

*IBM WebSphere Business Integration Connect
Enterprise and Advanced Editions*



Integration Overview

Version 4.2.1

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices and Trademarks," on page 109.

20 February 2004

This edition applies to Version 4, Release 2, Modification 1, of IBM[®] WebSphere[®] Business Integration Connect Enterprise Edition (5724-E87) and Advanced Edition (5724-E75), and to all subsequent releases and modifications until otherwise indicated in new editions.

IBM welcomes your comments. You can send to the following address:

*IBM Burlingame Laboratory
Information Development
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A*

Include the title and order number of this book, and the page number or topic related to your comment.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2003, 2004. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this book.....	5
Who should read this book.....	5
Related documents	5
Chapter 1. Backend Integration	7
Message transport protocols.....	8
Supported transport protocols	8
HTTP protocol	10
File System protocol for Enterprise and Advanced editions	12
JMS protocol.....	14
Setting up integration through the JMS transport protocol using WebSphere MQ 5.3.....	15
Message handling	18
Queued delivery	18
Communication error handling	18
Duplicate messages	19
Packaging	19
None packaging.....	19
Backend Integration packaging	19
Example of Backend Integration packaging over HTTP	29
Protocol-specific information	30
Web Services (SOAP)	30
cXML	31
RosettaNet.....	31
Chapter 2. Integrating with the WebSphere InterChange Server	37
Planning for integration	37
Which transport will you use?.....	37
Which packaging will you use?	38
Is the packaging available for the business protocol?.....	38
What types of business objects are required for the packaging?	39
Using the HTTP transport protocol.....	40
Overview of sending documents to the WebSphere InterChange Server	40
Setting up the environment for sending documents	43
Overview of sending documents to Business Integration Connect	52
Setting up the environment for sending documents	54
Summary of supported platforms and versions.....	55
Using Web Services	56
Using the JMS transport protocol	58
Overview of sending documents to the WebSphere InterChange Server.....	59
Overview of sending documents to the Business Integration Connect	60
Setting up the environment for sending and receiving documents	62
Creating business objects	63
Attachment Data Handler	65
Overview	65
Setting up the environment for the Attachment Data Handler.....	65
Configuring the Attachment Data Handler	66
Creating and modifying business objects.....	69
Converting messages to business objects.....	73
Converting business objects to messages.....	74
Chapter 3. Integrating with WebSphere Data Interchange	77
Introduction	77
Sending documents to WebSphere Data Interchange	78
Sending documents to Business Integration Connect.....	79
Sample scenario used in this chapter	79
Configuring your environment for message exchange	81

Configure WebSphere MQ communication	81
Configure WebSphere Data Interchange.....	84
Set up the JMS environment	90
Configure Business Integration Connect	92

Chapter 4. Routing EDI documents .. 105

Overview of EDI routing.....	105
Special considerations for AS packaging	107
Routing the inbound document	107
Routing the outbound document.....	107
Setting both IDs in the participant profile	107

Notices and Trademarks..... 109

Notices.....	109
Programming interface information	111
Trademarks and service marks	111

About this book

This document describes the Backend Integration interface, which is the mechanism that backend applications and IBM^(R) WebSphere^(R) Business Integration Connect use to communicate. The document then describes how to integrate WebSphere InterChange Server and WebSphere Data Interchange with Business Integration Connect using the Backend Integration interface.

Who should read this book

This book is intended for the person responsible for integrating Business Integration Connect with backend applications.

Related documents

The complete set of documentation available with this product describes the features and components of WebSphere Business Integration Connect Enterprise and Advanced Editions.

You can download this documentation or read it directly online at the following site:

<http://www.ibm.com/software/integration/wbiconnect/library/infocenter>

Chapter 1. Backend Integration

With Business Integration Connect, 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 backend system of your enterprise.

This document describes the interface used by backend applications to interact with Business Integration Connect. Consider the following example--a Community Participant sends a RosettaNet-formatted purchase order, intended for the Community Manager, to a particular target on Business Integration Connect. The Community Manager has a backend application that processes purchase orders and expects to receive the orders in RosettaNet Service Content (RNSC) format. When the connection between the Community Participant and Community Manager is established, it is agreed that:

- The document will be translated from RosettaNet to RNSC format
- The document routed to the backend application will have Backend Integration packaging, meaning that transport-level headers will be added to the document to convey information needed for the exchange

The backend application can then process the document.

You use the Community Console to establish the connection with your Community Participants and to specify the packaging that is used between Business Integration Connect and the backend application. The packaging can be None or Backend Integration.

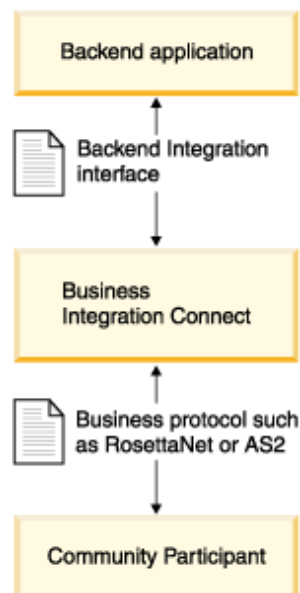
- None packaging causes Business Integration Connect to send the message to the backend application without any header data.
- Backend Integration packaging adds additional attributes to the message header and wraps the message contents if there are attachments. For information on these attributes, see [“Backend Integration packaging” on page 19](#).

