fields of the transport-level header from a RosettaNet message.

Table 6. Transport-level header fields and RosettaNet content

Header field	**Source of value: RosettaNet 2.0**	**Source of value: RosettaNet 1.1**
x-aux-sender-id	`<(DeliveryHeader)>` `<messageSenderIdentification>` `<PartnerIdentification>` `<GlobalBusinessIdentifier>`	`<ServiceHeader>` `<ProcessControl>` `<TransactionControl>` `<ActionControl>` *or* `<SignalControl>` `<PartnerRouter>` `<fromPartner>` `<PartnerDescription>` `<BusinessDescription>` `<GlobalBusinessIdentifier>`
x-aux-receiver-id	`<(DeliveryHeader)>` `<messageReceiverIdentification>` `<PartnerIdentification>` `<GlobalBusinessIdentifier>`	`<ServiceHeader>` `<ProcessControl>` `<TransactionControl>` `<ActionControl>` *or* `<SignalControl>` `<PartnerRouter>` `<toPartner>` `<PartnerDescription>` `<BusinessDescription>` `<GlobalBusinessIdentifier>`
x-aux-protocol	Set value for RosettaNet: RNSC	*Same as for RosettaNet 2.0*
x-aux-protocol-version	Set value: 1.0	*Same as for RosettaNet 2.0*
x-aux-process-type	The source XPath is: `/ServiceHeader/ProcessControl/pipCode/GlobalProcessIndicatorCode`	The source XPath is: `/ServiceHeader/ProcessControl/ProcessIdentity/GlobalProcessIndicatorCode`
x-aux-process-version	The source XPath is: `/ServiceHeader/ProcessControl/pipVersion/VersionIdentifier` The value of the version identifier of each PIP is in its PIP specification.	The source XPath is: `/ServiceHeader/ProcessControl/ProcessIdentity/VersionIdentifier` The value of the version identifier of each PIP is in its PIP specification.
x-aux-payload-root-tag	Name of the PIP, such as Pip3A4PurchaseOrderRequest	*Same as for RosettaNet 2.0*

Table 6. Transport-level header fields and RosettaNet content (continued)

Header field	Source of value: RosettaNet 2.0	Source of value: RosettaNet 1.1
x-aux-process-instance-id	For processes initiated by the application, the value is the ID of the process instance. For processes initiated by a community participant that are not pass-through workflow, the value is the process ID in the initial RosettaNet request: `<ServiceHeader>` `<ProcessControl>` `<pipInstanceId>` `<InstanceIdentifier>`	`<ServiceHeader>` `<ProcessControl>` `<ProcessIdentity>` `<InstanceIdentifier>`
x-aux-msg-id	`<(RNPipServiceContent)>` `<thisDocumentIdentifier>` `<ProprietaryDocumentIdentifier>`	*Same as for RosettaNet 2.0*
x-aux-production	`<ServiceHeader>` `<ProcessIndicator>` `<GlobalUsageCode>`	`<Preamble>` `<GlobalUsageCode>`

Transport-level header and an AS2 message: Table 7 describes where WebSphere Partner Gateway obtains values for the fields of the transport-level header from an AS2 message.

Note: The values are case-sensitive.

Table 7. Transport-level header fields from AS2 content

Header field	Source of value when a community participant sends an AS/2 message to the hub	Source of value when an AS2 message is sent to a community participant
x-aux-sender-id	The AS2-From header field of the AS2 message is set in the x-aux-sender-id field of the back-end integration message that is sent to the Community Manager.	The x-aux-sender-id field of the incoming back-end integration message is used as the AS2-From header value of the AS2 message.
x-aux-receiver-id	The AS2-To header field of the AS2 message is set in the x-aux-receiver-id field of the back-end integration message that is sent to the Community Manager.	The x-aux-receiver-id field of the incoming back-end integration message is used as the AS2-To header value of the AS2 message.
x-aux-protocol	The ToProtocol of the participant connection is set in the x-aux-protocol field of the back-end integration message that is sent to the Community Manager.	The x-aux-protocol field of the incoming back-end integration message is used to determine the FromProtocol of the participant connection.
x-aux-protocol-version	The ToProtocolVersion of the participant connection is set in the x-aux-protocol-version field of the back-end integration message that is sent to the Community Manager.	The x-aux-protocol-version field of the incoming back-end integration message is used as the FromProtocolVersion of the participant connection.
x-aux-process-type	The ToProcessCode of the participant connection is used to set the x-aux-process-type field of the back-end integration message that is sent to the Community Manager.	The x-aux-process-type field of the incoming back-end integration message is used as the FromProcessCode of the participant connection.

Table 7. Transport-level header fields from AS2 content (continued)

Header field	Source of value when a community participant sends an AS/2 message to the hub	Source of value when an AS2 message is sent to a community participant
x-aux-process-version	The ToProcessVersion of the participant connection is set in the x-aux-process-version field of the back-end integration message that is sent to the Community Manager.	The x-aux-process-version field of the incoming back-end integration message is used as the FromProcessVersion of the participant connection.
x-aux-payload-root-tag	For custom XML protocol only, the Root tag specified in the XPATH is parsed out of the message and used in the x-aux-payload-root-tag field.	This field does not need to be set in the incoming back-end integration message.
x-aux-process-instance-id	This field is not used for AS2.	This field is not used for AS2.
x-aux-msg-id	For custom XML protocol only, the Doc ID specified in the XPATH is parsed out of the message and used in the x-aux-msg-id field.	This field does not need to be set in the incoming back-end integration message
x-aux-system-msg-id	This field is set to the internally generated unique ID for this message.	This field does not need to be set in the incoming back-end integration message
x-aux-production	This field is not used for AS2.	This field is not used for AS2.

Transport-level header and an AS1 message: Table 8 describes where WebSphere Partner Gateway obtains values for fields in the transport-level header from an AS1 message.

Note: The values are case-sensitive.

Table 8. Transport-level header fields from AS1 content

Header field	Source of value when a community participant sends an AS/1 message to the hub	Source of value when an AS/1 message is sent to a community participant
x-aux-sender-id	The *FromID* in the "Subject: *ToID;FromID*" header field of the AS1 message is set in the x-aux-sender-id field of the back-end integration message that is sent to the Community Manager.	The x-aux-sender-id field of the incoming back-end integration message is used as *FromID* in the "Subject: *ToID;FromID*" header value of the AS1 message.
x-aux-receiver-id	The *ToID* in the "Subject: *ToID;FromID*" header field of the AS1 message is set in the x-aux-receiver-id field of the back-end integration message that is sent to the Community Manager.	The x-aux-receiver-id field of the incoming back-end integration message is used as *ToID* in the "Subject: *ToID;FromID*" header value of the AS1 message.
x-aux-protocol	The ToProtocol of the participant connection is set in the x-aux-protocol field of the back-end integration message that is sent to the Community Manager.	The x-aux-protocol field of the incoming back-end integration message is used as the FromProtocol of the participant connection.
x-aux-protocol-version	The ToProtocolVersion of the participant connection is set in the x-aux-protocol-version field of the back-end integration message that is sent to the Community Manager.	The x-aux-protocol-version field of the incoming back-end integration message is used as the FromProtocolVersion of the participant connection.

Table 8. Transport-level header fields from AS1 content (continued)

Header field	Source of value when a community participant sends an AS/1 message to the hub	Source of value when an AS/1 message is sent to a community participant
x-aux-process-type	The ToProcessCode of the participant connection is set in the x-aux-process-type field of the back-end integration message that is sent to the Community Manager.	The x-aux-process-type field of the incoming back-end integration message is used as the FromProcessCode of the participant connection.
x-aux-process-version	The ToProcessVersion of the participant connection is set in the x-aux-process-version field of the back-end integration message that is sent to the Community Manager.	The x-aux-process-version field of the incoming back-end integration message is used as the FromProcessVersion of the participant connection.
x-aux-payload-root-tag	For custom XML protocol only, the Root tag specified in the XPATH is parsed out of the message and set in the x-aux-payload-root-tag field.	This field does not need to be set in the incoming back-end integration message.
x-aux-process-instance-id	This field is not used for AS1.	This field is not used for AS1.
x-aux-msg-id	For custom XML protocol only, the Doc ID specified in the XPATH is parsed out of the message and used in the x-aux-msg-id field.	This field does not need to be set in the incoming back-end integration message.
x-aux-system-msg-id	This field is set to the internally generated unique ID for this message.	This field does not need to be set in the incoming back-end integration message.
x-aux-production	This field is not used for AS1.	This field is not used for AS1.

Payload

The payload of the message contains the actual content of the message. The location of the payload depends on the transport protocol that is sending the message, as Table 9 shows.

Table 9. Location of payload

Transport protocol	Location of payload
HTTP protocol messages	In the body of the HTTP post
JMS protocol messages	In the body of the JMS message
RosettaNet messages	The service content from the PIP
EDI	The EDI envelope
ROD or XML document	The ROD or XML document

The payload can be Base64-encoded and in an XML *transport envelope* in either of the following cases:

- If the document contains an attachment

  A document with attachments *must* be wrapped in an XML transport envelope. For more information on how attachments are formatted, see "Attachments" on page 26.
- If you set the envelope flag for Backend Integration packaging to Yes

  To wrap a document in the XML transport envelope *regardless* of whether it contains attachments, set the Backend Integration envelope flag to Yes from the profile's B2B Capabilities screen. For example, to set this flag in the Community Manager's profile, perform the following tasks:

1. Click **Account Admin > Profiles**.
2. Enter the name of the Community Manager (or perform a search on all participants).
3. Click the **View details** icon next to the name of the Community Manager.
4. Click **B2B Capabilities**.
5. Click the **Edit** icon next to **Backend Integration**.
6. Set the **Envelop Flag** to **Yes**.

This XML transport envelope wraps the document in the `<transport-envelope>` root tag. Inside this root tag there is a `<payload>` tag that contains the document payload. If any attachments exist, each is contained in an `<attachment>` tag. For more information on the structure of these tags, see "Attachments."

WebSphere Partner Gateway includes the following W3C XML schema file that describes the Backend Integration XML transport envelope structure:

```
wbipackaging_v1.0_ns.xsd
```

This schema file is located in the following directory on the installation medium:

```
B2BIntegrate\packagingSchemas
```

You can use any XML editing tool to validate your Backend Integration XML against this schema file to ensure the document is valid before it is sent to the Document Manager.

Attachments

If the business message protocol permits them, each document can have one or more attachments. If the document has attachments, it *must* be wrapped in an XML transport envelope, as described in "Payload" on page 25. Table 10 describes the XML attributes in the payload and attachment tags.

Table 10. XML attributes of the payload and attachment tags

XML attribute	**Description**	**Required?**
`Content-Type`	Identifies the MIME type/subtype, such as text/xml or image/gif.	Yes
`Encoding`	Identifies the encoding. Because the attachment and payload must be Base64-encoded, the only valid value for this attribute is ″Base64″.	No

Figure 13 shows an example of a document in an XML transport envelope that contains the payload and one attachment.

Note: The namespace in this example is required:

```
xmlns="http://www.ibm.com/websphere/bcg/2003/v1.0/wbipackaging"
```

```
<?xml version="1.0" encoding="utf-8"?>
<transport-envelope
    xmlns="http://www.ibm.com/websphere/bcg/2003/v1.0/wbipackaging">
  <payload encoding="base64" contentType="application/xml">
    ...base64 encoded XML message...
  </payload>
  <attachment encoding="base64" Content-Type="text/xml">
    ...base64 encoded XML attachment...
  </attachment>
</transport-envelope>
```

Figure 13. Sample XML transport envelope for payload and one attachment

Notes:

1. To process documents wrapped in the XML transport envelope with the WebSphere Interchange Server, WebSphere Partner Gateway provides the Attachment data handler. For more information, see "Handling documents with attachments" on page 123.
2. To process documents with attachments on WebSphere Process Server, WebSphere Partner Gateway provides Backend Integration packaging data binding. See "Handling Backend Integration Packaging messages" on page 55.

Which packaging type works with your documents?

Documents in certain business protocols can use only certain types of packaging. For example, a RosettaNet document can be processed *only* when a packaging of Backend Integration has been specified. See Table 12 on page 29, Table 13 on page 29, and Table 14 on page 30 for a complete list of which document types can be associated with which types of packaging.

Example of Backend Integration packaging over HTTP

Figure 14 shows an example of a message from WebSphere Partner Gateway to an application using the HTTP transport protocol. Note that the message does not contain an attachment.

```
POST /sample/receive HTTP/1.1
Host: sample. COM
Content-Type: application/xml
Content-Length: nnn
x-aux-sender-id: 000000001
x-aux-receiver-id: 000000002
x-aux-third-party-bus-id: 000000003
x-aux-create-datetime: 2002-10-28T23:05:02Z
x-aux-protocol: RNSC
x-aux-protocol-version: 1.0
x-aux-process-type: 3A4
x-aux-process-version: V02.00
x-aux-payload-root-tag: Pip3A4PurchaseOrderRequest
x-aux-msg-id: 1021358129419
x-aux-system-msg-id: 2
x-aux-production: Production
x-aux-process-instance-id: 123456
x-aux-transport-retry-count: 0

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Pip3A4PurchaseOrderRequest SYSTEM
     "3A4PurchaseOrderRequestMessageGuideline_v1_2.dtd">
<Pip3A4PurchaseOrderRequest>

   <PurchaseOrder>
      ...
   </PurchaseOrder>
      ...
   <thisDocumentIdentifier>
      <ProprietaryDocumentIdentifier>1021358129419
      </ProprietaryDocumentIdentifier>
   </thisDocumentIdentifier>
   <GlobalDocumentFunctionCode>Request</GlobalDocumentFunctionCode>

</Pip3A4PurchaseOrderRequest>
```

Figure 14. Sample message using HTTP transport protocol

Which message transport will you use?

When the back-end system and WebSphere Partner Gateway send messages to one another, each must use the same message-transport protocol. The *message transport protocol* defines the communication protocol in which the messages are sent.

WebSphere Partner Gateway communicates with a back-end system through its Backend Integration interface. Table 11 lists the transport protocols that this Backend Integration interface supports.

Table 11. Transport protocols supported by Backend Integration

Transport protocol	For more information
HTTP or HTTPS	"HTTP transport protocol" on page 31
File-system files	"File-system protocol" on page 35
JMS	"JMS protocol" on page 32

Table 12 on page 29 shows which transport protocols are supported for the packaging types and business protocols when the hub is sending documents to the back-end system.

Table 12. Supported transport protocols from WebSphere Partner Gateway to back-end system

Packaging type	Business protocol	HTTP or HTTPS?	JMS?	File system?
Backend Integration	RosettaNet (RNSC)	Yes	Yes	No
	Binary	Yes	Yes	No
	EDI (see Table 14 on page 30 for information on EDI)			
	XML	Yes	Yes	No
	ROD	Yes	Yes	No
None	EDI (see Table 14 on page 30 for information on EDI)			
	cXML only	Yes	No	No
	SOAP only	Yes	No	No
	Binary	Yes	Yes	Yes
	XML	Yes	Yes	Yes
	ROD	Yes	Yes	Yes

Table 13 shows which transport protocols are supported for the packaging types and business protocols when the back-end system is sending documents to the hub.

Table 13. Supported transport protocols from back-end system to WebSphere Partner Gateway

Packaging type	Business protocol	HTTP or HTTPS?	JMS?	File system?
Backend Integration	RosettaNet (RNSC)	Yes	Yes	No
	XML	Yes	Yes	No
	Binary	Yes	Yes	No
	ROD	Yes	Yes	No
None	XML only	Yes	Yes	Yes
	EDI (See Table 14 on page 30 for information on EDI).			
	cXML only	Yes	No	No
	SOAP only	Yes	No	No
	Binary only	No	No	No
	ROD only	Yes	Yes	Yes

Table 14 on page 30 shows which transport protocols and packaging types are supported for various EDI, XML, and record-oriented data (ROD) documents.

Table 14. Supported transport protocols between WebSphere Partner Gateway and the back-end system for EDI

Packaging type	Document	HTTP or HTTPS	JMS	File system
Backend Integration	Single interchange containing a single transaction (such as an X12 850 transaction within an envelope)	Yes	Yes	No
	Single interchange containing multiple transactions (such as an X12 850 transaction and an X12 890 transaction within the same envelope)	Yes	Yes	No
	Multiple interchanges containing a single transaction (such as two X12 envelopes within the same file, each of which contains a single transaction)	Yes	Yes	No
	Multiple interchanges containing multiple transactions (such as two X12 envelopes within the same file, each of which contains two or more transactions)	Yes	Yes	No
	EDI transaction (for example, an X12 850 transaction), which cannot be delivered by itself because a transaction must be within an EDI interchange	No	No	No
	Document (for example, XML) that is later transformed into an EDI transaction	Yes	Yes	No
None	Single interchange containing a single transaction	Yes	Yes	Yes
	Single interchange containing multiple transactions	Yes	Yes	Yes
	Multiple interchange containing a single transaction	Yes	Yes	Yes
	Multiple interchanges containing multiple transactions	Yes	Yes	Yes
	EDI transaction (not supported; must have interchange envelope)	No	No	No
	Document (for example, XML) that is later transformed into an EDI transaction	Yes	Yes	Yes

The previous tables list the transport protocols that are valid between the hub and the back-end system. The hub can use additional transport protocols to send documents to or receive documents from participants. For example, the hub can send a document to a remote FTP server by using the FTP Scripting transport. It can also receive documents using the FTP Scripting transport. The FTP Scripting transport, which is described in the *Hub Configuration Guide*, can be used to send and receive documents over the Internet but it must be used to send and receive documents from Value Added Networks (VANs).

HTTP transport protocol

To send messages using an HTTP protocol, WebSphere Partner Gateway uses HTTP/S 1.1. To receive messages from back-end systems, WebSphere Partner Gateway supports both HTTP/S version 1.0 and 1.1.

The HTTP message can include the integration packaging attributes. Whether these attributes are included depends on the packaging type associated with the participant connection, as follows:

- If the participant connection specifies that the HTTP message includes Backend Integration packaging, the transport-level header of the HTTP message includes additional attributes containing information about the message, such as the protocol of the content, the ID of the message, and the sender of the message. For a complete list of the fields in the header, see "Transport-level header content" on page 20.

  RosettaNet messages must use Backend Integration packaging.
- If the participant connection specifies None packaging, the HTTP message does *not* have these additional attributes, and WebSphere Partner Gateway parses the message to obtain this information.

  SOAP and cXML messages must use None packaging.

  Note: XML messages can use either None or Backend Integration packaging. Similarly, EDI documents can use either None or Backend Integration. Binary messages received from the back-end system must have the Backend Integration packaging; however, the reverse is not true because WebSphere Partner Gateway supports sending binary messages to the application using either type of packaging.

Process

When HTTP or HTTPS messages are sent between WebSphere Partner Gateway and an application for asynchronous exchanges, the following steps occur:

1. The source system (WebSphere Partner Gateway or the back-end system) posts an HTTP message to the target system using a specific URL.
2. The target system receives the message and sends the protocol-level acknowledgment, HTTP 200 or 202, to signify the change in ownership. The source system ignores the body of this acknowledgment message. If an error occurs during this processing, the target system sends an HTTP 500 message back to the source system.
3. If WebSphere Partner Gateway is the target system (that is, when WebSphere Partner Gateway receives a message), it then persists the message and releases the connection to the source system.
4. The target system can then process the message asynchronously.

When the exchange is synchronous (for example, for a SOAP or cXML document), a response is returned along with the HTTP 200 message in the same HTTP connection.

Sending messages from the back-end system using the HTTP protocol

To send a message to WebSphere Partner Gateway using the HTTP protocol, a back-end system takes the following steps:

1. Creates the message.

   The Content-Type attribute in the transport-level header gives the encoding used for the message.
2. Packages the message according to the packaging type set for the connection.

   For Backend Integration packaging, the back-end system adds the protocol header attributes that WebSphere Partner Gateway requires.
3. Posts the message to the URL that WebSphere Partner Gateway uses to receive these messages.
4. If the exchange is synchronous, the back-end system waits to receive a response in the same connection that was used for the request.

To enable HTTP message exchange in this direction, use the Target Details page of the Community Console to set up a target at the hub for inbound documents. This target specifies a URL. The back-end system needs to know this address to send documents to the hub.

Receiving messages at the back-end system using the HTTP protocol

To receive a message from WebSphere Partner Gateway using the HTTP protocol, a back-end system takes the following steps:

1. Listens for a message at a particular URL.
2. When a message is received, processes the message:
   - If the connection has None packaging, the back-end system must parse the message to determine how to handle it.
   - If the connection has Backend Integration packaging, the application can use the Backend Integration attributes to determine how to handle the message.
3. If the exchange is synchronous, the back-end system returns a response in the same connection used for the request.

To enable HTTP message exchange in this direction, use the Gateway page of the Community Console to set up a gateway that specifies where documents should be delivered to the back-end system.

JMS protocol

The JMS protocol is based on the Java$^{(TM)}$ Message Service (JMS) and transfers messages through transactional, persistent JMS queues provided by, for example, IBM WebSphere MQ. The JMS protocol supports the following JMS message types:

- StreamMessage (as a byte array)
- BytesMessage (as a byte array)
- TextMessage

In the JMS protocol, one system sends a JMS message to another. After the second system receives the message, it removes it from the queue. From this point forward, the receiving system can process the message asynchronously.

The JMS message can include integration packaging attributes. Whether these attributes are included depends on the packaging type associated with the participant connection, as follows:

- If the participant connection specifies that the JMS message includes Backend Integration packaging, the JMS message contains transport-level information (such as the protocol of the content, the ID of the message, and the sender of the message) as JMS properties within the message. For a complete list of the properties, see "Transport-level header content" on page 20.

Note: For compatibility with WebSphere MQ JMS, the properties in the JMS messages use underscores in the property names instead of hyphens. For example, in a JMS message, the property is x_aux_system_msg_id while the equivalent HTTP header field will be x-aux-system-msg-id. When WebSphere Partner Gateway processes a JMS message, it converts the underscores to hyphens in these properties.

- If the participant connection specifies None packaging, the JMS message does *not* have these additional attributes.

With the exception of binary messages, WebSphere Partner Gateway supports sending and receiving JMS messages with either type of packaging. Binary messages received from an application must have the Backend Integration packaging. The reverse is not true because WebSphere Partner Gateway supports sending binary messages to the application using either type of packaging.

Setting up the JMS environment

To set up your JMS environment, the following providers are required.

- JMS provider

  A JMS provider provides the implementation of JMS API support for messaging. The backend system with which you are exchanging documents determines which JMS provider you use.

  – If you are exchanging documents with WebSphere Integration Server, you use WebSphere MQ as the JMS provider.

  – If you are exchanging documents with WebSphere Process Server when WebSphere Partner Gateway is installed on WebSphere Application Server, you use Websphere Platform Messaging and a service integration bus as the JMS provider. If you are exchanging documents with WebSphere Process Server when WebSphere Partner Gateway is installed on an embedded application server, you use WebSphere Platform Messaging and a service integration bus as the JMS provider, and you use WebSphere MQ to communicate with WebSphere Platform Messaging. Details about the JMS providers available for use with WebSphere Process Server can be found in Chapter 5, "Integrating WebSphere Process Server with JMS as transport," on page 83.

  The JMS provider typically provides a program you can use to set up the JMS environment. For example, WebSphere MQ provides the JMSAdmin program, which lets you construct the objects required by JMS--the JMS connection factory and JMS queue objects. When these objects are constructed, references to them are stored in JNDI.

  Note: For messaging, WebSphere Partner Gateway supports the point-to-point model only.

- JNDI provider

  The JNDI provider supplies the implementation of JNDI, which is used to store references to JMS objects.

For a back-end application to send business documents to WebSphere Partner Gateway using the JMS protocol, a JMS target must be configured. The JMS target receives messages from a JMS queue, and the documents are introduced into the WebSphere Partner Gateway workflow. The JMS target configuration includes the required parameters for accessing the JNDI as well as the names of JMS objects.

For integration with the back-end system, the queue configured in the JMS target is the queue from which the back-end system is sending the JMS message.

Similarly, a JMS gateway is used by WebSphere Partner Gateway to send business documents to a queue where participants expect to receive them. Therefore, for sending messages to the back-end system, make sure a JMS gateway is configured in the profile of the Community Manager. The gateway should be configured to send to the queue on which the back-end system is receiving. The JMS gateway configuration includes the required parameters to access the JNDI as well as the names of JMS objects.

Overview of setting up the JMS environment

To communicate over the JMS transport protocol, WebSphere Partner Gateway and the back-end system require a JMS queue for *each* direction of the communication. Therefore, you must take the following steps to supply the appropriate JMS queues:

- Configure your JMS environment.
- Create a queue manager and the required queues including the transmission queue, remote queue, and receiver queue.

The JMS queue manager can exist on any computer, including the following:

- The computer where the back-end system resides
- The computer where WebSphere Partner Gateway resides

In addition, you can have a queue manager on *both* the computer where the back-end system resides and the computer where WebSphere Partner Gateway resides. In this case, use setup channels to tie the two queue managers together. Using this method, neither side needs to make client connections over the network.

Instructions for configuring a JMS transport-protocol mechanism using WebSphere MQ version 5.3 are provided in the *Hub Configuration Guide*. Instructions for configuring the JMS environment when you are exchanging documents with WebSphere Process Server are described in Chapter 5, “Integrating WebSphere Process Server with JMS as transport,” on page 83.

Sending messages from the back-end system using the JMS protocol

To send a message to WebSphere Partner Gateway using the JMS protocol, a back-end system takes the following steps:

1. Creates the message.
2. Packages the message according to the packaging type set for the connection.

   For Backend Integration packaging, the application adds the required JMS header attributes.
3. Sends the message to the JMS queue that the back-end system uses to send messages to WebSphere Partner Gateway.

Receiving messages at the back-end system using the JMS protocol

To receive a message from WebSphere Partner Gateway using the JMS protocol, a back-end system takes the following steps:

1. Listens for a message on the JMS queue.
2. When a message is received, processes the message:

- If the connection has None packaging, the back-end system must parse the message to determine how to handle it.
- If the connection has Backend Integration packaging, the application can use the Backend Integration attributes to determine how to handle the message.

File-system protocol

The File System protocol enables WebSphere Partner Gateway to send messages by placing them in a defined directory structure. WebSphere Partner Gateway receives messages by reading them from the directory structure. The file-system protocol supports only the None packaging type.

Sending messages from the back-end system using the file-system protocol

To send a message to WebSphere Partner Gateway using the file-system protocol, an application should take the following steps:

1. Create the message file in a temporary directory.
2. Once the file is ready, move the file to the directory that WebSphere Partner Gateway polls.

To enable file-system message exchange in this direction, use the Target Details page of the Community Console to set up a target for inbound documents. The target of the message determines the directory that WebSphere Partner Gateway polls. When you create a target, WebSphere Partner Gateway creates a Documents directory and its subdirectories for the target, as follows:

```
<doc_root>
    Documents
        Production
        Test
      <other destination types>
```

WebSphere Partner Gateway polls the Documents directories and their subdirectories regularly to detect message files. If it finds a message, WebSphere Partner Gateway persists the message and then deletes the message from the directory. WebSphere Partner Gateway then processes the message normally. See the *Hub Configuration Guide* for information on how to create a target.

Receiving messages at the back-end system using the file-system protocol

To receive messages using the file system protocol, an application should do the following:

1. Poll the appropriate directory for message files.
2. When there is a message, persist it.
3. Delete the message from the directory.
4. Process the message.

To enable file-system message exchange in this direction, use the Gateway page of the Community Console to set up a gateway that specifies where documents should be delivered. WebSphere Partner Gateway places the message file in the Documents directory, which the gateway defines. By defining the destination directory according to the gateway, each participant connection can have a different directory. For information on gateways, see the *Hub Configuration Guide*.

How do you access your back-end application?

WebSphere Partner Gateway provides the ability to integrate with many different back-end applications. Usually, a back-end application is accessed through a back-end system, such as an integration broker. Integration to the back-end systems listed in Table 15 is covered in this guide.

Table 15. Supported back-end systems for WebSphere Partner Gateway

Back-end system	For more information
WebSphere Process Server	Chapter 3, "Introduction to WebSphere Process Server integration," on page 49
WebSphere Interchange Server	Chapter 8, "Introduction to InterChange Server integration," on page 113
WebSphere Business Integration Message Broker	Chapter 11, "Integrating with WebSphere Business Integration Message Broker," on page 173
WebSphere Data Interchange	Chapter 12, "Integrating with WebSphere Data Interchange," on page 191

Message handling

This section describes how WebSphere Partner Gateway handles the following situations that affect the delivery of messages:

- "Queued delivery"
- "Communication error handling"
- "Duplicate messages" on page 37

Queued delivery

WebSphere Partner Gateway posts information on all documents that it wants to send to a particular gateway into a queue. The Document Manager processes these messages in the order the queue receives them (FIFO) and uses a thread for each message to send them. Note that if the gateway (for example, URL if the transport protocol is HTTP or JMS destination if the transport protocol is JMS) has been configured to be offline (see Communication error handling), the messages remain in the queue until the gateway is enabled (online). If the Document Manager receives an error in a thread, it stops other threads from attempting to deliver their messages. The Document Manager places these messages back into the queue until it is able to deliver the message that caused the error.

If the number of failed attempts exceeds the maximum number of attempts, the Document Manager places the message in a failed directory and then attempts to deliver the next message in the queue unless the gateway is offline.

Communication error handling

When WebSphere Partner Gateway is the sender and the application returns an error (for example, an HTTP Response message that is not a 200 or 202 message when using the HTTP protocol), WebSphere Partner Gateway may then try to send the message again depending on how it has been configured for this particular gateway. Each gateway (URL in the case of HTTP) has the following options that affect the number of retries and how the messages are sent:

Table 16. Gateway configuration options

Configuration options	Description
Retry Count	How many document retries to attempt if an error is received
Retry Interval	Time interval between retry attempts
Online/Offline	Starts and stops delivery attempts
Number of Threads	Number of posting threads that will process messages per gateway

If WebSphere Partner Gateway is not configured to retry sending the message or if all delivery attempts fail, WebSphere Partner Gateway signals the problem by doing any or all of the following actions:

- Presenting the errors in various screens of the Community Console such as the Document Viewer and RosettaNet Viewer
- Sending an e-mail to appropriate people to notify them of the problem so that they can take appropriate actions, if an e-mail alert for the delivery failed event has been set up
- Creating an event document and then sending that document to the receiver.

See "Managing gateway configurations" in the *Administrator Guide* for more information.

Duplicate messages

All messages sent to or received from WebSphere Partner Gateway must have a Global Unique Identifier (GUID). WebSphere Partner Gateway uses the GUID to detect duplicate messages. When Backend Integration packaging is used, each message carries its GUID in the transport-level header. For the HTTP protocol, for example, the GUID is carried in the `x-aux-system-msg-id` field (see "Transport-level header content" on page 20). The sender of the message generates the GUID. The file system protocol does not support checking for duplicate messages.

If the attempt to send a message results in an error, WebSphere Partner Gateway reuses the message's GUID in each retry. If WebSphere Partner Gateway receives a message that contains a duplicate GUID, it returns a positive acknowledgment (for example, HTTP 200) but does not process the duplicate message.

Note: WebSphere Partner Gateway checks for duplicate messages at the RosettaNet process level if RosettaNet is being used. It also checks for duplicate messages if XML is being used.

Configuring WebSphere Partner Gateway

"The hub configuration process" on page 4 provided a high-level description of the steps the Community Operator takes to configure the hub. This section summarizes steps for configuring WebSphere Partner Gateway for use with a back-end system. These configuration steps assume that you have already configured the community participants in your hub community. In particular, this section assumes that the following configuration has already been performed:

- A participant profile for the Community Manager has already been created.
- Community participants for the origin (or destination) of the documents have already been created.

- In the community participants' B2B programs, participant profiles for the Community Manager have been created.
- A target has been defined so that the WebSphere Partner Gateway Receiver can listen for incoming documents from the community participant over the appropriate transport protocol.
- B2B capabilities have been defined and enabled in the profile of the community participant (from which the document is received) so that WebSphere Partner Gateway expects documents from that source.
- Participant connections exist between the Community Manager and the community participants so that a participant and the Community Manager can receive (or send) a document.

Note: You should log in as the Community Operator (hub admin).

For a complete description of how to configure WebSphere Partner Gateway to support a hub community, see the *Hub Configuration Guide.*

After the community participants are configured, you must configure WebSphere Partner Gateway so that it can communicate with a back-end system. This section provides the following information to describe how to incorporate a back-end system into your hub community:

- "Sending documents to the back-end system"
- "Receiving documents from the back-end system" on page 42

Sending documents to the back-end system

To send a document to the back-end system, WebSphere Partner Gateway takes the following steps:

1. Receives a document from some community participant.

   The Receiver retrieves this source document from a target that has been defined at the hub for incoming messages from the community participant and its associated transfer protocol. When sending a document to the back-end system, the source document is the document that is received from some community participant; it is therefore referred to as the *participant document*.
2. Converts the participant document to the destination document, which is in the format that the back-end system requires.

   The WebSphere Partner Gateway Document Manager performs this conversion to the destination document. When sending a document to the back-end system, the destination document is the document that is sent to the back-end system; it is therefore referred to as the *back-end document*.
3. Sends the back-end document to the back-end system.

   The Document Manager sends the back-end document through a gateway that has been defined at the hub for outgoing messages to the back-end system.

Therefore, for the hub to be able to send a document to the back-end system, you must ensure that the configuration summarized in Table 17 has been performed within WebSphere Partner Gateway.

Table 17. Configuration steps to send documents to the back-end system

Configuration step	WebSphere Partner Gateway steps	For more information
1. Define where to send the document.	1. Create a gateway to the back-end system.	"Defining where to send the participant document" on page 39

Table 17. Configuration steps to send documents to the back-end system (continued)

Configuration step	WebSphere Partner Gateway steps	For more information
2. Define how to process the document.	2. Create document flow definitions for the source and destination formats. 3. Enable B2B capabilities for the document flow definition of the document sent to the back-end system. 4. Create a document flow definition interaction between the source and destination document flow definitions.	"Defining how to process the participant document" on page 40
3. Define how to connect to the back-end system.	5. Create a participant connection that sends documents to the back-end system.	"Defining how to connect to the back-end system" on page 41

Defining where to send the participant document

To send documents to the back-end system, the hub must have a gateway defined. This gateway specifies the destination for the converted documents; that is, it specifies the location (as a URI) to which the hub sends the back-end document. This location is the same one at which the back-end system listens for incoming messages. The gateway identifies the entrance point into the enterprise application layer (within the back-end system). Within WebSphere Partner Gateway, it is Document Manager that checks for a gateway. Once the Document Manager has processed the document, it sends the converted document to the back-end system at the location specified in the gateway.

To define a gateway within WebSphere Partner Gateway

1. Click **Account Admin > Profiles**.
2. Click Search to display a list of participants.
3. Select the **View details** icon next to the Community Manager.
4. Click **Gateways**.
5. Click **Create**.

When you define the gateway, you specify the transport protocol that the hub and back-end system both use to transfer the back-end document. As Table 12 on page 29 shows, the choice of transport protocol depends on the format of the document. Its format includes its packaging type and business protocol, which are defined in its document flow definition.

Note: For more information on how to create a gateway in WebSphere Partner Gateway, see the *Hub Configuration Guide.*

The choice of transport protocol also depends on the transport protocols your particular back-end system supports. For more information, refer to the chapter in this guide for integrating your particular back-end system.

Once you have selected a valid transport protocol for your document, you can provide the other information you need to define for the gateway in the Gateways screen.

Defining how to process the participant document

For the Document Manager to be able to process the participant document, it must know the format to which it needs to convert this document; that is, it needs to know the format of the back-end document. As part of the back-end integration, you must ensure that the following entities are defined within your WebSphere Partner Gateway:

- Document flow definitions must exist to define the format of both the participant document and the back-end document.
- The Community Manager's B2B capabilities must include enablement of the back-end document's document flow definition as a destination (target).
- A document flow definition interaction must exist that brings together the participant document as the source and the back-end document as the destination.

Defining the document flow definition: Each document flow definition defines how WebSphere Partner Gateway processes a particular document. It includes the packaging type and business protocol of the document. WebSphere Partner Gateway provides some predefined packaging types and protocol definitions. If these predefined formats correctly define your participant and back-end documents, you do not need to define any document flow definition. However, if the predefined formats do not adequately define your participant or back-end document, you must create a valid document flow definition for that document. To define a document flow definition within WebSphere Partner Gateway, you use the Manage Document Flow Definitions page (**Hub Admin > Hub Configuration > Document Flow Definition > Create Document Flow Definition**).

Note: For more information on predefined document flow definitions as well as how to create document flow definitions in WebSphere Partner Gateway, see the *Hub Configuration Guide.*

For back-end integration, the packaging type of the back-end document must be one of the following:

- None packaging
- Backend Integration packaging

You must determine which of these packaging types applies, based on the business protocol of your document and the particular back-end system you are using. For information on packaging types with back-end systems, see "Which packaging will you use?" on page 19. For information on supported back-end systems, see "How do you access your back-end application?" on page 36.

Setting the B2B capabilities for sending: Before the Document Manager can convert the source document, it must determine whether it can handle the format of the desired destination document. To make this determination, the Document Manager checks the B2B capabilities of the Community Manager profile, which define which document flow definitions have been enabled for the Community Manager. Supported document flow definitions have each of their component document types (such as packaging type, business protocol, and document) enabled. To enable a particular document flow definition, you use the B2B Capabilities page of WebSphere Partner Gateway. To access this screen, perform the following steps:

1. Click **Account Admin > Profiles > Community Participant.**
2. Click **Search** to display a list of participants.
3. Select the **View details** icon next to the Community Manager.

4. Click **B2B Capabilities**.
5. For back-end integration, make sure each of the component document types for the back-end document have been enabled to serve as a destination (target). Under **Set Target**, enable each document-type component in the document flow definition of the back-end document.
6. If the hub will also be receiving documents from the back-end system, you might want to enable B2B capabilities required while you still have the B2B Capabilities page displayed. In this case, you enable, under **Set Source**, the document-type component of the back-end document.

Note: For more information about how to set B2B capabilities in WebSphere Partner Gateway, see the *Hub Configuration Guide.*

Defining the document flow interaction for sending: For the Document Manager to know how to convert the participant document, it must be able to locate an interaction that combines the document flow definitions for the participant document and the back-end document and identifies which is the source and which is the destination participant.

When the Document Manager is ready to send the converted document to the back-end system, it must be able to locate a participant connection between the source participant and the destination participant (back-end system). However, for a participant connection to exist, a valid interaction between the source and the destination documents must exist. To define a document flow definition interaction within WebSphere Partner Gateway, click: **Hub Admin > Hub Configuration > Document Flow Definition > Manage Interactions > Create Interaction**.

Note: For more information about how to create document flow definition interactions in WebSphere Partner Gateway, see the *Hub Configuration Guide.*

To send documents to the back-end system, define an interaction between the source and destination (target) documents, as summarized in Table 18.

Table 18. Creating an interaction for sending a document

Manage Interactions section	**Action**
Source	Select the component document-types in the *participant* document's document flow definition.
Target	Select the component document-types in the *back-end* document's document flow definition.

Defining how to connect to the back-end system

For the Document Manager to be able to send the converted document to the back-end system, it must find a valid participant connection, which identifies the source and destination participants and provides the location through which these two participants communicate. To create a participant connection, you use the Manage Connections page in WebSphere Partner Gateway. To access this screen, click: **Account Admin > Participant Connections**.

For a participant connection to be defined, a document flow definition interaction between the source and destination documents must already exist. On the Manage Connections screen, you first check for an existing interaction by specifying the source and destination (target) participants. Table 19 lists the participants to choose on the Manage Connections page to define a participant connection for sending a document to the back-end system.

Table 19. Creating a participant connection for sending a document

Manage Connection dropdown list	Name of community participant
Source	Name of the community participant that is sending the document to the Community Manager
Target	Name of the Community Manager, who receives the document from the community participant

Once you specify the Source and Target, you then click **Search** to check for an existing document flow definition interaction. If no interaction exists, you must create one before you can proceed with the creation of a participant connection. If an interaction does exist (one whose source is the participant document flow definition and whose target is the back-end document flow definition), you can configure the participant connection for communication with the back-end system.

Note: For more information about how to create a participant connection in WebSphere Partner Gateway, see the *Hub Configuration Guide.*

For back-end integration, this participant connection should specify as its target gateway the gateway you defined in “Defining where to send the participant document” on page 39.

Receiving documents from the back-end system

To receive a document from the back-end system, the hub takes the following steps:

1. Receives a document from the back-end system.

   The WebSphere Partner Gateway Receiver retrieves this source document from a target that has been defined in the hub for incoming messages from the back-end system and the associated transfer protocol. When receiving a document from the back-end system, the source document is the document that is received from the back-end system; therefore, this document is referred to as the *back-end document*.
2. Converts the back-end document to the destination document, which is in the format that the designated community participant requires.

   The Document Manager performs this conversion to the destination document. When receiving a document from the back-end system, the destination document is the document that is sent to some community participant; therefore, this document is referred to as the *participant document*.
3. Sends the participant document to the appropriate community participant.

   The Document Manager sends the participant document through a gateway that has been defined in the hub for outgoing messages to the appropriate community participant.

Therefore, for the hub to be able to receive a document from the back-end system, you must ensure that the configuration summarized in Table 20 has been performed within WebSphere Partner Gateway.

Table 20. Configuration steps to receive documents from the back-end system

Configuration step	WebSphere Partner Gateway steps	For more information
1. Define where to retrieve the document.	1. Create a target that receives incoming messages from the back-end system.	"Defining where to retrieve the back-end document"
2. Define how to process the document.	2. Create document flow definitions for the source and destination formats. 3. Enable B2B capabilities for the document flow definition of the document received from the back-end system. 4. Create a document flow definition interaction between the source and destination document flow definitions.	"Defining how to process the back-end document"
3. Define how to connect to WebSphere Partner Gateway.	5. Create a participant connection that sends documents to WebSphere Partner Gateway.	"Defining how to connect to WebSphere Partner Gateway" on page 44

Defining where to retrieve the back-end document

To receive documents from the back-end system, the hub must have a target defined. This target specifies the source of the documents; that is, it identifies the location (as a URI) at which the hub listens for incoming documents. This location is the same one to which the back-end system sends documents. The target identifies the entrance point into the Receiver (within WebSphere Partner Gateway). Within WebSphere Partner Gateway, it is the Receiver that checks for a target. Once the Receiver has processed the document, it saves the converted document to the persistent shared storage for later retrieval by the Document Manager.

To define a target within WebSphere Partner Gateway, click **Hub Admin > Hub Configuration > Targets.**

Note: For more information on how to create a target in WebSphere Partner Gateway, see the *Hub Configuration Guide.*

When you define the target, you specify the transport protocol that the hub and back-end system both use to transfer the back-end document. As Table 13 on page 29 shows, the choice of transport protocol depends on the format of the document. Its format includes its packaging type and business protocol, which are defined in its document flow definition.

Note: The choice of transport protocol also depends on the transport protocols your particular back-end system supports. For more information, refer to the chapter in this guide for integrating to your particular back-end system.

Once you have selected a valid transport protocol for your document, you can provide the other information you need to define for the target in the Target Details screen.

Defining how to process the back-end document

For the Document Manager to be able to process the back-end document, it must know the format to which it needs to convert this document; that is, it needs to know the format of the participant document. As part of the back-end integration, you must ensure that the entities summarized in Table 21 are defined within your

WebSphere Partner Gateway.

Table 21. Defining how to convert the back-end document

Step	For more information
1. Document flow definitions must exist to define the format of *both* the participant document and the back-end document.	"Defining the document flow definition" on page 40
2. The Community Manager's B2B capabilities must include enablement of the back-end document's document flow definition as a source.	"Setting B2B capabilities for receiving"
3. A document flow definition interaction must exist that brings together the back-end document as the source and the participant document as the destination.	"Defining an interaction for receiving"

Setting B2B capabilities for receiving: For a summary of B2B capabilities as they apply to back-end integration, see "Setting the B2B capabilities for sending" on page 40.

If you have not already done so, enable, under **Set Source**, the document-type component of the back-end document.

Defining an interaction for receiving: For a summary of document flow definitions interactions as they apply to back-end integration, see "Defining the document flow interaction for sending" on page 41. This section summarizes how to define the interaction for receiving a document from the back-end system.

To receive documents from the back-end system, define an interaction between the document flow definitions of the source and destination (target) documents as summarized in Table 22.

Table 22. Creating an interaction for receiving a document

Manage Interactions section	Action
Source	Select the component document-types in the *back-end* document's document flow definition.
Target	Select the component document-types in the *participant* document's document flow definition.

Defining how to connect to WebSphere Partner Gateway

For the Receiver to be able to retrieve the document from the back-end system, it must find a valid participant connection, which identifies the source and destination participants and provides the location through which these two participants communicate. For a summary of participant connections as they apply to back-end integration, see "Defining how to connect to the back-end system" on page 41.

Table 23 lists the participants to choose on the Manage Connections page in WebSphere Partner Gateway to define a participant connection for receiving a document from the back-end system.

Table 23. Creating a participant connection for receiving a document

Manage Connection dropdown list	Name of community participant
Source	Name of the Community Manager
Target	Name of the community participant that is receiving the document from the Community Manager

Once you specify the Source and Target, you then click **Search** to check for an existing document flow definition interaction. If no interaction exists, you must create one *before* you can proceed with the creation of a participant connection. If an interaction does exist (one whose source is the back-end document flow definition and whose target is the participant document flow definition), you can configure the participant connection for communication with the back-end system.

Note: For more information about how to create a participant connection in WebSphere Partner Gateway, see the *Hub Configuration Guide.*

Part 2. Integrating with WebSphere Process Server

Chapter 3. Introduction to WebSphere Process Server integration

This chapter provides an overview of integration between WebSphere Partner Gateway and WebSphere Process Server.

Note: For a description of the general process used to integrate WebSphere Partner Gateway with a back-end system, see Chapter 2, "Planning for back-end integration," on page 9.

This chapter includes the following sections:

Overview

This section describes how WebSphere Partner Gateway can be used to provide the B2B capabilities for WebSphere Process Server.

Note: For detailed information about WebSphere Process Server, see the WebSphere Process Server information center.

WebSphere Process Server uses the B2B capabilities of WebSphere Partner Gateway to manage interactions with trading partners (known, in WebSphere Partner Gateway, as *community participants*). For example, suppose a service running on WebSphere Process Server needs to send a document to a community participant. WebSphere Process Server sends the document to WebSphere Partner Gateway, which determines the transformation map (if any) that should be used to transform the document into the form that the community participant is expecting. WebSphere Partner Gateway also handles all the community participant (partner) profile information.

Now suppose WebSphere Process Server needs to send a document to multiple community participants. The community participants receive the document in different formats. WebSphere Process Server needs to deal with only one type of format (the one it sends to WebSphere Partner Gateway). WebSphere Partner Gateway handles the interaction with the community participants.

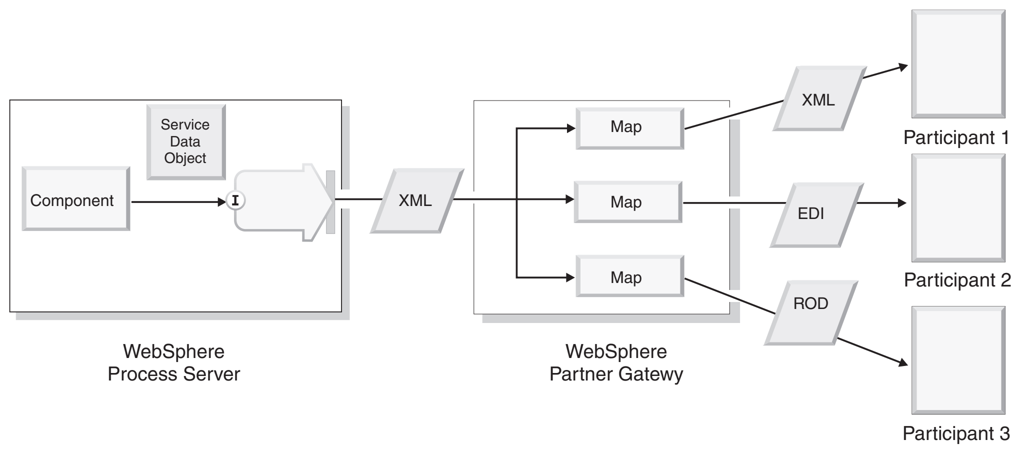

Figure 15. WebSphere Partner Gateway sends documents in multiple formats to community participants

You can develop the transformation maps that convert the document to the format required by the community participant, or you can import maps from the Data Interchange Services client program.