The process for establishing connections is described in the [Administrator Guide](#).

Note that documents exchanged between the Community Participant and Business Integration Connect can be in a variety of formats, in addition to RosettaNet. Documents can be in the SOAP, cXML, XML, EDI, or binary formats. 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 Business Integration Connect and the backend application of the Community Manager as well as the transport types associated with the documents are shown in [“Supported transport protocols” on page 8](#).

The following graphic illustrates how Business Integration Connect uses the backend integration interface to communicate with the backend application at the Community Manager. Note that the arrows go in both directions; that is, the document can originate from the backend application of the Community Manager.



The role of the business protocol and packaging in the flow of documents

Message transport protocols

The Backend Integration interface supports a number of message transport protocols. They are HTTP, JMS, and File System. See [“Supported transport protocols”](#) for information on which transport protocols are valid for a particular combination of message content and Backend Integration packaging.

Supported transport protocols

The business protocol and the message transport protocol have a role in determining whether the messages must have None packaging or Business Integration packaging or if there is a choice between the two. The following table shows whether a transport protocol is valid for sending messages from Business Integration Connect to an application for a particular combination of message content and integration packaging.

Supported transport protocols for Business Integration Connect to application

Business protocol and packaging combination	HTTP or HTTPS	JMS	File System
RosettaNet (RNSC) with Backend Integration packaging	Yes	Yes	No
XML with Backend Integration packaging	Yes	Yes	No
XML only (None packaging)	Yes	Yes	Yes
EDI only (None packaging)	Yes	Yes	Yes
cXML only (None packaging)	Yes	No	No
SOAP only (None Packaging)	Yes	No	No
Binary with Backend Integration packaging	Yes	Yes	No
Binary only (None packaging)	Yes	Yes	No

For example, while you can send a RosettaNet message with Backend Integration packaging over HTTP, HTTPS, or JMS protocols, you cannot use the File System protocol. In addition, RosettaNet messages with None packaging are not supported.

The following table shows whether a transport protocol is valid for sending messages from an application to Business Integration Connect for a particular combination of message content and integration packaging:

Supported transport protocols for application to Business Integration Connect

Message content and transport packaging combination	HTTP or HTTPS	JMS	File System
RosettaNet (RNSC) with Backend Integration packaging	Yes	Yes	No
XML with Backend Integration packaging	Yes	Yes	No
XML only (None packaging)	Yes	Yes	Yes
EDI only (None packaging)	Yes	Yes	Yes
cXML only (None packaging)	Yes	No	No
SOAP only (None packaging)	Yes	No	No
Binary with Backend Integration packaging	Yes	Yes	No
Binary only (None packaging)	No	No	No

All other message content and transport packaging combinations are not supported.

HTTP protocol

To send messages, Business Integration Connect uses HTTP/S 1.1. To receive messages from backend applications, Business Integration Connect supports both HTTP/S version 1.0 and 1.1. The HTTP message can include the integration packaging attributes depending on what the setting is for the participant connection.

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 19](#). If the connection specifies None packaging, the HTTP message does not have these additional attributes, and Business Integration Connect parses the message to obtain this information.

- EDI, SOAP, and cXML messages must use None packaging.
- RosettaNet messages must use Backend Integration packaging.
- XML messages can use either None or Backend Integration packaging.
- Binary messages received from the backend application must have the Backend Integration packaging; however, the reverse is not true because Business Integration Connect supports sending binary messages to the application using either type of packaging.

Process

The following describes what happens when HTTP or HTTPS messages are sent between Business Integration Connect and an application for asynchronous exchanges:

1. The source system (Business Integration Connect or the application) 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 Business Integration Connect is the target system, 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 and receiving messages using the HTTP protocol

To send a message using the HTTP protocol, a backend application does the following:

1. Creates the message. The Content-Type header attribute gives the encoding used for the message.
2. Packages the message according to the packaging set for the connection. For Backend Integration packaging, the application adds the Business Integration Connect required protocol header attributes.
3. Posts the message to the URL used by Business Integration Connect to receive these messages.
4. If the exchange is synchronous, the application waits to receive a response in the same connection used for the request.

To receive messages using the HTTP protocol, an application does the following:

1. Listens for a message at a particular URL.
2. Processes the message. If the connection has None packaging, the application 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 application returns a response in the same connection used for the request.

To enable HTTP message exchanges, you use the Target Details screen of the Community Console to set up a target for inbound documents and the Gateway screen to configure the gateway for outbound messages. See “Configuring targets in the [Administrator Guide](#) for information on setting up targets. For information on gateways, see “Managing gateway configurations” in the [Administrator Guide](#).

File System protocol for Enterprise and Advanced editions

The File System protocol enables Business Integration Connect to send messages by placing them in a defined directory structure. Business Integration Connect receives messages by reading them from the directory structure.

The only document types supported by the file system protocol are EDI and XML documents. The only integration packaging that Business Integration Connect supports is the None option. That is, the files cannot contain the additional attributes.

Process

When the application sends a message to Business Integration Connect Enterprise or Advanced Edition, it must put the message file into a specific directory. The target of the message determines the directory. When you create a target, Business Integration Connect creates a Documents directory and its subdirectories for the target.

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

See “Configuring targets” in the [Administrator Guide](#) for information on creating targets.

Business Integration Connect polls the Documents directories and their subdirectories regularly to detect message files. If it finds a message, Business Integration Connect persists the message and then deletes the message from the directory. Business Integration Connect then processes the message normally.

When Business Integration Connect is the source of the message, it places the message file in the Documents directory defined by the gateway. By defining the destination directory according to the gateway, each participant connection can have a different directory. For information on gateways, see “Managing gateway configurations” in the [Administrator Guide](#). The application should persist the message and then delete it from the directory before processing it. See “[Sending and receiving messages using the file system protocol](#)” on page 13.

Sending and receiving messages using the file system protocol

To send a message using the file system protocol, an application should do the following:

1. Create the message file in a temporary directory.
2. Once the file is ready, move the file to the appropriate directory polled by Business Integration Connect.

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

1. Poll the appropriate directory for message files.
Note that temporary files (those with extensions `.tmp` or `.tmp1`) should be ignored. The application must not pick up or delete these temporary files.
2. When there is a message, persist it.
3. Delete the message from the directory.
4. Process the message.

JMS protocol

The JMS protocol is based on the Java Message Service (JMS) and transfers messages through transactional, persistent JMS queues provided by, for example, WebSphere MQ. The queues and JMS properties are set in the Community Console. See the [Administrator Guide](#) for more information.

The JMS Protocol supports the following JMS message types:

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

In JMS protocol, the sending system sends a JMS message to the receiving system using the enqueue operation. The receiving system gets the message from the queue, persists the message, and then performs the dequeue operation to remove the message from the queue. From this point forward, the receiving system can process the message asynchronously.

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 19](#). Note that 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 Business Integration Connect 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, Business Integration Connect 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 Business Integration Connect supports sending binary messages to the application using either type of packaging.

Sending and receiving messages using the JMS protocol

To send a message using the JMS protocol, a backend application does the following:

1. Creates the message. The `content_type` header attribute sets the content type for the message and the `content_length` header attribute specifies the length of the message (in bytes).
2. Packages the message according to the packaging set for the connection. For Backend Integration packaging, the application adds the required JMS header attributes.
3. Sends the message to the JMS queue used by the application to send messages to Business Integration Connect.

To receive messages using the JMS protocol, an application does the following:

1. Listens for a message on the JMS queue.
2. Processes the message. If the connection has None packaging, the application 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.

To enable JMS message exchanges, you use the Target Details screen of the Community Console to set up a target for inbound documents. You use the Gateway screen to configure the gateway for outbound messages. See “Configuring targets” in the [Administrator Guide](#) for information on setting up targets. For information on gateways, see “Managing gateway configurations” in the [Administrator Guide](#).

Setting up integration through the JMS transport protocol using WebSphere MQ 5.3

This section describes the steps you perform to set up backend integration through the JMS transport protocol.

To use the JMS transport protocol to send and receive documents with a backend application, you:

1. Define a queue manager (if necessary) and create associated queues and channels.
2. Create a JMS bindings file for WebSphere MQ 5.3
3. Create a JMS target
4. Create a JMS gateway

Creating queues and channels

If you have not already defined a queue manager for Business Integration Connect and for the backend application, do so now. Then, create the following objects:

- Transmission queue (make sure the usage is set to transmission)
- Remote queue
- Receiver queue
- Sender channel
- Receiver channel

The way you create these objects depends on the platform you are using. Refer to the WebSphere MQ documentation for instructions on creating these objects.

Creating the JMS bindings file

The WebSphere MQ documentation describes how to create a JMS bindings file. The following is an overview of that process.

To create the JMS bindings file:

1. Open a command window and navigate to the folder `<MQ Root>\java\bin`, where `<MQ Root>` is the installation directory of WebSphere MQ.

2. Open the JMSAdmin.config file for edit.

3. Comment out the following line:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
```

4. Remove the comment from the line:

```
INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
```

5. Change the path for the PROVIDER_URL variable to the directory where you want the JMS bindings file to be placed (for example, `PROVIDER_URL=file:/d:/filesender/config`).

This directory must exist and your user account must have write permission to this folder.

6. Save the file.

7. In the command window, launch JMSAdmin.

8. To create the JMS context, type the following at the InitCtx prompt:

```
def ctx(<contextname>)
```

9. Change your active context by typing:

```
chg ctx(<contextname>)
```

where `<contextname>` is the context you created in the previous step.

10. To define the queue connection factory, type:

```
def qcf(<connection factory name>) qmgr (<queue manager name>) tran(client)
chan(<java channel name>) host (<MQ Host Name>) port (<MQ port>)
```

11. To define the queues, type the following for each queue:

```
def q(<queue alias name>) qmgr (<queue manager name>) queue (<queue name>)
```

12. To exit JMSAdmin, type:

```
end
```

The bindings file is created in a subfolder of the folder configured in the PROVIDER_URL field of the JMSAdmin.config file. The bindings file is created as “.bindings”. The name of the subfolder is the name you chose for your JMS context.

Creating the JMS target

Copy the bindings file created in [“Creating the JMS bindings file”](#) on page 15 to the directory where you want it to reside. If you want to preserve the JMS context, copy the subfolder (with the same name as the context) and the bindings file to the directory, so that the full path to the bindings file is `/<parent directory>/<context subdirectory>/bindings` (for example, `/mydir/myctx/.bindings`).

From the Targets screen of the Community Console, create a target, specifying the following information:

- Transport: JMS
- JMS Provider URL: The file system path to the directory where the context subfolder (if there is a context) and the bindings file are located, in the following form:
`file:///<parent directory containing the context subdirectory with the .bindings file>` (for example, `file:///mydir`).
- JMS Queue Name: The JMS queue alias name you specified when creating the bindings file, including the context (for example, `myctx/myqueue`)
- JMS Factory Name: The JMS factory name you specified when creating the bindings file, including the context (for example, `myctx/myfact`)
- JNDI Factory Name: `com.sun.jndi.fscontext.RefFSContextFactory`

Note that the target must be able to access the directory where the subfolder and bindings file are located.

Creating the JMS gateway

From the Gateways screen of the Community Console, create a gateway, specifying the following information:

- Transport: JMS
- Target URI: The file system path to the directory where the context subfolder (if there is a context) and the bindings file are located, in the following form:
`file:///<parent directory containing context subdirectory with bindings file>` (for example, `file:///mydir`)
- JMS Factory Name: The JMS factory name you specified when creating the bindings file, including the context (for example, `myctx/myfact`)
- JMS Message Class: `StreamMessage`, `BytesMessage`, or `TextMessage`
- JMS Queue Name: The JMS queue alias name you specified when creating the bindings file, including the context (for example, `myctx/myqueue`)
- JNDI Factory Name: `com.sun.jndi.fscontext.RefFSContextFactory`

Message handling

This section describes how Business Integration Connect handles various situations that impact the delivery of messages.

Queued delivery

Business Integration Connect posts information on all documents that it wants to send to a particular gateway into a queue. The Delivery Manager system 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 Delivery Manager receives an error in a thread, it stops other threads from attempting to deliver their messages. The Delivery 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 Business Integration Connect 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), Business Integration Connect may then retry 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:

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 Business Integration Connect is not configured to retry sending the message or if all delivery attempts fail, Business Integration Connect 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 Process 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 receiver.

See “Managing gateway configurations” in the [Administrator Guide](#) for more information.

Duplicate messages

All messages sent to or received from Business Integration Connect must have a Global Unique Identifier (GUID). Business Integration Connect 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 19). 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, Business Integration Connect reuses the message's GUID in each retry. If Business Integration Connect 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 that Business Integration Connect also checks for duplicate messages at the RosettaNet process level if RosettaNet is being used.

Packaging

This section describes the two types of packaging: Backend Integration and None.

None packaging

When packaging is set to None, Business Integration Connect does not add a transport-level header when it sends a message to a backend application and does not expect one when it receives a message from a backend application.

None packaging is required for EDI, SOAP, and cXML documents.

Backend Integration packaging

When packaging is set to Backend Integration, messages sent to or received from a backend application must have a transport level header, which contains meta information about the message, and a payload, which contains the content of the message. Messages may also have attachments. The header and payload are mandatory while attachments are optional. This section describes these features of Backend Integration packaging.

Transport level header content

The transport level header contains information that Business Integration Connect 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 Business Integration Connect have the mandatory fields and any of the optional fields that apply.

The following table lists the transport level header fields:

Transport header values

Header field	Description	Required ?
x-aux-sender-id	Identifier of the message sender such as a DUNS number. This is a required field.	Yes
x-aux-receiver-id	Identifier of the message receiver such as a DUNS number. This is a required field.	Yes
x-aux-protocol	Protocol of the message content. Valid values include RNSC for RosettaNet service content, XMLEvent, and Binary. For Business Integration Connect, 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
x-aux-process-type	Process to be performed or what type of message is being sent. For RosettaNet messages, this is the PIP code such as 3A4. For event messages, it is XMLEvent and for Binary messages, it is Binary. For Business Integration Connect, 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: <ul style="list-style-type: none"> • Production • 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, Business Integration Connect 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

Transport header values

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 32 .	
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 Business Integration Connect 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.	
content-type	The content type of the message.	
content-length	The length of the message (in bytes).	

Note: For JMS protocol messages, the fields 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.

RosettaNet to transport level header fields

The following table describes where Business Integration Connect obtains values for the transport level header fields for RosettaNet messages.

Mapping transport level header fields to RosettaNet content

Header field	Source of value
x-aux-sender-id	For RosettaNet 2.0: <(DeliveryHeader)> <messageSenderIdentification> <PartnerIdentification> <GlobalBusinessIdentifier> For RosettaNet 1.1: <ServiceHeader> <ProcessControl> <TransactionControl> <ActionControl> or <SignalControl> <PartnerRouter> <fromPartner> <PartnerDescription> <BusinessDescription> <GlobalBusinessIdentifier>
x-aux-receiver-id	For RosettaNet 2.0: <(DeliveryHeader)> <messageReceiverIdentification> <PartnerIdentification> <GlobalBusinessIdentifier> For RosettaNet 1.1: <ServiceHeader> <ProcessControl> <TransactionControl> <ActionControl> or <SignalControl> <PartnerRouter> <toPartner> <PartnerDescription> <BusinessDescription> <GlobalBusinessIdentifier>
x-aux-protocol	Set value for RosettaNet: RNSC
x-aux-protocol-version	Set value: 1.0

Mapping transport level header fields to RosettaNet content

x-aux-process-type	<p>For RosettaNet 2.0, the source XPath is: /ServiceHeader/ProcessControl/pipCode/GlobalProcessIndicatorCode</p> <p>For RosettaNet 1.1, the source XPath is: /ServiceHeader/ProcessControl/ProcessIdentity/GlobalProcessIndicatorCode</p>
x-aux-process-version	<p>For RosettaNet 2.0, the source XPath is: /ServiceHeader/ProcessControl/pipVersion/VersionIdentifier</p> <p>For RosettaNet 1.1, the source XPath is: /ServiceHeader/ProcessControl/ProcessIdentity/VersionIdentifier</p> <p>The value of the version identifier of each PIP is in its PIP specification.</p>
x-aux-payload-root-tag	Name of the PIP such as Pip3A4PurchaseOrderRequest
x-aux-process-instance-id	<p>For processes initiated by the application, the value is the ID of the process instance.</p> <p>For processes initiated by a community participant that are not pass-through workflow, the value is the process ID in the initial RosettaNet request.</p> <p>For RosettaNet 2.0:</p> <pre><ServiceHeader> <ProcessControl> <pipInstanceId> <InstanceIdentifier></pre> <p>For RosettaNet 1.1:</p> <pre><ServiceHeader> <ProcessControl> <ProcessIdentity> <InstanceIdentifier></pre>
x-aux-msg-id	<pre><(RNPipServiceContent)> <thisDocumentIdentifier> <ProprietaryDocumentIdentifier></pre>
x-aux-production	<p>For RosettaNet 2.0:</p> <pre><ServiceHeader> <ProcessIndicator> <GlobalUsageCode></pre> <p>For RosettaNet 1.1:</p> <pre><Preamble> <GlobalUsageCode></pre>

AS2 to transport level header fields

The following table describes where Business Integration Connect obtains values for the transport level header fields for AS2 messages.

Note: The values are case-sensitive.

Mapping transport level header fields to AS2 content