Similarly, when WebSphere Process Server receives documents sent from community participants, those documents are processed by WebSphere Partner Gateway. The documents can be in a variety of formats. WebSphere Partner Gateway transforms the documents and sends them to the gateway defined for the Community Manager on WebSphere Process Server.

How WebSphere Process Server and WebSphere Partner Gateway communicate

WebSphere Partner Gateway sends a document from a community participant to WebSphere Process Server so that the document can be processed by a service on WebSphere Process Server. The way a service on WebSphere Process Server makes itself available for use by other applications (including WebSphere Partner Gateway) is through its *export* binding.

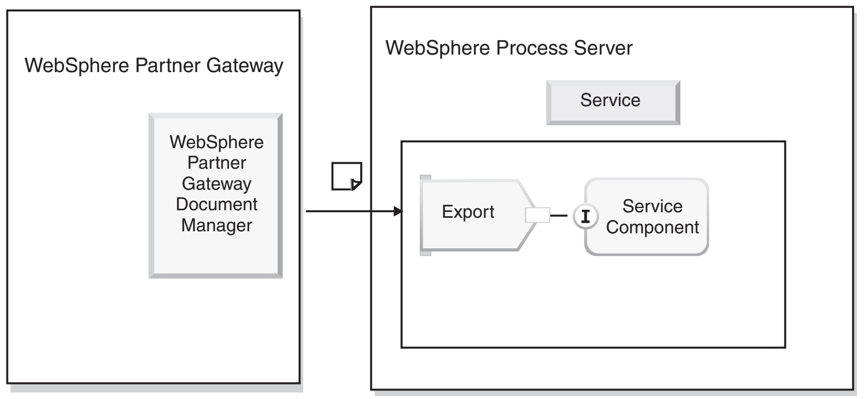
WebSphere Partner Gateway
WebSphere Partner Gateway Document Manager
WebSphere Process Server
Service
Export
Service Component

Figure 16. How WebSphere Partner Gateway uses the export binding to send documents to WebSphere Process Server

The WebSphere Process Server component has an interface that describes the service (the methods available and the input and output data) and a binding (in this case, an export binding).

Figure 16 on page 50 presents a generic view of how the export binding invokes a service on WebSphere Process Server. The transport type (for example, JMS or HTTP) you use to send the message affects where the message is sent and how it is retrieved, as described in "Message transports that WebSphere Process Server supports" on page 52.

Similarly, when a service on WebSphere Process Server wants to send a business document to a community participant, it uses its *import* binding. Imports identify services outside of a module, so they can be called from within the module. In this case, WebSphere Process Server uses the import binding to call WebSphere Partner Gateway, which processes the document and sends it to the community participant.

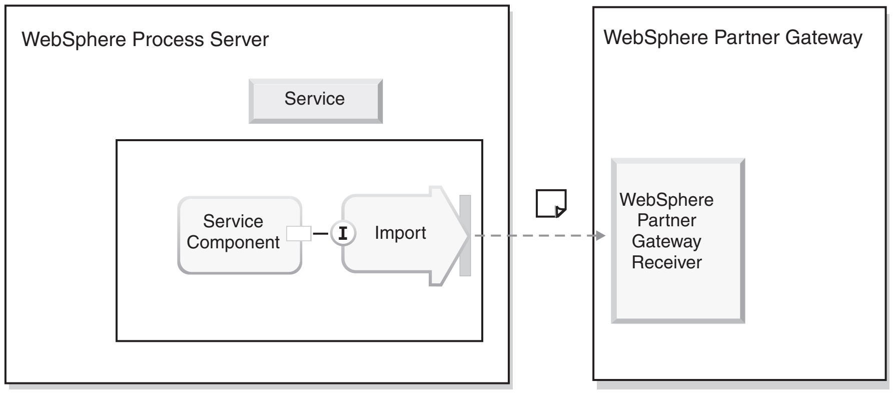

Figure 17. How WebSphere Process Server uses the import binding to invoke WebSphere Partner Gateway

Planning for integration with WebSphere Process Server

To plan for your integration with WebSphere Process Server, follow the steps outlined in Chapter 2, "Planning for back-end integration," on page 9.

WebSphere Process Server versions that WebSphere Partner Gateway supports

Version 6.0 of WebSphere Partner Gateway supports integration with WebSphere Process Server version 6.0

WebSphere Process Server is available on several platforms, including Windows 2000 and several UNIX-based platforms. For more information, consult your installation guide for WebSphere Process Server in the WebSphere Process Server information center.

Supported installation scenarios

In the following table, each row shows a supported combination of WebSphere Partner Gateway installations and WebSphere Process Server installations.

Note: During installation, WebSphere Partner Gateway creates separate WebSphere Application Server profiles for each of its components. These profiles are for WebSphere Partner Gateway use only. Do not deploy WebSphere Process Server or any other WebSphere Application Server application into these profiles.

Table 24. Supported installation scenarios

WebSphere Partner Gateway installation	WebSphere Process Server installation	Details
Installed with embedded WebSphere Application Server.	Either of the following: • WebSphere Process Server is installed on an existing installation of WebSphere Application Server 6.0 or WebSphere Application Server ND 6.0. • The WebSphere Process Server installer installs WebSphere Application Server ND V6.0 along with WebSphere Process Server.	Supported. In these scenarios, WebSphere Partner Gateway components will execute on the embedded WebSphere Application Server instance, and WebSphere Process Server will execute on the application server on which it is installed. Note: WebSphere Partner Gateway cannot be part of the WebSphere Application Server ND cell.
Installed on one or more existing installations of WebSphere Application Server 6.0.	WebSphere Process Server installer installs WebSphere Application Server ND V6.0 along with WebSphere Process Server.	Supported. Note: WebSphere Partner Gateway cannot be part of the WebSphere Application Server ND cell.
Installed on one or more existing instances of WebSphere Application Server 6.0.	Installed on the same installation, but not the same instance, of WebSphere Application Server 6.0 on which WebSphere Partner Gateway is installed. The profile used for WebSphere Process Server must be different than the one used by WebSphere Partner Gateway components.	This is supported only on the platforms (operating systems and versions) that are supported by both WebSphere Partner Gateway and WebSphere Process Server.

Message transports that WebSphere Process Server supports

When WebSphere Partner Gateway sends a message to WebSphere Process Server over a particular message transport protocol, the transport-specific gateway defined for the Community Manager sends the message to the WebSphere Process Server end point. WebSphere Process Server retrieves the message from the end point and processes the message. The type of message transport determines how the message is handled after it arrives at WebSphere Process Server:

- For the JMS transport, you set up a JMS export to retrieve the message from the JMS queue.
- For the HTTP transport, you create a servlet on WebSphere Process Server that handles the receipt of the message from WebSphere Partner Gateway.
- For SOAP documents (which are sent over the HTTP transport protocol), you set up a Web Service export binding in WebSphere Process Server to retrieve the SOAP request.
- For the file-system, you set up an inbound Flat File adapter to route the message to WebSphere Process Server.

When you send messages from WebSphere Process Server to WebSphere Partner Gateway, you send them to the transport-specific target (for example, a JMS queue or a URL) on WebSphere Partner Gateway. The type of message transport determines how the message is sent.

- For the JMS transport, you set up a JMS import to send the message to the JMS queue.
- For the HTTP transport, you create a component on WebSphere Process Server that does an HTTP POST to the URL specified for the WebSphere Partner Gateway target.
- For SOAP documents (which are sent over the HTTP transport protocol), you set up a Web Service import binding in WebSphere Process Server to send the SOAP request to a WebSphere Partner Gateway URL.
- For the file-system, you set up an outbound Flat File adapter to route the message to a directory on WebSphere Partner Gateway.

Information about sending and receiving messages using the supported transports can be found in the following sections:

- Chapter 4, "Integrating WebSphere Process Server with HTTP as transport," on page 75
- Chapter 5, "Integrating WebSphere Process Server with JMS as transport," on page 83
- Chapter 6, "Integrating WebSphere Process Server with SOAP/HTTP," on page 101
- Chapter 7, "Integrating WebSphere Process Server with File-system as transport," on page 107

Support for WebSphere Process Server integration

This section describes the samples, documentation, and utility classes that WebSphere Partner Gateway provides to assist you with WebSphere Process Server integration.

Samples

WebSphere Partner Gateway provides samples of using the JMS transport protocol to assist you in the integration process with WebSphere Process Server. These samples reside in the following subdirectory of the WebSphere Partner Gateway product directory:

```
Integration/WBI/WPG/samples
```

Table 25 lists the subdirectories of the `samples` directory.

Table 25. Samples for WebSphere Process Server integration

Type of sample	Samples subdirectory
General samples	JMS
RosettaNet-specific samples	RosettaNet/JMS

Documentation

In addition to the information in this document, WebSphere Partner Gateway provides the *PIP Sample for WebSphere Process Server*, which gives you step-by-step instructions on how to set up a PIP flow between a community participant and WebSphere Process Server.

Utility classes

WebSphere Partner Gateway provides two utility classes that you can use to transform a Backend Integration packaging message into a business object or to transform a business object into a Backend Integration packaging message:

- BCGBackEndIntegrationDataBindingUtil class

  This class implements the DataBinding interface and provides utility methods to read and write strings, streams, and byte arrays. You can use this class as is or develop a new data binding using this class.
- BCGBackEndIntegrationJMSDataBindingImpl class

  This class implements the JMSDataBinding interface. This class can be specified in SCA JMS export and import bindings. It creates a data object from a JMS message containing a payload or writes the data object to a JMS message.

Overview of tasks for integrating WebSphere Partner Gateway with WebSphere Process Server

This section lists the tasks that you perform so that WebSphere Partner Gateway can send documents to or receive documents from WebSphere Process Server. It provides a foundation for the transport-specific chapters that describe how to integrate with WebSphere Process Server.

On the WebSphere Partner Gateway system

This section provides a very brief overview of the tasks you perform to configure the hub so that you can send documents to and receive documents from WebSphere Partner Gateway. These tasks, which are described in detail in the *Hub Configuration Guide*, are performed at the Community Console of WebSphere Partner Gateway.

- Create a transport-specific target on the hub to receive documents sent to the hub from WebSphere Process Server or from community participants.
- Create a Community Manager profile (if one does not already exist), including a transport-specific gateway that WebSphere Partner Gateway will use to send documents to WebSphere Process Server.
- Create community participant profiles, including transport-specific gateways that WebSphere Partner Gateway will use to send documents to the participants.
- Import any WSDL files, transformation maps, RosettaNet packages, or other document definition mechanisms so that a document definition for the type of document you are exchanging appears on the Document Flow Definition page of the WebSphere Partner Gateway Community Console.
- Create interactions between the types of document the hub will receive (from WebSphere Process Server or from a community participant) and the types of documents the hub will send (to WebSphere Process Server or to the community participants).
- Create B2B capabilities in the profiles of the Community Manager and participants to indicate the types of documents they are able to send and receive.
- Create participant connections between the Community Manager and participants to indicate the source participant (the sender of the document), the target participant (the recipient of the document), and the action that the hub should take (if any) to transform the document.

On the WebSphere Process Server system

The module is the WebSphere Process Server artifact used to assemble and deploy a service. The first step, then, is to create a module, using the WebSphere Integration Developer.

After the module is created, you create the components and their interfaces and then specify the binding used.

1. Specify an interface for the component. You can import an interface (for example, you can import an existing WSDL file), or you can create the interface.

   When you create an interface, you define one or more operations performed by the component, and you define the inputs and outputs expected by the component.

   An interface for a component can be a WSDL or you can use a Java implementation for the interface. Refer to the WebSphere Process Server documentation for information on when to create a WSDL interface and when to create a Java interface.
2. Specify an implementation for the component. You can import an implementation (for example, you can import an existing Java program), or you can create the implementation.
3. Compose the application using the Assembly Editor of WebSphere Integration Developer. You create a service component, and then specify the interface for the component (which you created or imported in step 1). You also specify the implementation (which you created or imported in step 2).
4. Create an export binding to allow WebSphere Partner Gateway to send a document to the service, or create an import binding to allow WebSphere Process Server to send a document to WebSphere Partner Gateway.

   When you create the binding, you specify information needed to send and receive documents. For example, in the JMS Binding definition, you indicate the JMS queue and bus as well as the data binding that should be used to transform a business document to or from a business object. (See "Handling Backend Integration Packaging messages" for information on the use of and requirements for data binding.) The transport-specific requirements for the service bindings are described in subsequent chapters.
5. When you have finished assembling the components that make up the module, you deploy the module.

Handling Backend Integration Packaging messages

For certain protocols such as RosettaNet, WebSphere Partner Gateway expects backend applications to use Backend Integration packaging. WebSphere Partner Gateway supports Backend Integration packaging over HTTP and JMS protocols. This requires WebSphere Process Server services be able to handle Backend Integration packaging messages. WebSphere Process Server services use business objects. Therefore, to send Backend Integration packaging messages to WebSphere Partner Gateway, WebSphere Process Server services have to marshal business objects into Backend Integration packaging messages. Similarly to receive Backend Integration packaging message from WebSphere Partner Gateway, WebSphere Process Server services have to unmarshal business objects into Backend Integration packaging messages.

This section describes the data binding utility API and JMS data binding that WebSphere Partner Gateway provides, which can be used by WebSphere Process Server services for handling Backend Integration packaging messages.

Data binding refers to the mechanism used to:

- Transform a business object, sent from WebSphere Process Server, into a business document that WebSphere Partner Gateway can process
- Transform a business document, sent from WebSphere Partner Gateway, into a business object that WebSphere Process Server can process

The WebSphere Partner Gateway-provided Backend Integration packaging data binding utility API and JMS data binding provide a way to create a Backend Integration packaging message from a business object or to convert a Backend Integration packaging message into a business object. The WebSphere Partner Gateway-provided data binding can be used as is when you are sending and receiving documents with Backend Integration packaging. Or, if you want to customize the processing, you can create your own data binding using the data binding utility API provided by WebSphere Partner Gateway.

JMS transport

If you are using JMS as your transport protocol and you are sending or receiving an XML document that does not have BackEnd Integration packaging (in other words, it has a packaging of None specified), you can use the default JMS binding provided by WebSphere Process Server or you can use the WebSphere Process Server-provided business object APIs to create your own data binding. If you are sending or receiving a document other than XML that does not have BackEnd Integration packaging (in other words, it has a packaging of None specified), you have to create your own data binding. You select the JMS binding when you configure the SCA (Service Component Architecture) JMS Export or Import binding. Refer to the WebSphere Process Server information center for more information on using the default JMS binding.

If you are using JMS as your transport protocol and you are sending or receiving documents with BackEnd Integration packaging, you can use the BCGBackEndIntegrationJMSDataBindingImpl data binding as is, or, if you want to customize the processing, you can create your own data binding using BCGBackEndIntegrationJMSDataBindingImpl. To use BCGBackEndIntegrationJMSDataBindingImpl, you configure it as the data binding in the SCA JMS Export Binding and JMS Import Binding.

- For request processing, the export invokes the read method of the JMSDataBinding interface to convert the Backend Integration packaging JMS message from WebSphere Partner Gateway into a business object. For request-response processing, the export might also invoke the write method.
- For request processing, the import invokes the write method of the JMSDataBinding interface to construct a Backend Integration packaging JMS message from a business object before the message is sent to WebSphere Partner Gateway. For request-response processing, the import might also invoke the read method to read the response provided by the service.

Refer to the WebSphere Process Server information center for more information about how and when methods of data binding are called.

HTTP transport

If you are using HTTP as your transport protocol, you can write an HTTP data binding class that extends and overrides the generic BCGBackEndIntegrationDataBindingUtil class.

- An HTTP servlet can invoke methods of BCGBackendIntegrationDataBindingUtil to construct a business object from the HTTP stream that is sent from WebSphere Partner Gateway. The servlet can then use this business object to invoke a service.
- A component can be written to perform an HTTP POST to WebSphere Partner Gateway. A service can then invoke this component with the business object. If the component requires a Backend Integration packaging message, it can use the BCGBackendIntegrationDataBindingUtil class to construct the message from the business object.

Note: WebSphere Partner Gateway supports request-only invocations for Backend Integration packaging. You can, however, develop request-response flows with Backend Integration packaging data binding (via WebSphere Partner Gateway user exits). This utility class can be used as the basis for converting a response business object into a response stream or for converting a response stream into a response business object.

Top-level and child business objects

This section describes the top-level business object and the child business objects used by Backend Integration data binding.

Top-level business object

To use Backend Integration packaging data binding, you create a top-level business object that has three child attributes:

Table 26. Top-level business object

Attribute	Type
payload	This attribute is of type payload container business object. It can be of any type but must have the properties specified in Table 27 on page 58 or Table 28 on page 59
attachment	This attribute is of type attachment container business object. It can be of any type but must have the properties specified in Table 29 on page 59
packagingHeaders	This attribute is of type packaging header business object. It can be of any type but must have the properties specified in Table 30 on page 60

When a Backend Integration packaging message is converted to a top-level object, the top-level object is populated with the data from the message. When a top-level object is converted to a Backend Integration packaging message, the top-level object is the input to the data binding.

The following illustration shows the top-level business object and its child objects:

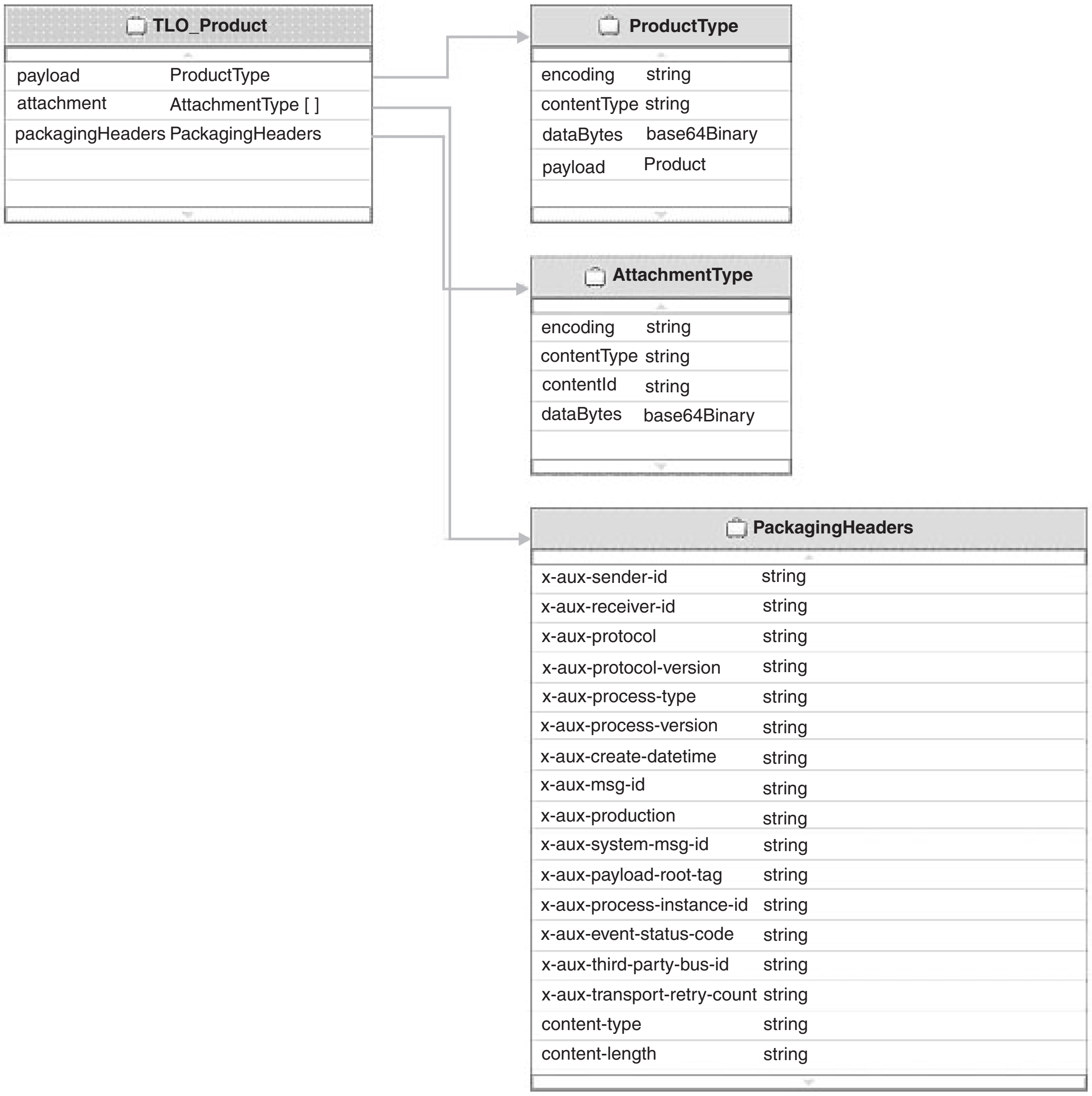

Figure 18. Top-level business object

Payload Container business object

The structure of the payload container business object depends on whether the payload is XML or non-XML.

XML payloads: If the payload is XML, the payload container has the following properties:

Table 27. Payload container object for XML payloads

Attribute	Type
contentType	xsd:string
encoding	xsd:string
dataBytes	xsd:base64Binary

Table 27. Payload container object for XML payloads (continued)

Attribute	Type
payload	A reference to the payload business object. For example, if the payload itself is contained in a Pip3A4PurchaseOrderRequest object, the payload attribute is of type PIP3A4PurchaseOrderRequest.

"Creating business objects for RosettaNet service content" on page 60 describes how to create a business object for a PIP.

Non-XML payloads: If the payload is not XML, the payload container has the following properties:

Table 28. Payload container object for non-XML payloads

Attribute	Type
contentType	xsd:string
encoding	xsd:string
dataBytes	xsd:base64Binary
dataString	xsd:string

The payload is the value of dataBytes or dataString.

- If your payload is not XML, while converting the Backend Integration packaging message into a top-level object, the Backend Integration packaging data binding will not convert the value of the <payload> element from the message into the payload business object. Instead, it will set the value of the dataBytes (or dataString) attribute of the payload container business object as the bytes (or string) value of the payload.
- Similarly, while constructing the Backend Integration packaging message from a top-level object, the Backend Integration packaging data binding will set the contents of the dataBytes (or dataString) attribute of the payload container business object as the value of the <payload> element of the Backend Integration packaging message.

Attachment Container business object

This business object represents an attachment. It can be of any complex type; however it must have the following attributes:

Table 29. Attachment container business object attributes

Attribute	Type
contentID	xsd:string
contentType	xsd:string
encoding	xsd:string
dataBytes	xsd:base64Binary
dataString	xsd:string

The attachment is the value of dataBytes or dataString.

Packaging header business object

This business object holds the transport-level headers. The x-aux transport headers are described in "Transport-level header content" on page 20. This business object must have the following attributes.

Table 30. Packaging header business object attributes

Attribute	**Type**
x-aux-sender-id	string
x-aux-receiver-id	string
x-aux-protocol	string
x-aux-protocol-version	string
x-aux-process-type	string
x-aux-process-version	string
x-aux-create-datetime	string
x-aux-msg-id	string
x-aux-production	string
x-aux-system-msg-id	string
x-aux-payload-root-tag	string
x-aux-process-instance-id	string
x-aux-event-status-code	string
x-aux-third-party-bus-id	string
x-aux-transport-retry-count	string
content-type	string
content-length	string

Creating business objects for RosettaNet service content

Backend Integration packaging data binding requires a top-level object. You can create the top level object as described in "Top-level business object" on page 57. WebSphere Partner Gateway expects RosettaNet service content XML as the payload of the Backend Integration packaging message. To create the payload container object, you need a payload business object that represents the RosettaNet service content XML. You can create the RosettaNet service content business object using one of the following approaches:

- If the structure of the PIP service content provided by RosettaNet is in XML schema format, this XML schema can be used as your payload business object.
- If the structure of the PIP service content provided by RosettaNet is in DTD format, you need to convert this DTD into XML schema. To convert PIP service content DTD into XML schema, follow the procedure in *Hub Configuration Guide*. This XML schema can be used as your payload business object.
- Alternatively, if the PIP is in DTD format and a WebSphere Partner Gateway-provided PIP package exists for the PIP, follow this procedure to create the PIP service content business object.
  1. Navigate to the RosettaNet service content PIP package for the RNIF version you are interested in. For example, if you want to create a PIP3A4PurchaseOrderRequest business object, you can use the BCG_Package_RNSC1.0_RNIFV02.02.zip file in the WebSphere Partner Gateway product directory.
  2. Using WebSphere Integration Developer, import the ZIP file into the WebSphere Process Server modules you have set up for the PIP.
  3. Expand the Data Types folder and right-click the business object that corresponds to the root element of your service content. For example, in the case of a PIP 3A4 request, right-click the Pip3A4PurchaseOrder from the Data Types folder; then select **Open with the Text Editor**.

4. In the text editor, change the includes as follows.

   ```
   ../../common/
   ```

   to

   ```
   ./
   ```
5. Save your changes, and close the text editor.

How the Backend Integration packaging data binding works

This section provides a description of how Backend Integration data binding creates a Backend Integration packaging message from a business object or creates a business object from an incoming message.

Backend Integration packaging message to top-level object conversion

This section describes how a Backend Integration packaging message is converted to a business object that can be used by services on WebSphere Process Server.

As mentioned in “Top-level and child business objects” on page 57, Backend Integration packaging data binding works with a specific type of top-level business object. You must create this business object before you can use the data binding.

The top-level business object is described in “Top-level business object” on page 57.

Types of messages: The way the Backend Integration packaging message from WebSphere Partner Gateway is converted depends upon the type of message. The message can be one of three types:

- An XML message with a root tag of <transport envelope> and a URI of “http://www.ibm.com/websphere/bcg/2003/v1.1/wbipackaging” or ″http://www.ibm.com/websphere/bcg/2003/v1.0/wbipackaging” .

  An example of a message inside a transport envelope is:

  ```
  <transport-envelope
       xmlns="http://www.ibm.com/websphere/bcg/2003/v1.1/wbipackaging">
       <payload encoding="base64"  contentType="application/xml"
  contentId="111111111111">
             ...base64 encoded XML message...
        </payload>
        <attachment encoding="base64" contentType="text/xml"
  contentId="2222222222222">
             ...base64 encoded XML attachment...
         </attachment>
         <attachment encoding="base64" contentType="application/pdf"
  contentId="3333333333">
             ...base64 encoded PDF attachment...
         </attachment>
  </transport-envelope>
  ```
- Any other XML message
- A non-XML message

Methods used to convert the message: To convert a Backend Integration packaging JMS message into a top-level object, you use the read method of BCGBackEndIntegrationJMSDataBindingImpl.

To convert a Backend Integration packaging message into a top-level object, you can use the following methods of BCGBackEndIntegrationDataBindingUtil:

- read():

If your Backend Integration packaging message is contained in an input stream, you can use this method.

- setFromByteArray:

  If your Backend Integration packaging message is contained in a byte array, you can use this method.
- setFromString

  If your backend integration packaging is contained in a string, you can use this method.

Before calling any of these methods, you can customize the processing of data binding by calling the following methods:

- setTLOTypeName()

  You can use setTLOTypeName on the object to name it and to specify its URI.
- setBOPrefix()

  You can use setBOPrefix to specify the business object prefix. If you have not done this, the default prefix of TLO_ is used.
  - If the payload business object type can be determined, the top-level object is: <BOPrefix><PayloadBOTypeName>.
  - If the payload business object is not determined (for non-XML payloads), the name is BCG_TLO_BackendPackaging (the default).

If you specify both setTLOTypeName and setBOPrefix, setBOPrefix is ignored.

For non-XML payloads, if you do not specify the top-level object name by calling setTLOTypeName before calling the read method, the default top-level object is used. The payload is not converted into a payload business object. Instead, the data bytes are set as is in the default top-level object. The default top-level object is shown in Figure 19 on page 63.

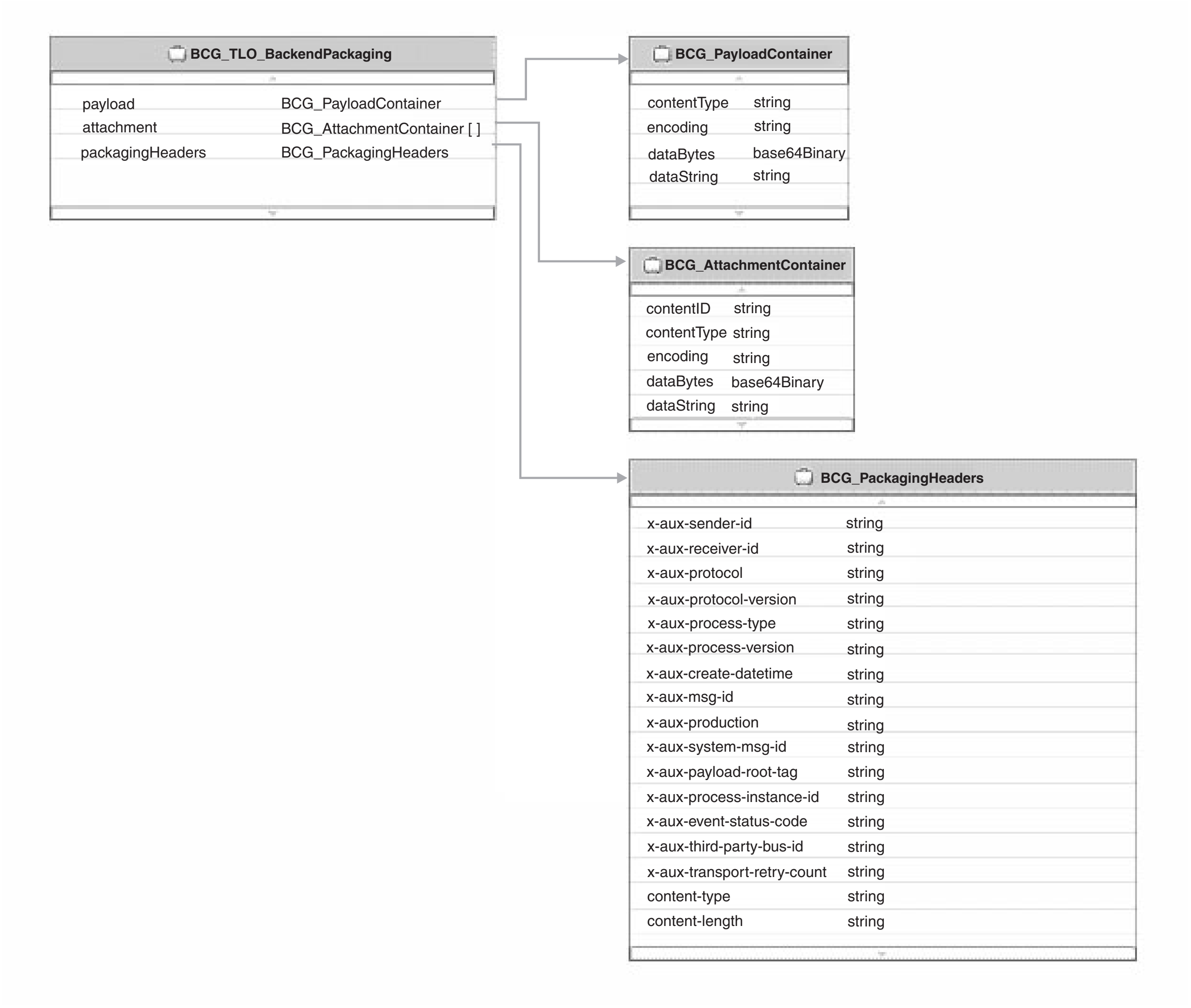

Figure 19. Default top-level business object

If the message has transport headers, you need to read those transport headers and set them using setxAuxHeaders (). The setxAuxHeaders method must be called before calling read(). The setxAuxHeaders method does not apply to the BCGBackEndIntegrationJMSDataBindingImpl data binding, because the JMS transport headers will be read from the given JMS message in the read method.

How messages are converted: The following section describes how the read, setFromByteArray, and setFromString methods of BCGBackEndIntegrationDataBindingUtil and the read method of BCGBackEndIntegrationJMSDataBindingImpl work. The method:

1. Determines the XML content type from the Content-type header (if it is available in the message). For example:

   ```
   Content-Type: application/xml
   ```

   If the contentType is not available, the method inspects the first few bytes of the message to determine its type.

   The method obtains the payload based on the content type, as follows:

- For an XML message, it deserializes the message to obtain the payload business object.
- For an XML message that has a root tag of <transport envelope>, it Base64 decodes the payload to obtain the payload bytes, and parses the message to determine whether it is XML or non-XML.
  - If the message is XML, it deserializes the payload bytes to obtain the payload business object.
  - If the message is not XML, it does not deserialize the payload bytes. The payload bytes are used as is.

  For each attachment contained within the transport envelope, it Base64 decodes the attachment to obtain the attachment bytes. The attachment bytes are used as is; they are not deserialized.
- For an XML message that has a root tag of <EventNotification>, it deserializes the message to obtain the event-notification business object.
- For a message that is not XML, it does not deserialize the message. The bytes are used as is.

2. Determines the name of the top-level business object.
   - If the payload is XML:
     - If setTLOTypeName was invoked before calling read, setFromByteArray, or setFromString, the name specified in the method is directly used to create the top-level name.
     - If you use setBOPrefix, the top-level object name is made up of the prefix you specify plus the XML root element of the payload. The URI is also obtained from the payload.
     - If you do not use setTLOTypeName or setBOPrefix, the top-level object name is made up of the default prefix TLO_ plus the XML root element of the payload.
   - If the payload is not XML:
     - The default top-level object can be used.
     - The setTLOTypeName method can be used, but it must conform to the top-level object you created or to the default top-level object (BCG_TLO_BackendPackaging).
3. Instantiates the top-level object.
4. Instantiates the payload container business object and set its values. The way the values are set depends upon the type of message.
   - For an XML message with a root tag of <transport envelope>, the attributes are set as follows:
     - contentType

       The value of the contentType attribute of the <payload> tag.
     - encoding

       The value of the encoding attribute of the <payload> tag.
     - payload

       The name of the payload business object.
   - For an XML message or an XML message with a root tag of <event notification>, the attributes are set as follows:
     - contentType is not set
     - encoding is not set
     - payload

       The name of the payload business object.

- For a non-XML message, the attributes are set as follows:
  - contentType is not set
  - encoding is not set
  - dataBytes

    If the incoming message is received as bytes, the entire set of bytes is set as the value of this attribute.
  - dataString

    If the incoming message is received as a string, the entire string is set as the value of this attribute.
  - payload is not set.
5. Instantiates the attachment business objects and set their values as follows:
   - contentType

     The value of the contentType attribute of the <attachment> tag.
   - encoding

     The value of the encoding attribute of the <attachment> tag.
   - contentId

     The value of the contentId attribute of the <attachment> tag.
   - dataBytes

     The Base64-decoded attachment bytes.
6. Instantiates the transport headers business object and sets the transport headers read from the message. The name of this business object is set as the value of the packagingHeaders attribute in the top-level object.

Example code shows how the methods are used.

Top-level object to Backend Integration packaging message conversion

This section describes how a business object from WebSphere Process Server is serialized into a Backend Integration packaging message. The data binding puts the content and any attachments into a transport envelope in Base64 format. It also puts the transport headers from the top-level object into the envelope as a string.

Before invoking backend integration packaging data binding, a service has to create a top-level object, as described in “Top-level business object” on page 57.

Methods used to convert the object: A WebSphere Process Server service invokes the Backend Integration packaging data binding, sending it the top-level object. To convert the object into a Backend Integration packaging JMS message, you use the write method of BCGBackEndIntegrationJMSDataBindingImpl. To convert the object into a Backend Integration packaging message, you use the write, getAsByteArray, or getAsString method of BCGBackEndIntegrationDataBindingUtil.

Before calling one of these methods, you can call:
- The setPackagingSchema method

  This method specifies which packaging schema to use in constructing the Backend Integration packaging message.
- The setOptions method

  This method provides information (such as the root tag of the payload) that can be used to serialize the message. For the options you can set, see “setOptions method” on page 70.

How objects are converted: The following section describes how the write, getAsByteArray, and getAsString methods of BCGBackEndIntegrationDataBindingUtil and the write method of BCGBackEndIntegrationJMSDataBindingImpl work. The method:

1. Determines the content type of the payload. This information is obtained from the payload container business object. The way the payload is processed is based on its content type, as follows:
   - XML payload

     The dataBytes or dataString attribute of the payload container (if present) must be null and the payload container business object should have at least one attribute that is of complex type. This is a reference to the payload business object. The business object is serialized and is used as the XML payload. If there are more attributes of complex type set, the first complex type attribute is considered to be the XML payload.
   - EventNotification

     The first non-null attribute should be of type EventNotification. This is a reference to the event notification business object. The business object is serialized and is used as the XML payload.
   - Non-XML

     If the dataBytes attribute is not null, the value of the attribute is used as the payload.

     If the dataString attribute is not null, the value of the attribute is used as the payload.
2. Serializes the payload business object if it is of type XML or EventNotification.
3. Base64 encodes the payload, based on its type:

   If the payload container business object has a non-null value for the dataByte attribute, that value (the payload bytes) is Base64 encoded.

   If the payload container business object has a non-null value for the dataString attribute, the bytes are extracted based on the contentType attribute of the business object. The bytes are then Base64 encoded.

   If the payload was serialized from a business object, the bytes are extracted based on the contentType attribute of the business object. The bytes are then Base64 encoded.
4. Constructs an XML document based on the packaging schema specified.
   a. The <transport envelope> root tag is added.
   b. The <payload> tag, which is a child element of the <transport envelope> tag, is set with the base64-encoded string from step 3.
5. Processes any attachments.

   If the attachment container object has a dataByte attribute that is not null, the bytes are base64-encoded, and the string is set as the value of the <attachment> tag.

   If the attachment container object has a dataString attribute that is not null, the bytes are extracted based on the contentType attribute of the business object. The bytes are then base-64-encoded, and the string is set as the value of the <attachment> tag.

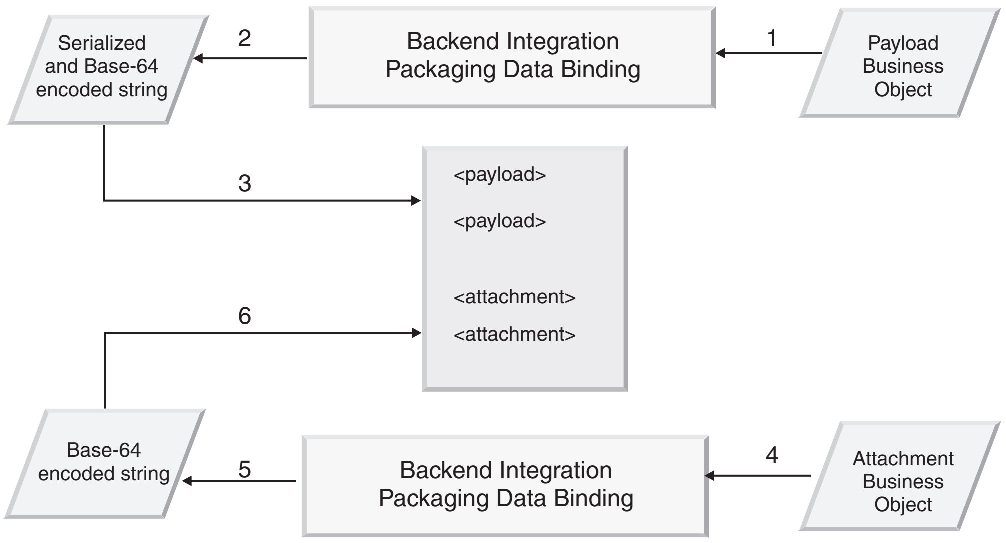
Serialized and Base-64 encoded string
2
Backend Integration Packaging Data Binding
1
Payload Business Object
3
<payload>
<payload>
<attachment>
<attachment>
6
Base-64 encoded string
5
Backend Integration Packaging Data Binding
4
Attachment Business Object

Figure 20. How data binding adds converted business objects to the Backend Integration packaging message

Example code shows how the methods are used.

BCGBackEndIntegrationDataBindingUtil class

This section describes the BCGBackEndIntegrationDataBindingUtil class and the methods of the class. You can use this class as it is, or, if you want to customize the processing of the data binding, you can create a new data binding class.

DataBinding interface

BCGBackEndIntegrationDataBindingUtil implements the following DataBinding interface:

```
DataBinding
public interface commonj.connector.runtime.DataBinding extends Serializable {
       public DataObject getDataObject()throws
commonj.connector.runtime.DataBindingException;
       public void setDataObject(DataObject dataObject) throws
commonj.connector.runtime.DataBindingException;
}
```

Methods

The BCGBackEndIntegrationDataBindingUtil methods are described in this section.

The BCGBackEndIntegrationDataBindingUtil class has two types of methods:

- Those that create a data object from a byte array, string, or data stream:
  - read(InputStream)
  - setFromByteArray(byte[ ])
  - setFromString(String)
  - setxAuxHeaders(HashMap)
  - setBOPrefix(String)
  - setTLOTypeName(String, String)
  - getDataObject()

Notes:

1. You can use one of the following methods to create the data object:
   - read(InputStream)
   - setFromByteArray(byte[ ])
   - setFromString(String)

   Before calling the method to create the data object, the program can invoke any of the following methods:
   - setxAuxHeaders
   - setBOPrefix
   - setTLOTypeName
2. If the setTLOTypeName method is used, the setBOPrefix method has no significance and is ignored.
3. Use the getDataObject method to obtain the top-level business object.

- Those that convert a data object into a byte, string, or data stream:
  - setDataObject(DataObject)
  - setPackagingSchema(String)
  - setOptions(HashMap)
  - getAsString()
  - getAsByteArray()
  - write(OutputStream)
  - getxAuxHeaders()

getAsByteArray method

This method returns the byte[ ] obtained from the top-level object.

Syntax:

```
public byte[ ] getAsByteArray()
```

getAsString method

This method returns a string form of the top-level object.

Syntax:

```
protected java.lang.String getAsString()
```

getDataObject method

This method returns the data object, if it has already been instantiated.

Syntax:

```
public commonj.sdo.DataObject getDataObject()
```

getxAuxHeader method

This method returns the x-aux headers from the transport envelope. See "Transport-level header content" on page 20 for information about the x-aux headers.

Syntax:

```
public java.util.HashMap getxAuxHeaders()
```

read method

This method takes an input stream, reads this stream, and converts it to a top-level business object. The actions taken by the read method are described in detail in "Backend Integration packaging message to top-level object conversion" on page 61.

Syntax:

```
public void read(java.io.InputStream inputStream)
          throws commonj.connector.runtime.DataBindingException
```

Parameter:

inputStream
: The input stream from which the data will be read.

setBOPrefix method

If the top-level object was not specified, you may specify a prefix by calling setBOPrefix (). If you do not specify a prefix, TLO_ is used as the default prefix.

Notes:

1. In the case of an XML payload, the namespace of the XML payload is used as the namespace of the top-level object.
2. In the case of a non-XML payload, the default namespace ″http://ibm.com/websphere/bcg/2005/wbi/bo″ is used as the namespace for the top-level object.

Syntax:

```
public void setBOPrefix(java.lang.String prefix)
```

setDataObject method

This method sets the data object.

Syntax:

```
public void setDataObject(commonj.sdo.DataObject topLevelbo)
                   throws commonj.connector.runtime.DataBindingException
```

Parameter:

topLevelbo
: The top-level business object. See "Top-level business object" on page 57 for information.

setDebugLevel method

By default, the debug level is set to error. Use this method to change the level.

Syntax:

```
public static void setDebugLevel(int debugLevel)
```

Parameter:

debugLevel
: Possible values for the debug level are:
- BCG_LOG_DEBUG
- BCG_LOG_ERROR
- BCG_LOG_WARNING
- BCG_LOG_INFO

setFromByteArray method

This method takes the payload data bytes or transport-envelope data bytes and creates a top-level object. Before calling this method, you may call one or more of the following methods:

- setTLOTypeName method
- setBOPrefix method
- setxAuxHeader method

Syntax:

```
public void setFromByteArray(byte[ ] dataBytes)
                      throws commonj.connector.runtime.DataBindingException
```

Parameter:

dataBytes
  The payload in the form of bytes.

setFromString method

This method takes a string as a parameter and converts it to a top-level object. Before calling this method, you may call one or more of the following methods:

- setTLOTypeName method
- setBOPrefix method
- setOptions method

Syntax:

```
public void setFromString(java.lang.String fromString)
                   throws commonj.connector.runtime.DataBindingException
```

Parameter:

fromString
  The string form of the top-level data object or payload.

setOptions method

This method sets the required options.

Syntax:

```
public void setOptions(java.util.HashMap options)
                throws commonj.connector.runtime.DataBindingException
```

Parameter:

Hashmap
  Hashmap can have the following key values:

  - BCG_ROOT_NODE_NAME

    This is the root name to be used to generate the payload.
  - BCG_APPEND_DTD

    This attribute indicates whether to append a DTD tag to the payload. The value can be BCG_APPEND_DTD_TRUE or BCG_APPEND_DTD_FALSE. The default is BCG_APPEND_DTD_FALSE.
  - BCG_SYSTEM_ID

    This is the system ID for the DTD. If the value of BCG_APPEND_DTD is BCG_APPEND_DTD_TRUE, this value may be set. If the value of BCG_APPEND_DTD is BCG_APPEND_DTD_FALSE, this value is ignored.
  - BCG_PUBLIC_ID

This is the public ID for the DTD. If the value of BCG_APPEND_DTD is BCG_APPEND_DTD_TRUE, this value is optional. If the value of BCG_APPEND_DTD is BCG_APPEND_DTD_FALSE, this option is ignored.

setPackagingSchema method

This method defines the packaging schema to use when generating Backend Integration packaging. You can specify one of the following packaging schemas:

- http://www.ibm.com/websphere/bcg/2003/v1.0/wbipackaging
- http://www.ibm.com/websphere/bcg/2003/v1.1/wbipackaging

If the packaging schema is not specified, http://www.ibm.com/websphere/bcg/2003/v1.1/wbipackaging is used.

The packaging schema is used when you call the write method.

Syntax:

```
public void setPackagingSchema(java.lang.String packagingSchema)
```

Parameter:

packagingSchema
The schema name for the top-level business object.

setTLOTypeName method

This method sets the top-level object name and its name space.

- If the payload is not XML and setTLOTypeName is not used, the default top-level object name is used.
- If the payload is an XML document, the top-level object name is derived as follow:

  `boPrefix + <` *`RootTag of the XML Payload`* `>.`

  For example, if setBOPrefix("IBM") is called, and the root element of the payload is "ABC", the top-level name is "IBM_ABC".

  The prefix can be set using the setBOPrefix method. If the top-level object name is not set and the payload is an XML doc, you must create a DataType of type boPrefix + < *RootTag* >.

Syntax:

```
public void setTLOTypeName(java.lang.String tns,
                           java.lang.String typeName)
```

Parameters:

tns
Namespace of the top-level business object.

typeName
The top-level object type.

setxAuxHeader method

This method sets the transport headers. If no transport headers are set, there are no transport headers to read. See "Transport-level header content" on page 20 for information about the x-aux headers.

Syntax:

```
public void setxAuxHeaders(java.util.HashMap xAuxHeaders)
```

write method

This method writes the top-level object into the specified output stream. The actions taken by the write method are described in detail in “Top-level object to Backend Integration packaging message conversion” on page 65.

Syntax:

```
public void write(java.io.OutputStream outputStream)
          throws commonj.connector.runtime.DataBindingException
```

Parameter:

outputStream
The output stream to which the data object will be written.

Example code

The following code shows how to create a top-level object from an input stream:

```
BCGBackEndIntegrationDataBindingUtil util = BCGBackEndIntegrationDataBindingUtil ();
util.setTLOTypeName ("TLO_URIName","TLOName");
//util.setBOPrefix ("BO_prefix");
//setBoPrefix is commented because setTLOTypeName () is being used
util.read (inputStream);
DataObject tlo = util.getDataObject ();
```

The following code shows how to get a stream from a data object:

```
BCGBackEndIntegrationDataBindingUtil util = BCGBackEndIntegrationDataBindingUtil ();
util.setOptions (options);
util.setDataObject (tlo);
byte [ ] tlo_bytes = util.getAsByteArray ();
```

BCGBackEndIntegrationJMSDataBindingImpl class

This class creates a business object from a JMS message containing a payload or writes the business object to a JMS message. When it reads the business object from a JMS message, the JMS transport headers are read before the business object is created. When it writes the business object to a JMS message, this class writes JMS transport headers if the business object has a packaging headers child business object.

BCGBackEndIntegrationJMSDataBindingImpl extends BCGBackEndIntegrationDataBindingUtil, which is described in “BCGBackEndIntegrationDataBindingUtil class” on page 67.

JMSDataBinding interface

BCGBackEndIntegrationJMSDataBindingImpl implements the JMSDataBinding interface:

```
public interface com.ibm.websphere.sca.jms.data.JMSDataBinding extends DataBinding {

      public void read(javax.jms.Message message) throws javax.jms.JMSException;
      public void write(javax.jms.Message message) throws javax.jms.JMSException;
      public int getMessageType();

      static public int OBJECT_MESSAGE = 0;
      static public int TEXT_MESSAGE = 1;
      static public int BYTES_MESSAGE = 2;
      static public int STREAM_MESSAGE = 3;
      static public int MAP_MESSAGE = 4;

}
```

Methods

The BCGBackEndIntegrationJMSDataBindingImpl methods are described in this section.

getMessageType

This method returns the type of message.

Syntax:

```
public int getMessageType()
```

isBusinessException

This method returns an indication of whether there are any business exceptions.

Syntax:

```
public boolean isBusinessException()
```

read

This method reads the transport headers and, depending on the message type, creates the top-level business object. The actions taken by the read method are described in detail in "Backend Integration packaging message to top-level object conversion" on page 61.

In this method, there is no need to set the AuxHeaders and read from the message itself.

Syntax:

```
public void read(javax.jms.Message jmsMessage)
         throws javax.jms.JMSException
```

Parameter:

jmsMessage
The message containing the payload and transport headers.

setBusinessException

This method sets an indicator of whether a business exception has occurred.

Syntax:

```
public void setBusinessException(boolean arg0)
```

write

This method writes the data object to the message, and, depending on the message type, sets the headers in the message. The actions taken by the write method are described in detail in "Top-level object to Backend Integration packaging message conversion" on page 65.

Syntax:

```
public void write(javax.jms.Message jmsMessage)
          throws javax.jms.JMSException
```

Including data-binding classes in your component implementation

While developing WebSphere Process Server components using WebSphere Integration Developer, you can use the WebSphere Partner Gateway-provided BCGBackEndIntegrationJMSDataBindingImpl and BCGBackEndIntegrationDataBindingUtil classes. These classes are available in databinding.jar, which is located in the Integration\WBI\WebSphereProcessServer\DataBinding directory on the product image.

To use these classes in your Business Integration project, you need to make sure databinding.jar is available in the EAR file of the Business Integration project you will deploy to WebSphere Process Server. To include the databinding.jar file in your generated EAR file, you can refer to the WebSphere Process Server and WebSphere Integration Developer information centers, or you can follow the procedure in this section. Before following this procedure, make sure you are in the Business Integration Perspective and that you have created the Business Integration project in which you are trying to import this jar file.

1. From WebSphere Integration Developer, import databinding.jar into your project. You need to import databindng.jar as "J2EE Utility jars". After a successful import, WebSphere Integration Developer will implicitly create a project for databinding.jar
2. Add this project as a dependency to your Business Integration project. To add the project as a dependency:

   Note: The following is an example of one way to add the project as a dependency. Refer to the WebSphere Process Server documentation for information on other ways to do to this.

   a. Double-click on your business integration project.

      This step opens the dependency editor.

   b. Expand the **Java** folder.

   c. Click the **Add** button.

      The list of projects is displayed.

   d. Select the project that was implicitly created by WebSphere Integration Developer for databinding.jar in step 1.

      The project created by WebSphere Integration Developer for databinding.jar is displayed as a dependency under Java.

   e. Select the project.

   f. Select the **Deploy with Module** check box.
3. Close the dependency editor and save the workspace.

After you complete these steps, databindng.jar will be available in your generated EAR file.

Chapter 4. Integrating WebSphere Process Server with HTTP as transport

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere Process Server over the HTTP transport protocol. This chapter provides the following information:

- "How messages are sent to WebSphere Process Server"
- "How messages are sent from WebSphere Process Server" on page 77
- "Sending documents to WebSphere Process Server" on page 79
- "Sending documents from WebSphere Process Server" on page 80

This chapter does not describe how SOAP requests sent over HTTP are processed. For information about sending and receiving SOAP requests over HTTP, see Chapter 6, "Integrating WebSphere Process Server with SOAP/HTTP," on page 101.

How messages are sent to WebSphere Process Server

This section describes how WebSphere Partner Gateway receives a message from a community participant and sends the message to WebSphere Process Server over HTTP.

Notes:

1. All document types except RosettaNet can have a packaging of None when sent from WebSphere Partner Gateway to WebSphere Process Server. RosettaNet documents must have Backend Integration packaging.
2. All document types except SOAP (Web service requests) can have a packaging of Backend Integration when sent from WebSphere Partner Gateway to WebSphere Process Server. SOAP requests must have a packaging of None specified.

   For information about sending and receiving SOAP documents over HTTP, see Chapter 6, "Integrating WebSphere Process Server with SOAP/HTTP," on page 101.

In order for WebSphere Process Server to receive a message sent from WebSphere Partner Gateway, you must write a servlet to retrieve the message and convert it to a business object.

WebSphere Partner Gateway sends messages to the URL configured in the Community Manager's "TO" HTTP gateway in the participant connection. The servlet is listening on this URL and receives the message. The service that will receive the converted business document has an SCA export binding. Figure 21 on page 76 shows how a message sent from a community participant is processed by WebSphere Partner Gateway and then sent to the HTTP servlet where, via the export binding, the business object invokes the service.

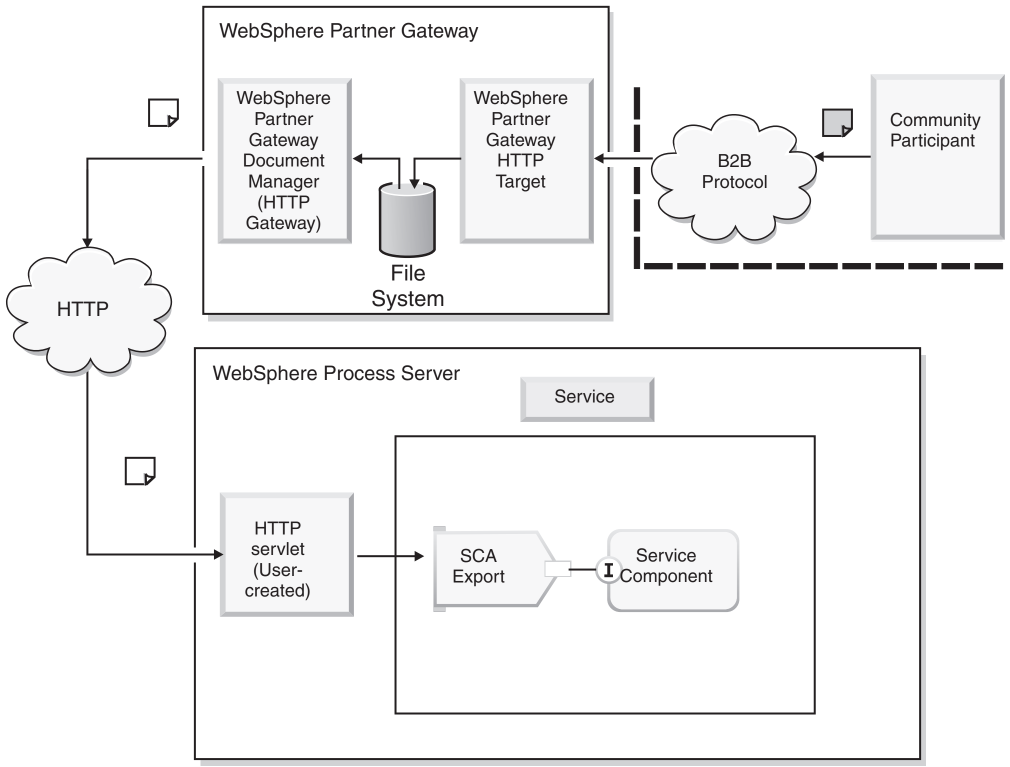
WebSphere Partner Gateway
WebSphere Partner Gateway Document Manager (HTTP Gateway)
WebSphere Partner Gateway HTTP Target
File System
B2B Protocol
Community Participant
HTTP
WebSphere Process Server
Service
HTTP servlet (User-created)
SCA Export
Service Component

Figure 21. How a message is sent from WebSphere Partner Gateway to the HTTP servlet

The HTTP servlet performs the following tasks:

1. From the request message it receives, the servlet determines which service and which method of that service need to be invoked.
2. The HTTP servlet constructs a business object from the incoming message, as described in “Creating a servlet” on page 80.
3. The HTTP servlet invokes the service using the appropriate SCA client API:
   a. If the SCA service method is request-only, there is no business response expected. If the HTTP servlet is able to invoke the SCA service, the servlet returns an HTTP 200 status code to WebSphere Partner Gateway. If the HTTP servlet is not able to invoke the SCA service, it returns the appropriate HTTP error status code.
   b. If the SCA service method is request-response, the SCA service returns a response business object. The HTTP servlet serializes this business object into a message. The HTTP servlet returns this message to WebSphere Partner Gateway in an HTTP response. If for some reason the HTTP servlet is not able to invoke the SCA service successfully, it returns the appropriate HTTP error status code.

      For example, for a cXML synchronous message received from participants, WebSphere Partner Gateway sends the cXML message to WebSphere Process Server over HTTP. WebSphere Partner Gateway expects WebSphere Process Server to provide a synchronous cXML response on the same HTTP connection.

Note: WebSphere Partner Gateway supports request-only invocations for Backend Integration packaging. You can, however, develop request-response flows with Backend Integration packaging data binding (via WebSphere Partner

Gateway user exits). The BCGBackEndIntegrationDataBindingUtil class can be used as the basis for converting a response business object into a response stream.

How messages are sent from WebSphere Process Server

This section describes how WebSphere Partner Gateway receives a message from WebSphere Process Server over HTTP and sends it to a community participant.

Notes:

1. All document types except RosettaNet and binary can have a packaging of None when sent from WebSphere Process Server to WebSphere Partner Gateway. RosettaNet and binary documents must have Backend Integration packaging.
2. All document types except SOAP (Web service requests) can have a packaging of Backend Integration when sent from WebSphere Process Server to WebSphere Partner Gateway. SOAP requests must have a packaging of None specified.

   For information about sending and receiving SOAP documents over HTTP, see Chapter 6, "Integrating WebSphere Process Server with SOAP/HTTP," on page 101.

When WebSphere Process Server sends a message to WebSphere Partner Gateway, it uses a component that does an HTTP POST to a WebSphere Partner Gateway HTTP target. to send an HTTP POST to a WebSphere Partner Gateway HTTP target. You develop the component and expose it as a service.

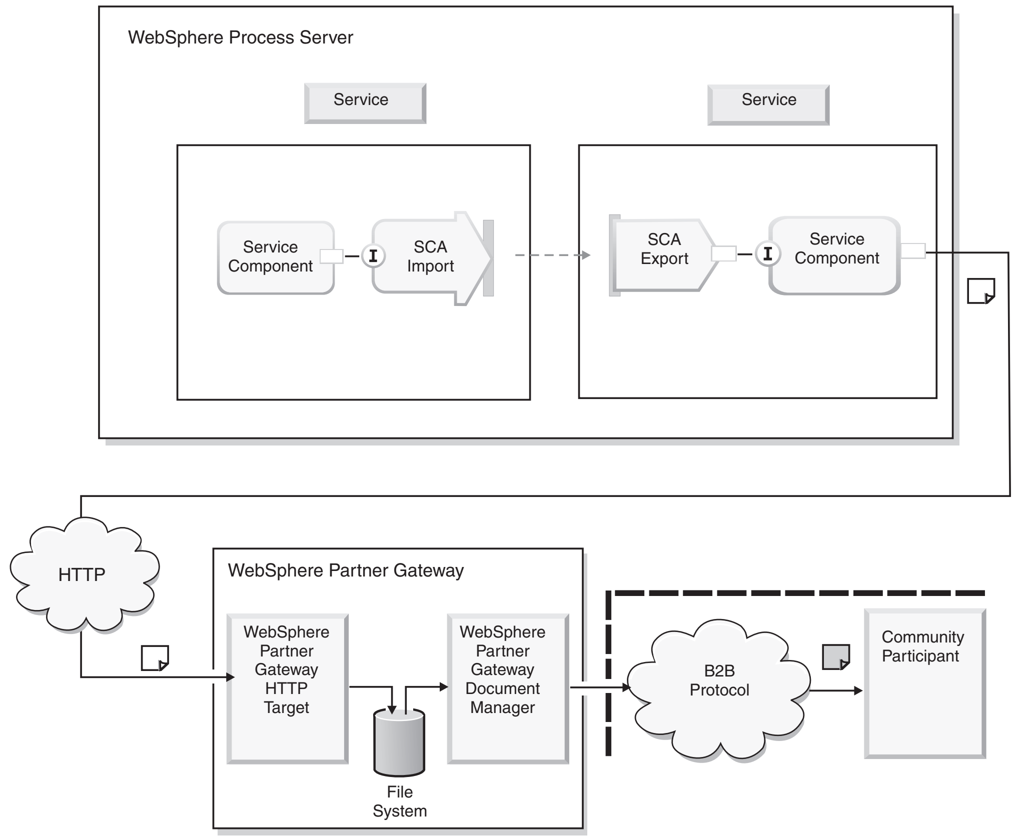

Figure 22. How a message is sent from WebSphere Process Server to the HTTP target on WebSphere Partner Gateway

WebSphere Process Server services send messages to WebSphere Partner Gateway as follows:

1. A WebSphere Process Server SCA service that needs to send business documents to WebSphere Partner Gateway over HTTP uses its SCA import binding to invoke the component that will perform the HTTP POST. The SCA import binding of the service invokes the component with a business object.
2. The component receives the business object on its export. The component then serializes the business object into a business document. See "Creating the component to convert and send the message" on page 81.
3. The component sends the message using an HTTP POST to the WebSphere Partner Gateway HTTP receiver URL.
4. WebSphere Partner Gateway receives this message on its HTTP receiver.

   The way WebSphere Partner Gateway responds depends on whether the message requires a synchronous response or whether it requires only an HTTP transport-level response. The HTTP target has a SyncCheck configuration point that makes this determination. You can configure SyncCheck when you create the target, or you can modify the target to specify it. See the *Hub Configuration Guide* for information on the SyncCheck configuration point.

   a. If WebSphere Partner Gateway determines that the message received from the WebSphere Process Server requires only a transport-level response, the WebSphere Partner Gateway receiver responds with the appropriate HTTP status code. It then routes the business document to the community participant.

   b. If WebSphere Partner Gateway determines that the message received from WebSphere Process Server requires a response business document, it keeps the HTTP transport connection open. WebSphere Partner Gateway routes the business document to a community participant. The community participant receives the request document and sends a response document to WebSphere Partner Gateway. WebSphere Partner Gateway returns the response business document as an HTTP response to WebSphere Process Server.
5. The component receives the HTTP response. It determines whether the response is an HTTP status code only or also a business document:
   a. If the response is an HTTP status code indicating success, the component returns the call. The WebSphere Process Server SCA service (which invoked the component) continues its processing. However, if the HTTP status code indicates failure, the component returns the appropriate fault. The WebSphere Process Server SCA service does the appropriate fault handling.
   b. If the response is a business document, the component converts this business document to a business object. It then returns the business object to the WebSphere Process Server SCA service, which processes the response business object.

      For example, for cXML synchronous messages received from WebSphere Process Server and intended for a community participant, WebSphere Partner Gateway sends the message to the participant and receives the response synchronously. WebSphere Partner Gateway then returns the response to WebSphere Process Server as an HTTP response of the same HTTP request that was originally sent from WebSphere Process server.

Note: WebSphere Partner Gateway supports request-only invocations for Backend Integration packaging. You can, however, develop request-response flows with Backend Integration packaging data binding (via WebSphere Partner Gateway user exits). The BCGBackEndIntegrationDataBindingUtil class can be used as the basis for converting a response business object into a response stream.

Sending documents to WebSphere Process Server

The section describes the steps you take to enable WebSphere Process Server to accept documents sent from WebSphere Partner Gateway

Setting up WebSphere Partner Gateway

You configure WebSphere Partner Gateway as described in the *Hub Configuration Guide*. Make note of the following as you configure WebSphere Partner Gateway:

1. Add the com.ibm.bcg.server.sync.SoapSyncHdlr handler to the syncCheck configuration point of the HTTP target (if you will be handling two-way requests).
2. Make sure the HTTP gateway of the Community Manager indicates the URL to which WebSphere Partner Gateway will send messages to WebSphere Process Server.

   The gateway should point to the HTTP servlet deployed on WebSphere Process Server.

Setting up WebSphere Process Server

This section describes how to set up WebSphere Process Server to receive messages from WebSphere Partner Gateway.

Creating a business object

If you are using the BCGBackendIntegrationDataBindingUtil class to perform the conversion from message to business object, you must use the top-level business object described in "Top-level and child business objects" on page 57. You can use the BO Editor to create the business object, or you can import the xsd file (BCG_TLO_BackendPackaging.xsd) that is available on the installation medium.

If you are not using Backend Integration packaging (in other words, if the packaging for the document was specified as None), you create the business object per the requirements of the document exchange.

Creating a servlet

This section describes the servlet you must create so that WebSphere Partner Gateway can send messages to WebSphere Process Server over the HTTP transport. Design the servlet to:

- Receive messages from WebSphere Partner Gateway
- Convert the business document to a business object
- Invoke the appropriate service on WebSphere Process Server

If you are using Backend Integration packaging, you can use the BCGBackEndIntegrationDataBindingUtil class to convert an HTTP request message into a request business object expected by your service. Also you can use BCGBackEndIntegrationDataBindingUtil to create an HTTP response message from the response business object returned by your service. An example on how to use the BCGBackEndIntegrationDataBindingUtil class to convert an HTTP request message into a request business object follows. This example uses the read method and the getDataObject method. If you know the name of the top level object in advance, you can also call the setTLOTypeName method.

```
try{
    DataObject rootBO = null;
    BCGBackEndIntegrationDataBindingUtil dataBinding = new
BCGBackEndIntegrationDataBindingUtil();

    // request.getInputStream() gives the
    // backend integration packaging input stream received by servlet
    dataBinding.read(request.getInputStream());
    rootBO = dataBinding.getDataObject();

}
catch(Exception exp){
    System.out.println("Error occurred while creating request business
object: " + exp);
}
```

After you create the servlet, deploy it. The URL at which this servlet is receiving should be specified as the URL in the TO gateway configured for the participant connection in WebSphere Partner Gateway.

Sending documents from WebSphere Process Server

The section describes the steps you take to enable WebSphere Process Server to send documents to WebSphere Partner Gateway

Setting up WebSphere Partner Gateway

You configure WebSphere Partner Gateway as described in the *Hub Configuration Guide*. Make note of the following as you configure WebSphere Partner Gateway:

1. Create an HTTP/S target on the hub to receive the documents sent from WebSphere Process Server (if one does not already exist). This target identifies the URL at which the Receiver component of WebSphere Partner Gateway listens for documents from WebSphere Process Server.
2. Add the com.ibm.bcg.server.sync.SoapSyncHdlr handler to the syncCheck configuration point of the HTTP target (if you will be handling two-way requests).

Setting up WebSphere Process Server

This section describes how to set up WebSphere Process Server to send messages to WebSphere Partner Gateway. For a WebSphere Process Service to send a message over HTTP to WebSphere Partner Gateway, you require a component that can do an HTTP POST to the URL at which the WebSphere Partner Gateway HTTP target is receiving documents. Whenever a WebSphere Process Server service sends a business document to a community participant, it will invoke this component.

Creating a business object

If you are using the BCGBackendIntegrationDataBindingUtil class to perform the conversion from business document to business object, you must use the top-level business object described in “Top-level and child business objects” on page 57. You can use the BO Editor to create the business object, or you can import the xsd file (BCG_TLO_BackendPackaging.xsd) that is available on the installation medium.

If you are not using Backend Integration packaging (in other words, if the packaging was specified as None), you create the business object per the requirements of the document exchange.

Creating the component to convert and send the message

Create a component that converts the business object into an input stream and then sends the HTTP message to WebSphere Partner Gateway. This component will serialize the business object obtained from the service into a business document. It can then use the JAVA API java.net.HTTPURLConnection to perform an HTTP POST to the URL at which the WebSphere Partner Gateway HTTP target is receiving documents.

The following example illustrates how this component can POST a message to WebSphere Partner Gateway, if NONE packaging is used.

```
String wpgHTTPTargetURL;     // URL of WebSphere Partner Gateway HTTP Target
byte payload[];   // message that needs to be sent to WebSphere Partner Gateway

// Set wpgHTTPTargetURL
// Set payload bytes. Convert business object received from service into bytes.
...

try{

           java.net.HttpURLConnection uc = (java.net.HttpURLConnection)
                new URL(wpgHTTPTargetURL).openConnection();

      uc.setDoInput(true);
      uc.setDoOutput(true);
      uc.setRequestMethod("POST");

      // Set the content type you want to send
      uc.setRequestProperty("Content-Type", "text/xml");

      uc.connect();
      BufferedOutputStream os = new BufferedOutputStream (uc.getOutputStream());
      os.write( payload);
      os.close();

      // If you are expecting response business document from WebSphere Partner
      // Gateway, then you have to read the response
      InputStream is = uc.getInputStream();
      if(is != null){
              BufferedInputStream bis = new BufferedInputStream
(uc.getInputStream());
              // Read bytes from BufferedInputStream obtained above
              // Convert response business document received from WebSphere
              // Partner Gateway into business object
              ...
              bis.close();
        }
    }
    catch(Throwable ex){
      System.out.println("Cannot post: " + ex.getMessage());
      ex.printStackTrace();
    }
  }
```

Important: If you are using Backend Integration packaging to send documents to WebSphere Process Server, you can use the BCGBackEndIntegrationDataBindingUtil class to convert business objects received from the service (or component) into business documents expected by WebSphere Partner Gateway. If you are expecting a response business document from WebSphere Partner Gateway, you can also use BCGBackEndIntegrationDataBindingUtil to create the HTTP response message from the response business object returned by your service.

Chapter 5. Integrating WebSphere Process Server with JMS as transport

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere Process Server over the JMS transport protocol. It contains procedures for configuring WebSphere Partner Gateway for JMS, when Websphere Platform Messaging (default messaging provider) and a service integration bus are used as the JMS provider and WebSphere Partner Gateway is installed on WebSphere Application Server. It also contains procedures for configuring WebSphere Partner Gateway for JMS, when Websphere Platform Messaging and a service integration bus are used as the JMS provider and WebSphere Partner Gateway is installed on embedded application server. When installed on embedded application server, WebSphere Partner Gateway uses WebSphere MQ to communicate with WebSphere Platform Messaging.

This chapter provides the following information on how to send and receive documents between WebSphere Partner Gateway and WebSphere Process Server using the JMS transport protocol:

- "Sending documents using the JMS transport protocol"
- "Receiving documents using the JMS transport protocol" on page 84
- "Setting up the WebSphere Process Server environment on WebSphere Application Server" on page 86
- "Configuring JMS when WebSphere Partner Gateway is installed on WebSphere Application Server" on page 89
- "Configuring JMS when WebSphere Partner Gateway is installed on the embedded application server" on page 94

Sending documents using the JMS transport protocol

For WebSphere Partner Gateway to send a document to WebSphere Process Server using the JMS transport protocol, JMS Export binding must be used to invoke the WebSphere Process Server component over JMS. Service Component Architecture (SCA) components can receive JMS messages from the JMS queue configured in their JMS Export binding. Figure 23 on page 84 provides an overview of how WebSphere Partner Gateway sends a document to WebSphere Process Server over the JMS transport protocol.

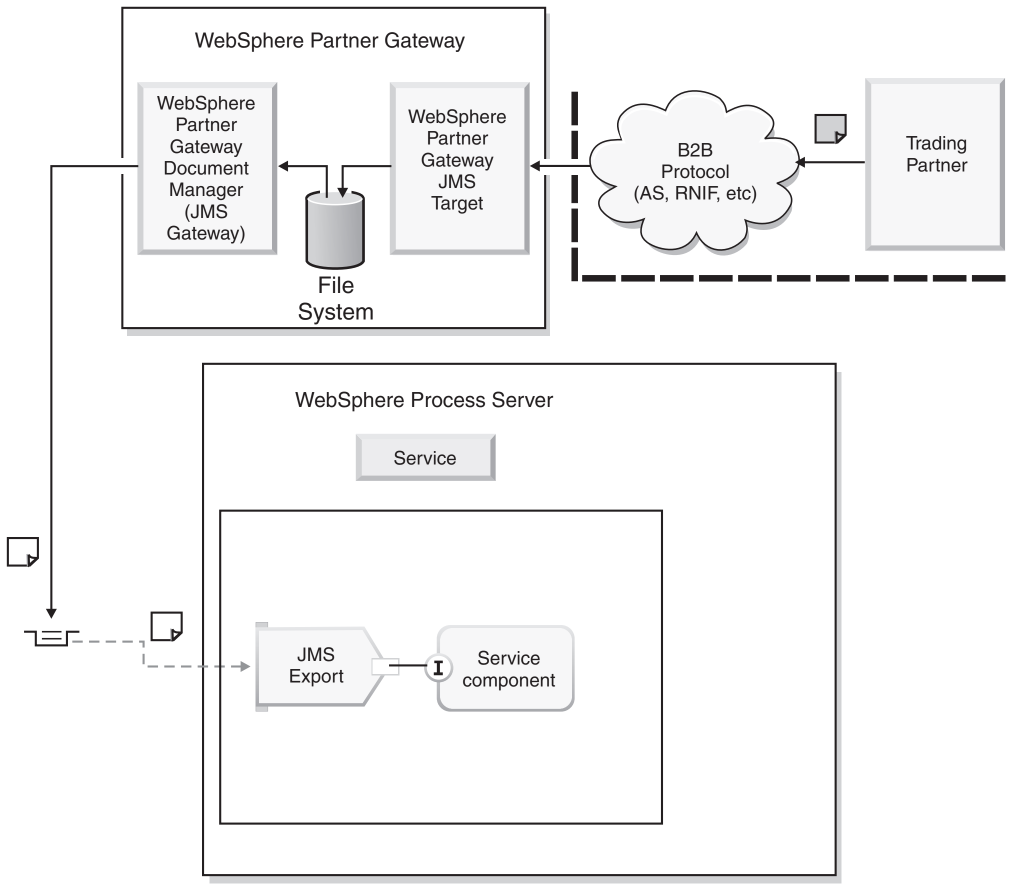
WebSphere Partner Gateway
WebSphere Partner Gateway Document Manager (JMS Gateway)
File System
WebSphere Partner Gateway JMS Target
B2B Protocol (AS, RNIF, etc)
Trading Partner
WebSphere Process Server
Service
JMS Export
Service component

Figure 23. JMS Export to invoke SCA services over JMS

The following steps describe how SCA JMS Export bindings are used to invoke SCA services over JMS:

1. Trading partner sends a business document to WebSphere Partner Gateway using a B2B protocol.
2. WebSphere Partner Gateway receives the business document from the trading partner.
3. Using the configured participant connection for this business document, WebSphere Partner Gateway routes the document to WebSphere Process Server.

For JMS based integration, it is expected that the ″To″ Gateway of the participant connection is a JMS gateway.

This JMS gateway is configured to send JMS messages to an SI Bus destination.

The SI Bus destination is the JMS queue configured in the SCA JMS Export binding of the SCA component.

Receiving documents using the JMS transport protocol

For WebSphere Partner Gateway to receive a document from WebSphere Process Server using the JMS transport protocol, JMS Import binding must be used to invoke external services over JMS. Service Component Architecture (SCA) components can send JMS messages to the JMS queue configured in their JMS

Import binding. Figure 24 provides an overview of how WebSphere Partner Gateway receives a document from WebSphere Process Server over the JMS transport protocol.

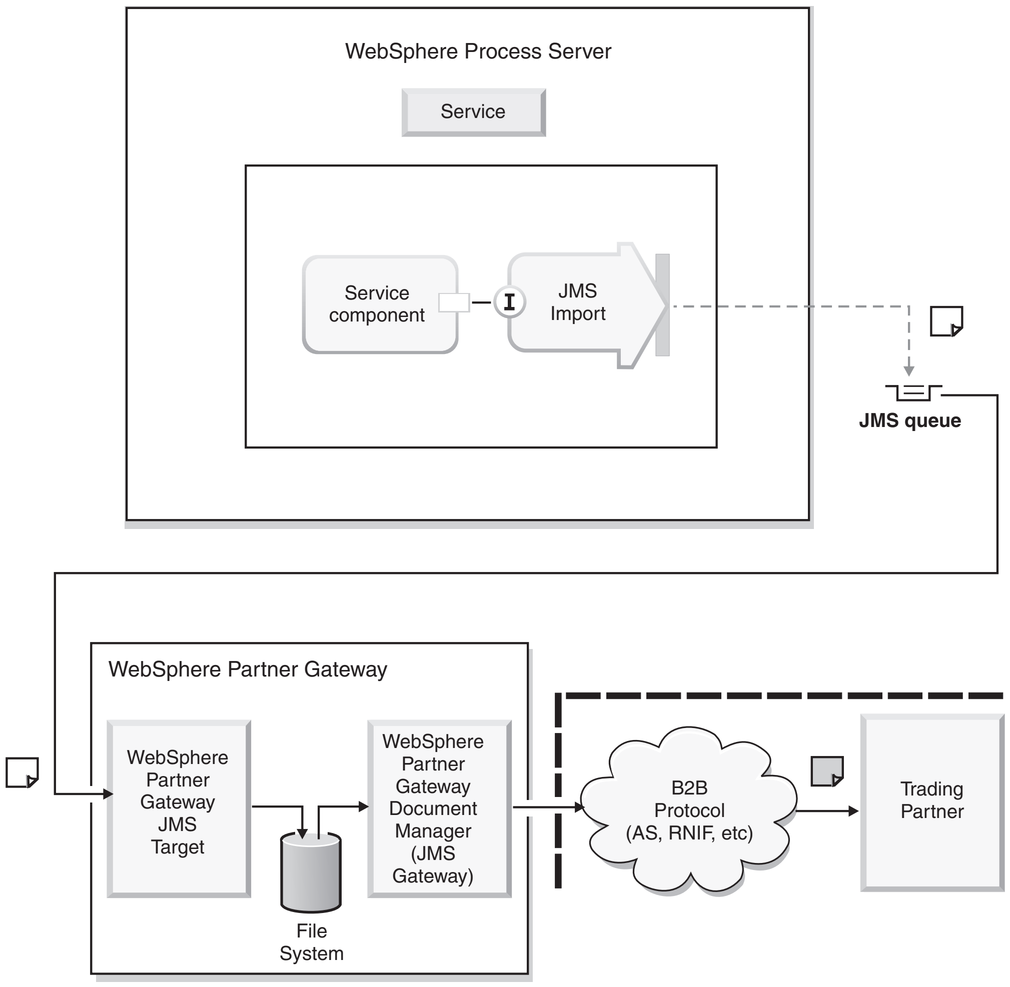
WebSphere Process Server
Service
Service component
JMS Import
JMS queue
WebSphere Partner Gateway
WebSphere Partner Gateway JMS Target
File System
WebSphere Partner Gateway Document Manager (JMS Gateway)
B2B Protocol (AS, RNIF, etc)
Trading Partner

Figure 24. JMS Import to receive a document over JMS

The following steps describe how SCA JMS Import bindings are used to receive a document from WebSphere Process Server:

1. WebSphere Partner Gateway JMS Receiver receives a business document from WebSphere Process Server.
2. Using the configured participant connection for this business document, WebSphere Partner Gateway routes this document to the trading partner.
3. Trading partner receives business document from WebSphere Partner Gateway over mutually agreed upon B2B protocol.

For JMS based integration, it is expected that the JMS queue configured in the JMS Target of WebSphere Partner Gateway Receiver is the JMS destination on which WebSphere Process Server Services will send JMS messages.

This JMS queue destination is the JMS queue destination configured in the SCA JMS Import binding of the SCA component.

Setting up the WebSphere Process Server environment on WebSphere Application Server

This section provides the steps for setting up the WebSphere Process Server environment for the JMS transport on WebSphere Application Server. This includes creating and configuring the WebSphere Process Server artifacts.

This section provides the following information:

- "Creating an SCA service with WSDL"
- "Customizing JMS Import and Export bindings"
- "Implementing JMS data binding" on page 87
- "Customizing a Function Selector" on page 87

Creating an SCA service with WSDL

An SCA service, combined with Export and Import bindings, is used to leverage WebSphere Partner Gateway's B2B and trading partner interaction capabilities. For instructions on how to use the Web Services Description Language (WSDL) to create and define an SCA service, see the WebSphere Process Server information center.

Customizing JMS Import and Export bindings

JMS Import binding of a component is used to invoke external services over JMS. SCA components can send JMS messages to the JMS queue configured in their JMS Import binding. To enable SCA components to send messages to the queue configured in the WebSphere Partner Gateway JMS Target:

1. Provide JMS data binding required by SCA JMS Import of the component. Refer to "Implementing JMS data binding" on page 87.
2. Specify JMS Import binding attributes specific to your environment. JMS Import of component places messages on the JMS destination specified in the JMS Import binding. The JMS destination you specify in JMS Import binding should refer to the JMS queue configured in WebSphere Partner Gateway JMS target. For more details on JMS Import binding attributes, see the WebSphere Process Server information center.

JMS Export binding of a component is used to invoke the WebSphere Process Server component over JMS. To enable SCA components to receive JMS messages from the JMS destination configured in their JMS Export binding:

1. Provide JMS data binding as required by SCA JMS Export binding of the component. Refer to "Implementing JMS data binding" on page 87.
2. Provide function selector as required by SCA JMS Export binding of the component. Refer to "Customizing a Function Selector" on page 87.
3. Specify JMS Export binding attributes specific to your environment. JMS Export of a component retrieves the JMS message from the JMS destination specified in the JMS export binding. The JMS destination you specify in JMS Export binding should refer to the JMS queue configured in WebSphere Partner Gateway JMS gateway. For more details of JMS Import binding attributes, see the WebSphere Process Server information center.

Implementing JMS data binding

WebSphere Process Server SCA JMS Import and Export components provide configuration information for specifying JMS data binding. JMS data binding is used by SCA JMS Import and Export to convert business objects into JMS messages and JMS messages into business objects:

- SCA JMS Export: SCA JMS Export of a service leverages a configured JMS data binding to convert a JMS message received from WebSphere Partner Gateway into business object as expected by the method of the service. This business object is used to invoke the method of a service.
- SCA JMS Import: SCA JMS Import of a service leverges a configured JMS data binding to convert a business object into a JMS message as expected by WebSphere Partner Gateway.

To develop and implement JMS data binding, see the WebSphere Process Server information center.

When JMS is used for backend integration, WebSphere Partner Gateway supports NONE packaging and backend integration packaging.

If NONE packaging is used, you can leverage the WebSphere Process Server provided default JMS data binding or you can implement your own data binding. Typically you will have to write your own data binding if the format of JMS messages exchanged between WebSphere Partner Gateway and WebSphere Process Server is other than XML.

If backend integration packaging is used, WebSphere Partner Gateway provided com.ibm.bcg.dataBinding.BCGBackEndIntegrationJMSDataBindingImpl can be leveraged. To customize the processing, you can implement data binding by utilizing com.ibm.bcg.dataBinding.BCGBackEndIntegrationDataBindingUtil or com.ibm.bcg.dataBinding.BCGBackEndIntegrationJMSDataBindingImpl provided by WebSphere Partner Gateway. For more information on these classes please refer to “BCGBackEndIntegrationJMSDataBindingImpl class” on page 72 and “BCGBackEndIntegrationDataBindingUtil class” on page 67.

To implement the JMS data binding interface for Backend Integration Packaging, create user-specified read and write methods.

Customizing a Function Selector

A Function Selector is required for SCA JMS Export binding. It is used to determine what service method will be invoked for a business document that is received at the JMS destination.

Implement a function selector using one of the following techniques:

- NONE packaging: If WebSphere Partner Gateway is sending JMS message to WebSphere Process Server using NONE packaging, the user must determine what method of their SCA service to invoke. With NONE packaging, the only way to determine the type of business document is to parse the document. You can use one of the following techniques to parse the document:
  - Dispatcher: For NONE packaging, IBM recommends that you do not parse the document using the function selector. This is redundant and expensive as documents need to be parsed in the data binding also. IBM recommends that you develop a dispatcher or function-selector component which will be the front end to the actual WebSphere Process Server component interested in the business document. JMS Export binding of this dispatcher component will

receive the business document from WebSphere Partner Gateway. This dispatcher component will have one method on which it can receive all possible business documents. JMS Export binding will un-marshal JMS messages into business objects. The dispatcher module will then determine which component of the service is interested in this business object and invoke the component with this business object.

  - Minimal Parsing: If a SCA JMS Export binding has multiple method bindings, the function selector can be implemented perform minimal parsing of the business document to determine the type of business document (for example PO, POConfirm, Invoice etc). The function selector can then return the type of business document as a native method name. At configuration time, the native method name in the method binding of the SCA JMS Export binding can be specified as a business document type. Using the method binding, SCA JMS Export can resolve which method of the service needs to be invoked.

    Note: Data binding performs a full parsing of the business document to construct the business object. Therefore, the function selector should do a very minimal parsing of the business document. If parsing in the function selector is too expensive, you should not use this technique.
  - Trivial function selector: If the user is expecting only one type of business document, you can develop a trivial function selector which will not parse the document and that will always return the same native method name. At configuration time, the native method name in the method binding of the SCA JMS Export binding can be specified as this method name. Using the method binding, SCA JMS Export will be able to determine which method of the service needs to invoke. An advantage of this technique is that no parsing of incoming an business document is required. However, the limitation of this approach is that there can be only one type of business document on the JMS queue.
- Backend integration packaging: If WebSphere Partner Gateway is sending a JMS message to WebSphere Process Server using backend integration packaging, the user can leverage backend integration packaging JMS headers to develop a function selector. The JMS properties can be read by the function selector from the JMS message input. The function selector can use the value of one of the backend integration packaging JMS headers to compose a native method name. For example, for a RosettaNet message, the x-aux-payload- root-tag JMS property can be read to determine the type of business document. The function selector can then compose the native method name to match the native method name in method binding of JMS Export.

Using the WebSphere Integration Developer, create a custom a function selector by implementing the commonj.connector.runtime.FunctionSelector interface and creating a custom function selector. For example:

```
public class PurchaseOrderSelector implements FunctionSelector {
        public String generateEISFunctionName(Object[] arg0)
                      throws SelectorException {
                        return "receiveMessage";
                }
}
```

Configuring JMS when WebSphere Partner Gateway is installed on WebSphere Application Server

This section provides the steps for setting up the WebSphere Partner Gateway environment for the JMS transport on WebSphere Application Server when using WebSphere Platform Messaging as the messaging provider. This section assumes that WebSphere Partner Gateway is installed on WebSphere Application Server.

If you have installed WebSphere Partner Gateway on the embedded application server, you must utilize MQ Link and WebSphere MQ to access WebSphere Platform Messaging. For information on setting up the WebSphere Partner Gateway environment for the JMS transport on the embedded application server, refer to "Configuring JMS when WebSphere Partner Gateway is installed on the embedded application server" on page 94.

The information in this section assumes that the user is familiar with WebSphere Platform Messaging and service integration buses.

This section provides the following information:
- "Creating and configuring buses, JMS queues, and Connection Factories" on page 90
- "Creating the JMS target" on page 92
- "Creating the JMS gateway" on page 93
- "Creating a destination queue" on page 93

Using the message engine (ME) and JNDI from the WebSphere Partner Gateway component's application server may restrict the bus topologies for the customer. Therefore, IBM recommends that WebSphere Partner Gateway components (JMS Receiver and JMS gateway) connect to the ME on a different application server, which may or may not be running WebSphere Process Server. See Figure 25 on page 90 for an overview of this topology.

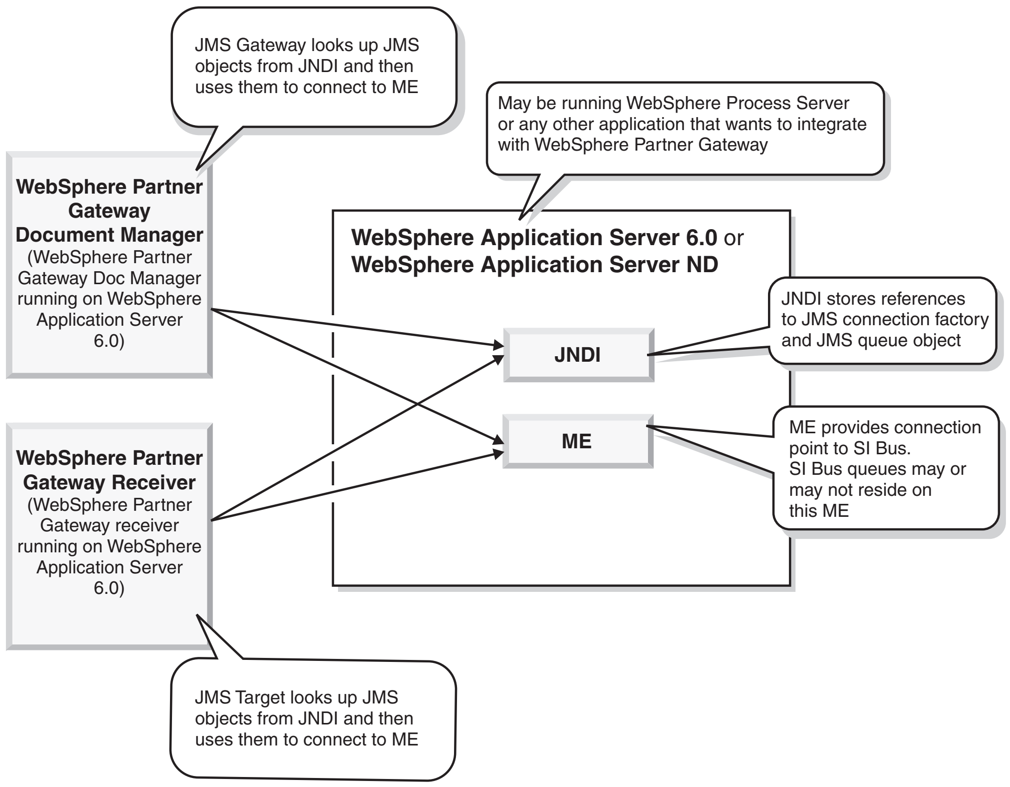
JMS Gateway looks up JMS objects from JNDI and then uses them to connect to ME
May be running WebSphere Process Server or any other application that wants to integrate with WebSphere Partner Gateway
WebSphere Partner Gateway Document Manager (WebSphere Partner Gateway Doc Manager running on WebSphere Application Server 6.0)
WebSphere Application Server 6.0 or WebSphere Application Server ND
JNDI stores references to JMS connection factory and JMS queue object
JNDI
ME
ME provides connection point to SI Bus. SI Bus queues may or may not reside on this ME
WebSphere Partner Gateway Receiver (WebSphere Partner Gateway receiver running on WebSphere Application Server 6.0)
JMS Target looks up JMS objects from JNDI and then uses them to connect to ME

Figure 25. Accessing the ME and JNDI from another WebSphere Application Server instance

Creating and configuring buses, JMS queues, and Connection Factories

This section describes how to create and configure buses, JMS queues, and Connection Factories that will be used by WebSphere Partner Gateway to send and receive messages.

Creating a service integration bus

A service integration bus supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

Using the WebSphere Administrative Console:

1. Create and name a bus.
   a. Click **Service integration** > **Buses**.
   b. Click **New** and provide a bus name. For example, SIBUS.
   c. Click **Apply**.
   d. Save the configuration by clicking **Save** in the Messages window that appears. This message is to confirm that you want to apply the changes to the master configuration.
   e. Click **Save** again when asked to update the master repository with your changes.
2. Add bus members to the bus.
   a. Click on the name of the newly created bus.

b. In the Additional Properties pane, click **Bus members**.

c. Click **Add** and select the server or cluster to be added.

d. Click **Next** and then click **Finish** to confirm the addition of the new bus members.

e. Save the configuration by clicking **Save** in the Messages window that appears. This message is to confirm that you want to apply the changes to the master configuration.

f. Click **Save** again when asked to update the master repository with your changes.

3. Create a destination queue:
   a. In the WebSphere Application Server default Console, click on **System Integration** in the left panel.
   b. Click on **Buses** < **SIBUS** (or the name of the bus created in step 1).
   c. In the Additional Properties pane, click **Destinations**. Click **New**.
   d. Select the **Queue** radio button for the destination type and click **Next**.
   e. Enter an **Identifier**. For example, Request. This will create the destination queue on the bus.
4. Save the configuration by clicking **Save** in the Messages window that appears. This message is to confirm that you want to apply the changes to the master configuration.
5. Click **Save** again when asked to update the master repository with your changes.

Creating a JMS Queue Connection Factory

A JMS queue connection factory is used to create connections to the associated JMS provider of JMS queues, for point-to-point messaging.

Using the WebSphere Administrative Console:

1. Create a Queue Connection Factory by populating the **Name** and **JNDI name** fields using the following syntax:
   - **Name**: SIBUS.JMSTargetQCF
   - **JNDI name**: SIBUS/JMSTargetQCF

   Where SIBUS is the name of the bus created in the previous steps.
2. Select the bus. For example, **SIBUS**.
3. In the resulting window, click on the Queue Connection Factory that you just created and input the **Provider endpoints** as:
   ```
   IPaddress/Name:7276:BootstrapBasicMessaging
   ```
   Where IPaddress is the IP address or the name of the machine on which WebSphere Application Server is running. It is expected that the Message Engine for this service integration bus is running on this machine. 7276 is the port number specified for SIB_ENDPOINT_ADDRESS for the WebSphere instance. If your Messaging Engine is running on the system with the IP address 9.26.234.100 and SIB_ENDPOINT_ADDRESS for the WebSphere instance running on this server is specified as 7276, you will specify Provider endpoints as follows:
   ```
   9.26.234.100:7276:BootstrapBasicMessaging
   ```
4. Save the configuration by clicking **Save** in the Messages window that appears. This message is to confirm that you want to apply the changes to the master configuration.

5. Click **Save** again when asked to update the master repository with your changes.

Creating a JMS queue

A JMS queue is used as a destination for point-to-point messaging.

Using the WebSphere Administrative Console:

1. Expand the Resources menu and click **JMS Providers** > **Default messaging**.
2. Click **JMS queues** in the Destinations section of the resulting page.
3. Click **New**.
4. Input a queue name in both the **Name** and **JNDI name** fields using the following syntax:
   - **Name**: Request.JMSTarget
   - **JNDI name**: Request/JMSTarget
5. Select the **Bus Name** (For example, SIBUS) and **Queue name** from the drop-down lists.
6. Click **OK**.
7. Save the configuration by clicking **Save** in the Messages window that appears. This message is to confirm that you want to apply the changes to the master configuration.
8. Click **Save** again when asked to update the master repository with your changes.

Creating the JMS target

This section provides directions for creating a target in WebSphere Partner Gateway.

Using the WebSphere Partner Gateway Community Console:

1. Click **Hub Admin > Hub Configuration > Targets** to display the Target List.
2. From the Target List page, click **Create Target**.
3. In the Target Details section, perform the following steps:
   a. Type a name for the target. For example, you might call the target JMSTarget. This is a required field. The name you enter here will be displayed on the Targets list.
   b. Optionally indicate the status of the target. Enabled is the default. A target that is enabled is ready to accept documents. A target that is disabled cannot accept documents.
   c. Optionally enter a description of the target.
4. Select **JMS** from the Transport list.
5. Enter the JMS provider URL. For example,
   ```
   iiop://systemip:2809/
   ```
   Where systemip is the IP address of the system where WebSphere Platform Messaging is running and 2809 is the default port where the BOOTSTRAP server is running.
6. Enter a value for JMS queue name. This is a required field. This name should match the JNDI name of the JMS queue created in the previous section. For example, Request/JMSTarget.
7. Enter a value for the JMS factory name. This is a required field. This name should match the Queue Connection Factory name created earlier. For example, SIBUS/JMSTargetQCF.

8. Enter the JNDI factory name as follows:
   ```
   com.ibm.websphere.naming.WsnInitialContextFactory
   ```
9. Click **Save**.

Creating the JMS gateway

This section provides directions for creating the Gateway in WebSphere Partner Gateway.

Using the WebSphere Partner Gateway Community Console, create a JMS gateway:

1. Click **Account Admin > Profiles > Community Participant**.
2. Enter search criteria and click **Search**, or click **Search** without entering any search criteria to display a list of all participants.
3. Click the View details icon to display the participant's profile.
4. Click **Gateways**.
5. Click **Create**.
6. From the Gateway List page, type a name to identify the gateway. This is a required field.
7. Select **JMS** from the **Transport** type list.
8. In the **Address** field, enter the URI where the document will be delivered. For example:
   ```
    iiop//systemip:2809
   ```
   Where systemip is the IP address of the system on which WebSphere Platform Messaging is running.
9. In the **JMS Factory Name** field, enter the name of the Java class that the JMS provider uses to connect to the JMS queue. This will be the JNDI name for the JMS factory we created earlier. For example:
   ```
   SIBUS/JMSTargetQCF
   ```
10. In the **JMS Queue Name** field, enter the name of the JMS queue where documents are to be sent. This will be the JNDI name for the queue where messages must be placed. For example:
    ```
    Response/JMSTarget
    ```
11. In the **JMS JNDI Factory Name** field, enter the following factory:
    ```
    com.ibm.websphere.naming.WsnInitialContextFactory
    ```
12. Click **Save**.