Header field	Source of value
x-aux-sender-id	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, the AS2-From header field of the AS2 message is set in the x-aux-sender-id field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS2 message goes out to a community participant, the x-aux-sender-id field of the incoming backend integration message is used as the AS2-From header value of the AS2 message.</p>
x-aux-receiver-id	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, the AS2-To header field of the AS2 message is set in the x-aux-receiver-id field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS2 message goes out to a community participant, the x-aux-receiver-id field of the incoming backend integration message is used as the AS2-To header value of the AS2 message.</p>
x-aux-protocol	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, the ToProtocol of the participant connection is set in the x-aux-protocol field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS2 message goes out to a community participant, the x-aux-protocol field of the incoming backend integration message is used to determine the FromProtocol of the participant connection.</p>
x-aux-protocol-version	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, the ToProtocolVersion of the participant connection is set in the x-aux-protocol-version field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS2 message goes out to a community participant, the x-aux-protocol-version field of the incoming backend integration message is used as the FromProtocolVersion of the participant connection.</p>
x-aux-process-type	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, the ToProcessCode of the participant connection is used to set in the x-aux-process-type field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS2 message goes out to a community participant, the x-aux-process-type field of the incoming backend integration message is used as the FromProcessCode of the participant connection.</p>

Mapping transport level header fields to AS2 content

x-aux-process-version	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, the ToProcessVersion of the participant connection is set in the x-aux-process-version field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS2 message goes out to a community participant, the x-aux-process-version field of the incoming backend integration message is used as the FromProcessVersion of the participant connection.</p>
x-aux-payload-root-tag	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, 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.</p> <p>When an AS2 message goes out to a community participant, this field doesn't need to be set in the incoming backend integration message.</p>
x-aux-process-instance-id	<p>This field is not used for AS2.</p>
x-aux-msg-id	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, for custom XML protocol only, the Doc ID specified in the XPATH is parsed out of the message and used in the x-aux-payload-root-tag field.</p> <p>When an AS2 message goes out to a community participant, this field doesn't need to be set in the incoming backend integration message.</p>
x-aux-system-msg-id	<p>When a community participant sends an AS2 message to Business Integration Connect Enterprise or Advanced edition, this field is set to the internally generated unique ID for this message.</p> <p>When an AS2 message goes out to a community participant, this field doesn't need to be set in the incoming backend integration message.</p>
x-aux-production	<p>This field is not used for AS2.</p>

AS1 to transport level header fields

The following table describes where Business Integration Connect obtains values for the transport level header fields for AS1 messages.

Note: The values are case-sensitive.

Mapping transport level header fields to AS1 content

Header field	Source of value
x-aux-sender-id	<p>When a community participant sends an AS1 message to Business Integration Connect Enterprise or Advanced edition, the <FromID> in the "Subject: <ToID>;<FromID>" header field of the AS1 message is set in the x-aux-sender-id field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS1 message goes out to a community participant, the x-aux-sender-id field of the incoming backend integration message is used as <FromID> in the "Subject: <ToID>;<FromID>" header value of the AS1 message.</p>
x-aux-receiver-id	<p>When a community participant sends an AS1 message to Business Integration Connect Enterprise or Advanced edition, <ToID> in the "Subject: <ToID>;<FromID>" header field of the AS1 message is set in the x-aux-receiver-id field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS1 message goes out to a community participant, the x-aux-receiver-id field of the incoming backend integration message is used as <ToID> in the "Subject: <ToID>;<FromID>" header value of the AS1 message.</p>
x-aux-protocol	<p>When a community participant sends an AS1 message to Business Integration Connect Enterprise or Advanced edition, the ToProtocol of the participant connection is set in the x-aux-protocol field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS1 message goes out to a community participant, the x-aux-protocol field of the incoming backend integration message is used as the FromProtocol of the participant connection.</p>
x-aux-protocol-version	<p>When a community participant sends an AS1 message to WebSphere Business Integration Connect Enterprise or Advanced edition, the ToProtocolVersion of the participant connection is set in the x-aux-protocol-version field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS1 message goes out to a community participant, the x-aux-protocol-version field of the incoming backend integration message is used as the FromProtocolVersion of the participant connection.</p>
x-aux-process-type	<p>When a community participant sends an AS1 message to Business Integration Connect Enterprise or Advanced edition, the ToProcessCode of the participant connection is set in the x-aux-process-type field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS1 message goes out to a community participant, the x-aux-process-type field of the incoming backend integration message is used as the FromProcessCode of the participant connection.</p>

Mapping transport level header fields to AS1 content

x-aux-process-version	<p>When a community participant sends an AS1 message to Business Integration Connect Enterprise or Advanced edition, the ToProcessVersion of the participant connection is set in the x-aux-process-version field of the backend integration message that is sent to the Community Manager.</p> <p>When an AS1 message goes out to a community participant, the x-aux-process-version field of the incoming backend integration message is used as the FromProcessVersion of the participant connection.</p>
x-aux-payload-root-tag	<p>When a community participant sends an AS1 message to Business Integration Connect Enterprise or Advanced edition, 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.</p> <p>When an AS1 message goes out to a community participant, this field doesn't need to be set in the incoming backend integration message.</p>
x-aux-process-instance-id	<p>This field is not used for AS1.</p>
x-aux-msg-id	<p>When a community participant sends an AS1 message to Business Integration Connect Enterprise or Advanced edition, for custom XML protocol only, the Doc ID specified in the XPATH is parsed out of the message and used in the x-aux-payload-root-tag field.</p> <p>When an AS1 message goes out to a community participant, this field doesn't need to be set in the incoming backend integration message.</p>
x-aux-system-msg-id	<p>When a community participant sends an AS1 message to Business Integration Connect Enterprise or Advanced edition, this field is set to the internally generated unique ID for this message.</p> <p>When an AS1 message goes out to a community participant, this field doesn't need to be set in the incoming backend integration message.</p>
x-aux-production	<p>This field is not used for AS1.</p>

Payload

For HTTP and JMS protocol messages, the payload of the message is in the body of the HTTP post or JMS message. For RosettaNet messages, the payload is the service content from the PIP. If you are sending EDI over AS2, the payload is the EDI message. The payload is not wrapped with an XML envelope unless the message also carries one or more attachments. For information on the XML envelope and tags used to wrap the attachments, see [“Attachments” on page 28](#).

If the message contains an attachment, the payload must be Base64 encoded in the XML envelope.

Attachments

If the business message protocol permits them, each message can have one or more attachments. If the message has attachments, it must have the <transport-envelope> root tag. Inside this root tag there is a <payload> tag that contains the message payload and an <attachment> tag for each attachment. The payload and attachment tags have two attributes:

Content-Type - to identify the MIME type/subtype, such as text/xml or image/gif. This is a mandatory attribute.

Encoding - to identify the encoding used. Because the attachment and payload must be Base64 encoded, the only valid value for this attribute is "Base64".

The following is an example of a message payload that has one XML attachment. Note that the namespace (xmlns="http://www.ibm.com/websphere/bcg/2003/v1.0/wbipackaging") is required.

```
<?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>
```

Business Integration Connect provides the Attachment Data Handler to process attachments exchanged with the WebSphere Integration Server, as described in ["Attachment Data Handler" on page 65](#).

Schema for Backend Integration Packaging

A W3C XML schema file that describes the Backend Integration XML envelope structure is included with Business Integration Connect. The file is named:

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

and 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.

Example of Backend Integration packaging over HTTP

The following is an example of a message from Business Integration Connect 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</ProprietaryDocumentI
dentifier>
</thisDocumentIdentifier>
<GlobalDocumentFunctionCode>Request</GlobalDocumentFunctionCode>
</Pip3A4PurchaseOrderRequest>
```

Protocol-specific information

This section describes information that is specific to the business protocol.

Web Services (SOAP)

Business Integration Connect can make Web Services provided by the Community Manager available to Community Participants. You will have to provide your Community Participant with the public WSDL that Business Integration Connect 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. Business Integration Connect 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.

Business Integration Connect can make Web Services provided by Community Participants available to the Community Manager. It is important to note that the same Web Service Interface can be provided by multiple partners. Business Integration Connect 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 [Administrator Guide](#) for more details. Business Integration Connect 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 [Administrator 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 Business Integration Connect receives a cXML document from a Community Participant, it validates the document and translates it (if specified) before sending it to the backend application at the Community Manager. Note that translation should not be used for synchronous cXML messages. In a synchronous exchange, the backend application generates a response, which Business Integration Connect returns to the Community Participant (if appropriate for the message).