Creating a destination queue

Using the WebSphere Administrative Console, create a destination queue on the WebSphere Application Server:

1. Click on **System Integration** in the left panel.
2. Click on **Buses** < **SIBUS** (or the name of the bus created in the previous section).
3. In the Additional Properties pane, click **Destinations**. Click **New**.
4. Select the **Queue** radio button for the destination type and click **Next**.
5. Enter an **Identifier**. For example, Response. This will create the destination queue on the bus.

Configuring JMS when WebSphere Partner Gateway is installed on the embedded application server

This section provides the steps for setting up the WebSphere Partner Gateway environment for the JMS transport when using WebSphere Platform Messaging as the messaging provider. This section assumes that WebSphere Partner Gateway is installed on the embedded application server. For this scenario, Websphere MQ must be installed. You will use MQ Link and WebSphere MQ to access WebSphere Platform Messaging.

The information in this section assumes that the user is familiar with WebSphere Platform Messaging and service integration buses.

MQ Link creates a bridge between the ME and WebSphere MQ. Figure 26 provides an overview of MQ Link.

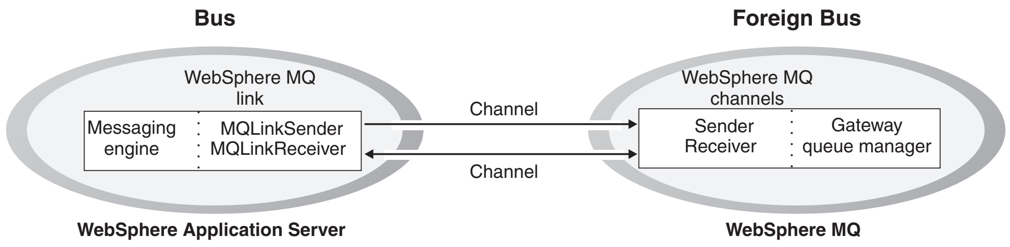

Figure 26. MQ Link used as a bridge between ME and WebSphere MQ

For information on setting up the WebSphere Partner Gateway environment for the JMS transport on WebSphere Application Server, refer to “Setting up the WebSphere Process Server environment on WebSphere Application Server” on page 86.

This section provides the following information:
- “Creating and configuring buses”
- “Setting up MQ Link and MQ queues” on page 95
- “Defining WebSphere Partner Gateway inbound queues” on page 98
- “Defining WebSphere Partner Gateway outbound queues” on page 98
- “Creating a JMS queue” on page 99
- “Creating the JMS target” on page 99
- “Creating the JMS gateway” on page 100

Creating and configuring buses

Using the WebSphere Administrative Console:

Note: The name of the queue manager should be substituted where <queue_manager_name> appears in the following steps.
1. Create and name a bus. For example, SIBUS.
2. Add bus members to the bus.
3. Create a foreign bus using the following information:

   Foreign bus properties:
   - **Name**: FB.<queue_manager_name>

- **Description**: Foreign bus needed to send messages to queue manager <queue_manager_name>

  Routing definition type:
- **Routing type**: Direct WebSphere MQ Link

  Routing definition properties:
- **Inbound userid**:
- **Outbound userid**:

4. Click **Finish**.

Setting up MQ Link and MQ queues

The procedures in this section describe how to configure MQ Link to create a bridge between the ME and WebSphere MQ. It also provides steps to use the JMS implementation of WebSphere MQ to set up the JMS environment.

Configuring MQ Link

To configure MQ Link, perform the following steps using the WebSphere Administrative Console:

1. Click on the bus that was created in the previous section. For example, SIBUS.
2. Click on **Messaging engines** under Additional Properties.
3. Click on the bus shown in the **Name** column.
4. Click on **WebSphere MQ Links** under Additional Properties.
5. Click on **NEW** to create the MQ Link.
6. Populate the fields using the following information:
   - **Name**: TO.QM01
   - **Description**: Link to send and receive data to and from queue manager.
   - **Foreign bus name**: Select the bus created earlier from the drop-down list. For example, FB.SIBUS
   - **Queue manager name**: SIBUS
   - **Batch size**: 50 (default)
   - **Maximum message size**: 4194304 (default)
   - **Heartbeat interval**: 300 (default)
   - **Sequence wrap**: 999999999 (default)
   - **Nonpersistant message speed**: Fast (default)
   - **Adoptable**: Ticked (default)
   - **Initial state**: Started (default)
7. Click **Next**.
8. Populate the sender channel fields using the following information:
   - **Sender channel name**: BUS1.TO.QM01
   - **Host name**: Name/IP of the machine where WebSphere MQ is running
   - **Port**: 1414 (default)
   - **Disconnect interval**: 900 (default)
   - **Short retry count**: 10 (default)
   - **Short retry interval**: 60 (default)
   - **Long retry count**: 999999999 (default)
   - **Long retry interval**: 1200 (default)
   - **Initial state**: Started (default)
9. Click **Next**.

10. Populate the receiver channel fields using the following information:
    - **Receiver channel name**: QM01.TO.BUS1
    - **Inbound nonpersistant message reliability**: Reliable (default)
    - **Inbound persistent message reliability**: Assured (default)
    - **Initial state**: Started (default)
11. Click **Next**.
12. Click **Finish**
13. Save the configuration by clicking **Save** in the Messages window that appears. This message is to confirm that you want to apply the changes to the master configuration.
14. Click **Save** again when asked to update the master repository with your changes.

Creating a JMS Queue Connection Factory

Create a JMS Queue Connection Factory. Refer to "Creating a JMS Queue Connection Factory" on page 91 for instructions.

Creating WebSphere MQ queues and the channel

In this section, you use WebSphere MQ to create the queues you will use to send and receive documents and the channel for this communication. The name of the queue manager should be substituted where <queue_manager_name> appears in the following steps.

1. Open a command prompt.
2. Create the queue manager by entering the following:
   ```
   crtmqm -q <queue_manager_name>
   ```
   For example:
   ```
   crtmqm -q QM01
   ```
3. Start the queue manager with the following command:
   ```
   strmqm <queue_manager_name>
   ```
4. Start the command server with the following command:
   ```
   strmqcsv <queue_manager_name>
   ```
5. Start the listener with the following command:
   ```
   runmqlsr -t tcp -p <port_number> -m <queue_manager_name>
   ```
6. Enter the following command to start the WebSphere MQ command environment:
   ```
   runmqsc <queue_manager_name>
   ```
7. Define a server connection channel, named java.channel, with the following command:
   ```
   DEFINE CHL(java.channel)
   CHLTYPE(SVRCONN)
   ```
8. Define the Sender channel by running the following commands:
   ```
   DEFINE CHL(QM01.TO.BUS1)
   CHLTYPE(SDR)
   TRPTYPE(TCP)
   CONNAME('hostname/hostIP(5558)'
   XMITQ(SIBUS)
   ```
   The above defines a TCP sender channel called QM01.TO.BUS1 with a transmission queue called SIBUS to WebSphere MQ.
9. Define the Receiver Channel by running the following commands:

```
DEFINE CHL(BUS1.TO.QM01)
CHLTYPE(RCVR)
TRPTYPE(TCP)
```

The above defines a TCP receiver channel called BUS1.TO.QM01.

10. Define the transmission queue by running the following commands:

    ```
    DEFINE QL(BUS1)
    USAGE(XMITQ)
    ```

    The above defines a TCP receiver channel called BUS1.TO.QM01.
11. Define a local queue, for example, SIBUS by typing the following command:

    ```
    DEFINE QL(SIBUS)
    ```

    Leave all other channel properties set to the default.
12. Enter the following command to exit the WebSphere MQ command environment:

    ```
    end
    ```

Modifying the default JMS configuration

In this section, you will update the JMSAdmin.config file, which is part of the WebSphere MQ installation, to change the context factory and provider URL. This enables WebSphere Partner Gateway's JMS Receiver and Sender to access files on the WebSphere MQ queue.

1. Navigate to the Java\bin directory of WebSphere MQ. For example, in a Windows installation, you would navigate to: C:\IBM\MQ\Java\bin
2. Open the JMSAdmin.config file using a plain text editor, such as Notepad or vi.
3. Add the # character to the front of the following lines:

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
   PROVIDER_URL=ldap://polaris/o=ibm,c=us
   ```
4. Remove the # character from the front of the following lines:

   ```
   #INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
   #PROVIDER_URL=file:/C:/JNDI-Directory
   ```
5. Change the PROVIDER_URL=file:/C:/JNDI-Directory line if necessary. For example, if you set up a C:/temp/JMS directory, the line would look like this:

   ```
   PROVIDER_URL=file:/C:/temp/JMS
   ```
6. Save the file.
7. Open a command window and go to the <WebSphere MQ install dir>/java/bin directory and execute the JMSAdmin.bat file.
8. Enter the following lines at the command prompt:

   ```
   def ctx(context name)  // creates a context
   chg ctx(context name)  // changes the context to context provided.
   def qcf(queue connection factory name) qmgr(qmgr name) tran(client) chan
   (channel name of server connection type) host(IP of the WMQ)
   port(port no. on which Listener is running);
   def q(queue alias name) qmgr(queue_manager_name) queue(queue name where
   receiver will be picking files from)
   end // to exit
   ```

   For example:

   ```
   def ctx(WPGJMS)
   chg ctx(WPGJMS)
   def qcf(WPGHub) qmgr(QM01) tran(client) chan(java.channel) host(vishals)
   port(1414)
   def q(LQ1) qmgr(QM01) queue(LQ1)
   ```

   Based on this example, a bindings file is created at C:/JNDI-Directory/WPGJMS.

This completes the creation of the binding file that will be used for the WebSphere Partner Gateway Receiver to pick up the document from the queue.

To create a binding file that will be used for the WebSphere Partner Gateway Sender, change the context name, so that it is created in a different directory and queue name from where you want the JMS gateway to put the file.

Defining WebSphere Partner Gateway inbound queues

The following steps must be completed to enable WebSphere Application Server to send messages to the WebSphere MQ queues.

Using the WebSphere Administrative Console:

1. Define an alias destination on WebSphere Application Server by selecting the **Alias** radio button for the destination type and click **Next**.
2. Populate the fields using the following information:
   - **Identifier**: AQ1
   - **Description**: Alias for target queue LQ1
   - **Bus**: leave this blank to let it default
   - **Target Identifier**: LQ1@QM01
   - **Target Bus**: FB.QM01
   - **Reliability**: Inherit - default
   - **Maximum Reliability**: Inherit - default
   - **Producer can override quality of service**: Inherit - default
   - **Send allowed**: Inherit - default
   - **Receive allowed**: Inherit - default
   - **Default priority**: -1 - default
   - **Reply destination**: leave this blank to let it default
   - **Reply destination bus**: leave this blank to let it default
   - **Default Forward Routing Path**: leave this blank to let it default

   Leave all other channel properties set to the default.
3. Create a JMS queue. Refer to “Creating a JMS queue” on page 92. Replace the Bus Name with Foreign Bus Name and Queue name with the Alias Queue Name created in step 2 above.

   In this example, the Bus Name is FB.QM01 and the Queue name is LQ1@QM01, which is the Target Identifer for Queue AQ1 created above in step 2 above.
4. Define a local queue on WebSphere MQ by typing the following command:
   ```
   DEFINE QL(LQ1)
   ```
   Leave all other channel properties set to the default.

   An application connected to the WebSphere Application Server sends messages to the alias destination named AQ1. These messages will then be routed to LQ1 on the target queue manager (QM01).

Defining WebSphere Partner Gateway outbound queues

The following steps must be completed to enable a JMS gateway on WebSphere Partner Gateway to send messages to the WebSphere MQ queues. These messages will then be sent to the queue on the WebSphere Application Server.

1. Define a remote queue on WebSphere MQ. For example:

```
DEFINE
QR(RQ1)
RNAME(Q1)
RQMNAME(SIBUS)
XMITQ(SIBUS)
```

Leave all other channel properties set to the default.

The example defines a remote queue definition called RQ1. This definition specifies that the remote queue name is Q1 and the remote queue manager name is SIBUS. The transmission queue associated with the remote queue definition is SIBUS.

2. Define a queue destination on the WebSphere Application Server:

   Using the WebSphere Administrative Console:

   a. Click on **System Integration** in the left panel.
   b. Click on **Buses** < **SIBUS** (or the name of the bus created in the previous section).
   c. In the Additional Properties pane, click **Destinations**. Click **New**.
   d. Select the **Queue** radio button for the destination type and click **Next**.
   e. Enter the following:

      ```
      Identifier (Q1)
      Description: (Local queue to receive messages from a WebSphere MQ network)
      Reliability: Assured persistent (default)
      ```

      An application connected to queue manager QM01 would put messages to remote queue RQ1. These messages would be routed to Q1 on the target messaging engine.

Creating a JMS queue

Create a JMS queue. Refer to “Creating a JMS queue” on page 92. In this scenario, the Bus Name is SIBUS and the Queue name is Q1.

Creating the JMS target

A JMS target polls a JMS queue (according to the schedule you specify) to look for new documents.

Using the WebSphere Partner Gateway Community Console, create a JMS target:

1. Click **Hub Admin** > **Hub Configuration** > **Targets** to display the Targets List page.
2. From the Target List page, click **Create Target**.
3. Type a name for the target. For example, JMSTarget1. This is a required field. The name you enter here will be displayed on the Targets list.
4. Select **JMS** from the Transport list.
5. Enter the JMS provider URL. This should match the value you entered (the file system path to the bindings file).
6. Enter the user ID and password required to access the JMS queue, if a user ID and password are required.
7. Enter a value for JMS queue name. This is a required field. This name should match the one you specified with the define q command when you created the bindings file.
8. Enter a value for the JMS factory name. This is a required field. This name should match the one you specified with the define qcf command when you created the bindings file.

9. Enter the JNDI factory name. If you do not enter anything, the value com.sun.jndi.fscontext.RefFSContextFactory is used. This is a required field.
10. Click **Save** .

Creating the JMS gateway

A JMS gateway is the gateway to your back-end system.

Using the WebSphere Partner Gateway Community Console, create a JMS gateway:

1. 1. Click **Account Admin** > **Profiles** > **Community Participant**.
2. Enter search criteria and click **Search**, or click **Search** without entering any search criteria to display a list of all participants.
3. Click the View details icon to display the participant's profile.
4. Click **Gateways**.
5. Click **Create**.
6. Type a name to identify the gateway. This is a required field.
7. Select **JMS** from the Transport list.
8. In the **Address** field, enter the URI where the document will be delivered. This field is required. For WebSphere MQ JMS, the format of the target URI is as follows:

   ```
   file:///<user_defined_MQ_JNDI_bindings_path>
   ```

   For example:

   ```
   file:///opt/JNDI-Directory
   ```

   The directory contains the .bindings file for the file-based JNDI. This file indicates to WebSphere Partner Gateway how to route the document to its intended destination.
9. In the **JMS Factory Name** field, enter the name of the Java class the JMS provider uses to connect to the JMS queue. This field is required. This name should match the one you specified with the define qcf command when you created the bindings file.
10. In the **JMS Queue Name** field, enter the name of the JMS queue where documents are to be sent. This field is required. This name should match the one you specified with the define q command when you created the bindings file.
11. In the **JMS JNDI Factory Name** field, enter the factory name used to connect to the name service. This field is required.
12. Click **Save**.

Chapter 6. Integrating WebSphere Process Server with SOAP/HTTP

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere Process Server using SOAP over the HTTP transport protocol. It includes the following information:

- "How SOAP messages are exchanged over the HTTP protocol"
- "Invoking Web services hosted by WebSphere Process Server" on page 103
- "Invoking Web services hosted by community participants" on page 104

How SOAP messages are exchanged over the HTTP protocol

When a community participant requests a Web service from WebSphere Process Server, or when WebSphere Process Server requests a Web service from a community participant, the associated SOAP message is sent to a target on WebSphere Partner Gateway. WebSphere Partner Gateway acts as a proxy, passing the request through to the URL where the service is located.

How Web services hosted by WebSphere Process Server are invoked

When a community participant invokes a Web Service hosted on WebSphere Process Server, the following flow occurs:

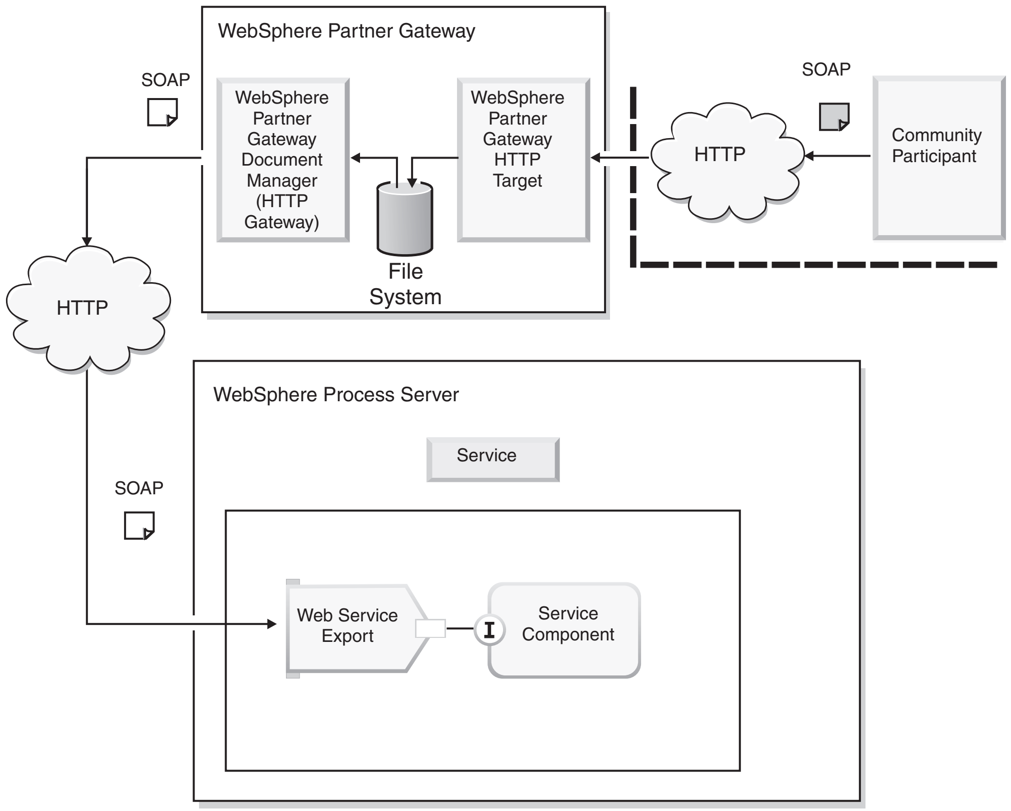
WebSphere Partner Gateway
SOAP
WebSphere Partner Gateway Document Manager (HTTP Gateway)
WebSphere Partner Gateway HTTP Target
File System
SOAP
HTTP
Community Participant
HTTP
WebSphere Process Server
Service
SOAP
Web Service Export
I
Service Component

Figure 27. Community participant invoking a Web service

WebSphere Partner Gateway routes the Web service request to WebSphere Process Server, where the Web Service Export binding is used to invoke the service.

You provide your community participant with the public WSDL, which has the URL of WebSphere Partner Gateway specified as the end point. See "Configuring WebSphere Process Server" on page 105 for information on specifying this URL. WebSphere Partner Gateway acts as a proxy. It receives a SOAP message from the participant and figures out the corresponding private Web service. It then invokes the private Web service on WebSphere Process Server using the same SOAP message. The HTTP response returned by the WebSphere Process Server (both the transport-level response and the Web service response) is then returned to the participant

How Web services hosted by participants are invoked

When WebSphere Process Server invokes a Web service from a community participant, the following flow occurs:

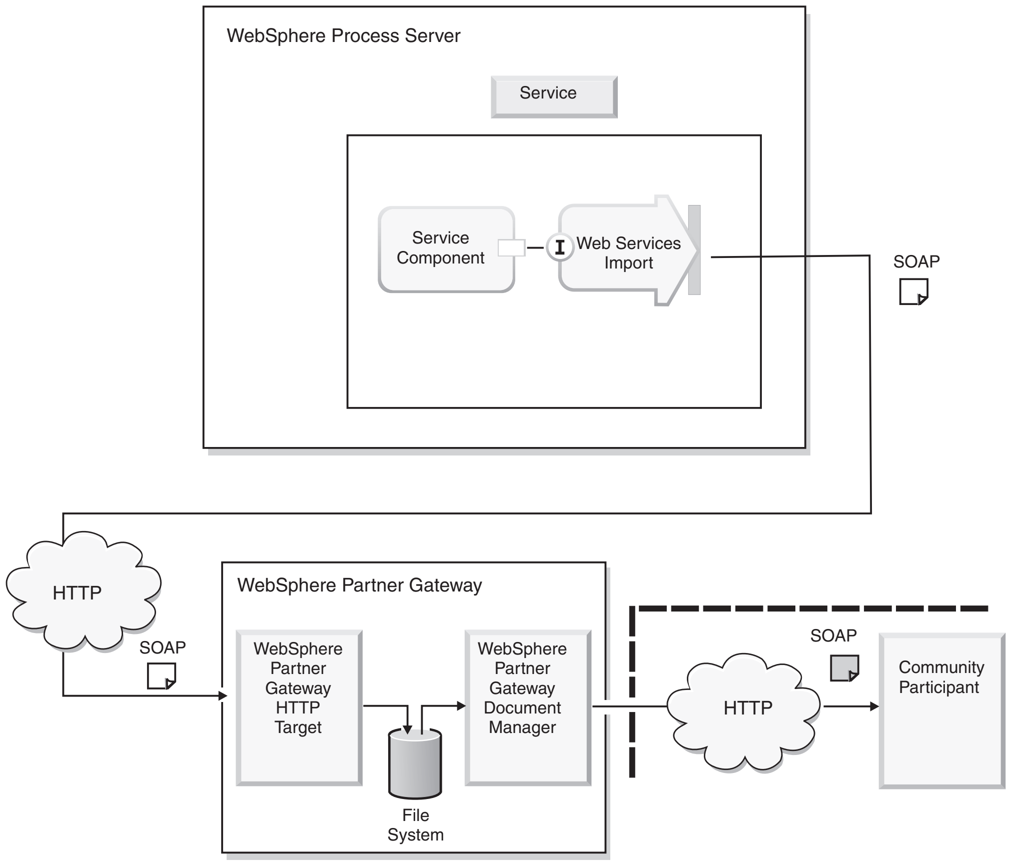

Figure 28. Service on WebSphere Process Server invoking a Web service

WebSphere Process Server uses its Web Services Import binding to invoke the Web service. WebSphere Partner Gateway routes the Web service request from WebSphere Process Server to the appropriate community participant.

WebSphere Partner Gateway makes the Web service available to WebSphere Process Server at the Web service URL specified when the Web service is uploaded in the Community Console. Additionally the service on WebSphere Process Server must provide the URL parameter to identify "To Partner". Refer to the *Hub Configuration Guide* for more details. WebSphere Partner Gateway acts as a proxy. It receives a SOAP message from WebSphere Process Server and figures out the corresponding Web service and the "To Partner". It then invokes the Web service provided by the community participant using the same SOAP message. The HTTP response returned by the community participant (both the transport-level response and the Web service response) is returned to WebSphere Process Server.

The response received from the community participant is returned to WebSphere Process Server in the same HTTP connection as the request. The behavior is the same for both request-only and request-response Web services.

Invoking Web services hosted by WebSphere Process Server

This section describes how a community participant invokes a Web service hosted on WebSphere Process Server.

A community participant sends the request to an HTTP target on the WebSphere Partner Gateway hub. WebSphere Partner Gateway authenticates the user, looks up the connection between the community participant and Community Manager, and forwards the request to a gateway defined at the Community Manager.

Note: When the community participant sends the SOAP message, the community participant must authenticate itself to WebSphere Partner Gateway. The community participant can use HTTP Basic Authentication, supplying the participant's Business ID, console name, and console password. Alternatively, the participant can present an SSL client certificate that has been previously set up in WebSphere Partner Gateway. See the *Hub Configuration Guide* for more information on certificates.

Configuring WebSphere Partner Gateway

You configure WebSphere Partner Gateway for Web services as described in the *Hub Configuration Guide*. Make note of the following as you configure WebSphere Partner Gateway:

1. Add the com.ibm.bcg.server.sync.SoapSyncHdlr handler to the syncCheck configuration point of the HTTP target (if you will be handling two-way requests).
2. From WebSphere Process Server, obtain the WSDL generated by the Web Service Export binding of your component.
3. Make sure the HTTP gateway of the Community Manager indicates the URL to which WebSphere Partner Gateway will send messages to WebSphere Process Server. This gateway points to the EndPointURL specified in the WSDL you uploaded in step 2.

Configuring WebSphere Process Server

Set up components on WebSphere Process Server per the documentation in the WebSphere Process Server information center. When you create an export for the component, be sure to select **Web Service Binding**.

Invoking Web services hosted by community participants

When a service on WebSphere Process Server makes a SOAP request to a URL on WebSphere Partner Gateway, WebSphere Partner Gateway identifies the participant capable of processing the SOAP request. It uses the basic authentication supplied by WebSphere Process Server against its partner profile. If the SOAP request is two-way, WebSphere Partner Gateway gets the response from the community participant and sends the response back to WebSphere Process Server.

Configuring WebSphere Partner Gateway

You configure WebSphere Partner Gateway for Web services as described in the *Hub Configuration Guide*. Make note of the following as you configure WebSphere Partner Gateway:

1. Add the com.ibm.bcg.server.sync.SoapSyncHdlr handler to the syncCheck configuration point of the HTTP target (if you will be handling two-way requests).
2. Make sure the HTTP gateway of the community participant indicates the URL to which WebSphere Partner Gateway will send messages. This gateway points to the EndPointURL specified in the WSDL you upload as part of the configuration process.

Configuring WebSphere Process Server

Set up components on WebSphere Process Server per the documentation in the WebSphere Process Server information center. Note that you must obtain the WSDL file that describes the Web service from the provider of the service. You import the WSDL file into the module for the components.

WebSphere Partner Gateway makes the community participant-provided Web services available at its URL. Therefore, WebSphere Process Server must invoke the service at the URL of the WebSphere Partner Gateway HTTP target. WebSphere Partner Gateway also requires the Community Manager to provide a user name and password as part of HTTP basic authentication. See the *Hub Configuration Guide* for information on the user name and password. To enable WebSphere Process Server to provide HTTP basic authentication and also to change the end point of the Web service that will be invoked by WebSphere Process Server, follow these steps:

To change the end-point address:

1. Log in to the WebSphere Process Server administrative console
2. Navigate to the Web service client bindings of the service you are invoking from WebSphere Partner Gateway. You can locate it by following this path: **Enterprise Applications >** <*your_application*> **> EJB Modules >** <*your_JAR_file*> **> Web service client bindings**
3. Specify the **Overridden Endpoint URL** for the service port you want to invoke from WebSphere Partner Gateway. Specify the URL of the WebSphere Partner Gateway HTTP target that will receive the Web service request from WebSphere Process Server. The endpoint URL takes the form:

   ```
   <IP_address:port_number>bcgreceiver/<target_name>
   ```
4. Click **Apply**, and then click **OK**.

To specify the user name and password required for basic authentication, perform the following steps:

1. Log in to the WebSphere Process Server administrative console
2. Navigate to the Web service client bindings of the service you are invoking from WebSphere Partner Gateway. You can locate it by following this path: **Enterprise Applications >** <*your_application*> **> EJB Modules >** <*your_JAR_files*> **> Web services:Client security bindings > HTTP basic authentication**
3. Specify the Basic authentication ID and Basic authentication password.

   The authentication ID takes the form:

   ```
   BusinessID/username
   ```

   For example, if the business ID of the Community Manager is 987654321, and the user name of the Community Manager is admin, the authentication ID would be:

   ```
   987654321/admin
   ```
4. Click **Apply** and then click **OK**.

Chapter 7. Integrating WebSphere Process Server with File-system as transport

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere Process Server using the File-system protocol.

The File-system protocol enables WebSphere Partner Gateway to send messages by placing them in a defined directory structure. WebSphere Partner Gateway receives messages by reading them from the directory structure. The file-system protocol supports the None packaging type.

This chapter provides the following information:
- "Sending documents using the File-system protocol"
- "Receiving documents using the File-system protocol"
- "Setting up the WebSphere Process Server environment"
- "Setting up the WebSphere Partner Gateway environment" on page 109

Sending documents using the File-system protocol

WebSphere Process Server components leverage the Service Component Architecture (SCA) J2C import binding to send business data to the WebSphere Adapter for Flat Files. The adapter writes the business data to a directory in the File-system. WebSphere Partner Gateway's File-system receiver target reads this business document from the directory on the File-system and routes it to the appropriate trading partner.

Receiving documents using the File-system protocol

WebSphere Partner Gateway receives a business document from the trading partner. WebSphere Partner Gateway's Document Manager writes the document to the File-system directory. The WebSphere Adapter for Flat Files polls events from this directory and reads the business document. It then invokes the Service Component Architecture (SCA) service export that is expecting events form the adapter. The SCA service receives the business document.

Note: The WebSphere Adapter for Flat Files expects a string or raw data that it can read from or write to a file. Therefore, the WebSphere Process Server environment must serialize the business object into either a string or raw data. WebSphere Process Server services must create a component that will serialize a business object into a business document and vice versa.

Setting up the WebSphere Process Server environment

This section provides the steps for setting up the WebSphere Process Server environment for the File-system transport on WebSphere Application Server. This includes creating and configuring the WebSphere Process Server artifacts.

This section contains the following information:
- "Deploying and configuring the WebSphere Adapter for Flat Files" on page 108
- "Creating SCA components" on page 109

Deploying and configuring the WebSphere Adapter for Flat Files

For File-system based integration WebSphere Process Server leverages the WebSphere Adapter for Flat Files, a bi-directional adapter which can read and write business data from the File-system.

For inbound communication, the adapter supports the Read function.

For outbound communication, the adapter supports the following functions:
- Create
- Append
- Delete
- Overwrite
- Retrieve
- List
- Exists

The WebSphere Adapter for Flat Files installation provides a Resource Adapter Archive (RAR) file that is deployable on the WebSphere Process Server. The RAR file contains the files that are shipped with the adapter. Before you deploy the WebSphere Adapter for Flat Files, you must import the adapter RAR file and create the project in WebSphere Integration Developer.

The following sections contain an overview of the steps that need to be completed to deploy and configure the WebSphere Adapter for Flat Files. Please refer to the WebSphere Integration Developer documentation for detailed information.

Importing the RAR file:
1. Switch to the J2EE perspective and import the RAR file.
2. Specify the location from where you will import the RAR file (the same location where you copied your adapter file during installation), and specify a project name.

   This creates a new J2EE Connector project in the workspace.

Adding external dependencies to the project:

Copy the external dependencies into the connectorModule in your WebSphere Integration Developer project and add them to the project as internal libraries. This is necessary to bundle the dependencies into the EAR file, which will be exported. For details, refer to the WebSphere Integration Developer documentation.

Configuring the adapter:

The configuration process is done using the Enterprise Service Discovery wizard in the WebSphere Integration Developer. This process allows you to enter all the information necessary to configure the adapter for the first time. The output from the Enterprise Service Discovery wizard is saved to a business integration module, which contains the business objects, the import file (which describes outbound processing, as defined by the ActivationSpec), the export file (which describes inbound event processing, as defined by the InteractionSpec), and the Web Services Description Language (WSDL) file. This results in a component in your module which you can use to read and write from file system.

Creating SCA components

If you want your SCA component to receive business documents from WebSphere Partner Gateway the File-system protocol, you can leverage the Enterprise Service Discovery tool to generate the required import bindings for the flat file adapter. To generate the required import bindings, you must specify the service type as Inbound in the Enterprise Service Discovery wizard. The method name is READ for inbound. Wire your SCA component export to the import binding generated by the Enterprise Service discovery tool.

If you want your SCA component to send business documents to WebSphere Partner Gateway the File-system protocol, you can leverage the Enterprise Service Discovery tool to generate the required export binding for the flat file adaper. To generate the required export binding, you must specify the service type as Outbound in the Enterprise Service Discovery wizard. Wire the export binding generated by the Enterprise Service discovery tool to the import binding of your SCA component.

Setting up the WebSphere Partner Gateway environment

This section provides the steps for setting up the WebSphere Partner Gateway environment for the File-system transport on WebSphere Application Server.

Using the WebSphere Partner Gateway Community Console:

1. Create a File-system target on the hub to receive documents sent to the hub from WebSphere Process Server or from community participants.
2. Create a Community Manager profile (if one does not already exist), including a File-system gateway that WebSphere Partner Gateway will use to send documents to WebSphere Process Server.
3. Create community participant profiles, including File-system gateways that WebSphere Partner Gateway will use to send documents to the participants.
4. Import any WSDL files, transformation maps, RosettaNet packages, or other document definition mechanisms so that a document definition for the type of document you are exchanging appears on the Document Flow Definition page of the WebSphere Partner Gateway Community Console.
5. Create interactions between the types of document the hub will receive (from WebSphere Process Server or from a community participant) and the types of document the hub will send (to WebSphere Process Server or to the community participants).
6. Create B2B capabilities in the profiles of the Community Manager and participants to indicate the types of documents they are able to send and receive.
7. Create interactions between the types of document the hub will receive (from WebSphere Process Server or from a community participant) and the types of document the hub will send (to WebSphere Process Server or to the community participants).
8. Create participant connections between the Community Manager and participants to indicate the source participant (the sender of the document), the target participant (the recipient of the document), and the action that the hub should take (if any) to transform the document.

Part 3. Integrating with WebSphere InterChange Server

Chapter 8. Introduction to InterChange Server integration

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere InterChange Server.

Notes:

1. For a description of the general process used to integrate WebSphere Partner Gateway with a back-end system, see Chapter 2, "Planning for back-end integration," on page 9.
2. This chapter assumes that you are familiar with WebSphere InterChange Server and associated components, such as collaborations, business objects, and adapters.

Often integration of WebSphere Partner Gateway with a back-end system is done by two separate people or roles. Each role configures a particular component, for which that role has expertise. Therefore, this chapter separates the integration with WebSphere InterChange Server into the configuration of WebSphere Partner Gateway and the configuration of InterChange Server. Table 31 lists these configuration roles along with the places in this chapter to obtain the associated configuration information.

Table 31. Roles for InterChange Server integration

Configuration role	For more information
Configuration of WebSphere Partner Gateway	1. "Planning for integration with InterChange Server." 2. "Configuring WebSphere Partner Gateway for InterChange Server" on page 116.
Configuration of WebSphere InterChange Server	1. "Planning for integration with InterChange Server." 2. "Configuring InterChange Server" on page 119.

Note: While each of these configuration roles can be performed separately, each also requires common information so that the two components can communicate.

This chapter provides the following information:

- "Planning for integration with InterChange Server"
- "Configuring WebSphere Partner Gateway for InterChange Server" on page 116
- "Configuring InterChange Server" on page 119
- "Handling documents with attachments" on page 123

Planning for integration with InterChange Server

To plan for your integration to WebSphere InterChange Server, follow the steps outlined in Chapter 2, "Planning for back-end integration," on page 9. Table 32 summarizes the integration steps to integrate WebSphere Partner Gateway with InterChange Server (ICS).

Table 32. Planning for integration with WebSphere InterChange Server

Integration step	**For more information**
1. Confirm that you have a supported version of WebSphere InterChange Server installed and available to WebSphere Partner Gateway.	Chapter 3: "InterChange Server versions that WebSphere Partner Gateway supports"
2. Determine the business protocol of the WebSphere Partner Gateway document.	Chapter 2: "Which business protocol are you using?" on page 9
3. Determine the packaging type for the document: None or Backend Integration.	Chapter 2: "Which packaging will you use?" on page 19
4. Determine the transport protocol to use between WebSphere Partner Gateway and WebSphere InterChange Server.	Chapter 3: "Message transports that InterChange Server supports"
5. Configure WebSphere Partner Gateway.	Chapter 3: "Configuring WebSphere Partner Gateway for InterChange Server" on page 116
6. Configure WebSphere InterChange Server components for use over the chosen transport protocol.	Chapter 3: "Configuring InterChange Server" on page 119

InterChange Server versions that WebSphere Partner Gateway supports

Version 6.0 of WebSphere Partner Gateway supports integration with the following versions of InterChange Server:

- 4.2.2
- 4.3.0

InterChange Server is available on several platforms, including Windows 2000 and several UNIX-based platforms. For more information, consult your installation guide for InterChange Server in the WebSphere InterChange Server documentation set.

Message transports that InterChange Server supports

When WebSphere Partner Gateway sends your message to InterChange Server over a particular message transport protocol, it sends that message to the appropriate adapter, which understands the particular transport protocol and routes the message to InterChange Server. Similarly when InterChange Server sends a message to WebSphere Partner Gateway, it sends the message to the appropriate adapter for routing to WebSphere Partner Gateway over the appropriate transport protocol.

The adapter works with a data handler, which handles the actual conversion from serialized data to business object, or from business object to serialized data. The data handler appropriate for the payload type is used to perform these conversions.

The following two message-transport protocols are supported by InterChange Server:

- HTTP transport protocol

Note: The exchange of Web Services over HTTP is handled in a separate section because Web Services are exchanged in a manner that is different from other documents transmitted over HTTP. See "Sending SOAP documents over HTTP/S" on page 154.

- JMS transport protocol

Note: InterChange Server provides other types of integration options, such as file-based integration. Refer to the WebSphere InterChange Server documentation for details on enabling the exchange of documents through file-based integration.

Use the transport protocol that best suits the needs of your business. Consider the following:

- First and foremost, determine that the transport protocol you are using between the community participant and WebSphere Partner Gateway is available with the integration mechanism used. See "Which message transport will you use?" on page 28.
- Sending SOAP documents to and receiving SOAP documents from the WebSphere InterChange Server requires use of the HTTP transport protocol. For more information, see "Sending SOAP documents over HTTP/S" on page 154.

HTTP

Which adapters are required to send and receive documents between WebSphere Partner Gateway and InterChange Server over HTTP depends on the following:

- The type of document you are sending
- The version of InterChange Server with which you are integrating.

Note: All references to the HTTP transport protocol apply to HTTPS as well.

Table 33 summarizes where to find information on how to configure adapters for use with InterChange Server.

Table 33. Configuring for HTTP transport with InterChange Server

Condition	For more information
If you are transferring non-SOAP documents with InterChange Server	"Using HTTP transport protocol with ICS" on page 141
If you are sending SOAP documents	"Sending SOAP documents over HTTP/S" on page 154

JMS

The components required to send and receive documents between WebSphere Partner Gateway and InterChange Server over JMS are summarized in Table 59 on page 158. Basically, support for JMS involves the use of the WebSphere Business Integration Adapter for JMS. The Adapter for JMS invokes collaborations within InterChange Server asynchronously.

Benefits of using each transport

As you decide which message transport to use with InterChange Server, consider the following benefits of each transport:

- The Adapter for HTTP supports synchronous communication. If you require synchronous transactions, you *must* use the HTTP transport protocol.
- The Adapter for JMS can provide "guaranteed event delivery" from WebSphere Partner Gateway to the WebSphere InterChange Server.

  Guaranteed event delivery ensures that events are never lost or sent twice.

Support for InterChange Server integration

WebSphere Partner Gateway provides samples to assist you in the integration process with InterChange Server. These samples reside in the following subdirectory of the WebSphere Partner Gateway product directory:

`Integration/WBI/WICS/samples`

Table 34 lists the subdirectories of the `samples` directory for the different transport protocols that InterChange Server supports.

Table 34. Samples for InterChange Server integration

Transport protocol	**InterChange Server version**	**Samples subdirectory**
HTTP	All supported versions	General samples: `HTTP` RosettaNet-specific samples: `RosettaNet/HTTP`
JMS	All supported versions	General samples: `JMS` RosettaNet-specific samples: `RosettaNet/JMS`

Configuring WebSphere Partner Gateway for InterChange Server

A general overview of how to configure WebSphere Partner Gateway to communicate with a back-end system is provided in "Configuring WebSphere Partner Gateway" on page 37. This section summarizes the steps needed to configure WebSphere Partner Gateway to communicate with InterChange Server.

Configuration of WebSphere Partner Gateway involves the following steps:

- Configuring for support of outgoing documents

  For information on sending documents from WebSphere Partner Gateway to InterChange Server, see "Providing support for outgoing documents."
- Configuring for incoming documents

  For information on sending documents from InterChange Server to WebSphere Partner Gateway, see "Providing support for incoming documents" on page 117.

Providing support for outgoing documents

For WebSphere Partner Gateway to send documents to any back-end system, you must perform the steps described in "Defining where to send the participant document" on page 39. When your back-end system is InterChange Server, you need to create a gateway whose transport type matches the transport protocol used for messages between WebSphere Partner Gateway and InterChange Server. When the hub sends a document to InterChange Server, it must know where to route the document. This location must conform with the transport protocol being used. The transport protocol must be one that InterChange Server supports (see "Message transports that InterChange Server supports" on page 114).

The following sections summarize how to create gateways for following transport protocols, which InterChange Server supports:

- "Configuration for sending documents to ICS over the HTTP transport protocol" on page 117
- "Configuration for sending documents to ICS over the JMS transport protocol" on page 117

Configuration for sending documents to ICS over the HTTP transport protocol

When the hub sends a document to InterChange Server over the HTTP protocol, the hub routes the message through the defined gateway. This gateway identifies the URL where the document can be received by InterChange Server. When InterChange Server uses the HTTP protocol, an adapter receives the document at the appropriate URL, where it can then send it to InterChange Server.

For the hub to be able to send documents through a gateway over the HTTP transport protocol, you must create a gateway from the Gateway Details page of the Community Console. This gateway must be configured to use the HTTP 1.1. transport protocol and to write to the URL on which the appropriate adapter is listening.

Note: An overview of how to create a gateway is provided in "Defining where to send the participant document" on page 39.

Configuration for sending documents to ICS over the JMS transport protocol

When the hub sends documents to InterChange Server over the JMS protocol, the hub routes the document to the appropriate JMS queue, where it can be retrieved by InterChange Server. For the hub to obtain this JMS location, you must create a gateway in WebSphere Partner Gateway, one that uses the JMS transport protocol. This gateway must be configured to write to the queue on which the Adapter for JMS polls.

Note: For an overview of how to create a gateway, see "Defining where to send the participant document" on page 39.

For the hub to be able to send documents through a gateway over the JMS transport protocol, create a gateway from the Gateway Details page of the Community Console. If you are using WebSphere MQ, version 5.3 as your JMS provider, refer to the *Hub Configuration Guide* for detailed steps. In addition, provide the information specified in Table 35 for the JMS protocol in the Gateway Details page.

Table 35. JMS values for the Gateway Details page for communication with ICS

Gateway Details field	Value	Notes and restrictions
JMS Message Class	`TextMessage`, `BytesMessage`, or `StreamMessage`	
JMS Queue Name	Same JMS queue name as the input queue for the Adapter for JMS	This queue must be included in the list of input queues of the Adapter for JMS; that is, the adapter must poll this queue for incoming events. For more information, see "Identifying the JMS queues" on page 164.

Providing support for incoming documents

For WebSphere Partner Gateway to receive messages from any back-end system, you must perform the steps described in "Defining where to retrieve the back-end document" on page 43. When your back-end system is InterChange Server, you need to take the following steps in your hub:

1. As part of your participant profile for the Community Manager, define the gateway type and provide the associated IP address on which the Receiver will listen.
2. Create a target whose transport type matches the transport protocol used for documents between WebSphere Partner Gateway and InterChange Server.

   For the hub to receive a document from InterChange Server, it must know the location at which to retrieve the messages. This location must conform with the transport protocol to be used. The transport protocol must be one that InterChange Server supports (see “Message transports that InterChange Server supports” on page 114).

The following sections summarize how to create targets for transport protocols that InterChange Server supports.

Configuration for receiving documents from ICS over the HTTP transport protocol

When the hub receives a document over the HTTP transport protocol, its Receiver retrieves the document from the defined target. This target identifies the URL at which the Receiver listens for documents from InterChange Server. When InterChange Server uses the HTTP transport protocol, an adapter sends the document to the appropriate URL, where it can be received by the hub.

For the hub to receive documents through a target over the HTTP transport protocol, you must create a target from the Target List page of the Community Console. This target must use the HTTP 1.1 transport protocol. The hub determines this URL as a combination of the following information:

- The IP address of the host computer, obtained from within the Community Manager’s participant profile
- The target URL, obtained from the URL field of the target definition

Note: An overview of how to create a target is provided in “Defining where to retrieve the back-end document” on page 43.

For InterChange Server to be able to send documents to this target, its adapter must be configured to send documents to this URL. Therefore, you must ensure that this target URL is available for the InterChange Server configuration.

Configuration for receiving documents from ICS over the JMS transport protocol

When the hub receives documents from InterChange Server over the JMS protocol, the hub obtains the document from the appropriate JMS queue, where InterChange Server has sent it. For the hub to be able to obtain this JMS location, you must create a target in WebSphere Partner Gateway, one that uses the JMS transport protocol. Through the target, the hub listens for any documents on its input queue and retrieves them.

Note: For an overview of how to create a target, see “Defining where to retrieve the back-end document” on page 43.

For the hub to receive documents through a target over the JMS transport, you must create a target from the Target List page of the Community Console. If you are using WebSphere MQ, version 5.3 as your JMS provider, refer to the *Hub Configuration Guide* for the detailed steps. In addition, provide the information specified in Table 36 for the JMS protocol in the Target Details page.

Table 36. JMS values for the Target Details page for communication with ICS

Target Details field	Value	Notes and restrictions
JMS Message Class	TextMessage	
JMS Queue Name	Same JMS queue name as the output queue for the Adapter for JMS	This queue must be listed as the output queue of the Adapter for JMS; that is, the adapter must send documents to this queue. For more information, see “Identifying the JMS queues” on page 164.

Configuring InterChange Server

For your interactions between WebSphere Partner Gateway and InterChange Server, you must create an Integration Component Library (ICL) within the System Manager tool. This ICL will include the following artifacts:

- Business object definitions
- Connector objects
- Collaboration templates and collaboration objects

You must also create a user product and select from the ICL those artifacts required for your particular interaction between InterChange Server and WebSphere Partner Gateway.

Note: For more information on how to create ICLs and configure InterChange Server, see the *System Implementation Guide* in the WebSphere InterChange Server documentation set.

Creating business object definitions

WebSphere Partner Gateway sends your message to an adapter, which routes the message to InterChange Server in the form of one or more *business objects*. For InterChange Server to recognize a business object, it must first locate a template, called a *business object definition*, to describe the structure of the information in the business object. Each piece of information in a business object definition is held in an *attribute*. Therefore, you must create business object definitions to represent the information in your message. To create business object definitions, use the Business Object Designer tool.

Note: Business Object Designer is included as part of both the WebSphere InterChange Server and the WebSphere Business Integration Adapter products. For more information on the use of this tool, see the *Business Object Development Guide*.

InterChange Server uses business objects for the following information:

- “Business object for the document”
- “Business objects for configuration information” on page 122

Business object for the document

To hold the payload of the WebSphere Partner Gateway document or message, you must define a business object definition to represent the *payload business object*. It is in the form of a payload business object that the adapter transfers the document into (or out of) InterChange Server. This section provides the following information on the payload business object:

- "Business object structure"
- "Business object conversion"
- "InterChange Server terminology" on page 121

Business object structure: The payload business object must be designed so that each piece of information in the document that you want to transfer has an attribute in its associated payload business object definition. As Table 37 shows, the contents of the payload business object depend on the structure of the document and the packaging type that the document uses.

Table 37. Relationship of packaging to the structure of the payload business object

Document structure	Packaging type	Payload business object definition
Payload only	None	Holds the payload information of the document.
Payload only	Backend Integration	Holds: • The payload information of the document • Transport-level headers
Payload and attachments	None	*Not applicable.* You must use Backend Integration packaging if your document contains attachments.
Payload and attachments The document contains an XML wrapper, called a transport envelope, in which both the payload and attachments are wrapped.	Backend Integration	Holds: • The payload information of the document • Transport-level headers • The attachment container, which holds the attachment data and any attachment business objects A WebSphere Partner Gateway-supplied data handler, called the Attachment data handler, is required to process the transport envelope. For more information, see "Handling documents with attachments" on page 123.

The payload business object must also be designed according to the requirements of the particular adapter used for integration with WebSphere Partner Gateway. Table 38 provides information on where to find details of how to create the payload business object for transfer over a particular transport protocol.

Table 38. Creating payload business objects for different transport protocols

Transport protocol	Notes and restrictions	For more information
HTTP		"Creating business object definitions for ICS over HTTP" on page 147
JMS	If document uses Backend Integration packaging	"Creating business object definitions for JMS" on page 165
All	If document has attachments	"Creating attachment-related business object definitions" on page 134

Business object conversion: Usually, the adapter uses a data handler to convert between the format of the document and its business-object representation. This data handler is called the *payload data handler*. The adapter must be configured to call the appropriate data handler for the payload's content type. Usually, the WebSphere Business Integration Data Handler for XML is configured as the payload data handler because it converts between XML messages and business objects. However, you can create custom data handlers for any message formats that do not have a corresponding data handler provided by WebSphere Business Integration Server.

Note: For processing XML messages, make sure you are using the WebSphere Business Integration Data Handler for XML, version 2.3.1 or higher. For cXML messages, you must use the Data Handler for XML, version 2.4.1 or higher.

You must make sure the payload data handlers you are using can ignore the child meta-objects that are required by the transport protocol you are using. Before using a data handler (whether it is supplied by WebSphere Business Integration or whether it is a custom data handler), make sure it provides support for child meta-objects. Refer to the section on the `cw_mo_`*`label`* tag in the business object's application-specific information in the appropriate section for your transport protocol (see Table 38 on page 120).

To indicate which data handler to use to convert the payload, you must take the following steps:

1. Identify the MIME type that the data handler must support to convert the payload and locate a data handler that can handle this MIME type.

   The *Data Handler Guide* in the WebSphere Business Integration Adapter documentation set describes the data handlers that IBM provides. If none of these data handlers can work, you can create a custom data handler.
2. In Business Object Designer, create a child meta-object for the data handler you need to use. If you are using an IBM-provided data handler, refer to the *Data Handler Guide* for information on the structure of the child meta-object.
3. In Business Object Designer, update the top-level data-handler meta-object for connectors to include an attribute for the supported MIME type. The attribute type for this attribute is the data handler's child meta-object.
4. In Connector Configurator, set the appropriate connector configuration properties to identify the data handler to use:
   - Set the `DataHandlerConfigMO` and `DataHandlerMimeType` properties with the name of the top-level data-handler meta-object and the supported MIME type, respectively.
   - Set the `DataHandlerClassName` property with the name of the data-handler class to instantiate.

   Note: You set *either* the `DataHandlerConfigMO` and `DataHandlerMimeType` properties *or* the `DataHandlerClassName` property.
5. In Connector Configurator, include the top-level data-handler meta-object in the list of supported business objects.

InterChange Server terminology: For InterChange Server, the name of the payload business object depends on the direction of the communication, as follows:

- When WebSphere Partner Gateway *sends* a document to InterChange Server, it is participating in InterChange Server's *event notification*.

  In this case, the data business object is called the *event business object* (sometimes called just an event), which notifies InterChange Server of an event that occurred in some community participant.
- When WebSphere Partner Gateway *receives* a document from InterChange Server, it is participating in InterChange Server's *request processing*.

In this case, the data business object is a *request business object*, which InterChange Server has sent to request information from some community participant. In response, InterChange Server might send a *response business object* back to the hub community.

Business objects for configuration information

For many of the adapters, you create business object definitions to hold configuration information. Such business objects are often called *meta-objects*.

Table 39 provides information on where to find details of how to create the data business object for transfer over a particular transport protocol.

Table 39. Sections that describe business-object creation

Transport protocol	Related component	For more information
HTTP	Adapter for HTTP	"Creating HTTP transport-level header information for ICS" on page 151
JMS	Adapter for JMS	"Creating JMS header information" on page 166
All	Attachment data handler	"Creating the Attachment child meta-object" on page 130

Creating the connectors

You must create a connector object for the adapter you will be using. This connector object represents an instance of the adapter at run-time. You create connector objects within InterChange Server's System Manager tool.

Note: For information on how to create connector objects, refer to the *System Implementation Guide* in the WebSphere InterChange Server documentation set.

Table 40 summarizes where to find information about how to create connector objects, based on the transport protocol you are using.

Table 40. Creating connector objects for different transport protocols

Transport protocol	Adapter	For more information
HTTP	Adapter for HTTP	"Creating the HTTP connector object" on page 154
JMS	Adapter for JMS	"Creating the JMS connector object" on page 170

Creating the collaborations

It is the *collaboration*, within InterChange Server, that performs the actual business process you need. Therefore, the appropriate collaboration must exist for InterChange Server to correctly process your WebSphere Partner Gateway documents. Make sure you take the following steps to make the appropriate collaboration available at run-time:

1. Ensure that a collaboration template exists that provides the business process you need:
   - If such a collaboration template does *not* currently exist, you must create one and compile it.

- If a collaboration template does exist, you must understand how to use it sufficiently to be able to configure its collaboration object.

2. Create a collaboration object and bind its ports, as follows:
   - For request processing, set the ″to″ port, which sends requests to WebSphere Partner Gateway, to the adapter.
   - For event notification, set the ″from″ port, which receives events from WebSphere Partner Gateway, to the adapter.

Table 41 summarizes where to find information about how to create connector objects, based on the transport protocol you are using.

Table 41. Collaboration binding for different transport protocols

Transport protocol	**Adapter**	**For more information**
HTTP	Adapter for HTTP	"Binding collaborations to communicate with Adapter for HTTP" on page 154
JMS	Adapter for JMS	"Binding collaborations to communicate with Adapter for JMS" on page 170

Deploying the project

After your user project contains the artifacts that define the run-time components needed, you must deploy it to the InterChange Server repository. You deploy a user project within System Manager.

Handling documents with attachments

WebSphere Partner Gateway provides the Attachment data handler to process documents that are sent between WebSphere Partner Gateway and InterChange Server. The Attachment data handler converts a document within the XML transport envelope (with or without attachments) between its serialized format and its business-object representation. You should configure the Attachment data handler as the payload data handler in either of the following cases:

- The Envelope Flag for Backend Integration packaging has been set to Yes.

  When this flag is set to Yes, WebSphere Partner Gateway always wraps a document in an XML transport envelope, regardless of whether it contains attachments. You set this flag to Yes for Backend Integration packaging as part of the profile's B2B Capabilities page. For more information, see "Payload" on page 25.
- The document to be processed can contain attachments.

  When a document contains attachments, WebSphere Partner Gateway wraps it in an XML transport envelope. In any document flow, there is one payload and, optionally, multiple attachments. If you are sending or receiving documents that contain attachments, your payload business object needs to contain attachment information.

  Note: The Attachment data handler is *not* required for SOAP documents that contain attachments. For information on how SOAP documents are handled, see "Sending SOAP documents over HTTP/S" on page 154.

The Attachment data handler is called from a WebSphere Business Integration adapter.

- If WebSphere Partner Gateway and InterChange Server use the HTTP transport protocol, it is the Adapter for HTTP that calls the Attachment data handler.
- If WebSphere Partner Gateway and InterChange Server use the JMS transport protocol, it is the Adapter for JMS that calls the Attachment data handler.

When the adapter receives a document within an XML transport envelope, it calls the Attachment data handler to convert this document to its appropriate business-object representation. For example, Figure 36 on page 159 shows the Adapter for JMS calling the Attachment data handler to convert the serialized format of the document to its business-object representation. Conversely, when the adapter receives a business-object representation for a document within an XML transport envelope, it calls the Attachment data handler to convert this business-object structure to its appropriate document format. For example, Figure 37 on page 161 shows the Adapter for JMS calling the Attachment data handler to convert the business-object representation of the document to its serialized format.

This section provides the following information on the Attachment data handler:
- "How the Attachment data handler performs the conversion"
- "Setting up the environment for the Attachment data handler" on page 129
- "Configuring the Attachment data handler" on page 130
- "Creating attachment-related business object definitions" on page 134

How the Attachment data handler performs the conversion

The Attachment data handler can interpret the structure of the XML transport envelope and handle the conversion between the contained data and the corresponding business-object representation, as described in the following sections:
- "How documents are converted to business objects" to send a document to InterChange Server
- "How business objects are converted to documents" on page 127 to receive a document from InterChange Server

How documents are converted to business objects

Before WebSphere Partner Gateway sends a document to InterChange Server, it must determine whether to wrap the contents in an XML transport envelope. If WebSphere Partner Gateway creates the transport envelope, the payload and any attachments are Base64-encoded. WebSphere Partner Gateway then sends the XML transport envelope to the appropriate adapter with the appropriate transport-level headers. This adapter can be configured to call the Attachment data handler to handle the conversion of payload and any attachments in an XML-wrapped document into the corresponding business-object representation.

To convert a document wrapped in an XML transport envelope to its business-object representation, the calling entity instantiates the Attachment data handler, passing it the document (in its transport envelope). The Attachment data handler then takes the following steps:

1. Loads the content-type maps defined in the data handler's child meta-object.

   The content-type maps are defined in the `ContentTypeMap_x` configuration properties of the child meta-object. The child meta-object is a business object that contains configuration information for the Attachment data handler.

Attributes in this business object associate content-type maps with content types. For more information, see "Creating the Attachment child meta-object" on page 130.

2. Checks the document to see whether it is wrapped in an XML transport envelope.
   - If the Attachment data handler does *not* detect the transport envelope, it does not need to extract the payload from this envelope structure.

     The document contains only a payload, which the Attachment data handler must convert to its associated business-object representation. For more information, see "How documents without a transport envelope are processed."
   - If the Attachment data handler *does* detect the transport envelope, it must extract the payload and any attachments from this envelope structure.

     The document contains a payload and possibly some attachments. Therefore, the Attachment data handler needs to convert the payload *and* any attachments to their associated business-object representation. For more information, see "How documents in a transport envelope are processed."
3. Sets the resulting payload business object and returns this business object to the calling entity.

How documents without a transport envelope are processed: If the Attachment data handler determines that the document is *not* contained in an XML transport envelope, it does not need to extract the payload data from the envelope structure. Therefore, the data handler uses the `PayloadDataHandlerMimeType` configuration property (defined in its child meta-object) to obtain the MIME type that identifies the default payload data handler to instantiate for the document payload. This data handler converts the payload data to its corresponding payload business object and returns the resulting payload business object to its calling entity.

How documents in a transport envelope are processed: If the Attachment data handler determines that the document is contained in an XML transport envelope, it must extract the payload and any attachments from this envelope structure before it can process them. Therefore, the data handler takes the following steps to process and convert the document:

1. Extracts the payload and any attachments from the transport envelope and decodes the payload data.

   The payload is contained in the `<payload>` XML tag. Each attachment is contained in an `<attachment>` XML tag.
2. Searches the content-type maps for a content type that matches that of the payload.

   The Attachment data handler uses the MIME type specified in the matching content-type map to create an instance of a data handler. This data handler converts the payload data to its corresponding payload business object and returns the resulting business object to the Attachment data handler.
3. Creates the content-information business object for the payload.

   The Attachment data handler examines the business-object-level application-specific information of the payload business object definition and determines the name of the content-information business object, whose attribute name is specified by the `cw_mo_bcg_content_info` tag. It then creates an instance of this content-information business object and sets the values for the payload content type and encoding.
4. Creates the attachment-container business object for the payload.

The Attachment data handler examines the business-object-level application-specific information of the payload business object and determines the name of the attachment-container business object, whose attribute name is specified by the `cw_mo_bcg_attachment` tag. It then creates an instance of the attachment-container business object and saves it in the appropriate attribute of the payload business object.

If the `cw_mo_bcg_attachment` tag does not exist (or is empty), assume that the document does not contain any attachments. Therefore, no further processing steps are required. The Attachment data handler returns the converted payload business object.

5. Creates the default attachment business object for the attachment container.

   The Attachment data handler examines the business-object-level application-specific information of the attachment-container business object and determines the name of the default attachment business object, whose attribute name is specified by the `cw_mo_bcg_default_attribute` tag. It then creates an instance of the default attachment business object and saves it in the appropriate attribute of the attachment-container business object.
6. Determines whether the attachment needs to be converted to a business object by searching the content-type maps for a content type that matches that of the attachment.

   The Attachment data handler examines the content type and character-set encoding from the attachment and checks to see whether there is a corresponding entry in a content-type map.

   - If a corresponding content-type map is *not* found, the Attachment data handler does not create a business object for the attachment data.

     Therefore, the data handler creates an instance of the default attachment business object, sets the values for the content type and encoding within its content-information business object, and sets the base64-encoded attachment data (as a string) in the `attachment` attribute.

     The Attachment data handler then populates the attachment-container business object with the default attachment business object.
   - If a content-type map *is* found, the Attachment data handler checks to see whether the attachment needs to be converted to a business object:
     - If the `ConvertAttachment` configuration property in the matching content-type map is false, the Attachment data handler creates an instance of the default attachment business object, sets the values for the content type and encoding within its content-information business object, and sets the base64-encoded attachment data (as a string) in the `attachment` attribute.

       The Attachment data handler then populates the attachment-container business object with the default attachment business object.
     - If the `ConvertAttachment` configuration property in the matching content-type map is true, the Attachment data handler decodes the attachment data and creates an instance of a data handler to process the attachment data. This data handler processes the decoded bytes and returns the corresponding attachment business object.

       The Attachment data handler then examines the business-object-level application-specific information of the attachment business object definition and determines the name of the content-information business object, whose attribute name is specified by the `cw_mo_bcg_content_info` tag. If this tag exists, the data handler creates the content-information business object for the attachment and sets the values for attachment's content type and encoding.

Finally, the Attachment data handler populates the attachment-container business object with the attachment business object.

How business objects are converted to documents

Before WebSphere Partner Gateway receives a document from InterChange Server, an adapter must determine whether to wrap the business-object representation of the payload and any attachments in the XML transport envelope. InterChange Server sends the business object to the appropriate data handler, which handles the actual conversion. This data handler can be configured to call the Attachment data handler to handle the conversion of payload and any attachment business objects into the corresponding payload and attachments as well as the creation of an XML transport envelope.

To convert a payload business object with attachments to its transport-envelope representation, the calling entity instantiates the Attachment data handler, passing it the payload business object. The Attachment data handler takes the following steps:

1. Loads the content-type maps defined in its configuration meta-object.

   The content-type maps are defined in the `ContentTypeMap_x` configuration properties of the child meta-object. The child meta-object is a business object that contains configuration information for the Attachment data handler. Attributes in this business object associate content-type maps with content types. For more information, see "Creating the Attachment child meta-object" on page 130.
2. Checks the business object to determine whether to create an XML transport envelope.
   - If the Attachment data handler does *not* determine that the document requires a transport envelope, it does not need to wrap the payload in this envelope structure.

     The document contains only a payload, which the Attachment data handler must create from its associated business-object representation. The data handler does not need to create a transport envelope for the document. For more information, see "How documents without a transport envelope are created."
   - If the Attachment data handler does determine that the document requires a transport envelope, it must wrap the payload and any attachments in this envelope structure.

     The document contains a payload and possibly some attachments. Therefore, the Attachment data handler needs to convert the payload business-object representation to a payload *and* any attachments and wrap these components in a transport envelope. For more information, see "How documents with a transport envelope are created" on page 128.
3. Sets the resulting payload and any attachment tags in the WebSphere Partner Gateway document and returns this document to the calling entity.

How documents without a transport envelope are created: If the Attachment data handler determines that the payload business object does *not* require an XML transport envelope, it does not need to wrap the payload data in the envelope structure. Therefore, the data handler uses the default payload data handler to convert the payload business object to its corresponding payload document. The `PayloadDataHandlerMimeType` configuration property (defined in the Attachment data handler's child meta-object) contains the MIME type that identifies the default payload data handler to instantiate for the payload business object. This data handler receives the payload business object as an argument and returns the resulting payload document to its calling entity.

How documents with a transport envelope are created: If the Attachment data handler determines that the payload business object *does* require an XML transport envelope, it must wrap the payload and attachment documents in this envelope structure. Therefore, the data handler takes the following steps to process and convert the business object:

1. Gets the content type and character-set encoding for the payload.

   The `cw_mo_bcg_content_info` tag in the business-object-level application-specific information of the payload business object specifies the name of the content-information attribute. This attribute contains the content type and encoding for the payload.

   Note: If the content-information attribute does not exist, use the default data handler (identified by the MIME type contained in the `PayloadDataHandlerMimeType` configuration property, in the Attachment data handler's child meta-object) to convert the payload business object.
2. Searches the content-type maps for a content type that matches that of the payload.

   The Attachment data handler uses the MIME type specified in the matching content-type map to create an instance of a payload data handler. This data handler converts the payload business object to its corresponding payload document and returns the resulting document to the Attachment data handler. From the string that is returned by the payload data handler, the Attachment data handler encodes the bytes using Base64 and stores the result in the payload tag of the XML transport envelope.
3. Gets the attachment container from the payload business object.

   The attachment container resides in the attachment-container attribute of the payload business object. The business-object-level application-specific information of the payload business object contains the `cw_mo_bcg_attachment` tag, which identifies the attachment-container attribute. This attribute contains the attachments.

   If the `cw_mo_bcg_attachment` tag does not exist (or is empty), assume that the document does not contain any attachments. Therefore, no further processing steps are required. The Attachment data handler returns the converted payload in its transport envelope.
4. For each attachment, determines whether the attachment is represented as a business object or just data.
   - If the attachment is just attachment data, the business-object-level application-specific information of the attachment-container business object contains the `cw_mo_bcg_default_attribute` tag, which identifies the default-attachment attribute. This attribute contains the attachment data, which the Attachment data handler retrieves, extracts the Base64-encoded data, and stores the result in the document.
   - If the attachment is represented by a business object, its attribute-level application-specific information contains the `wbic_type` tag to indicate that it contains an attachment business object.

     The Attachment data handler takes the following steps to process the attachment business object:

     a. Retrieves the contents of the attachment attribute and gets the content type and encoding for the attachment.

The business-object-level application-specific information of the attachment business object contains the `cw_mo_bcg_content_info` tag, which identifies the content-information attribute. This attribute contains the content type and encoding for this attachment. The Attachment data handler stores this content information in the attachment tag of the document.

b. Searches the content-type maps for a content type that matches that of the attachment.

   The Attachment data handler uses the MIME type in the matching content-type map to create an instance of a data handler. This data handler converts the attachment business object to its corresponding attachment document and returns the resulting document (as a string) to the Attachment data handler.

c. Stores the encoded result in the attachment tag of the XML wrapper for the document.

   The Attachment data handler gets the bytes from the returned string (using the character set, if one was present) and encodes the bytes using Base64. It then stores the result in the attachment tag.

Setting up the environment for the Attachment data handler

Use of the WebSphere Partner Gateway-supplied Attachment data handler involves the following steps:

- "Specifying which schema to use"
- "Deploying the Attachment data handler"
- "Configuring the Attachment data handler" on page 130

Specifying which schema to use

You have the option of using the default schema for the Attachment data handler or using a schema (wbipackaging_v1.1_ns.xsd) that lets you pass the contentId in the Backend Integration packaging.

To use the wbipackaging_v1.1_ns.xsd schema, configure the wbipackaging_version property in the bcg.properties file. (The bcg.properties file is described in the *Administrator Guide*.) This property is specified as:

```
wbipackaging_version=1.n
```

where *n* is 0 or 1. The default value of this property is 1.0.

The meta-object of the Attachment data handler has a wbipackaging_version attribute, which can have a value of 1.0 or 1.1. If you specify 1.1, the Attachment data handler parses and generates the XML messages containing the contentId of the attachment.

To specify the content ID of the attachment, the encoding business object uses the contentId attribute. When the Attachment Data Handler generates the XML from the business object, it uses this attribute to create the contentId tag for the attachment. When the Attachment Data Handler generates the business object from the XML, it sets this attribute, using the value specified in the contentId tag in the XML message.

Deploying the Attachment data handler

The Attachment data handler and associated repository file are available on the WebSphere Partner Gateway installation medium, in the locations listed in Table 42.

Table 42. Location of the components for Attachment data handler

Component	Location
Attachment data handler	`Integration/WBI/WICS/Attachment/bcgwbiattachmentdh.jar`
Repository file	`Integration/WBI/WICS/Attachment/MO_DataHandler_DefaultAttachmentConfigV1.0.in` or `Integration/WBI/WICS/Attachment/MO_DataHandler_DefaultAttachmentConfigV1.1.in`

Deploy the files into the Web server according to the documentation for the Web server.

Specifying the location of the Attachment data handler

WebSphere InterChange Server needs to know the location of the Attachment data handler, so that it can load it at run-time. To specify the location of the Attachment data handler, take the following steps:

1. Edit the ICS startup script, `start_server.bat`, which is located in the `bin` subdirectory of the InterChange Server product directory (on the computer where InterChange Server resides).
2. To the CLASSPATH variable in this file, add the jar file for the Attachment data handler: `bcgwbiattachmentdh.jar`

Configuring the Attachment data handler

Configuring the Attachment data handler consists of the following steps to create the configuration business objects:

- "Creating the Attachment child meta-object"
- "Updating the top-level data-handler meta-object" on page 133

Note: You must also create the attachment-related business object definitions for the Attachment data handler. For more information, see "Creating attachment-related business object definitions" on page 134.

Creating the Attachment child meta-object

To configure the Attachment data handler, you must create a child meta-object to provide the class name and configuration properties that the Attachment data handler needs. To create this meta-object, you create a business object definition that contains the attributes listed in Table 43. Use Business Object Designer, which is part of the WebSphere Business Integration Toolset, to create this business object definition

The child meta-object provides the class name and configuration properties that the Attachment data handler needs. In the Business Object Designer tool, create a child meta-object that includes MIME types for the payload and for the types of attachments you expect to receive.

The attributes of the child meta-object are shown in Table 43. An example of a child meta-object for the Attachment data handler is shown in Figure 29 on page 133.

Note: The sample business objects shown in this chapter do *not* include the standard attributes (such as ObjectEventId) required by WebSphere InterChange

Server but not used by the Attachment data handler.

Table 43. Configuration properties in the Attachment child meta-object

Attribute Name	**Description**
`ClassName`	Class name (required), which points to the following data handler class: `com.ibm.bcg.DataHandlers.AttachmentDataHandler`
`ContentTypeMap_`*`x`*	The content-type map for the payload and for each type of attachment you expect to receive in the XML wrapper. For more information, see "Content-type maps."
`PayloadDataHandlerMimeType`	MIME type used to identify the default data handler, which processes a payload that does *not* have associated attachments.
wbipackaging_version	This attribute can have a value of 1.0 or 1.1. If you specify 1.1, the Attachment data handler parses and generates the XML messages containing the contentId of the attachment.

Important: To assign a value to the attributes in Table 43, set the default value of the attribute. For example, if the Attachment data handler is to use the XML data handler for its default data handler, set the Default Value of the `PayloadDataHandlerMimeType` attribute to `text/xml`.

Content-type maps: The content-type map determines the data handler that the Attachment data handler calls to convert information formatted in the associated content type. For example, if the content type of the payload is `application/xml`, the Attachment data handler looks for a content-type map whose `ContentType` attribute contains the value `application/xml`. If no matching content type can be found, the data handler assumes that it should *not* convert the associated attachment to a business object.

You would create a content-type map for each of these content types, with the attribute-level application-specific information as shown in Table 45.

When you create an attribute in the child meta-object that represents a content-type map, keep the following in mind:

- The name of the content-type-map attribute has the following format:

  `ContentTypeMap_`*`x`*

  where *x* is an integer that uniquely identifies the content-type map within the business object definition.

  Note: You must order the `ContentTypeMap_`*`x`* attributes in sequence. For example, if you have three content-type maps, their attributes must be named `ContentType_1`, `ContentType_2`, and `ContentType_3`.
- The default value of the content-type-map attribute must contain some combination of valid tags.

  Table 44 lists the tags that the default value for this attribute can contain.

Table 44. Valid tags for default value of content-type-map attribute

Tag name	**Description**	**Required?**
`ContentType`	Actual content type that comes in the transport envelope (for example, `text/xml`).	Yes

Table 44. Valid tags for default value of content-type-map attribute (continued)

Tag name	Description	Required?
`MimeType`	MIME type used to identify the data handler to convert the associated content type to a business object. If you do not specify `MimeType`, the data handler uses the value of `ContentType` to instantiate the data handler.	No
`CharSet`	Name of a character set (for example, UTF-8) that the Attachment data handler uses to convert bytes to a string or a string to bytes. If you do not specify `CharSet`, the Attachment data handler takes the following actions: • For inbound data, the data bytes that result from decoding the message from base64 are used for the conversion to the business object. • For outbound data, calls are made to the method of the child data handler that returns bytes (and not a string).	No
`ConvertAttachment`	Boolean value to indicate whether the attachment should be converted to a business object. The default is `false`.	No

The content-type map can also specify the character set for encoding as well as whether an attachment should be converted to a business object. For a description of the child meta-object attributes and an example, see "Creating the Attachment child meta-object" on page 130.

For example, suppose you have the following content types in your document:
- application/xml
- text/xml
- application/octet-stream

Table 45. Sample content-type maps

Content type	Attribute name	Default value
text/xml	`ContentType_1`	`ContentType=text/xml;MimeType=myxml; CharSet=UTF-8;ConvertAttachment=false;`
application/xml	`ContentType_2`	`ContentType=application/xml; MimeType=mynewxml;CharSet=UTF-16; ConvertAttachment=true;`
application/octet-stream	`ContentType_3`	`ContentType=application/octet-stream; MimeType=myoctet`

Sample child meta-object: WebSphere Partner Gateway provides the following InterChange Server repository input files, which contains a sample child meta-object for the Attachment data handler:

ProductDir`/Integration/WBI/WICS/Attachment/`
`MO_DataHandler_DefaultAttachmentConfigV1.0.in`

ProductDir`/Integration/WBI/WICS/Attachment/`
`MO_DataHandler_DefaultAttachmentConfigV1.1.in`

where *ProductDir* is the directory of your installed WebSphere Partner Gateway product. The repository files define a single Attachment data handler whose associated child meta-object is `MO_DataHandler_DefaultAttachmentConfig`. Figure 29 shows the sample child meta-object for the Attachment data handler. This meta-object defines two content-type maps, `ContentTypeMap_1` and `ContentTypeMap_2`.

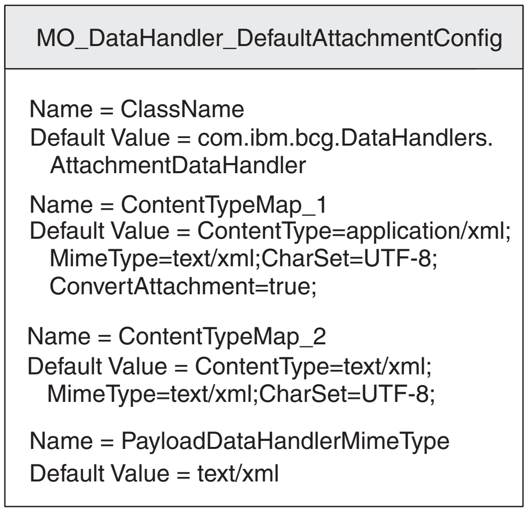
MO_DataHandler_DefaultAttachmentConfig
Name = ClassName
Default Value = com.ibm.bcg.DataHandlers.
AttachmentDataHandler
Name = ContentTypeMap_1
Default Value = ContentType=application/xml;
MimeType=text/xml;CharSet=UTF-8;
ConvertAttachment=true;
Name = ContentTypeMap_2
Default Value = ContentType=text/xml;
MimeType=text/xml;CharSet=UTF-8;
Name = PayloadDataHandlerMimeType
Default Value = text/xml

Figure 29. Sample child meta-object for the Attachment data handler

MO_DataHandler_DefaultAttachmentConfigV1.1.in contains the additional attribute:

```
[Attribute]
    Name = WBIPackaging_Version
    Type = String
    MaxLength = 255
    IsKey = false
    IsForeignKey = false
    IsRequired = true
    DefaultValue = 1.0
    IsRequiredServerBound = false
    [End]
```

Updating the top-level data-handler meta-object

A WebSphere Business Integration Adapter (such as the Adapter for JMS) uses the `MO_DataHandler_Default` meta-object to identify the data handlers it can use. Add a reference to the Attachment data handler in the meta-object.

To the `MO_DataHandler_Default` meta-object, you make the following modifications:

1. Add an attribute whose name identifies the MIME type associated with the Attachment data handler instance; that is, for a document that contains this MIME type, the associated data handler can handle its conversion to a business object.

   The attribute type of this attribute is the business object definition for the Attachment data handler's child meta-object (see "Creating the Attachment child meta-object" on page 130).
2. Add an attribute for each of the supported attachment MIME types, if these do not already exist in the top-level data-handler meta-object.

   The attribute type of these attributes would be the child meta-object of the associated data handler.

For example, suppose you have the Attachment data handler as configured in Figure 29 on page 133. Figure 30 shows the `MO_DataHandler_Default` meta-object with an attribute that associates the `wbic_attachment` MIME type with the instance of the Attachment data handler that is configured by the `MO_DataHandler_DefaultAttachmentConfig` child meta-object. This top-level data-handler meta-object also associates the document MIME type (`text/xml`) with the XML data handler's child meta-object.

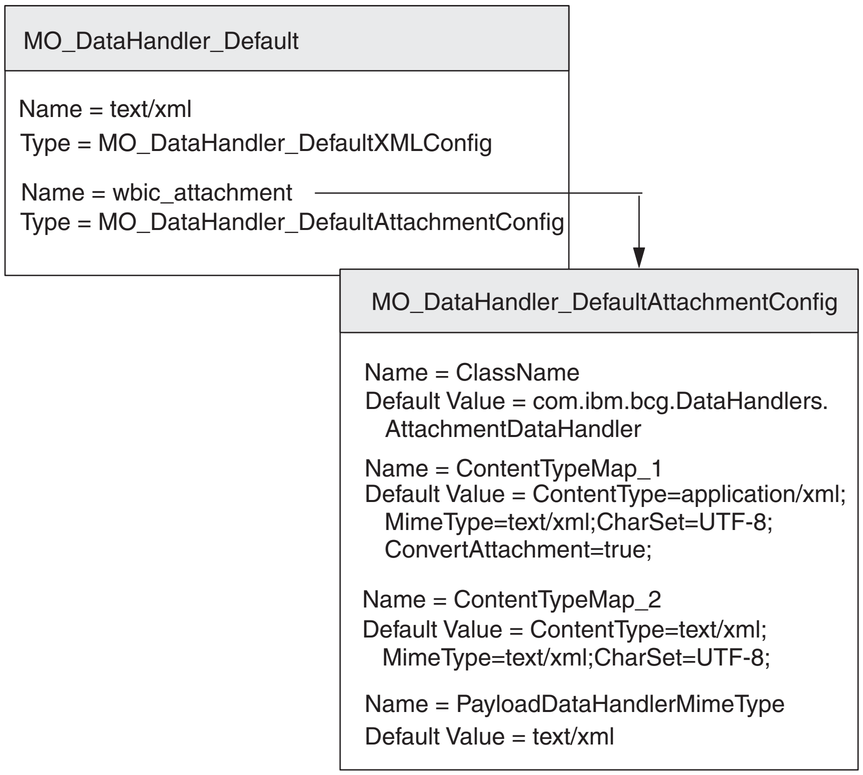
MO_DataHandler_Default
Name = text/xml
Type = MO_DataHandler_DefaultXMLConfig
Name = wbic_attachment
Type = MO_DataHandler_DefaultAttachmentConfig
MO_DataHandler_DefaultAttachmentConfig
Name = ClassName
Default Value = com.ibm.bcg.DataHandlers.AttachmentDataHandler
Name = ContentTypeMap_1
Default Value = ContentType=application/xml;MimeType=text/xml;CharSet=UTF-8;ConvertAttachment=true;
Name = ContentTypeMap_2
Default Value = ContentType=text/xml;MimeType=text/xml;CharSet=UTF-8;
Name = PayloadDataHandlerMimeType
Default Value = text/xml

Figure 30. Associating the wbic_attachment MIME type with the Attachment data handler

For each unique combination of supported content types that you need to support, repeat the process by adding an attribute in the appropriate top-level data-handler meta-object, whose attribute name is the MIME type associated with the Attachment data handler instance and whose type is the name of the associated child meta-object. Also ensure that the configured MIME types (and their child meta-objects) exist in the top-level meta-object.

Creating attachment-related business object definitions

If you are sending or receiving documents that are wrapped in an XML transport envelope, your payload business object needs to contain attachment information. In any document flow, there is one payload and, optionally, multiple attachments. The Attachment data handler expects this attachment information to be in *attachment-related business objects*. Therefore, you must create business object definitions to represent this information. A business object definition is the form of information that InterChange Server uses. You use the Business Object Designer tool to create business object definitions.

Figure 31 shows the business-object structure for a payload that is wrapped in an XML transport envelope.

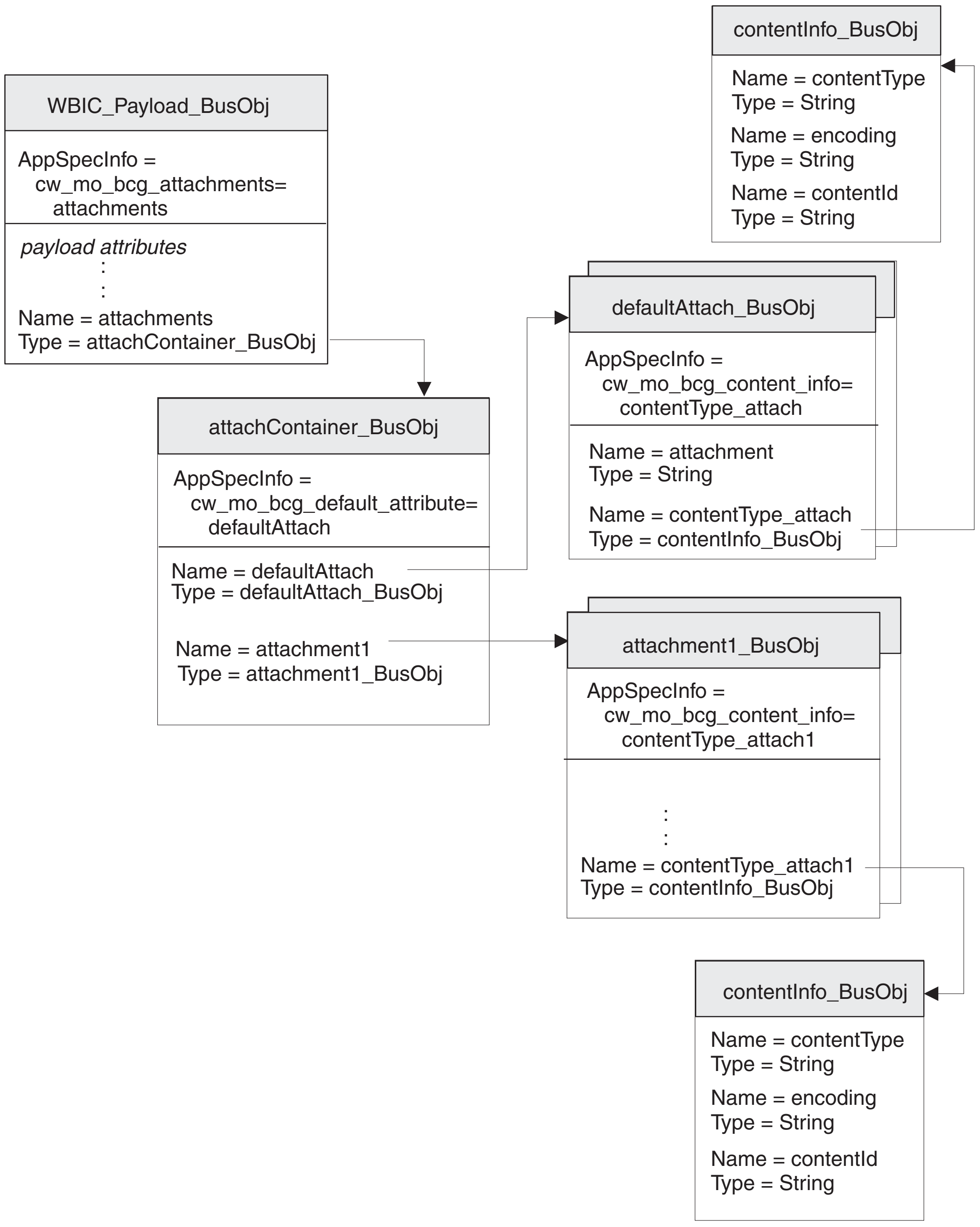

Figure 31. Relationship of the payload business object to the attachment business objects

As Figure 31 shows, all the attachments are contained in the attachment-container business object. If there are attachments, the payload business object has an attribute that corresponds to the attachment-container business object.

Make sure your business-object structure includes attachment-required business objects by taking the following steps:

1. Create a business object definition to hold the content-type-encoding properties required by the Backend Integration packaging.
2. Create a business object definition for each type of attachment.

3. Create a business object definition for the attachment-container business object.
4. Modify the business object definition for your payload business object.

Each of these steps is described in the sections below.

Representing the content information

To store the content type and encoding of the associated payload or attachment, you create the *content-information business object*. To create a content-information business object definition, create the attributes shown in Table 46.

Table 46. Attributes of the content-information business object

Attribute	Attribute type	Description	Is key attribute?
`contentType`	String	The content type for the associated payload or attachment.	Yes
`encoding`	String	The character encoding for the associated payload or attachment	No

In Figure 31 on page 135, the `contentInfo_BusObj` business object definition contains attributes for the content type and encoding of the attachment. These attributes all have attribute-level application-specific information to specify the name of the related protocol header. For example, the `x-aux-sender-id` attribute has the application-specific information set as follows:

```
name=x-aux-sender-id
```

You can choose whatever name you want to identify the content-information business object definition. The application-specific information of the attachment business object determines if this is a Content Type Encoding business object type. Figure 31 on page 135 shows an example of a content-type-encoding business object definition called `contentType_BusObj`.

Representing attachment data

For attachment data that is not to be converted into a business object, you create the *default attachment business object*. This business object is useful for containing base64-encoded data that comes from the transport envelope.

To create a default-attachment business object definition, take the following steps:

1. Create the attributes shown in Table 47.
2. If you create a content-information business object, in the application-specific information for the default attachment business object definition, add the `cw_mo_bcg_content_info` tag to identify the attribute that contains the content information.

   This `cw_mo_bcg_content_info` tag has the following format:

   ```
   cw_mo_bcg_content_info=contentInfoAttr
   ```

   where *contentInfoAttr* is the name of the attribute that contains the attachment-container business object.

Table 47. Attributes of the default attachment business object

Attribute	Attribute type	Description	Is key attribute?
`attachment`	String	The piece of attachment data. **Note:** This attribute is the key attribute of the business object definition.	Yes

Table 47. Attributes of the default attachment business object (continued)

Attribute	**Attribute type**	**Description**	**Is key attribute?**
An attribute to hold the content information	Business object	An optional attribute to hold the content-information business object, which provides the content type and encoding for the attachment data. This attribute should have single cardinality. **Note:** If this attribute does *not* exist, the Attachment data handler does not set the attachment data in the business object. For more information on the format of the content-information business object, see "Representing the content information" on page 136.	No

In Figure 31 on page 135, the `defaultAttach_BusObj` business object definition contains attributes for the piece of attachment data, including a content-information business object to hold its content type and encoding. The piece of attachment data that this default attachment business object represents does have a content-type encoding, represented by the `contentType_attach` attribute. Therefore, the default attachment business object definition includes the following tag in its business-object-level application-specific information:

```
cw_mo_bcg_content_info=contentType_attach
```

Representing the attachments

For each kind of attachment in your document that converts to a business object, you must create a separate *attachment business object definition*. The attachment business object definition represents the actual data in a document attachment. To create an attachment business object definition, take the following steps:

1. Create an attribute for each piece of attachment data.

   Possible attribute types can include String (for simple pieces of data) or a business object definition (for complex data).
2. If the attachment requires content-type encoding:
   a. Create a content-type-encoding attribute.

      The attribute type for this attribute is the content-type-encoding business object definition (see "Representing the content information" on page 136).
   b. Add to the business-object-level application-specific information of the attachment business object definition the `cw_mo_bcg_content_info` tag, to identify the attribute that contains the content-type encoding.

      This `cw_mo_bcg_content_info` tag has the following format:

      ```
      cw_mo_bcg_content_info=contentTypeEncodingAttr
      ```

      where *contentTypeEncodingAttr* is the name of the attribute that contains the content-type-encoding business object.

In Figure 31 on page 135, the payload document has one attachment, represented by the `attachment1_BusObj` business object definition. This attachment does have a content-type encoding, represented by `contentTypeEncoding` attribute. Therefore, the attachment business object definition includes the following tag in its business-object-level application-specific information:

```
cw_mo_bcg_content_info=contentTypeEncoding
```

Representing the attachment container

The attachment container contains all document attachments in the transport envelope. To represent the attachment container for InterChange Server, you create the attachment-container business object. Each attribute in the attachment-container business object represents one attachment.

To create the attachment-container business object definition, take the following steps:

1. Add an attribute for each attachment in the document that is to be converted to a business object.

   The attribute type for each of these attributes is the associated attachment business object (see "Representing the attachments" on page 137). Each attribute should have multiple cardinality.
2. Add to the application-specific information for each attribute the `wbic_type` tag to identify the attribute as an attachment.

   The `wbic_type` tag has the following format:

   ```
   wbic_type=Attachment
   ```

   Note: An attachment attribute can have multiple cardinality.
3. If the payload contains attachment data that should *not* be converted to a business object:
   a. Add an attribute for the default attachment business object.

      The attribute type for this attribute is the default attachment business object (see "Representing attachment data" on page 136). It is the key attribute for the attachment-container business object. This attribute does *not* require the `wbic_type` tag in its application-specific information.

      Note: The attachment-container business object can contain only *one* default attachment attribute. However, this attribute can have multiple cardinality.
   b. Add to the business-object-level application-specific information of the attachment business object definition the `cw_mo_bcg_default_attribute` tag, to identify the attribute that contains the attachment data.

      This `cw_mo_bcg_default_attribute` tag has the following format:

      cw_mo_bcg_content_info=*defaultAttachmentAttr*

      where *defaultAttachmentAttr* is the name of the attribute that contains the default attachment business object.

   Important: If no default-attachment attribute exists, the Attachment data handler *cannot* convert any attachments that do not have an associated content-type map or attachments that are not converted to business objects. These attachments will be lost during the conversion to business-object representation.

In Figure 31 on page 135, the attachment container is represented by the `attachContainer_BusObj` business object definition. This attachment-container business object definition has the following attributes:

- The `attachment1` attribute represents the single attachment for the document. Therefore, the attachment-container business object definition includes the following tag in its attribute-level application-specific information:

  ```
  wbic_type=Attachment
  ```

  This attachment is represented by the `attachment1_BusObj` business object definition.

- The `defaultAttach` attribute represents the attachment data that does *not* require conversion to the business-object representation. Therefore, the attachment-container business object definition includes the following tag in its business-object-level application-specific information:

  ```
  cw_mo_bcg_default_attribute=defaultAttach
  ```

Modifying the payload business object definition

The payload business object definition represents the information in your document. It contains an attribute for each piece of information you are transferring between WebSphere Partner Gateway and InterChange Server. For information on the creation of the payload business object definition, see "Business object for the document" on page 119.

If you are sending or receiving documents that contain attachments, your payload business object needs to contain attachment information. In any document flow, there is one payload and, optionally, multiple attachments. If the payload of your document contains attachments, you must modify the payload business object definition as follows:

1. Create an attribute to hold the payload data.

   You might find it easier to use if your actual payload data is stored in a separate payload business object definition. In this case, the top-level payload business object contains an attribute for the payload data whose attribute type is the business object definition of actual payload data.
2. Add an attachment container:
   a. Add an attribute to hold the attachment container.

      The attribute type of this attribute is the attachment-container business object definition (see "Representing the attachment container" on page 138). This attribute should have single cardinality.
   b. In the application-specific information for the payload business object definition, add the `cw_mo_bcg_attachment` tag to identify the attribute that contains the attachment container.

      This `cw_mo_bcg_attachment` tag has the following format:

      cw_mo_bcg_attachment=*attachContainerAttr*

      where *attachContainerAttr* is the name of the attribute that contains the attachment-container business object.
3. Optionally, you can specify the content type of the payload. The Attachment data handler uses this content type to determine which data handler to instantiate to convert the payload data. If it finds a matching content type in the content-type maps, it instantiates the data handler for this content type.
   a. Add a content-information attribute, which is an optional attribute to hold the content type and encoding for the payload. This attribute should have single cardinality.

      Note: If this attribute does *not* exist, the Attachment data handler obtains the data handler to convert the payload from the setting of the `PayloadDataHandlerMimeType` configuration property, in its child meta object.
   b. In the application-specific information for the payload business object definition, add the `cw_mo_bcg_content_info` tag to identify the attribute that contains the content information.

      This `cw_mo_bcg_content_info` tag has the following format:

      cw_mo_bcg_attachment=*contentInfoAttr*

where *contentInfoAttr* is the name of the attribute that contains the content-information business object. For more information on the format of the content-information business object, see "Representing the content information" on page 136.

4. Add any configuration attributes required for your transport protocol.

   For example, if you are using the JMS transport protocol, your payload business object definition must contain the JMS dynamic business object. For more information, see the section on how to create business object definitions in support of your transport protocol.

Chapter 9. Integrating InterChange Server over HTTP

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere InterChange Server over the HTTP transport protocol. It provides information on how to configure InterChange Server (ICS) and the adapters required for communication over HTTP.

Note: For information on how to configure WebSphere Partner Gateway to communicate with InterChange Server over HTTP, see "Configuring WebSphere Partner Gateway for InterChange Server" on page 116. For general information on how to configure InterChange Server, see "Configuring InterChange Server" on page 119.

This chapter provides the following information on how to send and receive documents between WebSphere Partner Gateway and WebSphere InterChange Server through the use of the HTTP transport protocol:
- "Using HTTP transport protocol with ICS"
- "Sending SOAP documents over HTTP/S" on page 154

Using HTTP transport protocol with ICS

WebSphere Partner Gateway can send and receive documents with WebSphere InterChange Server (ICS) over the HTTP transport protocol

Note: If you are exchanging SOAP documents over the HTTP transport protocol, see "Sending SOAP documents over HTTP/S" on page 154.

This section provides the following information on how to configure InterChange Server and the appropriate adapters for use with WebSphere Partner Gateway over HTTP:
- "Components required for documents to ICS over HTTP transport"
- "Setting up the environment for HTTP transport with ICS" on page 144
- "Creating business object definitions for ICS over HTTP" on page 147
- "Creating ICS artifacts for HTTP" on page 153

Components required for documents to ICS over HTTP transport

For WebSphere Partner Gateway to communicate with InterChange Server using the HTTP transport protocol requires that these two components be configured. Table 48 summarizes these configuration steps.

Table 48. Configuring WebSphere Partner Gateway and InterChange Server

Component	Version	For more information
WebSphere Partner Gateway	6.0	"Configuration for sending documents to ICS over the HTTP transport protocol" on page 117 "Configuration for receiving documents from ICS over the HTTP transport protocol" on page 118

Table 48. Configuring WebSphere Partner Gateway and InterChange Server (continued)

Component	Version	For more information
WebSphere InterChange Server	4.2.2 or higher	"Creating ICS artifacts for HTTP" on page 153

In addition, to send or receive a document between WebSphere Partner Gateway and InterChange Server using the HTTP transport protocol, you use the components listed in Table 49.

Table 49. Components required to transfer documents with InterChange Server through HTTP

Component	Description	Notes and restrictions
WebSphere Business Integration Adapter for HTTP (Adapter for HTTP)	This adapter allows InterChange Server to exchange business objects with applications that send or receive data in the form of HTTP streams.	Use version 4.2.1 of the Adapter for HTTP.
A payload data handler	This data handler converts the document payload between its document format (usually XML) and its business-object representation.	This data handler is required and must support the MIME type of your payload document.
Attachment data handler	This data handler handles attachment documents for your document message.	This data handler is required *only* if your documents include attachments.

The following sections describe how the components in Table 49 work together to send and receive documents between WebSphere Partner Gateway and InterChange Server over the HTTP transport protocol.

How documents are sent to ICS through HTTP

For WebSphere Partner Gateway to send a document to InterChange Server using the HTTP transport protocol, you use the Adapter for HTTP to retrieve the document that WebSphere Partner Gateway has sent as an HTTP stream. The adapter then routes the document to InterChange Server. Figure 32 provides an overview of how WebSphere Partner Gateway sends documents to InterChange Server over the HTTP transport protocol.

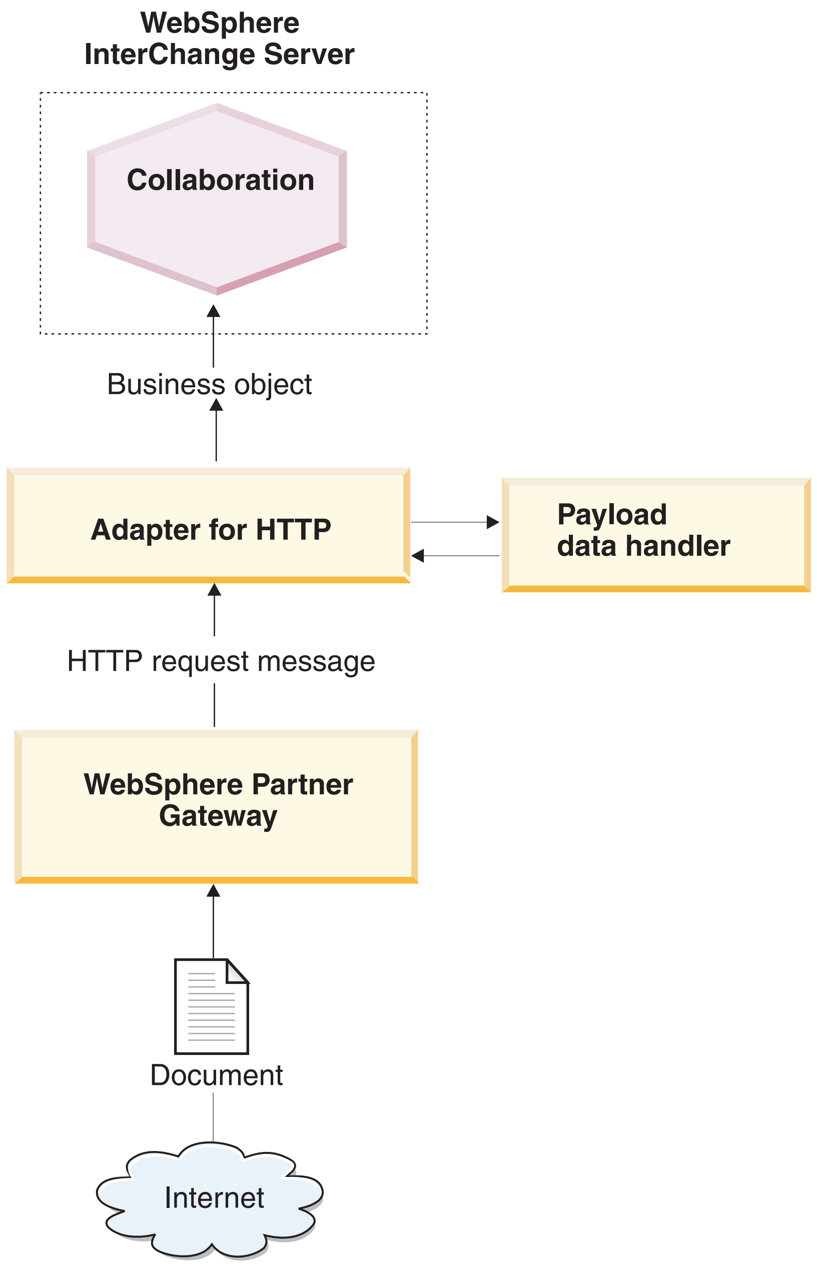

Figure 32. Message flow from WebSphere Partner Gateway to a collaboration through the HTTP transport protocol

How documents are received from ICS through HTTP

For WebSphere Partner Gateway to receive a document from InterChange Server using the HTTP transport protocol, you use the Adapter for HTTP, which sends the message it receives from InterChange Server as an HTTP stream for WebSphere Partner Gateway to retrieve. Figure 33 provides an overview of how WebSphere Partner Gateway receives documents from InterChange Server over the HTTP transport protocol.

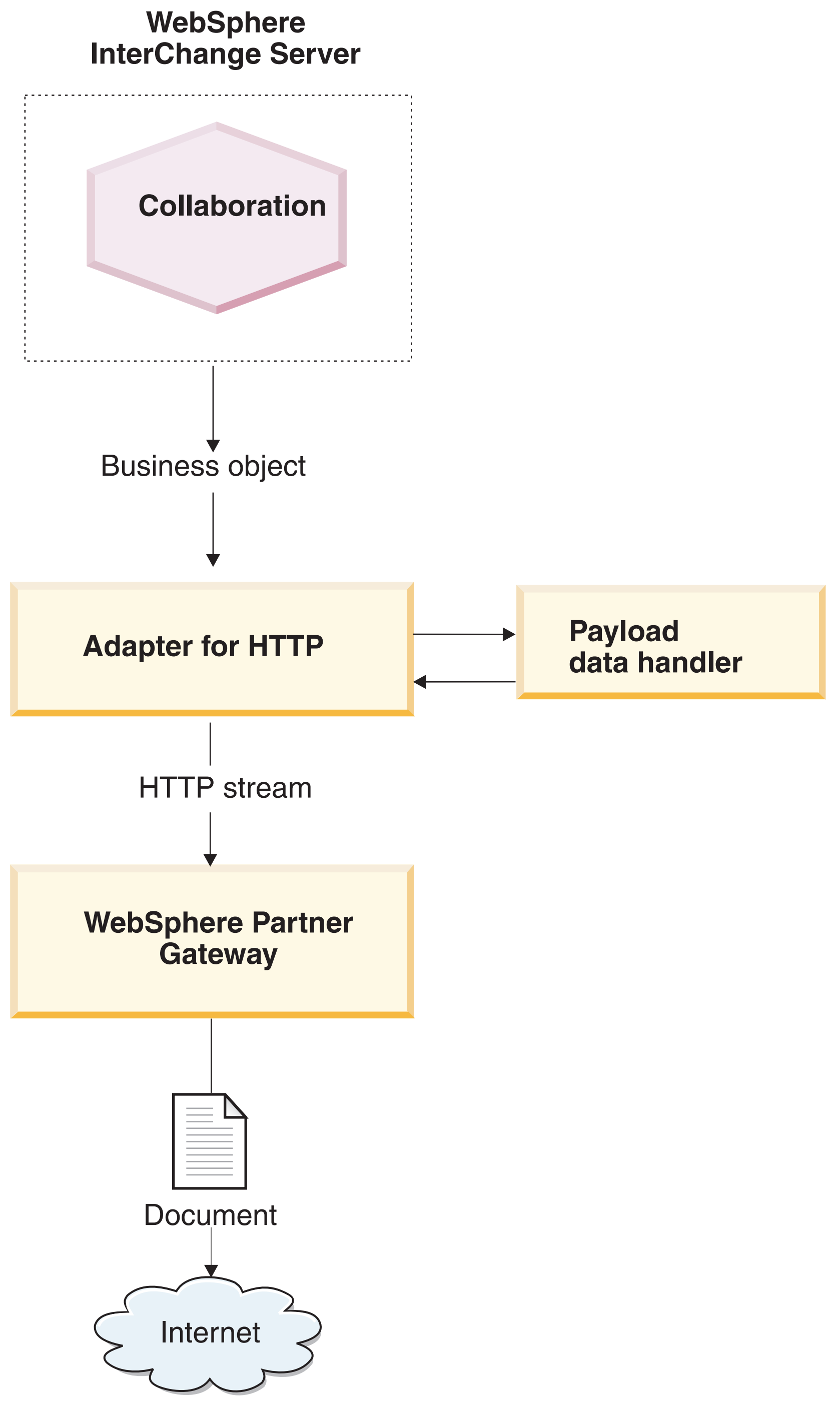

Figure 33. Message flow from a collaboration to WebSphere Partner Gateway through the HTTP transport protocol

Setting up the environment for HTTP transport with ICS

Because the sending and receiving of documents to and from InterChange Server involves adapters and data handlers, you must perform the setup and configuration tasks on the Adapter for HTTP. For information on how to configure WebSphere Partner Gateway for use with InterChange Server over HTTP, see "Configuring WebSphere Partner Gateway for InterChange Server" on page 116.

The Adapter for HTTP allows WebSphere Partner Gateway to exchange documents with InterChange Server in the form of HTTP messages. It supports the following interactions with InterChange Server:

- For request processing, it receives the request business object from InterChange Server, converts it to an HTTP stream, and sends it to the specified URL, where it can be received by WebSphere Partner Gateway.
- For event notification, it listens at a specified URL, where WebSphere Partner Gateway sends documents. When it receives a document, it converts it to an event business object (using a data handler) and sends it to InterChange Server.

Important: WebSphere Partner Gateway does *not* include the WebSphere Business Integration Adapter for HTTP. You must obtain this product separately and install it according to the instructions in its *Adapter for HTTP User Guide*. Refer to the adapter documentation to ensure that the version of the adapter is compatible with the version of InterChange Server you are using.

When you have configured the Adapter for HTTP to communicate with InterChange Server, follow the steps in these sections to configure this adapter to listen for HTTP messages from WebSphere Partner Gateway:

Specifying the payload data handler

As Figure 33 on page 144 shows, the Adapter for HTTP uses a data handler to convert the business objects it receives from InterChange Server into the appropriate HTTP streams.

Note: The data handler that the Adapter for HTTP calls converts the payload of the document. If your document is wrapped in an XML transport envelope (it contains attachments or the Envelope Flag is Yes), configure the Attachment data handler as the payload data handler. For more information, see “Handling documents with attachments” on page 123.

To indicate which data handler to use to convert the payload, you must take the steps listed in “Business object conversion” on page 120. In addition, you must configure the Adapter for HTTP to use this payload data handler. You can set the payload data handler in either of the following ways:

- In Connector Configurator, set the `DataHandlerMetaObjectName` connector configuration property to specify the name of the top-level data-handler meta-object that the Adapter for HTTP uses to identify data handlers. Make sure you include the top-level data-handler meta-object in the list of supported business objects for the adapter.
- In the top-level business object, use the `MimeType` attribute to hold the MIME type to identify the payload data handler. For more information on this business object, see “Top-level business object” on page 148.

Configuring the protocol-handler package name

The Adapter for HTTP uses the `JavaProtocolHandlerPackages` connector configuration property to identify the name of the Java Protocol Handler packages. For integration with WebSphere Partner Gateway, make sure that the `JavaProtocolHandlerPackage` property is set to its default value:

```
com.ibm.net.ssl.internal.www.protocol
```

Configuring the HTTP protocol listener

The Adapter for HTTP supports hierarchical configuration properties to obtain the information it needs to configure its protocol listeners. The top-level configuration property is called `ProtocolListenerFramework`. Within this top-level property are several levels of subproperties. To configure the protocol handlers for use with the Adapter for HTTP, make sure that the properties are configured in the `ProtocolListener` property, as described in the following steps:

1. Configure a protocol listener with subproperties under the following configuration property:

   ```
   ProtocolListenerFramework
      ProtocolListeners
         HttpListener1
   ```

   To configure your protocol listener, set the subproperties listed in Table 50.

   Table 50. Configuring the protocol listener

Property	Description	Value
`Protocol`	Type of protocol listener: • HTTP • HTTPS	`http` or `https`
`Host`	IP address on which the protocol listener listens	IP address of the local computer on which WebSphere Partner Gateway is running
`Port`	Port on which the protocol listener listens for requests	`8080`

2. Configure the URL configurations that the protocol listener supports with subproperties under the following configuration property:

   ```
   ProtocolListenerFramework
      ProtocolListeners
         HttpListener1
            URLsConfiguration
               URL1
   ```

   Set the `ContextPath` property to the URI for the HTTP requests that the protocol listener receives.

   Note: This directory must be the same one that WebSphere Partner Gateway specifies as its Target URI. For more information, see "Configuration for sending documents to ICS over the HTTP transport protocol" on page 117.

3. If your document contains attachments, you must configure a transformation for the protocol listener by setting subproperties of the following configuration property:

   ```
   ProtocolListenerFramework
      ProtocolListeners
         HttpListener1
            URLsConfiguration
               URL1
                  TransformationRules
                     TransformationRule1
   ```

   To configure the attachment transformation for your protocol listener, set the subproperties listed in Table 51. You need one transformation rule for each instance of the Attachment data handler you are using. For more information on the Attachment data handler, see "Handling documents with attachments" on page 123.

   Table 51. Configuring the attachment transformation for the protocol listener

Property	Description	Value
`ContentType`	Content type of the data to be transformed with a data handler	Content type associated with the attachment data

Table 51. Configuring the attachment transformation for the protocol listener (continued)

Property	Description	Value
`MimeType`	MIME type to use to identify the data handler to call	MIME type associated with the instance of the Attachment data handler
`Charset`	Character set to use when transforming data of the specified content type	Character set for the attachment data

For more information on these properties, see the *Adapter for HTTP User Guide*.

Creating business object definitions for ICS over HTTP

The Adapter for HTTP sends and receives your document to InterChange Server in the form of a payload business object. The Adapter for HTTP invokes the payload data handler to handle this business object when it receives or sends a WebSphere Partner Gateway document, as follows:

- For request processing, the payload data handler converts the request business object to its corresponding HTTP stream.
- For event notification, the data handler converts the HTTP stream to an event business object.

Therefore, you must create the business object definitions shown in Table 52 to represent the payload business-object structure that the Adapter for HTTP expects.

Table 52. Business object definitions for the Adapter for HTTP

Condition	Business object definition	For more information
If you are using None or Backend Integration packaging for your message *and* your documents do *not* have attachments	Payload business object: • Top-level business object • Request business object • Response business object (optional) • Fault business object (optional)	"Creating the payload business-object structure for ICS over HTTP"
If you are using Backend Integration packaging for your message	Add to the payload business object the business objects to hold the message header information: • Dynamic meta-object • HTTP-properties business object	"Creating HTTP transport-level header information for ICS" on page 151.
If the document includes attachments	You must also create additional business objects to represent the attachments.	"Creating attachment-related business object definitions" on page 134

Creating the payload business-object structure for ICS over HTTP

The Adapter for HTTP expects a payload business-object structure that consists of the following business objects:

- A top-level business object
- A request business object
- A fault business object (optional)

- A response business object (optional)

Figure 34 shows a sample business-object structure for a payload business object definition for use with InterChange Server over the HTTP transport protocol.

Note: For a detailed description of this business-object structure, refer to the *Adapter for HTTP User Guide*.

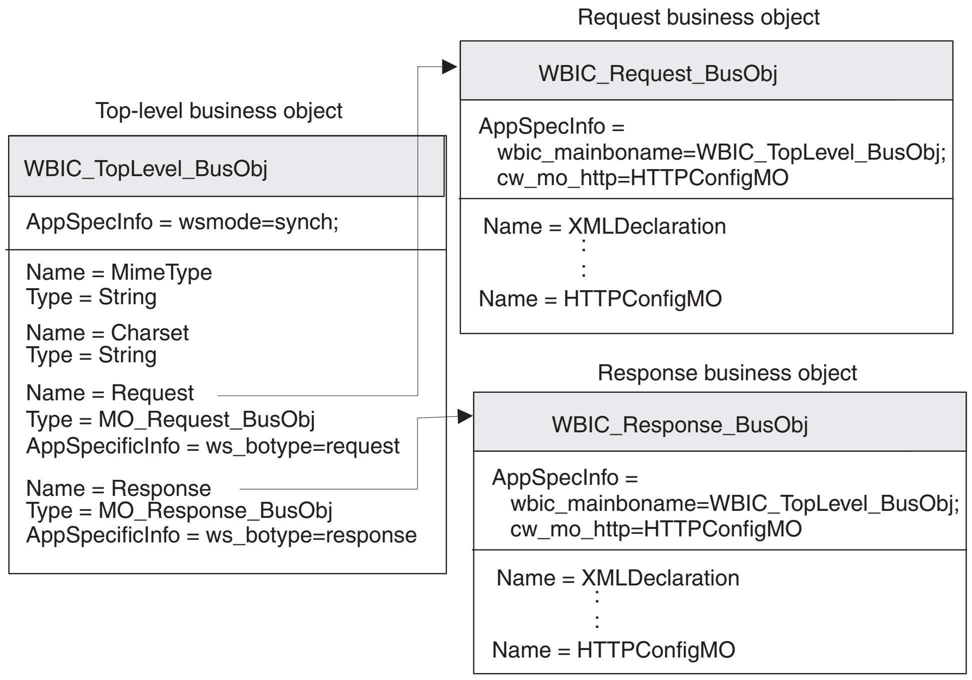
Request business object
WBIC_Request_BusObj
AppSpecInfo = wbic_mainboname=WBIC_TopLevel_BusObj; cw_mo_http=HTTPConfigMO
Name = XMLDeclaration
Name = HTTPConfigMO
Top-level business object
WBIC_TopLevel_BusObj
AppSpecInfo = wsmode=synch;
Name = MimeType
Type = String
Name = Charset
Type = String
Name = Request
Type = MO_Request_BusObj
AppSpecificInfo = ws_botype=request
Name = Response
Type = MO_Response_BusObj
AppSpecificInfo = ws_botype=response
Response business object
WBIC_Response_BusObj
AppSpecInfo = wbic_mainboname=WBIC_TopLevel_BusObj; cw_mo_http=HTTPConfigMO
Name = XMLDeclaration
Name = HTTPConfigMO

Figure 34. Business-object structure for the HTTP payload business object for ICS

Top-level business object: The top-level business object is a wrapper for the request and response business objects. You must create a business object definition for this business object. Table 53 summarizes the attributes of the top-level business object definition.

Table 53. Attributes of top-level business object

Attribute	Attribute type	Description
MimeType	String	Defines the content type and format of the data that is being passed to the URL.
Charset	String	Used to determine which data handler to call.
Request	Business object	Child business object that represents the request message. The purpose of this business object depends on whether it participates in request processing or event notification. For more information on the structure of this business object, see "Request business object" on page 149.

Table 53. Attributes of top-level business object (continued)

Attribute	Attribute type	Description
Response	Business object	Child business object that represents the response message (if you are expecting a response). The purpose of this business object depends on whether it participates in request processing or event notification. For more information on the structure of this business object, see "Response business object" on page 150.

Note: When using the Adapter for HTTP with WebSphere Partner Gateway, you do *not* need to include fault business objects in your top-level business object.

Table 54 summarizes the application-specific information that the top-level business object definition can have.

Table 54. Application-specific information for the top-level business object definition

Application-specific information	Tag	Description
Business-object level	`ws_mode`	Defines whether the interaction is synchronous or asynchronous
Attribute level	`ws_botype`	Defines which attribute contains the request or response business object

For a complete description of the structure of the top-level business object and its application-specific information, see the *Adapter for HTTP User Guide*.

Request business object: The request business object contains the data to be passed to the URL. It represents the HTTP request message. The purpose of this request business object depends on which InterChange Server task it is participating in, as follows:

- For event notification (sending a document to InterChange Server), the request business object contains the request message from WebSphere Partner Gateway, which is the event to be sent to InterChange Server.
- For request processing (receiving a document from InterChange Server), the request business object contains the request that InterChange Server is making to WebSphere Partner Gateway.

Note: The top-level business object identifies its child business objects as its request and response business objects. However, this structure is used in *both* request processing and event notification.

For the basic description of the request business object's structure, refer to the *Adapter for HTTP User Guide*. For use with WebSphere Partner Gateway, there are two customizations you must make to the structure of the request business object definition:

- If the document that WebSphere Partner Gateway sends to InterChange Server uses Backend Integration packaging, you must add to the request business object definition a special attribute to identify the HTTP protocol-configuration meta-object.

This attribute provides configuration information for the transport-level headers of the message. For more information, see "Creating HTTP transport-level header information for ICS" on page 151.

- To the business-object-level application-specific information of the request business object definition, add the tags shown in Table 55.

Table 55. Tags in application-specific information of request business object

Application-specific-information tag	Description	Required?
`ws_tloname`	Gives the name of the top-level business object	Only required if business object definition participates in event notification
`cw_mo_http`	Specifies the HTTP protocol-configuration meta-object, which contains the HTTP transport-level header fields. For more information, see "Creating HTTP transport-level header information for ICS" on page 151.	Only required if you are using Backend Integration packaging

Note: If you are using the Attachment data handler to process documents wrapped in an XML transport envelope, you must modify your request business object to hold the attachments, as described in "Creating attachment-related business object definitions" on page 134.

Response business object: The response business object contains the data to be received from the URL. It contains attributes for the various XML tags in the response message. The purpose of this response business object depends on which InterChange Server task it is participating in, as follows:

- For event notification, the response business object contains the response message, which is sent from the collaboration in InterChange Server.
- For request processing, the response business object contains the information from WebSphere Partner Gateway in response to the request that InterChange Server sent.

Regardless of whether the response is part of event notification or request processing, a response business object is sent *only* if the exchange between WebSphere Partner Gateway and InterChange Server is *synchronous* and a business response is expected in response to your request.

For the basic description of the fault business object's structure, refer to the *Adapter for HTTP User Guide*. For use with WebSphere Partner Gateway, there are customizations you must make to the structure of the request business object definition:

- If the document that WebSphere Partner Gateway sends to InterChange Server uses Backend Integration packaging, you must add to the response business object definition a special attribute to identify the HTTP protocol-configuration meta-object.

  This attribute provides configuration information for the transport-level headers of the message. For more information, see "Creating HTTP transport-level header information for ICS" on page 151.
- To the business-object-level application-specific information of the response business object definition, add the tags shown in Table 55.

- In the top-level business object, add the `ws_botype` tag to the attribute-level application-specific information for the attribute that corresponds to the response business object.

  This tag has the following syntax:

  ```
  ws_botype=response
  ```

If the exchange between WebSphere Partner Gateway and InterChange Server is *asynchronous*, WebSphere Partner Gateway does *not* expect a response, so you do not need to create a response business object.

Creating HTTP transport-level header information for ICS

If you are sending documents with Backend Integration packaging over the HTTP transport protocol, your request business object needs to contain custom transport-level header information. The Adapter for HTTP expects this custom header information to be in a *dynamic meta-object*.

Figure 35 shows the business-object structure for a request business object that represents a WebSphere Partner Gateway document with Backend Integration packaging over the HTTP transport protocol.

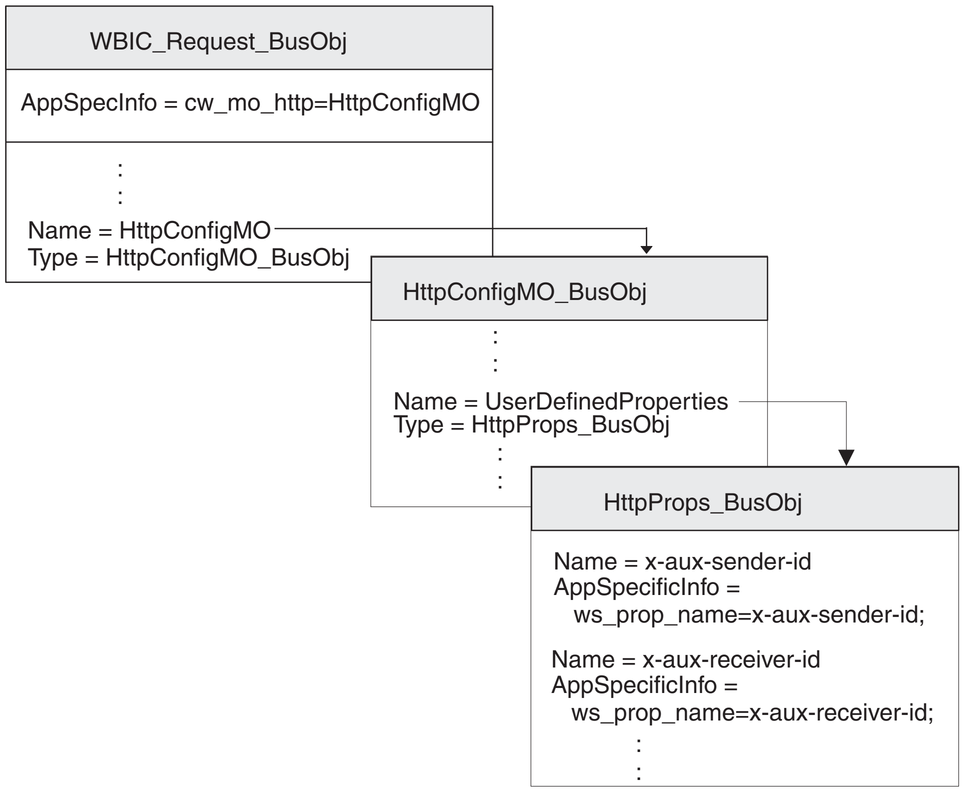

Figure 35. Relationship of the request business object to the HTTP protocol-configuration meta-object

Make sure your business-object structure includes an HTTP protocol-configuration meta-object by taking the following steps:

1. Create a business object definition to hold the HTTP properties required by the Backend Integration packaging.

2. Create a business object definition for the HTTP protocol-configuration meta-object.
3. Modify the business object definition for your request business object to include an attribute for the HTTP protocol-configuration meta-object.

Each of these steps is described in the sections below.

Creating the user-defined-properties business object: The Adapter for HTTP supports a *user-defined-properties business object* to hold custom properties in the HTTP protocol-configuration meta-object. WebSphere Partner Gateway uses this business object to hold HTTP properties required by the Backend Integration packaging. It can also contain the `Content-Type` attribute, which specifies the content-type header to set in the request message, and the content-length attribute, which specifies the length of the message, in bytes. Table 5 on page 20 describes each of the valid transport-header fields.

To create a user-defined-properties business object definition for the HTTP header fields, take the following steps:

1. Create an attribute within the business object definition for each of the transport-header fields.

   All attributes should have an attribute type of String. You can name the attribute with the exact name of the HTTP property (as listed in the `Header field` column of Table 5 on page 20).
2. For each of the attributes in the HTTP-properties business object, add application-specific information to identify the purpose of the associated attribute.

   This attribute-level application-specific information has the following format:

   ```
   ws_prop_name=HTTPproperty
   ```

   where *HTTPproperty* is one of the values in the `Header field` column of Table 5 on page 20.

In Figure 35 on page 151, the `HttpProps_BusObj` business object definition contains attributes for the various transport-header fields. These attributes all have attribute-level application-specific information to specify the name of the related protocol header. For example, the `x-aux-sender-id` attribute has the application-specific information set as follows:

```
ws_prop_name=x-aux-sender-id
```

Creating the HTTP protocol-configuration meta-object: For event notification, the request, response, or fault business object can contain a dynamic meta-object called the *HTTP protocol configuration meta-object* to hold configuration information (such as header information).

For the basic description of the HTTP protocol-configuration business object's structure, refer to the *Adapter for HTTP User Guide*. For use with WebSphere Partner Gateway, you must make the following customizations to the structure of the HTTP protocol-configuration business object definition:

1. Create an attribute within the business object definition for any of the fields you require.

   All attributes should have an attribute type of String.

   Note: For a complete list of attributes in the HTTP protocol-configuration meta-object, see the *Adapter for HTTP User Guide*.
2. Add the `UserDefinedProperties` attribute to this business object definition.

The attribute type of this attribute is the business object definition for the user-defined-properties business object (see "Creating the user-defined-properties business object" on page 152).

For example, in Figure 35 on page 151, the `HttpConfigMO_BusObj` business object definition contains the `UserDefinedProperties` attribute, whose attribute type is `HttpProps_BusObj`.

Modify the request business object definition: The request business object definition represents the information requested from WebSphere Partner Gateway. For information on how to create the request business object, see "Request business object" on page 149. To incorporate the dynamic meta-object into your payload business-object structure, you must make the following modifications to your request business object definition:

1. Add an attribute to your request business object definition to hold the HTTP protocol-configuration meta-object.

   The attribute type for this attribute is the business object definition for the HTTP protocol-configuration meta-object (see "Creating the HTTP protocol-configuration meta-object" on page 152).
2. Add the `cw_mo_http` tag to the business-object-level application-specific information of your request business object definition to identify the attribute that contains the HTTP protocol-configuration meta-object.

   The `cw_mo_http` tag has the following format:

   `cw_mo_http=`*`HttpConfigMetaObjAttr`*

   where *HttpConfigMetaObjAttr* is the name of the attribute in the request business object that holds the HTTP protocol-configuration meta-object.

For example, in Figure 35 on page 151, an attribute named `HttpConfigMO` has been added to the request business object definition, `hub_HttpRequest_BusObj`. This attribute contains the dynamic meta-object, which is a child business object of type `HttpConfigMO_BusObj`. In addition, the application-specific information of the request business object has been modified to include the following `cw_mo_http` tag to identify this dynamic meta-object:

```
cw_mo_http=HttpConfigMO
```

Creating ICS artifacts for HTTP

To configure InterChange Server for communication with WebSphere Partner Gateway over the HTTP transport protocol, you must create the InterChange Server artifacts shown in Table 56.

Table 56. Artifacts for communicating with ICS over the HTTP transport protocol

ICS artifact	Purpose	For more information
Business object definitions	Represent the document	"Creating business object definitions for ICS over HTTP" on page 147
Connector object	Represents the Adapter for HTTP at run-time	"Creating the HTTP connector object" on page 154
Collaboration template and collaboration object	Represents the business process that InterChange Server uses to process the document	"Binding collaborations to communicate with Adapter for HTTP" on page 154

Creating the HTTP connector object

To obtain an instance of the Adapter for HTTP at run-time, you must take the following steps within System Manager:

1. Create the connector objects:
   - Create a connector object to represent an instance of the Adapter for HTTP.

     Note: In the Supported Business Objects tab of Connector Configurator, make sure that you specify all business object definitions you created for use with the Adapter for HTTP. For a description of these business object definitions, see "Creating business object definitions for ICS over HTTP" on page 147.
   - If required by your collaboration, create a connector object for the Port Connector.
2. Configure the connector objects

   For information on how to configure your Adapter for HTTP connector object for use with WebSphere Partner Gateway, see "Setting up the environment for HTTP transport with ICS" on page 144.

Binding collaborations to communicate with Adapter for HTTP

As described in "Creating the collaborations" on page 122, a collaboration object must exist at run-time for InterChange Server to know where to receive and send business objects. When you create the collaboration object for the collaboration that uses the Adapter for HTTP to send information to and receive it from WebSphere Partner Gateway, you bind the collaboration ports, as follows:

- For request processing, set the "to" port, which sends requests to WebSphere Partner Gateway, to the connector object you created for the Adapter for HTTP; that is, the Adapter for HTTP is the *destination* adapter.
- For event notification, set the "from" port, which receives events from WebSphere Partner Gateway, to the connector object you created for the Adapter for HTTP; that is, the Adapter for HTTP is the *source* adapter.

Sending SOAP documents over HTTP/S

SOAP documents differ from other types of documents exchanged over HTTP/S. They use the standard Adapter for Web Services, which calls the SOAP data handler to transform SOAP messages into business objects and to transform business objects into SOAP messages. This section describes how to send and receive SOAP documents between WebSphere Partner Gateway and WebSphere InterChange Server over the HTTP transport protocol.

Note: To send and receive non-SOAP documents between WebSphere Partner Gateway and WebSphere InterChange Server over the HTTP transport protocol, see "Using HTTP transport protocol with ICS" on page 141.

Refer to the Adapter for Web Services documentation for information on the business-object structure and on the WSDL Object Discovery Agent (ODA), a design-time tool you can use to generate SOAP business objects that include information about the target Web services.

As described in the *Hub Configuration Guide*, you must have set up a target to receive Web service invocations from a back-end system (the Web services target) as well as a target to receive Web service invocations from a community participant (the external Web services target).

Components required for sending and receiving

To send a SOAP document from WebSphere Partner Gateway to InterChange Server using the HTTP transport protocol, you use the component listed in Table 57.

Table 57. Components required to send SOAP documents to InterChange Server through HTTP

Component	Description	Notes and restrictions
WebSphere Business Integration Adapter for Web Services	This adapter allows InterChange Server to exchange business objects with applications that send or receive data in the form of HTTP messages.	1. This adapter *cannot* be used with non-SOAP documents. 2. Make sure you are using the Adapter for Web Services 3.4.0 (or higher). Refer to the *Adapter for Web Services User Guide* to make sure that the level of the adapter is compatible with the version of WebSphere InterChange Server you are using.

Note: If a SOAP document contains attachments, you do not need to use the Attachment data handler to handle them.

How community participants invoke Web services

The following steps occur when a community participant sends a request for a collaboration that is exposed as a Web Service that the Community Manager provides:

1. The community participant sends a SOAP request message to the destination specified in the WSDL document generated for the collaboration. Note that the endpoint specified in the WSDL is the Web services target (URL) of WebSphere Partner Gateway instead of the actual endpoint.
2. WebSphere Partner Gateway receives and routes the message to the Adapter for Web services.
3. The Adapter for Web Services sends the SOAP message to the SOAP data handler to convert the SOAP message to a business object. The adapter invokes the collaboration exposed as a Web service.
4. If this is a request/response operation, the collaboration returns a SOAP response (or fault) business object.
5. If the collaboration returned a SOAP response (or fault) business object, the Adapter for Web Services calls the SOAP data handler to convert the SOAP response (or fault) business object to a SOAP response message. The adapter returns the response to WebSphere Partner Gateway. If the collaboration did not return a SOAP response (or fault) business object, the Adapter for Web Services returns the appropriate HTTP response status code.
6. WebSphere Partner Gateway routes the response to the Web service.

How the Community Manager invokes Web services

The Public WSDL provided by WebSphere Partner Gateway can be used for creating business objects using WSDL ODA. It is important to note that when the Web service is provided by a community participant for use by the Community Manager, the public URL used by the Community Manager to invoke the Web service should contain the following query string:

```
?to=<Community participant Web Service Provider's business ID>
```

For example, the following address tells WebSphere Partner Gateway that the provider of the Web service is the participant with business ID 123456789:

```
http://<Hub_IP_address>/bcgreceiver/Receiver?to=123456789
```

The WSDL ODA will not add the query string in the default value of the URL attribute of the Web Service top-level business object.

The following steps occur when a collaboration sends a request (to the Adapter for Web Services) to invoke a Web service of a community participant:

1. The collaboration sends a service call request to the adapter, which calls the SOAP data handler to convert the business object to a SOAP request message.
2. The adapter invokes the Web service by sending the SOAP message to the external Web services target (URL) on WebSphere Partner Gateway.
3. WebSphere Partner Gateway acts as a proxy, sending the SOAP message to the endpoint corresponding to the destination (community participant) Web service. This invokes the Web service.
4. The invoked Web service receives the SOAP request message and performs the requested processing.
5. The invoked Web service sends a SOAP response (or fault) message. In the case of a one-way operation, the appropriate HTTP status code is returned.
6. If this is a request/response Web Service, WebSphere Partner Gateway routes the SOAP response (or fault) message to the adapter, which calls the data handler to convert it to a response or fault business object. The connector returns the SOAP response or fault business object to the collaboration.

Chapter 10. Integrating with InterChange Server over JMS

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere InterChange Server over the JMS transport protocol. It provides information on how to configure InterChange Server and the adapters required for communication over JMS.

Note: For information on how to configure WebSphere Partner Gateway to communicate with InterChange Server over JMS, see "Configuring WebSphere Partner Gateway for InterChange Server" on page 116. For general information on how to configure InterChange Server, see "Configuring InterChange Server" on page 119.

This chapter provides the following information on how to send and receive documents between WebSphere Partner Gateway and WebSphere InterChange Server through the use of the JMS transport protocol:
- "Components required for documents over JMS transport"
- "Setting up the environment for JMS transport" on page 162
- "Creating business object definitions for JMS" on page 165

Components required for documents over JMS transport

For WebSphere Partner Gateway to communicate with InterChange Server over the JMS transport protocol, the components must be configured to work with JMS. Table 58 summarizes these configuration steps.

Table 58. Configuring WebSphere Partner Gateway and InterChange Server for JMS transport protocol

Component	**Version**	**For more information**
WebSphere Partner Gateway	6.0	"Configuration for sending documents to ICS over the JMS transport protocol" on page 117 "Configuration for receiving documents from ICS over the JMS transport protocol" on page 118
WebSphere InterChange Server	4.2.2, 4.3	"Creating ICS artifacts for JMS" on page 169

In addition, to send or receive a document between WebSphere Partner Gateway and InterChange Server over the JMS transport protocol, you also use the components listed in Table 59.

Table 59. Components required to transfer documents to and from InterChange Server through JMS

Component	Description	Notes and restrictions
WebSphere Business Integration Adapter for JMS (Adapter for JMS)	This adapter allows InterChange Server to exchange business objects with applications that send or receive data in the form of JMS messages. The Adapter for JMS and WebSphere Partner Gateway communicate through JMS queues.	Make sure you are using the Adapter for JMS, version 2.7.0 (or higher), which provides support for custom header properties. Refer to the adapter documentation to make sure that the version of the adapter is compatible with the version of InterChange Server you are using.
A payload data handler	This data handler converts the payload between its document format and its business-object representation.	For more information, see "Specifying the payload data handler" on page 164.
Attachment data handler	This data handler converts documents with attachments into business objects.	This data handler is required *only* if your documents include attachments. For more information, see "Handling documents with attachments" on page 123.

The following sections describe how the components in Table 59 work together to send and receive documents between WebSphere Partner Gateway and InterChange Server over the JMS transport protocol.

How documents are sent over the JMS transport

For WebSphere Partner Gateway to send a document to InterChange Server using the JMS transport protocol, you use the Adapter for JMS to retrieve the message that WebSphere Partner Gateway has put on a JMS queue. The adapter then routes the message to InterChange Server. Figure 36 provides an overview of how WebSphere Partner Gateway sends documents to InterChange Server over the JMS transport protocol.

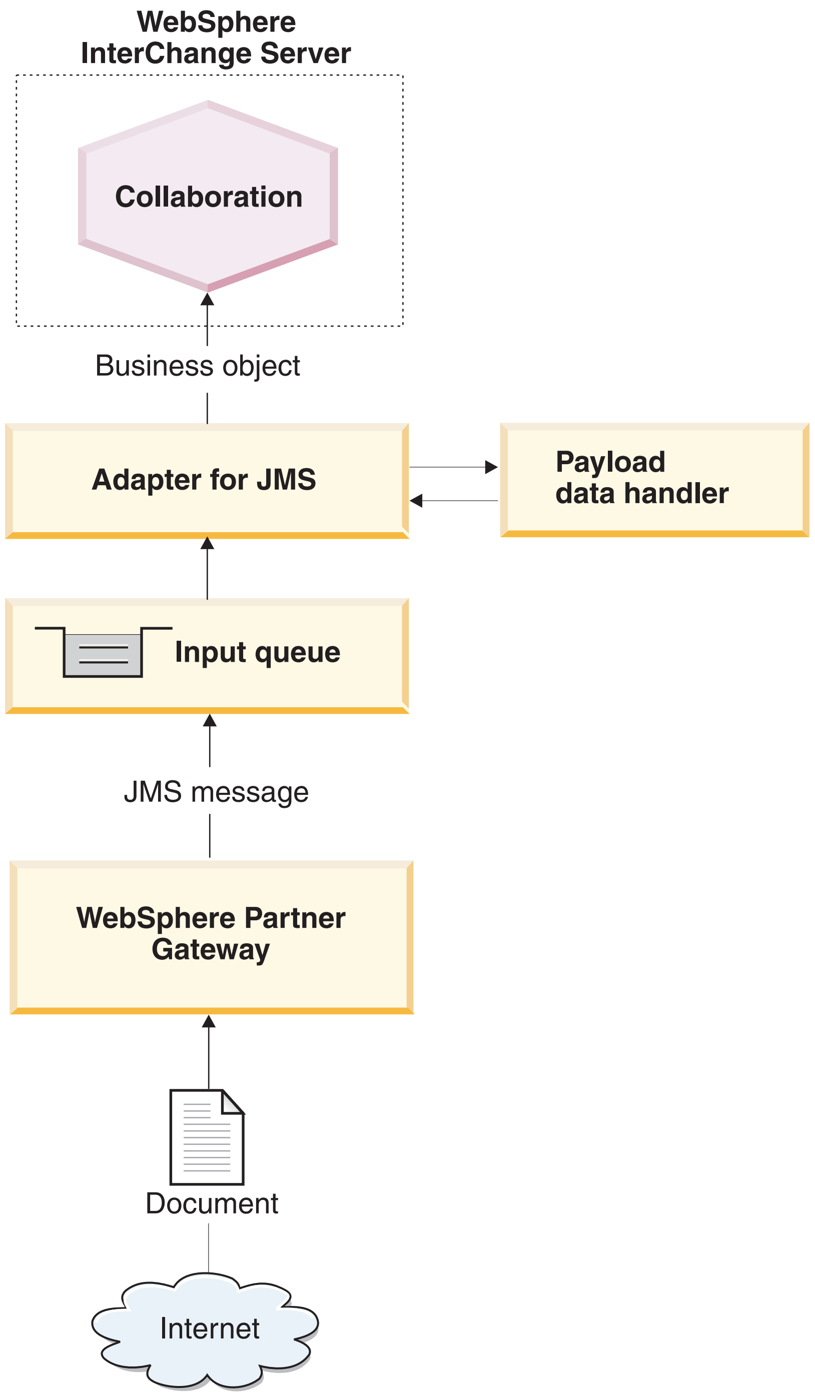

Figure 36. Message flow from WebSphere Partner Gateway to a collaboration through the JMS transport protocol

The following steps describe how WebSphere Partner Gateway participates in event notification by sending a document to a collaboration within InterChange Server over the JMS transport protocol:

1. WebSphere Partner Gateway posts a message to its JMS outbound queue.

   If the packaging type of the document is Backend Integration, WebSphere Partner Gateway has provided custom properties in this message. The JMS message header, JMSType, is set with the content type of the payload.

Note: Within WebSphere Partner Gateway, you must configure a gateway that identifies the JMS queue to which WebSphere Partner Gateway sends the message and on which the Adapter for JMS is polling.

2. When the Adapter for JMS sees a message on one of its input queues, it retrieves the message.

   The JMS queue that WebSphere Partner Gateway uses as its outbound queue is the same queue that the Adapter for JMS uses as its input queue. For information on how to set up this queue, see "Configuring the JMS queues" on page 163. For detailed information on the processing of the Adapter for JMS, see the *Adapter for JMS User Guide*.
3. The Adapter for JMS moves the message to its in-progress queue.
4. The Adapter for JMS extracts the body of the JMS message and invokes a data handler with the body of the message. This data handler converts the body of the JMS message to a business object.

   Note: If your messages have attachments, you can install the Attachment data handler and then configure the Adapter for JMS to call it to convert the body of the JMS message to a business object. For more information, see "Handling documents with attachments" on page 123.

   When Backend Integration is the packaging type and the document contains attachments, the configured data handler is responsible for handling the payload and attachments.
5. The data handler returns the business object to the Adapter for JMS.

   Note: If the Attachment data handler was used, this business object contains the payload as well as the attachments.
6. If the Adapter for JMS finds a child dynamic meta-object (specified using `cw_mo_conn` in the business-object level application specific information), the adapter populates the user-defined JMS headers present in the business object with the headers present in the JMS message.
7. The Adapter for JMS delivers the business object to the InterChange Server as part of a subscription delivery.

How documents are received over the JMS transport

For WebSphere Partner Gateway to receive a document from InterChange Server using the JMS transport protocol, you use the Adapter for JMS, which places the message it receives from InterChange Server on a JMS queue for WebSphere Partner Gateway to retrieve. Figure 37 provides an overview of how WebSphere Partner Gateway receives documents from InterChange Server over the JMS transport protocol.

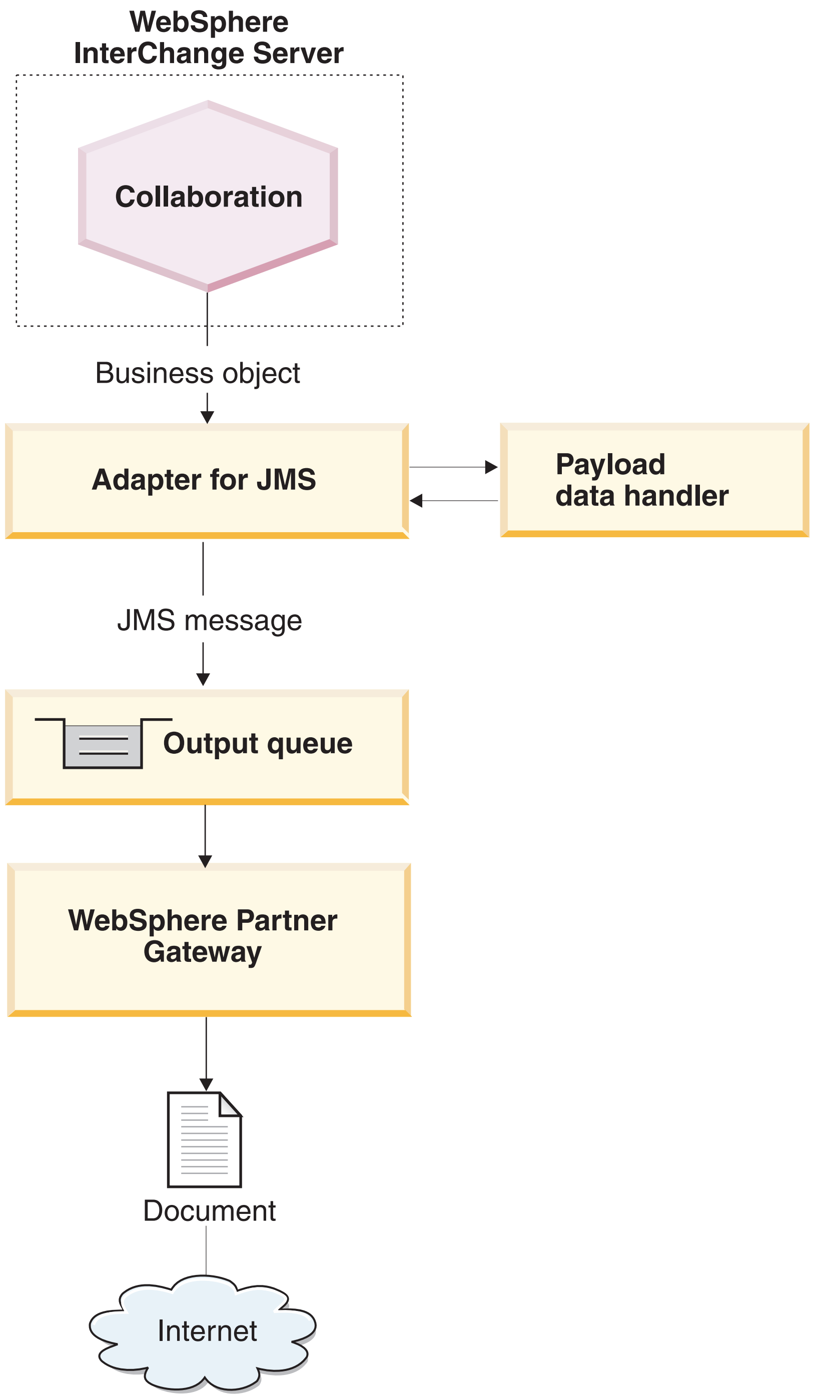

Figure 37. Message flow from a collaboration to WebSphere Partner Gateway through the JMS transport protocol

The following steps describe how WebSphere Partner Gateway participates in request processing by receiving a document from a collaboration within InterChange Server over the JMS transport protocol:

1. The collaboration within InterChange Server makes a service call to the Adapter for JMS, sending it the request business object.

   The request business object contains application-specific information pointing to a dynamic meta-object that contains the JMS transport-level header information, which WebSphere Partner Gateway expects.

2. The Adapter for JMS uses a data handler to convert the business object that the collaboration has sent it into a JMS message.

   The adapter reads the `DataHandlerMimeType` and `DataHandlerConfigMO` properties to determine the data handler to use. For more information, see "Specifying the payload data handler" on page 164.

   Note: If your documents have attachments, install the Attachment data handler and then configure the Adapter for JMS to call it to convert the request business object to a document with attachments. For more information, see "Handling documents with attachments" on page 123.
3. The data handler converts the business object to a string and returns it to the Adapter for JMS.
4. The Adapter for JMS determines, from the request business object, the name of the dynamic meta-object for custom JMS properties.

   The adapter searches the application-specific information of the request business object for the `cw_mo_conn` tag, which identifies the attribute that contains the dynamic meta-object. If you are using Backend Integration packaging for your document, you can specify transport-level header information in this dynamic meta-object.
5. The Adapter for JMS searches the dynamic meta-object for the `JMSProperties` attribute.

   If this attribute is populated, the adapter sets the transport-level header fields in the request document. Within the `JMSProperties` attribute, you can also specify the content-type standard JMS header. For more information, see "Creating JMS header information" on page 166.
6. The Adapter for JMS creates a JMS message, using the string returned by the data handler. It also sets any custom properties, as defined in the dynamic meta-object.
7. The Adapter for JMS sends the resulting request message to an output queue.

   The queue can be specified in the static meta-object or the dynamic meta-object. For information on specifying queues, see "Identifying the JMS queues" on page 164. WebSphere Partner Gateway listens on this JMS queue, which is configured as its inbound queue in its target definition.
8. WebSphere Partner Gateway receives the message from its JMS inbound queue, as configured in its target.

Note: WebSphere Partner Gateway supports only *asynchronous* interaction with InterChange Server over JMS. Therefore, you might not want to wait for the response. The response from the community participant or WebSphere Partner Gateway can come on a different queue. You can configure the Adapter for JMS to poll that queue. The response that comes on the queue can be delivered to InterChange Server as part of the event delivery.

Setting up the environment for JMS transport

Because the sending and receiving of documents to and from InterChange Server involves JMS queues and the Adapter for JMS, you must perform the setup and configuration tasks described in Table 60. For information on how to configure WebSphere Partner Gateway for use with InterChange Server over JMS, see "Configuring WebSphere Partner Gateway for InterChange Server" on page 116.

Table 60. Setting up the environment for use of JMS transport protocol

Configuration step	For more information
1. Configure your JMS queues.	"Configuring the JMS queues"
2. Configure the WebSphere Business Integration Adapter for JMS.	"Configuring the Adapter for JMS"

Note: If your documents contain attachments, you must also install and configure the Attachment data handler. For more information, see "Handling documents with attachments" on page 123.

Configuring the JMS queues

To use the JMS transport protocol with InterChange Server, you must set up the JMS system that WebSphere MQ provides. Supported versions of InterChange Server use version 5.3 of WebSphere MQ as a JMS provider. You can use the steps in the *Hub Configuration Guide* to set up the JMS transport-protocol mechanism.

Important: The steps in the *Hub Configuration Guide* must be performed on the computer on which WebSphere Partner Gateway resides. This guide assumes that the JMS transport-mechanism required by the Adapter for JMS and InterChange Server has already been set up as part of the InterChange Server installation.

When you create your JMS queues for use between WebSphere Partner Gateway and InterChange Server, consider the following points:

- Part of the InterChange Server installation process involves the creation of a WebSphere MQ queue manager. You can use this queue manager with WebSphere Partner Gateway.
- When you create your JMS queue aliases, you might want to name them to indicate the direction of flow between WebSphere Partner Gateway and InterChange Server. For example, if you create the queues listed in the `Original queue name` column of Table 61, you could rename these queues to indicate the InterChange Server directional flow, as shown in the `Directional queue name` column of Table 61.

Table 61. Naming JMS queues for InterChange Server to indicate direction

Original queue name	Directional queue name
inQ	ICS2HUB
outQ	HUB2ICS

Configuring the Adapter for JMS

The Adapter for JMS allows WebSphere Partner Gateway to exchange documents with InterChange Server in the form of JMS messages. It supports the following interactions with InterChange Server:

- For request processing, it receives the request business object from InterChange Server, converts it to a JMS message (using a data handler), and puts the JMS message on a JMS queue (see Figure 37 on page 161), where it can be picked up by WebSphere Partner Gateway.
- For event notification, it polls a JMS queue for JMS messages from WebSphere Partner Gateway. When it finds a JMS message, it converts it to an event business object (using a data handler) and sends it to InterChange Server.

Important: WebSphere Partner Gateway does *not* include the WebSphere Business Integration Adapter for JMS. You must obtain this product separately and install it according to the instructions in its *Adapter for JMS User Guide*. It is important that you read the steps described in this guide to correctly install and configure your Adapter for JMS.

When you have configured the Adapter for JMS to communicate with InterChange Server, follow the steps in this section to configure this adapter to accept JMS messages from WebSphere Partner Gateway:

- "Specifying the payload data handler"
- "Identifying the JMS queues"

Specifying the payload data handler

As Figure 37 on page 161 shows, the Adapter for JMS uses a data handler to convert the business objects it receives from InterChange Server into the appropriate JMS messages.

Note: The data handler that the Adapter for JMS calls converts the payload of the document. If your document is wrapped in an XML transport envelope (it contains attachments or the Envelope Flag is Yes), configure the Attachment data handler as the payload data handler. For more information, see "Handling documents with attachments" on page 123.

To indicate which data handler to use to convert the payload, you must take the steps listed in "Business object conversion" on page 120. In addition, you must configure the Adapter for JMS to use this payload data handler. In Connector Configurator, take the following steps:

1. Set the following connector configuration properties to identify the payload data handler:
   - Set the `DataHandlerConfigMO` and `DataHandlerMimeType` properties with the name of the top-level data-handler meta-object and the supported MIME type, respectively.
   - Set the `DataHandlerClassName` property with the name of the data-handler class to instantiate.

   Note: You set *either* the `DataHandlerConfigMO` and `DataHandlerMimeType` properties *or* the `DataHandlerClassName` property.
2. Include the top-level data-handler meta-object in the list of supported business objects.

You can also specify the data handler to use in the static or dynamic meta-object. The same properties (`DataHandlerMimeType`, `DataHandlerConfigMO`, and `DataHandlerClassName`) are available as attributes in these meta-objects. For a complete description, refer to the *Adapter for JMS User Guide*.

Identifying the JMS queues

When the Adapter for JMS receives a document from InterChange Server, it puts the message in its outbound queue, which is the one that the WebSphere Partner Gateway Receiver is polling. Similarly, when WebSphere Partner Gateway sends a document to InterChange Server, it puts the document in its outbound queue, which is the one that the Adapter for JMS is polling.

Table 62 summarizes how to configure the JMS queues that the Adapter for JMS uses to receive and send documents.

Note: For a complete description of how to configure JMS queues, refer to the *Adapter for JMS User Guide*.

Table 62. JMS queues

JMS queue	Configuration set
Input queue	Set the `InputDestination` connector configuration property to the name of the JMS queue that the Adapter for JMS will poll for incoming messages. Make sure that the name of this queue is the same as the one WebSphere Partner Gateway is using as its JMS outbound queue. If this queue is not specified in `InputDestination`, the Adapter for JMS will *not* poll the queue. **Note:** The `InputDestination` property contains a comma-separated list of input queues. If the Adapter for JMS polls multiple queues, make sure that this list includes the name of the JMS queue that WebSphere Partner Gateway is using as its JMS outbound queue.
Output queue	At run-time, the collaboration can dynamically set the `OutputQueue` attribute in the dynamic meta-object to the name of the JMS queue that the Adapter for JMS will use to send its outgoing message.

You must make sure that the static or dynamic meta-objects are configured so that they can write to the queue on which the WebSphere Partner Gateway target is listening.

Creating business object definitions for JMS

The Adapter for JMS sends and receives your document to InterChange Server in the form of a payload business object. The Adapter for JMS invokes the payload data handler to handle this business object when it receives or sends a WebSphere Partner Gateway document, as follows:

- For request processing, the payload data handler converts the request business object to its corresponding JMS message.
- For event notification, the data handler converts the JMS message to an event business object.

Therefore, you must create the business object definitions shown in Table 63 to represent the payload business-object structure that the Adapter for JMS expects.

Table 63. Business object definitions for the Adapter for JMS

Condition	Business object definition	For more information
If you are using None or Backend Integration packaging for your message *and* your documents do *not* have attachments	Payload business object	"Creating the payload business-object structure for JMS" on page 166.
If you are using Backend Integration packaging for your document	Business objects to hold the message header information: • Dynamic meta-object • JMS-properties business object	"Creating JMS header information" on page 166.
If the document includes attachments	You must also create additional business objects to represent the attachments.	"Creating attachment-related business object definitions" on page 134

Creating the payload business-object structure for JMS

The structure of the payload business object for the JMS transport protocol depends on the packaging type, as follows:

- If your document uses None packaging, there are no special requirements to create the payload business object for a document sent over the JMS transport protocol.

  As discussed in "Business object for the document" on page 119, you must create an attribute for each piece of payload information you need to transfer.
- If your document uses Backend Integration packaging, you must take the following steps:
  1. Add to the payload business object definition a special attribute to identify the dynamic meta-object. This attribute provides configuration information for the transport-level headers of the message.
  2. In the business-object-level application-specific information, add the `cw_mo_conn` tag to identify the attribute that contains the dynamic meta--object.

     For more information on these steps, see "Creating JMS header information."

Note: For request processing, the JMS transport protocol can *only* support asynchronous interactions. You can send a request business object, but you *cannot* obtain a response. Therefore, you must create a request business object definition but not a business object definition for a response.

Creating JMS header information

If you are sending or receiving documents that use Backend Integration packaging over the JMS transport protocol, your request business object needs to contain custom transport-level header information. The Adapter for JMS expects this custom header information to be in its *dynamic meta-object*.

Figure 38 shows the business-object structure that the Adapter for JMS uses for a request business object representing a WebSphere Partner Gateway document that uses Backend Integration packaging.

Note: The *Adapter for JMS User Guide* provides information about this required business-object structure. Refer to this guide when defining your business object definitions.

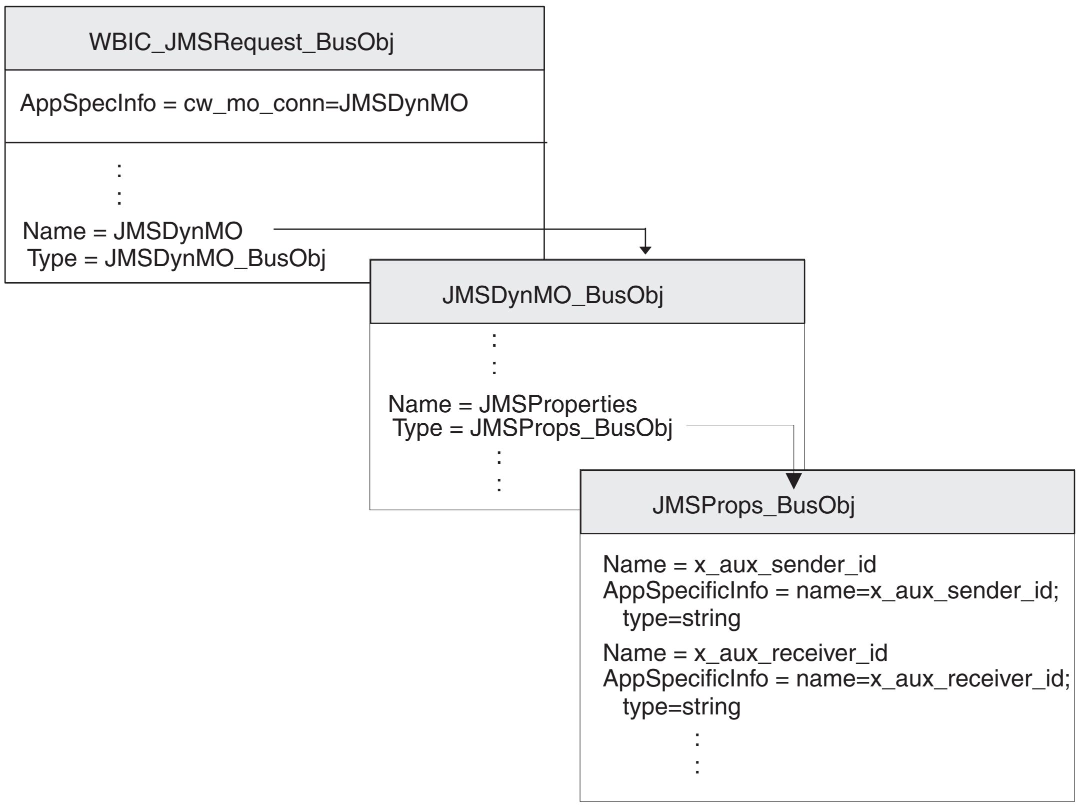

Figure 38. Relationship of the request business object to the JMS dynamic meta-object

Make sure your business-object structure includes a dynamic child meta-object by taking the following steps:

1. Create a business object definition to hold the JMS properties required by the Backend Integration packaging.
2. Create a business object definition for the dynamic meta-object.
3. Modify the business object definition for your request business object to include an attribute for the dynamic meta-object.

Each of these steps is described in the sections below.

Creating the JMS-properties business object

A *JMS-properties business object* contains JMS properties required for transport-level headings, which are needed by Backend Integration packaging. It can also contain the content-type attribute, which specifies the content-type header to set in the request message, and the content-length attribute, which specifies the length of the message, in bytes. Table 5 on page 20 describes each of the valid transport-header fields.

To create a JMS-properties business object definition, take the following steps:

1. Create an attribute within the business object definition for each of the transport-level header fields.

   All attributes should have an attribute type of String. For JMS messages, the names of transport-header fields use underscores instead of hyphens, as shown in Table 64 on page 168.

2. For each of the attributes in the JMS-properties business object definition, add application-specific information to identify the purpose of the associated attribute.

   This attribute-level application-specific information must have the following format:

   ```
   name=JMSproperty;type=string
   ```

   where *JMSproperty* is one of the values in the `JMS property name` column of Table 64.
3. For any of the attributes in the JMS-properties business object definition, you can add a default value to indicate the common (or only valid) value for that transport-level field.

Table 64. Attributes for the JMS-properties business object definition

Transport-header field	**JMS property name**
x-aux-sender-id	x_aux_sender_id
x-aux-receiver-id	x_aux_receiver_id
x-aux-protocol	x_aux_protocol
x-aux-protocol-version	x_aux_protocol_version
x-aux-process-type	x_aux_process_type
x-aux-process-version	x_aux_process_version
x-aux-create-datetime	x_aux_create_datetime
x-aux-msg-id	x_aux_msg_id
x-aux-production	x_aux_production
x-aux-system-msg-id	x_aux_system_msg_id
x-aux-payload-root-tag	x_aux_payload_root_tag
x-aux-process-instance-id	x_aux_process_instance_id
x-aux-event-status-code	x_aux_event_status_code
x-aux-third-party-bus-id	x_aux_third_party_bus_id
x-aux-transport-retry-count	x_aux_transport_retry_count
content-type	content_type
content-length	content_length

Note: Table 64 does *not* provide an exhaustive list of the headers required for back-end integration. For a complete list and description of the headers, see "Transport-level header content" on page 20. Make sure you substitute underscore characters for any hyphens in transport-header field names.

In Figure 38 on page 167, the `JMSProps_BusObj` business object definition contains attributes for the various transport-level header fields. These attributes all have attribute-level application-specific information to specify the name of the related protocol header. For example, the `x_aux_sender_id` attribute has the application-specific information set as follows:

```
name=x_aux_sender_id;type=string
```

Creating the JMS dynamic meta-object

This dynamic meta-object contains a child business object with configuration information for the Adapter for JMS. To create a business object definition for a dynamic meta-object, take the following steps:

1. Create an attribute named `JMSProperties`, whose attribute type is the business object definition for the JMS-properties business object (see "Creating the JMS-properties business object" on page 167).
2. Add other configuration properties as appropriate. For a list of valid attributes in the dynamic meta-object, see the *Adapter for JMS User Guide*. Consult this guide for information on how to create attributes to configure the dynamic meta-object.

For the Adapter for JMS to work with WebSphere Partner Gateway, the business object definition for the dynamic meta-object *must* include the attribute named `JMSProperties`, whose attribute type is the business object definition for the JMS-properties business object (see "Creating the JMS-properties business object" on page 167). For example, in Figure 38 on page 167, the `JMSDynMO_BusObj` business object definition contains attributes for various configuration properties (not shown) and includes the `JMSProperties` attribute.

Modifying the request business object definition

To incorporate the business-object structure into your request business object, you must make the following modifications to your request business object definition:

1. Add an attribute to your request business object definition to hold the dynamic child meta-object.

   The attribute type for this attribute is the business object definition for the dynamic meta-object (see "Creating the JMS dynamic meta-object" on page 168).
2. Add the `cw_mo_conn` tag to the business-object-level application-specific information of your request business object definition to identify the attribute that contains the dynamic meta-object.

   The `cw_mo_conn` tag has the following format:

   `cw_mo_conn=`*`dynamicMetaObjAttr`*

   where *dynamicMetaObjAttr* is the name of the attribute in the request business object that holds the dynamic meta-object.

For example, in Figure 38 on page 167, an attribute named `JMSDynMO` has been added to the request business object definition, `HUB_JMSRequest_BusObj`. This attribute contains the dynamic meta-object, which is a child business object of type `JMSDynMO_BusObj`. In addition, the application-specific information of the request business object has been modified to include the following `cw_mo_conn` tag to identify this dynamic meta-object:

```
cw_mo_conn=JMSDynMO
```

Creating ICS artifacts for JMS

To configure InterChange Server for communication with WebSphere Partner Gateway over the JMS transport protocol, you must create the InterChange Server artifacts shown in Table 65.

Table 65. ICS artifacts for communicating over the JMS transport protocol

ICS artifact	Purpose	For more information
Business object definitions	Represent the document	"Creating business object definitions for JMS" on page 165
Connector object	Represents the Adapter for JMS at run-time	"Creating the JMS connector object" on page 170

Table 65. ICS artifacts for communicating over the JMS transport protocol (continued)

ICS artifact	Purpose	For more information
Collaboration template and collaboration object	Represents the business process that InterChange Server uses to process the document	"Binding collaborations to communicate with Adapter for JMS"

Creating the JMS connector object

To obtain an instance of the Adapter for JMS at run-time, take the following steps within System Manager:

1. Create the connector objects:
   - Create a connector object to represent an instance of the Adapter for JMS.

     Note: In the Supported Business Objects tab of Connector Configurator, make sure that you specify all business object definitions you created for use with the Adapter for JMS. For a description of these business object definitions, see "Creating business object definitions for JMS" on page 165.
   - If required by your collaboration, create a connector object for the Port Connector.
2. Configure the connector objects.

   For information on how to configure your Adapter for JMS for use with WebSphere Partner Gateway, see "Configuring the Adapter for JMS" on page 163.

Binding collaborations to communicate with Adapter for JMS

As described in "Creating the collaborations" on page 122, a collaboration object must exist at run-time for InterChange Server to know where to receive and send business objects. When you create the collaboration object for the collaboration that uses the Adapter for JMS to send information to and receive it from WebSphere Partner Gateway, you bind the collaboration ports, as follows:

- For request processing, set the "to" port, which sends requests to WebSphere Partner Gateway, to the connector object you created for the Adapter for JMS; that is, the Adapter for JMS is the *destination* adapter.
- For event notification, set the "from" port, which receives events from WebSphere Partner Gateway, to the connector object you created for the Adapter for JMS; that is, the Adapter for JMS is the *source* adapter.

Part 4. Integrating with other back-end systems

Chapter 11. Integrating with WebSphere Business Integration Message Broker

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere Business Integration Message Broker.

Notes:

1. For a description of the general process used to integrate WebSphere Partner Gateway with a back-end system, see Chapter 2, "Planning for back-end integration," on page 9.
2. This chapter assumes that you are familiar with WebSphere Business Integration Message Broker and associated components, such as projects and message flows.

Often integration of WebSphere Partner Gateway with a back-end system is done by two separate people or roles. Each role configures a particular component, for which that role has expertise. Therefore, this chapter separates the integration with WebSphere Business Integration Message Broker into the configuration of WebSphere Partner Gateway and the configuration of Message Broker. Table 66 lists these configuration roles along with the places in this chapter to obtain the associated configuration information.

Table 66. Roles for Message Broker integration

Configuration role	For more information
Configuration of WebSphere Partner Gateway	1. "Planning for integration with Message Broker" on page 174 2. "Configuring WebSphere Partner Gateway for Message Broker" on page 175
Configuration of WebSphere Business Integration Message Broker	1. "Planning for integration with Message Broker" on page 174 2. "Configuring Message Broker" on page 178

Note: While each of these configuration roles can be performed separately, each also requires common information so that the two components can communicate.

This chapter provides the following information:

- "Planning for integration with Message Broker" on page 174
- "Configuring WebSphere Partner Gateway for Message Broker" on page 175
- "Configuring Message Broker" on page 178
- "Using HTTP transport protocol with Message Broker" on page 179
- "Sending SOAP documents" on page 182
- "Using JMS transport protocol with Message Broker" on page 183

Planning for integration with Message Broker

To plan for your integration to WebSphere Business Integration Message Broker, follow the steps outlined in Chapter 2, "Planning for back-end integration," on page 9. Table 67 summarizes the integration steps to integrate WebSphere Partner Gateway with Message Broker.

Table 67. Planning integration with WebSphere Business Integration Message Broker

Integration step	For more information
1. Confirm that you have a supported version of WebSphere Business Integration Message Broker installed and available to WebSphere Partner Gateway.	Chapter 11: "Message Broker versions that WebSphere Partner Gateway supports"
2. Determine the business protocol of the WebSphere Business Integration Message Broker document.	Chapter 2: "Which business protocol are you using?" on page 9
3. Determine the packaging type for the document: None or Backend Integration.	Chapter 2: "Which packaging will you use?" on page 19
4. Determine the message transport to use between WebSphere Partner Gateway and WebSphere Business Integration Message Broker.	Chapter 11: "Message transports that Message Broker supports"
5. Configure WebSphere Partner Gateway.	Chapter 11: "Configuring WebSphere Partner Gateway for Message Broker" on page 175

Message Broker versions that WebSphere Partner Gateway supports

WebSphere Partner Gateway supports integration with version 5.0 of Message Broker. Message Broker is available on several platforms, including Windows 2000 and several UNIX-based platforms. For more information, consult your installation guide for Message Broker in the WebSphere Business Integration Message Broker documentation set.

Message transports that Message Broker supports

The following two message-transport protocols are supported by WebSphere Business Integration Message Broker:

- HTTP transport protocol (including Web Services)
- JMS transport protocol

Support for these message transport protocols requires the installation and configuration of IBM WebSphere MQ.

HTTP

Message Broker uses the HTTP transport protocol for its Web Services transactions. You need no additional software to send and receive documents between WebSphere Partner Gateway and Message Broker over the HTTP protocol. However, to send the document out of Message Broker to some other destination, you need WebSphere MQ.

Note: WebSphere Partner Gateway supports both asynchronous and synchronous interactions with Message Broker over HTTP.

JMS

Message Broker uses the JMS transport protocol for most of its transactions. To send and receive documents between WebSphere Partner Gateway and Message Broker over the JMS transport protocol, you must use JMS queues. If these two components reside on different computers, you must create the JMS queues on each computer. Basically, support for JMS involves the use of a message flow within Message Broker and the underlying JMS queues. For more information on how to configure for JMS, see "Using JMS transport protocol with Message Broker" on page 183.

Note: WebSphere Partner Gateway supports only asynchronous interactions with Message Broker over JMS.

Support for Message Broker integration

WebSphere Partner Gateway provides samples to assist you in the integration process with Message Broker. These samples reside in the following subdirectory of the WebSphere Partner Gateway product directory:

```
Integration/WBI/WBIMB/samples
```

Configuring WebSphere Partner Gateway for Message Broker

A general overview of how to configure WebSphere Partner Gateway to communicate with a back-end system is provided in "Configuring WebSphere Partner Gateway" on page 37. This section summarizes the steps needed to configure WebSphere Partner Gateway to communicate with Message Broker.

Configuration of WebSphere Partner Gateway involves the following steps:

- Configuring for support of outgoing documents

  For information on sending documents from WebSphere Partner Gateway to Message Broker, see "Providing support for outgoing documents."
- Configuring for incoming documents

  For information on sending documents from Message Broker to WebSphere Partner Gateway, see "Providing support for incoming documents" on page 177.

Providing support for outgoing documents

For WebSphere Partner Gateway to send documents to any back-end system, you must perform the steps described in "Defining where to send the participant document" on page 39. When your back-end system is Message Broker, you need to create a gateway whose transport type matches the transport protocol used for messages between WebSphere Partner Gateway and Message Broker. When the hub sends a document to Message Broker, it must know where to route the document. This location must conform with the transport protocol being used. The transport protocol must be one that Message Broker supports (see "Message transports that Message Broker supports" on page 174).

The following sections summarize how to create gateways for the following transport protocols, which Message Broker supports:

- "Configuration for sending documents over the HTTP transport protocol" on page 176
- "Configuration for receiving documents over the JMS transport protocol" on page 176

Configuration for sending documents over the HTTP transport protocol

When the hub sends a document to Message Broker over the HTTP protocol, it routes the message through the defined gateway. This gateway identifies the URL where the document can be received by Message Broker. When Message Broker uses the HTTP protocol, it routes the document to the HTTPInput node of the message flow associated with the specified URL.

For the hub to be able to send documents through a gateway over the HTTP transport protocol, you must create a gateway from the Gateway Details page of the Community Console. This gateway must be configured to use the HTTP 1.1. transport protocol and to write to the URL on which the appropriate HTTPInput node is listening. As Table 68 shows, you provide this URL in the Target URI field of the gateway definition.

Note: An overview of how to create a gateway is provided in “Defining where to send the participant document” on page 39.

Table 68. HTTP values for Gateway Details page for communication with Message Broker

Target Details field	Value	Notes and restrictions
Target URI	The URL should be the same as the one configured for the HTTPInput node in the Message Broker message flow	Obtain this URL from the configuration of the message flow in the WebSphere Business Integration Message Broker integration .

Configuration for receiving documents over the JMS transport protocol

When the hub sends documents to Message Broker over the JMS protocol, it routes the document to the appropriate JMS queue, where it can be transferred to the JMS queue from which Message Broker can retrieve it. For the hub to obtain this JMS location, you must create a gateway in WebSphere Partner Gateway, one that uses the JMS transport protocol. This gateway must be configured to write to the queue whose contents are transferred to the queue on which Message Broker receives messages.

Note: For an overview of how to create a gateway, see “Defining where to send the participant document” on page 39.

For the hub to be able to send documents through a gateway over the JMS transport protocol, create a gateway from the Gateway Details page of the Community Console. When using WebSphere MQ, version 5.3 as your JMS provider, refer to the *Hub Configuration Guide* for the detailed steps. In addition, use the information specified in Table 69 for the JMS protocol in the Gateway Details page.

Table 69. JMS values for the Gateway Details page for communication with Message Broker

Gateway Details field	Value	Notes and restrictions
JMS Queue Name	Name of the JMS queue, on the computer where WebSphere Partner Gateway resides	Documents received on this queue are transferred to the JMS queue on the computer where Message Broker resides.

Providing support for incoming documents

For WebSphere Partner Gateway to receive messages from any back-end system, you must perform the steps described in "Defining where to retrieve the back-end document" on page 43. When your back-end system is Message Broker, you need to take the following steps:

1. As part of the participant profile for the Community Manager, define the gateway type and provide the associated IP address on which the Receiver will listen.
2. Create a target whose transport type matches the transport protocol used for documents between WebSphere Partner Gateway and Message Broker.

   For the hub to receive a document from Message Broker, it must know the location at which to retrieve the messages. This location must conform with the transport protocol to be used.

The following sections summarize how to create targets for transport protocols that Message Broker supports.

Configuring for incoming documents over HTTP transport protocol

When the hub receives a document over the HTTP transport protocol, its Receiver retrieves the document from the defined target. This target identifies the URL at which the Receiver listens for documents from Message Broker. When Message Broker uses the HTTP transport protocol, the HTTPRequest node sends the document to the appropriate URL, where it can be received by the hub.

For the hub to receive documents through a target over the HTTP transport protocol, you must create a target from the Target List page of the Community Console. This target must use the HTTP 1.1 transport protocol. The hub determines this URL as a combination of the following information:

- The IP address of the host computer, obtained from within the Community Manager's participant profile
- The target URL, obtained from the URL field of the target definition

Note: An overview of how to create a target is provided in "Defining where to retrieve the back-end document" on page 43.

For Message Broker to be able to send documents to this target, the HTTPRequest node of the message flow must be configured to send documents to this URL. Therefore, you must ensure that this target URL is available to the Message Broker configuration.

Configuring for incoming documents over JMS transport protocol

When the hub receives documents from Message Broker over the JMS protocol, it obtains the document from the appropriate JMS input queue, where it has been transferred from the JMS output queue where Message Broker has sent it. For the hub to be able to obtain this JMS location, you must create a target in WebSphere Partner Gateway, one that uses the JMS transport protocol. Through the target, the hub listens for any documents on its input queue and retrieves them.

Note: For an overview of how to create a target, see "Defining where to retrieve the back-end document" on page 43.

For the hub to receive documents through a target over the JMS transport, you must create a target from the Target List page of the Community Console. When using WebSphere MQ, version 5.3 as your JMS provider, refer to the *Hub Configuration Guide* for the detailed steps. In addition, use the information specified in Table 70 for the JMS protocol in the Target Details page.

Table 70. JMS values for the Target Details page for communication with Message Broker

Target Details field	Value	Notes and restrictions
JMS Queue Name	Name of the JMS input queue that receives documents from the output queue of Message Broker	Documents in this input queue are transferred from the JMS output queue on the computer where Message Broker resides

Configuring Message Broker

For your interactions between WebSphere Partner Gateway and Message Broker, you must create a Message flow project within the Broker Application Development Perspective of the Message Brokers Toolkit. This project will include the following artifacts:

- Message flows
- PIP files (RosettaNet only) or message definition files

Note: For more information on how to create message flow projects, see the WebSphere Business Integration Message Broker documentation set.

Creating the message flow

It is the *message flow*, within Message Broker, that performs the actual business logic you need to process information. Therefore, the appropriate message flows must exist for Message Broker to correctly process your WebSphere Partner Gateway documents. Make sure that a message flow exists that provides the business logic you need:

- If such a message flow does *not* currently exist, you must create or import one.
- If a message flow does exist, you must understand how to use it.

For Message Broker to handle incoming and outgoing documents, its message flow uses special transport nodes. The type of transport node to use depends on the particular transport protocol, as shown.

Table 71. Creating message flows for different transport protocols

Transport protocol	For more information
HTTP HTTP (SOAP documents)	"Creating the message flow for HTTP transport" on page 180
JMS	"Creating the message flow for JMS transport" on page 188

Deploying the project

After your message flow project contains the correct artifacts, you must deploy it to Message Broker. You deploy a message flow project with the Broker Administrator Perspective of the Message Brokers Toolkit.

Using HTTP transport protocol with Message Broker

This section describes how to send and receive documents between WebSphere Partner Gateway and WebSphere Business Integration Message Broker through the use of the HTTP transport protocol.

Note: All references to the HTTP transport protocol apply to HTTPS as well.

Components required for documents over HTTP transport

You need no additional software to send or receive a document between WebSphere Partner Gateway and Message Broker using the HTTP transport protocol. Only WebSphere Partner Gateway and Message Broker are required. For WebSphere Partner Gateway to communicate with version 5.0 of Message Broker using the HTTP transport protocol, these two components must be configured. Table 72 summarizes these configuration steps.

Table 72. Configuring WebSphere Partner Gateway and Message Broker

Component	Version	For more information
WebSphere Partner Gateway	6.0	"Configuration for sending documents over the HTTP transport protocol" on page 176 "Configuring for incoming documents over HTTP transport protocol" on page 177
WebSphere Business Integration Message Broker	5.0	"Configuring Message Broker" on page 178

In addition, to send or receive a document between WebSphere Partner Gateway and Message Broker using the HTTP transport protocol, you must use the version 5.3 IBM WebSphere MQ as your JMS provider.

Sending documents over HTTP transport

For WebSphere Partner Gateway to send a document to Message Broker over the HTTP transport protocol, you use special HTTP-transport nodes within the Message Broker message flow to retrieve the document that WebSphere Partner Gateway has sent as an HTTP stream. The nodes of the message flow perform the computations required and then route the document to some destination (a JMS output queue).

The following steps describe how WebSphere Partner Gateway sends a document to a message flow within Message Broker over the HTTP transport protocol:

1. WebSphere Partner Gateway sends an HTTP message to Message Broker.

   If the packaging type of the document was Backend Integration, WebSphere Partner Gateway has provided custom properties in this message.

   Note: Within WebSphere Partner Gateway, you must configure a gateway that identifies the URL to which WebSphere Partner Gateway sends the message and on which Message Broker is polling. For more information, see "Configuration for sending documents over the HTTP transport protocol" on page 176.
2. The HTTPInput node of the message flow picks up the document and sends it to the next node of the message flow. This node is usually a compute node.
3. The nodes of the message flow perform the business logic.

When business logic is complete, the message flow sends the resulting document to its HTTPReply node.

4. The HTTPReply node, by default, sends back the output message to the client (WebSphere Partner Gateway).

   Alternatively, the message flow can put the message into an MQOutput node. The MQOutput node receives the document and sends it to the appropriate JMS queue or other application.

Receiving documents over HTTP transport

For WebSphere Partner Gateway to receive a document from Message Broker using the HTTP transport protocol, you use special HTTP-transport nodes within the Message Broker message flow to send the document that WebSphere Partner Gateway is to receive as an HTTP stream. The nodes of the message flow perform the computations required and handle the request and response (if the interaction is synchronous) with WebSphere Partner Gateway.

The following steps describe how WebSphere Partner Gateway receives a document from a message flow within Message Broker over the HTTP transport protocol:

1. The message flow within Message Broker receives a document in its MQInput node (a JMS input queue).
2. The MQInput node of the message flow receives the document and sends it to the HTTPRequest node.
3. The HTTPRequest node handles the request and response interactions with the client (WebSphere Partner Gateway), using a specified URL.
4. WebSphere Partner Gateway receives the message from its URL, as configured in its target.

   For more information on the target, see "Configuring for incoming documents over HTTP transport protocol" on page 177.

Creating the message flow for HTTP transport

For a Message Broker message flow to handle documents over the HTTP transport protocol, it uses the following transport nodes:

- HTTPInput
- HTTPReply
- HTTPRequest

The order of use for these transport nodes depends on the direction of communication, as follows:

- When WebSphere Partner Gateway *sends* a document to Message Broker, the message flow includes the types of nodes in Table 73 (in the order shown) to describe the business logic.
- When WebSphere Partner Gateway *receives* a document from Message Broker, the message flow includes the types of nodes in Table 74 (in the order shown) to describe the business logic.

Table 73. Nodes for sending documents to Message Broker over HTTP

Node type	Purpose	Notes
HTTPInput	Receives the WebSphere Partner Gateway request document into the message flow	Set this transport node URL Selector field (in the Basic properties) to the URL where WebSphere Partner Gateway sends its documents (the URL configured in the WebSphere Partner Gateway target).The URL should have the following format: `http://hostName:port/path` where *hostName* is the name of the computer on which Message Broker resides, *port* is the HTTP port number on which the Message Broker is listening, and *path* identifies the location on this computer. For more information, see "Configuration for sending documents over the HTTP transport protocol" on page 176.
Compute	Performs business-logic tasks, such as updating header information	Use ESQL to perform the business logic. The compute node sends the resulting message to the HTTPReply node.
HTTPReply	Returns a response to WebSphere Partner Gateway	By default, this node sends the output message to the client. However, you can configure it to send it to an MQOutput node.
MQOutput	Receives the document from the HTTPReply node and sends it to WebSphere Partner Gateway	This transport node sends the resulting document to a JMS output queue, which routes it to its next destination.

Table 74. Nodes for receiving documents from Message Broker over HTTP

Node type	Purpose	Notes
MQInput	Receives the document from WebSphere Partner Gateway	This transport node receives the incoming document from a JMS input queue.
HTTPRequest	Handles request/response interactions with WebSphere Partner Gateway	This transport node must set its Web Services URL field (in the Basic Properties) to the URL where WebSphere Partner Gateway is listening for documents (the URL configured in the WebSphere Partner Gateway target). The URL should have the following format: `http://hostName:port/bcgreceiver/path` where *hostName* is the name of the computer on which WebSphere Partner Gateway resides, *port* is the HTTP port number on which the WebSphere Partner Gateway Receiver is listening, and *path* identifies the location on this computer. For more information, see "Configuring for incoming documents over HTTP transport protocol" on page 177.

For more detailed information on how to create and configure message flow nodes, see your WebSphere Business Integration Message Broker documentation.

Sending SOAP documents

SOAP documents differ from other types of documents exchanged over HTTP/S. This section describes how to send and receive SOAP documents between WebSphere Partner Gateway and WebSphere Business Integration Message Broker over the HTTP transport protocol.

The way to configure WebSphere Partner Gateway and Message Broker for the transfer of SOAP documents is very similar to the configuration for transferring non-SOAP documents over the HTTP protocol. Table 75 summarizes where to find information on how to configure these two integration components.

Table 75. Configuring WebSphere Partner Gateway and Message Broker for transfer of SOAP documents

Integration component	**Configuration step**	**For more information**
WebSphere Partner Gateway	You configure the target and gateway the same way for SOAP documents as for non-SOAP documents over HTTP.	"Configuration for sending documents over the HTTP transport protocol" on page 176 "Configuring for incoming documents over HTTP transport protocol" on page 177
WebSphere Business Integration Message Broker	The message flows to handle SOAP documents are very similar to those for non-SOAP documents over HTTP. Only one additional transport node is required to handle SOAP documents.	For sending a SOAP document to Message Broker, see Table 76. For receiving a SOAP document from Message Broker, see "Creating the message flow for HTTP transport" on page 180.

For Message Broker to correctly process a SOAP document that WebSphere Partner Gateway sends, the message flow must contain an HTTPRequest node to handle communication with the Web Services client. Table 76 lists the nodes in a Message Broker message flow needed to handle a SOAP document sent by WebSphere Partner Gateway.

Table 76. Nodes for sending SOAP documents to Message Broker

Node type	**Purpose**	**Notes**
HTTPInput	Receives the WebSphere Partner Gateway request document into the message flow	Set this transport node URL Selector field (in the Basic properties) to the URL where WebSphere Partner Gateway sends its documents (the URL configured in the WebSphere Partner Gateway gateway). The URL should have the following format: `http://hostName:port/path` where *hostName* is the name of the computer on which WebSphere Partner Gateway resides, *port* is the HTTP port number on which the WebSphere Partner Gateway Receiver is listening, and *path* identifies the location on this computer. For more information, see "Configuration for sending documents over the HTTP transport protocol" on page 176.
Compute	Performs business-logic tasks, such as updating header information	Use ESQL to perform the business logic. The compute node sends the resulting message to the HTTPReply node.

Table 76. Nodes for sending SOAP documents to Message Broker (continued)

Node type	Purpose	Notes
HTTPRequest	Sends the SOAP request to the external Web Service Provider (WebServices) and gets back a response from that Web Service.	None
HTTPReply	Returns a response to WebSphere Partner Gateway	By default, this node sends the output message to the client.

Using JMS transport protocol with Message Broker

This section describes how to configure components to send and receive documents between WebSphere Partner Gateway and WebSphere Business Integration Message Broker through the use of the JMS transport protocol. It provides the following information on how to send and receive documents:

- "Components required for documents over JMS transport"
- "How documents are sent over the JMS transport" on page 158
- "How documents are received over the JMS transport" on page 160

Components required for documents over JMS transport

To send or receive a document between WebSphere Partner Gateway and version 5.0 Message Broker using the JMS transport protocol, WebSphere MQ must be the JMS provider. The following sections describe how WebSphere Partner Gateway, Message Broker, and WebSphere MQ work together to exchange documents over the HTTP transport protocol.

How documents are sent over the JMS transport

For WebSphere Partner Gateway to send a document to Message Broker using the JMS transport protocol, you use the JMS queues. WebSphere Partner Gateway sends a document to its JMS output queue, where it is transferred to the JMS input queue on which Message Broker listens. When Message Broker receives a document, it retrieves it from its input queue. Message Broker's message flow contains special WebSphere MQ (JMS) transport nodes, which handle access to the JMS queues. Figure 39 provides an overview of how WebSphere Partner Gateway sends documents to Message Broker over the JMS transport protocol.

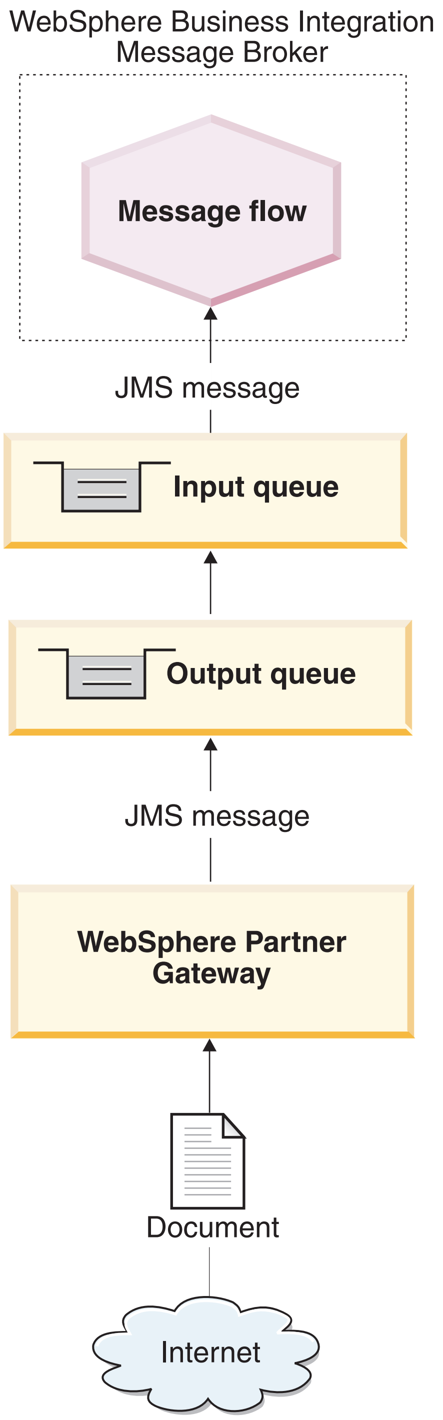

Figure 39. Message flow from WebSphere Partner Gateway to a message flow through the JMS transport protocol

The following steps describe how WebSphere Partner Gateway sends a document to a message flow within Message Broker over the JMS transport protocol:

1. WebSphere Partner Gateway posts a message to its JMS output queue.

   If the packaging type of the document was Backend Integration, WebSphere Partner Gateway has provided custom properties in this message. The JMS message header, JMSType, is set with the content type of the payload.

Note: Within WebSphere Partner Gateway, you must configure a gateway that identifies the JMS output queue to which WebSphere Partner Gateway sends the message and on which Message Broker is polling. For more information, see “Configuration for receiving documents over the JMS transport protocol” on page 176.

2. WebSphere MQ transfers the document from the output queue on the computer where WebSphere Partner Gateway resides to the input queue that Message Broker is polling.
3. When Message Broker sees a message on its input queue, it retrieves the message and sends it to the appropriate message flow.

   For information on how to set up this queue, see “Setting up the environment for JMS transport” on page 187.
4. The MQInput node sends the document to the next node of the message flow. This node is usually a compute node.
5. The nodes of the message flow perform the business logic.

   When business logic is complete, the message flow sends the resulting document to its MQOutput node.
6. The MQOutput node sends the document to the appropriate queue.

How documents are received over the JMS transport

For WebSphere Partner Gateway to receive a document from Message Broker over the JMS transport protocol, you use JMS queues. Message Broker sends a document to its JMS output queue, where it is transferred to the JMS input queue on which WebSphere Partner Gateway listens. When WebSphere Partner Gateway receives a document, it retrieves it from its input queue. Message Broker’s message flow contains special WebSphere MQ (JMS) transport nodes, which handle access to the JMS queues. Figure 40 provides an overview of how documents are sent from Message Broker to WebSphere Partner Gateway.

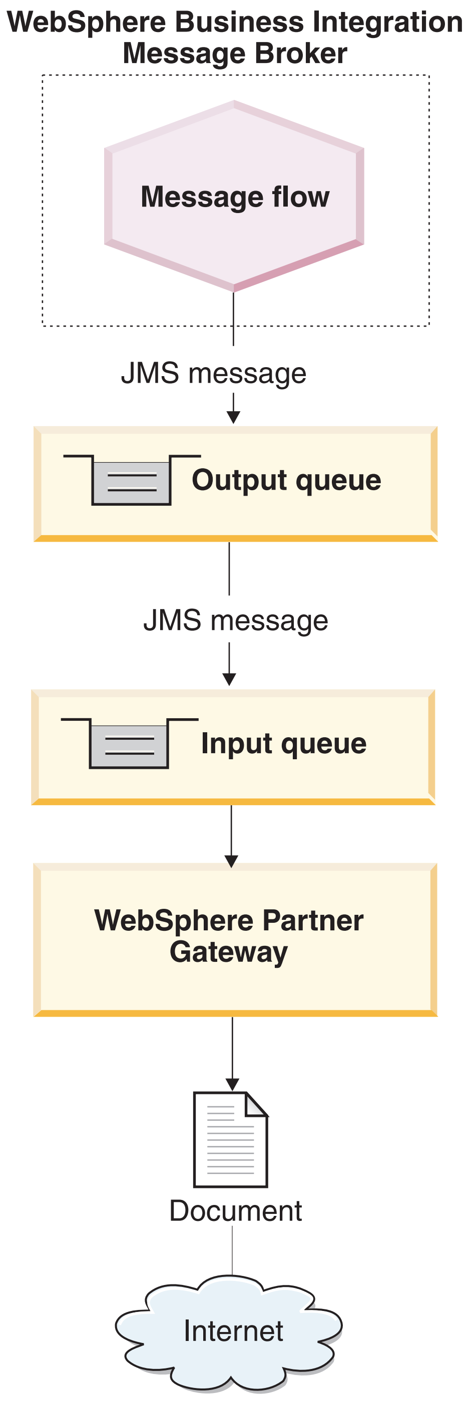

Figure 40. Message flow from a message flow to WebSphere Partner Gateway through the JMS transport protocol

The following steps describe how WebSphere Partner Gateway receives a document from a message flow within Message Broker over the JMS transport protocol:

1. The message flow within Message Broker receives a document in its MQInput node.

   The message flow receives its incoming message from a JMS input queue.
2. The MQInput queue of the message flow receives the document and sends it to the next node of the message flow. This node is usually a compute node.

3. The nodes of the message flow perform the business logic.

   When business logic is complete, the message flow sends the resulting document to its MQOutput node.
4. The MQOutput node sends the document to the appropriate JMS output queue.
5. WebSphere MQ transfers the document from the queue on the computer where Message Broker resides to the queue that WebSphere Partner Gateway is polling.
6. WebSphere Partner Gateway receives the message from its JMS input queue, as configured in its target.

   For more information on the target, see "Configuring for incoming documents over JMS transport protocol" on page 177. For information on how to set up this queue, see "Setting up the environment for JMS transport."

Setting up the environment for JMS transport

The sending and receiving of documents to and from Message Broker involves JMS queues (remote and local). For information on how to configure WebSphere Partner Gateway for use with Message Broker over JMS, see "Configuring WebSphere Partner Gateway for Message Broker" on page 175. To use the JMS transport protocol with Message Broker, you can set up the JMS system that WebSphere MQ provides. Version 5.0 of Message Broker uses version 5.3 of WebSphere MQ as a JMS provider. You can use the steps in the *Hub Configuration Guide* to set up the JMS transport-protocol mechanism.

Important: The steps in the *Hub Configuration Guide* must be performed on the computer on which WebSphere Partner Gateway resides. This guide assumes that the JMS transport-mechanism required by Message Broker has already been set up as part of the Message Broker installation.

When you create your JMS queues for use between WebSphere Partner Gateway and Message Broker, consider the following points:

- Part of the Message Broker installation process should involve the creation of the following queue managers:
  - A WebSphere MQ queue manager associated with the broker domain

    You can use the following command to create this queue manager and a set of named queues:

    ```
    mqsicreatebroker
    ```
  - A WebSphere MQ queue manager for Message Broker

    Because Message Broker uses a set of predetermined queue names, it requires a separate WebSphere MQ queue manager per broker. Message Broker can share this queue manager hosting with either its Configuration Manager or the optional User Name Server, or both.

  For more information, refer to your *WebSphere Business Integration Message Broker Installation and Configuration Guide*.
- When you create your JMS queue aliases, you might want to name them to indicate the direction of flow between WebSphere Partner Gateway and Message Broker.

  For example, if you create the queues listed in the `Original queue name` column of Table 61, you could rename these queues to indicate the direction of flow, as shown in the `Directional queue name` column of Table 77.

Table 77. Naming JMS queues for Message Broker to indicate direction

Original queue name	Directional queue name
inQ	MB2HUB
outQ	HUB2MB

Creating the message flow for JMS transport

For a Message Broker message flow to handle documents over the JMS transport protocol, it uses the following transport nodes:

- MQInput
- MQOutput

The order of use for these transport nodes depends on the direction of communication, as follows:

- When WebSphere Partner Gateway *sends* a document to Message Broker, the message flow includes the types of nodes in Table 78 (in the order shown) to describe the business logic.
- When WebSphere Partner Gateway *receives* a document from Message Broker, the message flow includes the types of nodes in Table 79 (in the order shown) to describe the business logic.

Table 78. Nodes for sending documents to Message Broker over JMS

Node type	Purpose	Notes and restrictions
MQInput	Receives the document from WebSphere Partner Gateway	The value in the Queue Name field (in the Basic properties) of this transport node is the message flow input queue. WebSphere MQ must be set up so that this JMS queue receives documents from the output queue of WebSphere Partner Gateway. For more information, see "Configuration for receiving documents over the JMS transport protocol" on page 176.
Compute	Performs business-logic tasks, such as removing header information	None
MQOutput	Receives the document from the compute node and sends it as the message-flow output	This transport node sends the resulting document to a JMS output queue, which routes it to its next destination.

Table 79. Nodes for receiving documents from Message Broker

Node type	Purpose	Notes and restrictions
MQInput	Receives the document into the message flow	This transport node receives the incoming document from a JMS input queue.
Compute	Performs business-logic tasks, such as updating header information	None
MQOutput	Receives the document from the compute node and sends it to WebSphere Partner Gateway	The value in the Queue Name field (in the Basic properties) of this transport node is the message flow output queue. WebSphere MQ must be set up so that this JMS queue sends documents to the input queue of WebSphere Partner Gateway. For more information, see "Configuring for incoming documents over JMS transport protocol" on page 177.

For more detailed information on how to create and configure message flow nodes, see your WebSphere Business Integration Message Broker documentation.

Chapter 12. Integrating with WebSphere Data Interchange

This chapter describes how to integrate WebSphere Partner Gateway with WebSphere Data Interchange.

Note: For a description of the general process used to integrate WebSphere Partner Gateway with a back-end system, see Chapter 2, "Planning for back-end integration," on page 9.

Who should read this chapter

WebSphere Partner Gateway now includes EDI processing capabilities (such as de-enveloping and transformation) that are similar to those found in WebSphere Data Interchange. You might be able to use these EDI capabilities in place of WebSphere Data Interchange, in which case you would not need the information in this chapter. Refer to the *Hub Configuration Guide* for information on setting up the hub to process EDI documents.

This chapter, therefore, is intended for those who:

- Were using a previous version of WebSphere Partner Gateway (formerly known as WebSphere Business Integration Connect) and want to continue to integrate with WebSphere Data Interchange
- Need the advanced capabilities of WebSphere Data Interchange not included in the WebSphere Partner Gateway EDI support

If you want to use WDI, follow the steps outlined in this chapter. Be aware that the steps for configuring EDI have changed in WebSphere Partner Gateway, version 6.0. You now select a specific document flow for the version of EDI you are using. For example, if you are exchanging EDI-X12 documents, you select ISA (instead of ALL) for Document Flow.

Resources you can use with this chapter

This chapter provides an explanation of the process by which documents are exchanged and then lists the steps for setting up a sample environment for such exchanges. The scenario used throughout this chapter is similar to the one presented in the *Integrating WebSphere Data Interchange V3.2 with WebSphere Business Integration Connect V4.2* tutorial, which is available on the following Web site:

```
www.ibm.com/developerworks/websphere/
```

The tutorial provides additional scripts (in the section on configuring WebSphere MQ) as well as sample transformation maps. By following the tutorial, you can set up the environment described in this chapter.

Note: The tutorial describes integration with WebSphere Business Integration Connect, version 4.2; however, the same steps, with slight modifications, apply to WebSphere Partner Gateway.

It is assumed that you are familiar with using WebSphere Data Interchange. See the WebSphere Data Interchange documentation for additional information as you read this chapter.

Introduction

WebSphere Data Interchange integrates electronic data interchange (EDI) into the WebSphere business process, messaging, and Internet-based B2B capabilities. You exchange documents and messages between WebSphere Partner Gateway and WebSphere Data Interchange through the JMS transport protocol. You must specify a packaging of None when sending a document to WebSphere Data Interchange.

Note: WebSphere Data Interchange provides other types of integration options, such as file-based integration. Refer to the WebSphere Data Interchange documentation for details on enabling the exchange of documents through file-based integration.

How documents are sent to WebSphere Data Interchange

For WebSphere Partner Gateway to send an EDI document to WebSphere Data Interchange, the following steps occur:

1. A community participant sends an EDI document to WebSphere Partner Gateway. The document is sent with a specific packaging over a transport protocol (in this example, AS2 packaging over HTTP). WebSphere Partner Gateway strips off the AS2 packaging from the EDI document.
2. WebSphere Partner Gateway places the EDI document on a queue.

   Note: WebSphere Partner Gateway determines the protocol used in the document by examining the first three characters of the EDI document. It then determines, from the protocol type, the sender and receiver information. See “Overview of EDI routing” on page 213 for details.
3. WebSphere Data Interchange reads the EDI document from the queue. It performs the tasks of unwrapping, validating, and translating the EDI document.

   Note: WebSphere Data Interchange must be configured with the necessary maps, trading partner profiles, and other information. See the WebSphere Data Interchange documentation for details.
4. WebSphere Data Interchange distributes the document to a back-end system. If the back-end system is WebSphere InterChange Server, WebSphere Data Interchange sends the document to the WebSphere Business Integration Adapter for MQ to create a business object and invoke a collaboration within InterChange Server.

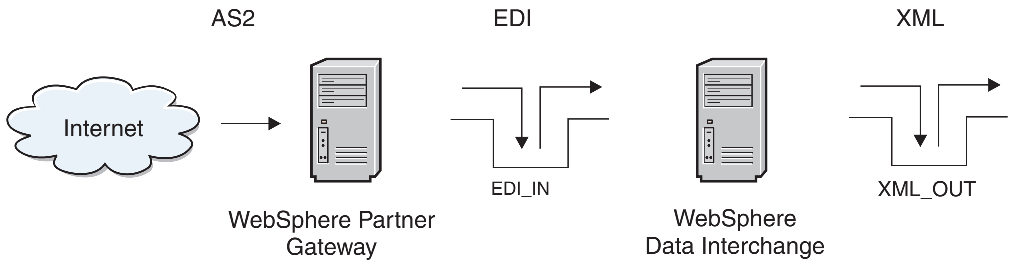

Figure 41. EDI document from WebSphere Partner Gateway

In Figure 41, a community participant sends an EDI document in AS packaging to WebSphere Partner Gateway, which, in turn, sends it to the EDI_IN queue on the WebSphere Data Interchange side. Note that the remote queue, transmission queue, receiver queue (in the example, EDI_IN), and the sender and receiver channels must be set up so that the message sent to WebSphere Partner Gateway is transmitted to the EDI_IN queue. The WebSphere Data Interchange server picks up the EDI document, searches for the user profiles, mappings, and so on, converts the document to XML, and puts it in the XML_OUT queue.

How documents are received from WebSphere Data Interchange

For WebSphere Partner Gateway to receive an EDI document from WebSphere Data Interchange, the following steps occur:

1. WebSphere Data Interchange places the EDI document on a queue.
2. WebSphere Partner Gateway reads the message from the queue.

   Note: WebSphere Partner Gateway determines how to route the document as described in "Overview of EDI routing" on page 213.
3. WebSphere Partner Gateway routes the document to the appropriate community participant.

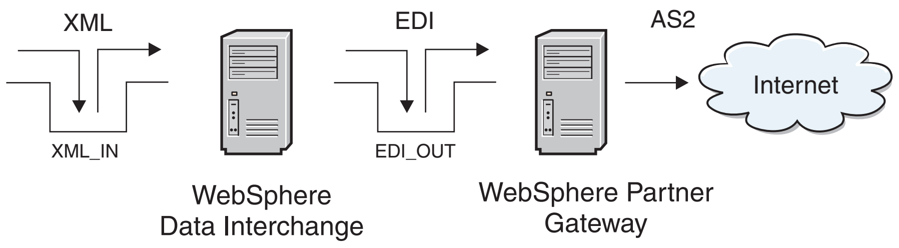

Figure 42. Sending an EDI document to WebSphere Partner Gateway

In Figure 42, an XML document is placed into the XML_IN queue for WebSphere Data Interchange to translate. It is assumed that the user profiles, mappings, and so on, are already performed. Upon receiving a valid XML document, WebSphere Data Interchange converts it into EDI format and places the output in the EDI_OUT queue (a remote queue). It is also assumed that the transmission queue, sender and receiver channels, and receiver queue on the WebSphere Partner Gateway side are set up. Upon receiving the document, WebSphere Partner Gateway routes it to the community participant.

Example scenario used in this chapter

Throughout this chapter, you will see the steps to set up the exchange of EDI documents between two trading partners. The EDI documents are sent over the internet, and AS2 (over HTTP) is used as the communication protocol.

In this sample, the trading partners are Partner One and Partner Two. Figure 43 illustrates the configurations of the two partners.

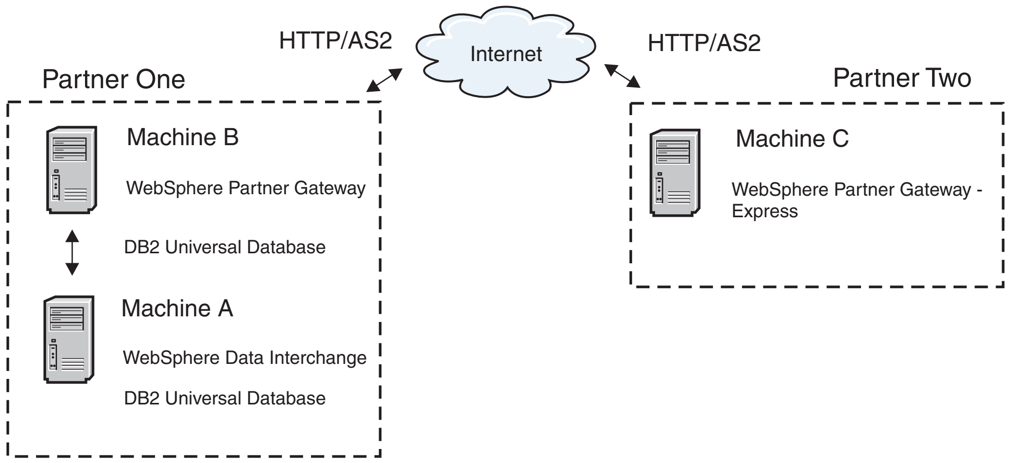

Figure 43. Configuration of two partners in example scenario

The three computers have the following software installed:

- Computer A contains WebSphere Data Interchange Server 3.2 and WebSphere Data Interchange Client 3.2 along with its prerequisite software.
- Computer B contains WebSphere Partner Gateway Enterprise Edition along with its prerequisite software.
- Computer C contains WebSphere Partner Gateway - Express.

Refer to the *WebSphere Partner Gateway Installation Guide* and to the WebSphere Data Interchange documentation for a complete list of software prerequisites.

In this example, partnerOne is operating two computers. Computer A has both WebSphere MQ and WebSphere Data Interchange Server installed. Computer B has WebSphere MQ as well as WebSphere Partner Gateway Enterprise Edition installed. Computer B supports the communications between the two trading partners.

WebSphere Data Interchange supports integration with WebSphere MQ, enabling interoperation with a wide range of enterprise applications and business process engines. WebSphere Partner Gateway employs WebSphere MQ as a JMS provider. As such, integration between WebSphere Data Interchange and WebSphere Partner Gateway is through MQ messages destined for JMS API clients.

WebSphere Partner Gateway is used to communicate EDI transactions over the Internet using the AS2 protocol.

Note that, in this example, partnerTwo is using WebSphere Partner Gateway - Express to accept transactions via AS2 and has its own WebSphere Data Interchange environment for handling translations and acknowledgments.

Throughout this chapter, you will see the details about configuring the computers used in this sample scenario. The flow of messages is bi-directional, and so both send and receive artifacts are included.

Configuring your environment for message exchange

To enable communication between WebSphere Data Interchange and WebSphere Partner Gateway, you perform the following setup and configuration tasks:

- "Configuring WebSphere MQ communication"
- "Configuring WebSphere Data Interchange" on page 196
- "Setting up the JMS environment" on page 201
- "Configuring WebSphere Partner Gateway Enterprise Edition" on page 202

Configuring WebSphere MQ communication

The first step in setting up the environment is to configure WebSphere MQ intercommunication. Intercommunication means sending messages from one queue manager to another. The first step is to define a queue manager (and associated objects) for the WebSphere Data Interchange system and the WebSphere Partner Gateway system. If you will be sending messages in both directions, you set up a source queue manager and a target queue manager on both systems. On the source queue manager, you define a sender channel, a remote queue definition, and a transmission queue. On the target queue manager, you define a receiver channel and a target queue.

Note: Refer to the WebSphere MQ documentation for additional details on defining queue managers.

This section shows you the values you would use to set up the queue managers and associated objects needed for the sample scenario. In the scenario, WebSphere MQ V5.3 is installed on both Computer A and Computer B. The first step, then, is to create a queue manager on both Computer A and Computer B for use with WebSphere Data Interchange and WebSphere Partner Gateway Enterprise Edition respectively.

Note: Your WebSphere Data Interchange queue manager should be configured to trigger the WebSphere Data Interchange Server using the WDI Adapter application.

- On Computer A, you would use the queue manager defined for use with WebSphere Data Interchange. For the remainder of this chapter, this queue manager is referred to as WDI32_QM.
- On Computer B, you would use the queue manager created during the initial installation and configuration of WebSphere Partner Gateway Enterprise Edition. For the remainder of this chapter, this queue manager is referred to as HUB_QM

To send messages from one queue manager to another using WebSphere MQ, you define the following objects:

- On the source queue manager:
  - Sender channel
  - Remote queue definition
  - Transmission queue
- On the target queue manager:
  - Receiver channel
  - Target queue

In the sample scenario, both Computer A and Computer B act as sender and receiver. Therefore, you would have to define a number of objects on each computer.

Table 80 lists the objects you would create to set Computer A and Computer B as sender and receiver.

Table 80. WebSphere MQ objects to create

WebSphere MQ object	Computer A	Computer B
Queue Manager	WDI32_QM	HUB_QM
Sender Channel	TO.HUB60	TO.WDI32
Receiver Channel	TO.WDI32	TO.HUB60
Remote Queue	EDI_OUT_A	EDI_OUT_B
Transmission Queue	XMITQ_A	XMITQ_B
Local Queue	EDI_IN_A	EDI_IN_B
Local Queue	XML_IN_A	XML_IN_B
Local Queue	XML_OUT_A	XML_OUT_B

Figure 44 shows the message flow between Computer A and Computer B, indicating the role of the WebSphere MQ objects listed in Table 80.

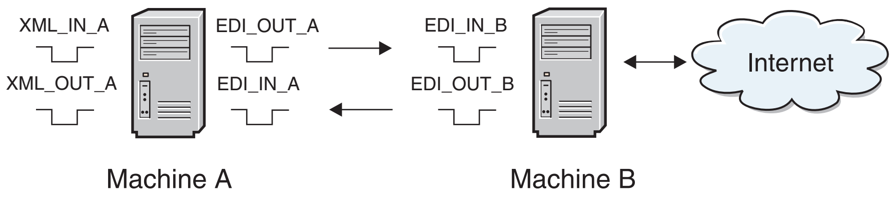

Figure 44. Message flow between Computer A and Computer B

You could use several different methods to define these objects, depending on your WebSphere MQ platform. For example, you could use WebSphere MQ Explorer on Windows to define the objects.

Configuring WebSphere Data Interchange

For WebSphere Data Interchange to receive messages from the WebSphere MQ queue and write EDI messages to a queue, you must configure profiles in the WebSphere Data Interchange Client. Using WebSphere Data Interchange Client, you would create the following profiles, which are described in the sections that follow:

- MQ Series queue profile
- Network profile
- Mailbox profile
- Service profile

In the sample scenario, WebSphere Data Interchange receives XML messages from the WebSphere MQ queue XML_IN_A and writes the result of translation to WebSphere MQ queue EDI_OUT_A. This is called the XML-to-EDI translation. WebSphere Data Interchange also receives EDI from WebSphere Partner Gateway Enterprise Edition on the WebSphere MQ queue EDI_IN_A and writes the result of translation to XML_OUT_A.

MQSeries$^{(R)}$ Queue profile

An MQSeries Queue profile contains information about a WebSphere MQ message queue. Table 81 shows the properties to configure for each profile.

Table 81. Properties in an MQSeries Queue profile

MQ property	Description
`Queue Profile ID`	The unique identifier to name the profile (logical name)
`Full Queue Name`	The actual name of the WebSphere MQ queue
`Queue Manager Name`	The actual name of the WebSphere MQ queue manager
`Description`	Any string to identify the purpose of the profile
`Maximum Length`	The largest possible message for the queue as configured in WebSphere MQ
`Destructive Reads`	If selected, these cause WebSphere Data Interchange to remove the message from the WebSphere MQ queue when reading.
`Syncpoint Control`	When checked, the reading and writing of queue messages is under syncpoint control. If syncpoint control is in effect, modifications to a message queue do not take place until WebSphere Data Interchange issues a syncpoint.

Because you are working with the WebSphere MQ queues, you require an MQSeries Queue profile in WebSphere Data Interchange for *each* queue. In all, you would create four MQSeries Queue profiles, one for each WebSphere MQ queue used in the message flow. From the setup area of WebSphere Data Interchange Client, you would:

1. Create an MQSeries Queue profile for XML_IN_A and EDI_OU_A.

   Table 82 lists the actual parameters specified in each MQSeries Queue profile you created. The queues represented here are used with XML-to-EDI translation.

Table 82. MQSeries Queue profile for XML_IN_A and EDI_OU_A

Queue property	Value for XML_IN_A	Value for EDI_OU_A
`Queue Profile ID`	`XML_IN_A`	`EDI_OU_A`
`Full Queue Name`	`XML_IN_A`	`EDI_OUT_A`
`Queue Manager Name`	`WDI32_QM`	`WDI32_QM`
`Destructive Reads`	Checked	Checked
`Syncpoint Control`	Checked	Checked

   Note: The Queue Profile ID is restricted to a maximum of eight characters. Therefore, the profile ID for the EDI_OUT_A queue must be named EDI_OU_A. All references to the WebSphere MQ queue EDI_OUT_A in WebSphere Data Interchange use EDI_OU_A.

2. Create an MQSeries Queue profile for EDI_IN_A and XML_OU_A. Table 83 defines the properties of each queue used in EDI-to-XML translation.

Table 83. MQSeries Queue profile for EDI_IN_A and XML_OU_A

Queue property	Value for EDI_IN_A	Value for XML_OU_A
Queue Profile ID	EDI_IN_A	XML_OU_A
Full Queue Name	EDI_IN_A	XML_OUT_A
Queue Manager Name	WDI32_QM	WDI32_QM
Destructive Reads	Checked	Checked
Syncpoint Control	Checked	Checked

Network profile

Network profiles define for WebSphere Data Interchange the characteristics of the networks you use for communications with trading partners. For this scenario, you would create and configure a Network Profile that communicates with the WebSphere MQ queues created earlier.

Table 84 shows the properties to configure for the Network profile.

Table 84. Properties in a Network profile

Network property	Description
Network ID	A unique identifier to name the profile
Communication Routine	The name of the program that builds network commands and invokes the network program to process the commands
Network Program	The program invoked by the communication routine to process requests
Network Parameters	Parameters required by the network program

For this sample scenario, you create and configure a Network profile that communicates with the WebSphere MQ queues created earlier (see "MQSeries$^{(R)}$ Queue profile" on page 196), as follows:

1. Create a new Network profile called HUB_IN.

   This network profile is used in the XML-to-EDI scenario. Table 85 lists the actual parameters specified for HUB_IN.

Table 85. Network profile for HUB_IN

Network property	Value for HUB_IN profile
Network ID	HUB_IN
Communication Routine	VANIMQ
Network Program	EDIMQSR
Network Parameters	SENDMQ=EDI_OU_A RECEIVEMQ=XML_IN_A

2. Create a second Network profile called HUB_OUT.

   This Network profile is used in the translation of EDI received from WebSphere Partner Gateway Enterprise Edition. A second Network profile is required, because WebSphere Partner Gateway Enterprise Edition places messages on the WebSphere MQ queues that include RFH2 headers. Table 86 on page 199 lists the properties of HUB_OUT.

Table 86. Network profile for HUB_OUT

Network property	Value for HUB_OUT profile
Network ID	HUB_OUT
Communication Routine	VANIMQ
Network Program	EDIRFH2
Network Parameters	SENDMQ=XML_OU_A RECEIVEMQ=EDI_IN_A

Mailbox profile

Mailbox profiles contain the information that WebSphere Data Interchange needs to identify the individuals and groups in your organization that receive documents to be translated. Table 87 shows the properties to configure for each Mailbox profile.

Table 87. Properties in a Mailbox profile

Mailbox property	Description
Mailbox ID	A unique identifier to name the profile
Network ID	The network ID of the network profile created earlier

You create mailbox profiles for each of the WebSphere MQ queues to identify the individuals and groups in the organization, as follows:

1. Create a Mailbox profile for each WebSphere MQ queue used.

   Table 88 lists the actual parameters in each of the Mailbox profiles.

Table 88. Mailbox profiles for XML_IN_A and EDI_OU_A

Mailbox property	Value for XML_IN_A	Value for EDI_OU_A
Mailbox ID	XML_IN_A	EDI_OU_A
Network ID	HUB_IN	HUB_IN
Receive File	XML_IN_A	EDI_OU_A

2. Create a second pair of mailboxes.

   Table 89 lists the properties for each.

Table 89. Mailbox profiles for EDI_IN_A and XML_OU_A

Mailbox property	Value for EDI_IN_A	Value for XML_OU_A
Mailbox ID	EDI_IN_A	XML_OU_A
Network ID	HUB_OUT	HUB_OUT
Receive File	EDI_IN_A	XML_OU_A

Service profile

Service profiles allow you to enter a utility command and define all the files that will be used during execution of that command.

For the sample scenario, you take the following steps:

1. Create a new Service Profile for XML_IN_A. You define the properties under the **General** tab, as follows:
   - Continue Command Chaining: **On Success**
   - PERFORM Command:

```
PERFORM TRANSFORM WHERE INFILE(XML_IN_A) SYNTAX(X)
OUTTYPE(MQ)OUTFILE(EDI_OU_A)
```

Table 90 lists the Common Files properties.

Table 90. Common Files for XML_IN_A

Common File property	**Value**
Tracking File	..\trk\xml_in.trk
Exception File	..\xex\xml_in.xex
Work File	..\wrk\xml_in.wrk
Report File	..\rpt\xml_in.rpt
Query File	..\qry\xml_in.qry

2. Enter the following in the **Output Files** tab:
   - Name in Command: **EDI_OU_A**
   - System File Name: **..\edi\edi_out.txt**

   Note: EDI_OU_A is used rather than EDI_OUT _A because of character length restrictions.
3. Create a second Service Profile for EDI_IN_A. You define the properties under the **General** tab, as follows:
   - Continue Command Chaining: **On Success**
   - PERFORM Command:

   ```
   PERFORM TRANSFORM WHERE INFILE(EDI_IN_A) SYNTAX(E)
   OUTTYPE(MQ) OUTFILE(XML_OU_A)
   ```

   Table 91 lists the Common Files properties.

Table 91. Common files for EDI_IN_A

Common File property	**Value**
Tracking File	..\trk\edi_in.trk
Exception File	..\xex\edi_in.xex
Work File	..\wrk\edi_in.wrk
Report File	..\rpt\edi_in.rpt
Query File	..\qry\edi_in.qry

4. Enter the following details under the **Output Files** tab:
   - Name in Command: **XML_OU_A**
   - System File Name: **..\xml\xml_out.txt**

   Note: XML_OU_A is used rather than XML_OUT_A because of character length restrictions. This restriction was eliminated with CSD10 of the WebSphere Interchange Server.

Import and compile data transformation maps

After you create the profiles, as described in the previous section, you can import any maps you need to transform your data. You then compile the transformation maps and set a rule for each. You use the WebSphere Data Interchange Client to perform these tasks. See the WebSphere Data Interchange documentation for information.

Setting up the JMS environment

As mentioned earlier in this chapter, WebSphere Partner Gateway Enterprise Edition can use the WebSphere MQ implementation of the Java Message Service (JMS) for integration with WebSphere Data Interchange.

Note: Alternatively, it is possible to use LDAP or WebSphere Application Server as a JNDI provider.

This section outlines the steps involved in creating a JMS environment on Computer B:

- "Configuring JMSAdmin"
- "Creating the JMS objects"

WebSphere MQ classes for Java and WebSphere MQ classes for JMS are built in to WebSphere MQ for Windows, version 5.3.

Configuring JMSAdmin

Use the JMSAdmin tool available in WebSphere MQ to create the JMS objects in JNDI. For information on how to create the default configuration file called `JMSAdmin.config`, see the *Hub Configuration Guide*.

To create the JMS objects for this tutorial:

1. To use a file-based JNDI provider, you would make sure the JMSAdmin.config file contains the lines shown below:
   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
   PROVIDER_URL=file:/opt/mqm/java/JNDI
   ```
2. If the JNDI directory does not already exist, create the JNDI directory under the following directory:
   ```
   /opt/mqm/java/bin
   ```

Before invoking the JMSAdmin tool, you would ensure your CLASSPATH contains the following entries:

```
/opt/mqm/java/lib/jms.jar
/opt/mqm/java/lib/com.ibm.mq.jar
/opt/mqm/java/lib/com.ibm.mqjms.jar
/opt/mqm/java/lib/jta.jar
/opt/mqm/java/lib/connector.jar
/opt/mqm/java/lib/jndi.jar
/opt/mqm/java/lib/providerutil.jar
/opt/mqm/java/lib/fscontext.jar
```

Note: The above entries, which relate to Linux$^{(TM))}$, assume you are using file-based JNDI.

Creating the JMS objects

To create the required JMS objects, you use the JMSAdmin tool. For the sample scenario, you would:

1. Define a new context:
   ```
   DEF CTX(WdiJms)
   ```
2. Change to the new context:
   ```
   CHG CTX(WdiJms)
   ```
3. Define a queue connection factory:
   ```
   DEF QCF(HUB60_QM_QCF) TRAN(CLIENT) HOST(IP_COMPUTER_B)
   PORT(9999) CHAN(java.channel) QMANAGER(HUB60_QM)
   ```

4. Define the EDI_IN_B queue:

```
DEF Q(EDI_IN_B) QMANAGER(HUB60_QM) QUEUE(EDI_IN_B)
```

5. Define the EDI_OUT_B queue:

```
DEF Q(EDI_OUT_B) QMANAGER(HUB60_QM) QUEUE(EDI_OUT_B)
```

6. End the JMSAdmin session

```
END
```

Configuring WebSphere Partner Gateway Enterprise Edition

WebSphere Partner Gateway is the communication layer between disparate community participants and internal processes. When setting up WebSphere Partner Gateway to work with EDI documents, you can configure it to:

- Send and receive EDI documents to and from WebSphere Data Interchange
- Communicate EDI transactions with external trading partners using AS2

The *Hub Configuration Guide* provides complete information on how to configure WebSphere Partner Gateway Enterprise and Advanced Editions. This section provides you with an example of configuring the WebSphere Partner Gateway Enterprise Edition that is described in the sample scenario. It describes the following steps:

1. "Creating participants"
2. "Setting the B2B capabilities" on page 204
3. "Creating gateways" on page 205
4. "Defining interactions" on page 206
5. "Creating participant connections" on page 207
6. "Creating targets" on page 208

Note: For information on how to configure WebSphere Partner Gateway - Express, see "Configuring WebSphere Partner Gateway - Express" on page 209.

Creating participants

A participant profile identifies companies to the system. You create participants for Partner One and Partner Two in the WebSphere Partner Gateway Enterprise Edition Community Console.

Create a participant for Partner One: Create a participant profile to represent Computer A and Computer B, which are the two systems that Partner One owns.

To create this participant profile, you take the following steps:

1. Open the WebSphere Partner Gateway Community Console.
2. Log in as **hubadmin**.
3. Verify that **Profiles** is already selected from the Account Admin menu.
4. Click **Create** and enter the details as listed in Table 92 below.

Table 92. Participant properties for Partner One

Field name	Value
Company Login Name	partnerOne
Participant Display Name	Partner One
Participant Type	Community Manager
Status	Enabled
Vendor Type	Other

Table 92. Participant properties for Partner One (continued)

Field name	Value
Web Site	http://*IP_COMPUTER_A* where *IP_COMPUTER_A* is the Internet protocol (IP) address of Computer A
Business ID Type	Freeform
Business ID Identifier	123456789
IP Address Gateway Type	Production
IP Address	*IP_COMPUTER_A* where *IP_COMPUTER_A* is the Internet protocol (IP) address of Computer A

Note: To create the Business ID Type and Business ID Identifier, you first click the **New** button below Business ID. The Business ID must be unique. Similarly, to create details relating to the IP Address, you click the **New** button below the IP Address header.

5. Click **Save**.

WebSphere Partner Gateway Enterprise Edition uses the Business ID Identifier (defined in Table 92 on page 202) to identify the sender or receiver of a document. When an ANSI X12 EDI transaction is received, the Interchange Sender and Receiver data is read to determine the source and target of the transaction.

Important: Make a note of the Administrator's Password for Partner One, as you will need it later. When you log on to the Community Console as Partner One, you will be asked to enter the password and then to change it.

Create a participant for Partner Two: Next, create a community participant to represent Partner Two. To create the participant, you take the following steps:

1. Click **Account Admin** on the main menu and **Profiles** on the horizontal navigation bar.
2. Click **Create**.
3. Enter the values listed in Table 93 below.

Table 93. Participant properties for Partner Two

Field name	Value
Company Login Name	partnerTwo
Participant Display Name	Partner Two
Participant Type	Community Participant
Status	Enabled
Vendor Type	Other
Web Site	http://*IP_COMPUTER_C* where *IP_COMPUTER_C* is the Internet protocol (IP) address of Computer C
Business ID Type	Freeform
Business ID Identifier	987654321
IP Address Gateway Type	Production

Table 93. Participant properties for Partner Two (continued)

Field name	Value
IP Address	*IP_COMPUTER_C* where *IP_COMPUTER_C* is the Internet protocol (IP) address of Computer C

4. Click **Save**.

Important: Make a note of the Administrator's Password for Partner Two, as you will need it later. When you logged on to the Community Console as Partner Two, you were asked to enter the password and then to change it.

Setting the B2B capabilities

You define the B2B capabilities for each participant in WebSphere Partner Gateway Enterprise Edition through the Community Console. After you define the B2B capabilities for participants, you can define a valid Document Flow Definition used to support specific business collaboration types between the participants.

Set the B2B capabilities for Partner One: To define the B2B Capabilities for Partner One, take the following steps:

1. Click **Account Admin** on the main menu and **Profiles** on the horizontal navigation bar.
2. Click **Search** to reveal a list of all participants defined in the system.
3. Click the **View details** icon next to **Partner One**, and then click **B2B Capabilities**.

B2B Capabilities are set to active by clicking on the **Role is not active** icon. For the purposes of this sample, you will configure only the B2B Capabilities required to implement the scenario.

To set the source and target packaging for Partner One to None, you would:

1. Click the **Role is not active** icon underneath **Set Source for Package: None** to enable it. Repeat this step for **Set Target**.
2. Click the **Expand** icon to drill down.
3. Click the **Role is not active** icon for **Protocol: EDI-X12 (ALL)** for both source and target.
4. Click the **Expand** icon.
5. Click the **Role is not active** icon for **Document Flow: ISA (ALL)** for both source and target.

Set the B2B capabilities for Partner Two: To define the B2B capabilities for Partner Two, take the following steps:

1. Click **Account Admin** on the main menu and **Profiles** on the horizontal navigation bar.
2. Click **Search** to reveal a list of all participants defined in the system.
3. Click the **View details** icon next to **Partner Two,** and then click **B2B Capabilities**.

To set the source and target packaging for Partner Two to AS, take the following steps:

1. Click the **Role is not active** icon underneath **Set Source** for **Package: AS** to enable it. Repeat this step for **Set Target**.

2. Click the **Expand** icon to drill down.
3. Click the **Role is not active** icon for **Protocol: EDI-X12 (ALL)** for both source and target.
4. Click the **Expand** icon.
5. Click the **Role is not active** icon for **Document Flow: ISA (ALL)** for both source and target.

Next, you update the AS definition for Partner Two, to ensure that Message Disposition Notifications (MDNs) for AS2 sent to Partner Two are returned to the correct address, as follows:

1. Click the **Edit** icon.
2. Enter an AS MDN E-mail address.

   This address is used to receive MDNs for AS1.
3. Enter an AS MDN HTTP URL:

   ```
   http://IP_COMPUTER_B:PORT/bcgreceiver/submit
   ```

   Note: The URL defined for AS2 uses the same parameters that will be defined for the AS2 Target later in this chapter.

Creating gateways

A gateway in WebSphere Partner Gateway defines a network point that acts as the entrance to another network. The gateway contains the information that tells WebSphere Partner Gateway how to deliver documents to the Enterprise Application Integration (EAI) layer.

Create a Gateway for Partner One: Partner Two sends EDI documents to Partner One using AS2. Partner One's gateway is used to send the EDI documents received via AS2 to a JMS queue and ultimately to WebSphere Data Interchange for translation.

To create a new gateway for Partner One, take the following steps:

1. Click **Account Admin** from the main menu and **Profiles** from the horizontal navigation bar.
2. Click **Search**.
3. Select Partner One by clicking the **View details** icon, and then select **Gateways**.
4. Click **Create** to create a new gateway for Partner One.
5. Enter the values for this new gateway are shown in Table 94.

Table 94. Properties for Partner One gateway

Field name	Value
Gateway Name	JMStoPartnerOne
Transport	JMS
Target URI	file:///opt/mqm/java/JNDI/WdiJms
JMS Factory Name	HUB60_QM_QCF
JMS Message Class	TextMessage
JMS Message Type	TextMessage
JMS Queue Name	EDI_OUT_B
JMS JNDI Factory Name	com.sun.jndi.fscontext.RefFSContextFactory

6. Click **Save**.

Make JMStoPartnerOne the default gateway for Partner One, as follows:

1. Click **View Default Gateways**.
2. From the **Production** list, select **JMS2toPartnerOne**.
3. Click **Save**.

Create a Gateway for Partner Two: Partner One sends EDI documents to WebSphere Partner Gateway Enterprise Edition over a JMS queue. Partner Two's gateway is used to send the received EDI documents to Partner Two via AS2.

To create a new gateway for Partner Two, take the following steps:

1. Click **Account Admin** from the main menu and **Profiles** from the horizontal navigation bar.
2. Click **Search**.
3. Select Partner Two by clicking the **View details** icon and then select **Gateways**.
4. Click **Create** to create a new gateway for Partner Two.
5. Enter the values for this gateway as shown in Table 95.

Table 95. Properties for Partner Two gateway

Gateway Name	AS2toPartnerTwo
Transport	HTTP/1.1
Target URI	http://*IP_COMPUTER_C*/input/AS2
User Name	partnerOne
Password	partnerOne

6. Click **Save**.

   Note: The User Name and Password as entered above refer to the Inbound Participant Mapping Method for HTTP as defined in WebSphere Partner Gateway - Express.

   For an example of setting these properties in WebSphere Partner Gateway - Express, see "Configuring WebSphere Partner Gateway - Express" on page 209.

   Notice that AS2toPartnerTwo is displayed as Online with a Status of **Enabled**.

Make AS2toPartnerTwo the default gateway for PartnerTwo, with the following steps:

1. Click **View Default Gateways**.
2. From the **Production** list, select **AS2toPartnerTwo**.
3. Click **Save**.

Defining interactions

A document flow definition is a collection of "meta-information" that defines the document processing capabilities of the participant. For the system to process a business document, two or more document flow definitions must be linked to create an interaction.

To create an interaction between Partner One and Partner Two, take the following steps:

1. Click **Hub Admin** from the main menu and **Document Flow Definition** from the horizontal navigation bar.
2. Click **Manage Interactions** and then **Create Interaction**.
3. From the Source column, select:

   a. Package: **None**
   b. Protocol: **EDI-X12 (ALL)**
   c. Document Flow: **ISA (ALL)**
4. From the Target column, select:
   a. Package: **AS**
   b. Protocol: **EDI-X12 (ALL)**
   c. Document Flow: **ISA (ALL)**
5. Set the Action as **Pass Through**.
6. Click **Save**.
7. Click **Create Interaction** again.
8. From the Source column, select:
   a. Package: **AS**
   b. Protocol: **EDI-X12 (ALL)**
   c. Document Flow: **ISA (ALL)**
9. From the Target column select:
   a. Package: **None**
   b. Protocol: **EDI-X12 (ALL)**
   c. Document Flow: **ISA (ALL)**
10. Set the Action as **Pass Through**.
11. Click **Save**.

Creating participant connections

Participant connections are the mechanism that enables the system to process and route documents between the Community Manager and its various participants. Connections contain the information necessary for the proper exchange of each document flow.

To create a participant connection between Partner One and Partner Two, take the following steps:

1. Click **Account Admin** from the main menu and **Participant Connections** from the horizontal navigation bar.
2. From the **Source** list, select **Partner One**.
3. From the **Target** list, select **Partner Two**.
4. Click **Search**.
5. Activate the Participant Connection that is displayed below by clicking on the **Activate** button. This should display the B2B Capabilities shown in Table 96.

Table 96. Activate Partner One-to-Partner Two participant connection

Document flow type	Source	Target
Package	None (N/A)	AS (N/A)
Protocol	EDI-X12 (ALL)	EDI-X12 (ALL)
Document Flow	ISA (ALL)	ISA (ALL)

To create a participant connection where Partner Two is the source and Partner One is the target, take the following steps:

1. Click **Account Admin** from the main menu and **Participant Connections** from the horizontal navigation bar.
2. From the **Source list**, select **Partner Two**.

3. From the **Target list**, select **Partner One**.
4. Click **Search**.
5. Activate the connection with the details shown in Table 97.

Table 97. Activate Partner Two-to-Partner One participant connection

Document flow type	Source	Target
Package	AS (N/A)	None (N/A)
Protocol	EDI-X12 (ALL)	EDI-X12 (ALL)
Document Flow	ISA (ALL)	ISA (ALL)

Creating targets

The Target List screen provides location information that enables WebSphere Partner Gateway's Document Manager to fetch documents from the appropriate system location based on the transport type of the incoming document. You can create separate target configurations based on transport type. The Document Manager can then poll the document repository locations of multiple Web, FTP, and POP mail servers--including internal directories and JMS queues--for incoming documents.

After the Document Manager retrieves a document from the location based on a pre-defined target, the routing infrastructure can process the document based on channel configuration.

To receive an EDI transaction from WebSphere Data Interchange, create a new JMS target by doing the following:

1. Click **Hub Admin** from the top-level menu.
2. Click **Targets** from the second-level menu, and then click **Create**.
3. Assign the properties shown in Table 98.

Table 98. Target properties for receipt over JMS

Target property	Value
Target Name	WdiJmsListener
Transport	JMS
Gateway Type	Production
JMS Provider URL	file:///opt/mqm/java/JNDI/WdiJms
JMS Queue Name	EDI_IN_B
JMS Factory Name	HUB60_QM_QCF
JNDI Factory Name	com.sun.jndi.fscontext.RefFSContextFactory

A second target is required for the receipt of EDI from Partner Two via AS2. Take the following steps to create this target:

1. Click **Hub Admin** from the top level menu.
2. Click **Targets** from the second level menu, and then click **Create**.
3. Assign the properties from Table 99 below:

Table 99. Target properties for receipt over AS2

Target Name	HubAS2Listener
Transport	HTTP/S

Table 99. Target properties for receipt over AS2 (continued)

Gateway Type	Production
URI	/bcgreceiver/submit **Note:** The URI for receipt of HTTP/S must always begin with `/bcgreceiver`

4. Click **Save**.

Configuring WebSphere Partner Gateway - Express

This section provides you with the steps to configure the community participant's environment. In this case, this environment is handled with a WebSphere Partner Gateway - Express system. In the sample scenario presented in this chapter, partnerTwo is using WebSphere Partner Gateway - Express to send and receive EDI using HTTP AS2.

To successfully receive EDI via HTTP AS2, take the following steps:

1. "Configuring My Profile"
2. "Creating a participant for Partner One"
3. "Configuring the Partner One participant" on page 210

Configuring My Profile

As the first step, you must create a profile for Partner Two in WebSphere Partner Gateway - Express. To create a profile for Partner Two, take the following steps:

1. Click **Configuration** from the main menu.
2. Click **My Profile** from the horizontal navigation bar.
3. Enter the details as outlined in Table 100.

Table 100. My Profile details

Receipt Address Unsecure Domain	*IP_COMPUTER_C* where *IP_COMPUTER_C* is the Internet protocol (IP) address of Computer C, where WebSphere Partner Gateway - Express is running
Receipt Address Unsecure Port	80 where 80 is the port assigned for use by WebSphere Partner Gateway - Express during installation.
AS2 Sender ID	987654321
Business ID Type	DUNS
Business Identifier	987654321

4. Click **Save**.

Creating a participant for Partner One

Partner One must be identified as a participant to WebSphere Partner Gateway - Express. To create Partner One as a participant, take the following steps:

1. Click **Configuration** from the main menu.
2. Click **Participants** from the horizontal navigation bar.

3. Click the **Create Participants** button.
4. Assign the following values:
   a. Participant Name: **partnerOne**
   b. AS2 Participant ID: **123456789**
5. Click **Save**.

From the Manage Participants view, you can see the details for partnerOne.

Configuring the Partner One participant

Once the participant for Partner One exists, you must configure Partner One for AS2 and HTTP. This configuration identifies the parameters required by WebSphere Partner Gateway - Express for both sending and receiving HTTP and AS2 to partnerOne.

To configure partnerOne for HTTP and AS2, take the following steps:

1. Click **Configuration** from the main menu.
2. Click **AS2** from the horizontal navigation bar.
3. Select **partnerOne** from the **Selected Participant** list and click **Edit**.
4. Define the Outbound Destination Address of partnerOne as:
   `http://`*`IP_COMPUTER_B`*`:7080/bcgreceiver/submit`
   Where *IP_COMPUTER_B* is the IP address of Computer B.
5. Click **Save**.
6. Click **HTTP** from the horizontal navigation bar. (**partnerOne** should still be displayed as the Selected Participant.)
7. Click **Edit**.
8. Set the Inbound User Name and Password:
   User Name: **partnerOne**
   Password: **partnerOne**
   Remember these were referenced earlier in the sample step of creating the default gateway for Partner Two in WebSphere Partner Gateway Enterprise Edition on Computer B.
9. Set the Outbound Destination Address to:
   `http://`*`IP_COMPUTER_B`*`:7080/bcgreceiver/submit`
10. Click **Save**.

Important: After making these changes in WebSphere Partner Gateway - Express, log out of the console and stop the gateway. Restart the gateway and console for all changes to take effect.

Summary

This chapter described the process by which WebSphere Partner Gateway interacts with WebSphere Data Interchange. It also provided you with procedures to set up the sample scenario described in "Example scenario used in this chapter" on page 193.

As mentioned at the beginning of this chapter, you can follow the *Integrating WebSphere Data Interchange V3.2 with WebSphere Business Integration Connect V4.2* tutorial to actually create a sample configuration. The tutorial provides sample scripts and maps to help you configure the environment and then shows you how to test a sample exchange. To access the tutorial, go to:

`www.ibm.com/developerworks/websphere/`

and search on the title of the tutorial.

Remember that you will have to select **ISA (ALL)** for Document Flow (in place of **ALL (ALL)** when you set the B2B Capabilities of the participants. The interactions you set up and the connections you activate will also have **ISA (ALL)** for the Document Flow.

Chapter 13. Routing EDI documents

This section describes the process by which WebSphere Partner Gateway determines the routing information for electronic data interchange (EDI) documents it sends and receives. It describes:

- The general flow of this processing (see Overview of EDI routing)
- Additional processing required when AS packaging has been specified (see "Special considerations for AS packaging" on page 214)

You can find additional information on how file-based integration can be used when routing EDI documents in "File-system protocol" on page 35.

Overview of EDI routing

An EDI document contains information, within the document, about the sender and the recipient of the document. WebSphere Partner Gateway uses this information when it routes the EDI document. The general flow is as follows:

1. WebSphere Partner Gateway determines the protocol used by examining the first three characters of the document. Table 101 shows the document-type protocol associated with each code.

Table 101. EDI codes and associated document types and protocols

Code	Document type	Document type protocol	Outbound as Content Type:
ISA	X12	EDI-X12	application/EDI-X12
GS	X12	EDI-X12	application/EDI-X12
UNB	Edifact	EDI-EDIFACT	application/EDIFACT
UNA	Edifact	EDI-EDIFACT	application/EDIFACT
ICS	ICS	EDI-X12	application/EDI-X12
STX	UNTDI	EDI-Consent	application/edi-consent
BG	UCS	EDI-Consent	application/edi-consent

2. WebSphere Partner Gateway extracts, from the EDI document, the sender information, based on the element and position for that particular document type, as described in Table 102.

Table 102. EDI codes and the location of the sender and receiver information

Code	From Qualifier	From ID	To Qualifier	To ID	EDI component support notes
ISA	Element 105 at position 5	Element 107 at position 6	Element 105 at position 7	Element 106 at position 8	Supported
GS	N/A	Element 142 at position 2	N/A	Element 124 at position 3	″GS-only″ addressing not supported

Table 102. EDI codes and the location of the sender and receiver information (continued)

Code	From Qualifier	From ID	To Qualifier	To ID	EDI component support notes
UNB UNA	Sub-element 0007 at position 2 of composite element S002 at position 20 (2nd composite) of the UNB segment	Sub-element 0004 at position 2 of composite element S002 at position 20 (2nd composite) of the UNB segment	Sub-element 0007 at position 2 of composite element S003 at position 30 (3rd composite) of the UNB segment	Sub-element 0010 at position 1 of composite element S003 at position 30 (3rd composite) of the UNB segment	Supported
ICS	Element X05 at position 4	Element X06 at position 5	Element X05 at position 6	Element X08 at position 7	Precursor to ISA - Not supported
STX	Element FROM1 at position 3	Element FROM2 at position 3	Element UNT1 at position 4	Element UNT2 at position 4	Not supported in this release
BG	N/A	Element BG03 at position 3	N/A	Element BG04 at position 4	Supported

3. WebSphere Partner Gateway determines the sender ID from the sender ID and qualifier of the EDI document.

   Note that some EDI envelopes (for example, GS) do not have the notion of qualifiers. In this case, WebSphere Partner Gateway uses only the ID.
4. WebSphere Partner Gateway concatenates the qualifier and ID with a dash (-) character to look up the sender ID from the WebSphere Partner Gateway profile repository. For example, if, in the EDI message for the sender, the qualifier is AB and the identifier is 1234567, WebSphere Partner Gateway expects to find a community participant with an identifier of AB-1234567 in the profile repository. If WebSphere Partner Gateway cannot find this ID, the EDI document is not routed.
5. To look up the receiving partner, WebSphere Partner Gateway determines the receiver qualifier and ID from the EDI message.
6. WebSphere Partner Gateway concatenates the qualifier and ID with a dash (-) character to look up the receiver ID in the profile repository.
7. WebSphere Partner Gateway routes the document to the intended recipient.

Special considerations for AS packaging

When the packaging of the document is specified as AS, WebSphere Partner Gateway performs some additional processing.

How inbound documents are routed

When an EDI document is received from a community participant:

1. WebSphere Partner Gateway first checks the AS1 or AS2 header information. Specifically, it checks the sender and receiver information to determine whether it matches IDs for valid community participants.
   - For AS1, it uses the Subject header field, which is in the form *ToID*;*FromID*.
   - For AS2, it uses the AS2-From and AS2-To header fields.

If the values in the header fields do not match valid IDs, WebSphere Partner Gateway does not route the document.

2. WebSphere Partner Gateway then performs the steps described in "Overview of EDI routing" on page 213.

How outbound documents are routed

When an EDI document is received from a back-end system, WebSphere Partner Gateway determines whether an AS BusinessID attribute has been specified for both the source packaging (None) and the target packaging (AS):

- If the AS BusinessId attribute has been specified, WebSphere Partner Gateway uses this information to generate the From and To IDs in the AS1 or AS2 header.
- If the attribute has not been specified, WebSphere Partner Gateway determines the protocol of the document, extracts the sender and receiver information and concatenates the result (as described in "Overview of EDI routing" on page 213) and then populates the header information.

Setting both IDs in the participant profile

Because WebSphere Partner Gateway uses both the AS1 or AS2 header information as well as the information derived from the EDI document, the IDs for the same participant could be in different forms. For example, the AS header information for the sender could be 123456789 while the information derived from the EDI document could be AB-12345678.

Make sure that you have listed both IDs in the profile for the community participant. Refer to the *Hub Configuration Guide* for information.

Notices

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program. General-use programming interfaces allow you to write application software that obtain the services of this program's tools. However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

i5/OS
IBM
the IBM logo
AIX
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator

WebSphere Partner Gateway Enterprise and Advanced Editions, version 6.0.0.1

Index

Printed in USA