A backend application at the Community Manager that needs to send a cXML document can do one of two things:

- Generate and send a cXML document, which Business Integration Connect passes through to the Community Participant
- Generate and send an XML document, which Business Integration Connect 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 will be returned asynchronously to the backend application.

Refer to the [Administrator Guide](#) for more information, including how to set up your document flow definitions for cXML.

RosettaNet

Business Integration Connect provides support for RosettaNet 1.1 and 2.0 provided the RosettaNet messages have Backend Integration packaging (that is, they must have transport level headers.) These messages must use the HTTP or JMS transport protocol. The transport level header retains meta-information that is not part of the PIP and enables Business Integration Connect to route the message appropriately.

For example, say 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, what PIP will be sent, and the version of the PIP along with other information. This information enables Business Integration Connect to send the correct PIP to the community participant.

You can find information about setting up RosettaNet support and configuring PIPS in the [Administrator Guide](#).

Event notification

Because Business Integration Connect separates the application from the Community Participant that is the RosettaNet service provider, Business Integration Connect provides event notification. Event notification enables Business Integration Connect to, for example, notify the application if Business Integration Connect is unable to send a PIP to the Participant. The application can then handle the failure of the service call.

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

Business Integration Connect can send an event notification message to backend applications when the following occurs:

- Business Integration Connect delivers a RosettaNet document to a Community Participant and receives a Receipt Acknowledgment. Event 100 is sent to the backend application.
- Business Integration Connect cancels a PIP by generating an 0A1 message and delivering it to the Community Participant. Event 800 is sent to the backend application.
- Business Integration Connect receives a Receipt Acknowledgment exception or a general exception from a Community Participant. Event 900 is sent to the backend application.

Business Integration Connect can send 0A1 message 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 Business Integration Connect 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 following schema:

```
<?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/xmleven
      tnotification"
    xmlns:evntf="http://www.ibm.com/websphere/bcg/2003/v1.0/xmleven
      tnotification"
    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>

<!-- 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>

```

The following table describes each field within the event payload:

Event notification XML fields

Field	Description
StatusCode	Type of message. The valid values are: <ul style="list-style-type: none"> • 100 - Business Integration Connect has delivered the document and received a receipt acknowledgment. • 800 - The application cancelled the PIP. • 900 - Business Integration Connect received a receipt acknowledgment exception, a general exception, or a 0A1Failure PIP from the community participant.
StatusMessage	Alpha-numeric description of this event notification message
EventMessageID	Alpha-numeric 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 datatype for dateTime (w3.org/TR/2001/REC-xmlschema-2-20010502#dateTime)

Event notification message example

The following is 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/xmleventno
tification">
  <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>
```

Chapter 2. Integrating with the WebSphere InterChange Server

[“Backend Integration” on page 7](#) described the general process used to integrate Business Integration Connect with a backend application. This chapter describes a specific implementation of that process—how to integrate Business Integration Connect with the WebSphere InterChange Server.

It is assumed that you are familiar with the WebSphere InterChange Server and associated components, such as collaborations, business objects, adapters, and the Server Access Interface.

Planning for integration

To set up integration with WebSphere InterChange Server, consider the following items:

Which transport will you use?

Two transport protocols are available to integrate with WebSphere 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 [“Using Web Services” on page 56](#).

- JMS transport protocol

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 Business Integration Connect is available with the integration mechanism used. See [“Message transport protocols” on page 8](#).
- If you require synchronous transactions, you must use the HTTP transport protocol.
- Sending documents (other than SOAP documents) to WebSphere InterChange Server over the HTTP transport protocol involves the use of a WebSphere Business Integration Connect-supplied servlet and Wrapper Data Handler. The servlet is used for receiving documents sent from Business Integration Connect. The servlet uses the Server Access Interface to invoke collaborations. The Server Access Interface invokes the collaborations synchronously.

Note: Even though some interactions between Business Integration Connect and backend applications are asynchronous, the Server Access Interface still invokes the collaboration synchronously and waits until the collaboration execution has completed. Refer to the WebSphere InterChange Server documentation for information on the Server Access Interface.

- Receiving documents from WebSphere InterChange Server over the HTTP transport protocol involves the use of a Business Integration Connect-supplied HTTP protocol handler for the Adapter for XML.
- Sending SOAP documents to and receiving SOAP documents from the WebSphere InterChange Server over the HTTP transport protocol involves the use of the Adapter for Web Services.
- Sending documents to and receiving documents from the WebSphere InterChange Server over the JMS transport protocol involves the use of the Adapter for JMS. The Adapter for JMS invokes collaborations asynchronously.
- The Adapter for JMS provides guaranteed delivery from Business Integration Connect to the WebSphere InterChange Server.

Note that WebSphere 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.

Which packaging will you use?

Two types of packaging are available for documents that are sent through the HTTP transport protocols:

- **Backend integration**, which means that transport-level headers are added to the document before it is sent to WebSphere InterChange Server. With backend integration packaging, you can optionally include attachments. If attachments are included, they are wrapped in an XML envelope.
- **None**, which means that no transport-level headers will be added to the document.

You specify the packaging to be used when you set up your partner connections. See the [Administrator Guide](#) for information on setting up partner connections.

Is the packaging available for the business protocol?

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. An EDI document can be processed only when a packaging of None has been specified. See [“Supported transport protocols” on page 8](#) for a complete list of which document types can be associated with which types of packaging.

What types of business objects are required for the packaging?

You define the business objects based on the type of packaging specified for the partner connection.

Relationship of packaging to business objects

Packaging	Business object
None	The business object corresponds to the payload of the document.
Backend integration without attachments	The business object corresponds to the payload of the document.
Backend integration with attachments	A data handler is required to process the XML envelope in which the payload and attachments are wrapped. See “Attachment Data Handler” on page 65 for information on the Business Integration Connect-supplied data handler you can use to process the XML envelope.

The appropriate data handler for the payload type is used to convert the documents or messages into business objects and to convert business objects into documents or messages. As mentioned earlier, the Data Handler for XML is used to convert XML messages into business objects. You can create custom data handlers for any message formats that do not have a corresponding data handler provided by WebSphere Business Integration Server.

The business object must be designed according to the requirements of the data handler and the requirements of the adapter used for integration with Business Integration Connect. For example, if you are using the Adapter for JMS, refer to the Adapter for JMS documentation for the required business object structure. If you are using JMS with the Backend Integration packaging, make sure your business objects have a dynamic child meta-object. For the HTTP transport protocol, see [“Step 8: Define the business objects” on page 48](#) for the business object requirements.

Additionally, make sure the data handlers you are using can ignore child meta-objects required by your connectivity. Refer to the section on the `cw_mo_label` application-specific information in the appropriate data handler documentation. 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.

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.

The remainder of this chapter describes the process for sending and receiving documents over each of the transport protocols and lists the steps you need to take to prepare to use the transport protocols.

Using the HTTP transport protocol

This section describes how to send and receive documents through the use of the HTTP transport protocol. All document types except SOAP make use of the Business Integration Connect-supplied servlet, described in the following sections. If you are exchanging SOAP documents over the HTTP transport protocol, see the section [“Using Web Services” on page 56](#).

All references to the HTTP transport protocol apply to HTTPS as well.

Overview of sending documents to the WebSphere InterChange Server

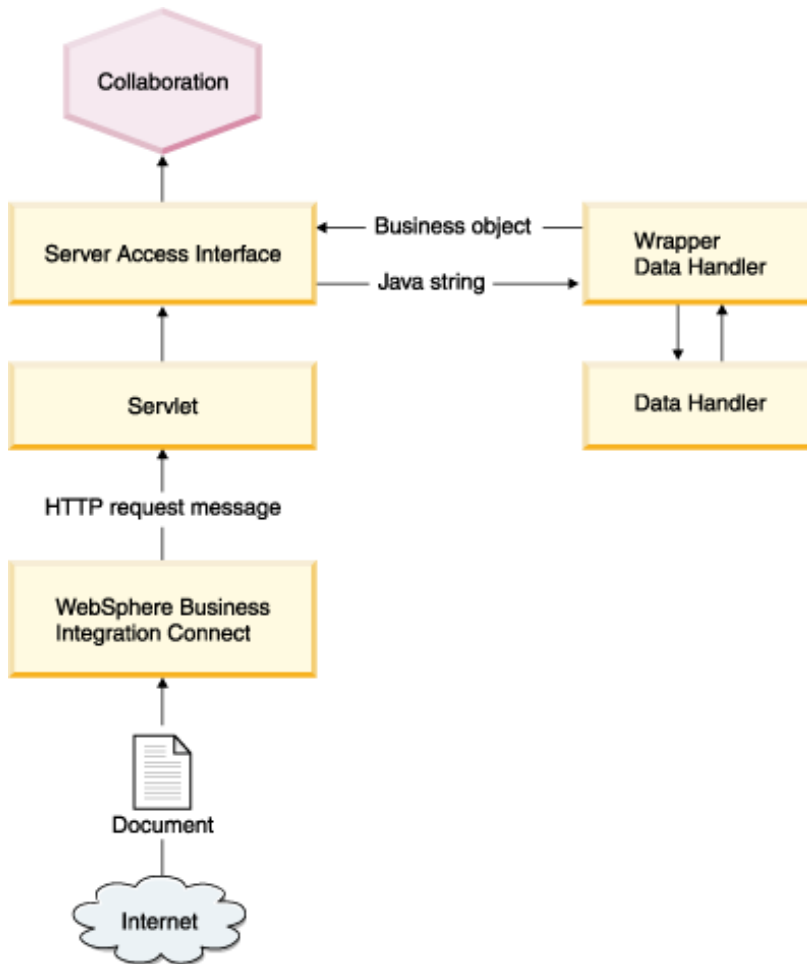
To send a document from Business Integration Connect to the WebSphere InterChange Server using the HTTP transport protocol, you interact with the following components, which are provided with Business Integration Connect:

- The WebSphere Business Integration Connect Servlet
- A Wrapper Data Handler
- The Attachment Data Handler (optional)

In addition, the servlet uses the Server Access Interface, and the Wrapper Data Handler invokes the data handler that is configured for your message. For example, if the payload is XML, the Wrapper Data Handler can be configured to call the Data Handler for XML.

The servlet can be used with WebSphere InterChange Server versions 4.1.1, 4.2.0, and 4.2.1. Note that the servlet cannot be used with WebSphere InterChange Server version 4.2.2.

The following illustration provides an overview of how the components interact. Note that the Wrapper Data Handler, the Attachment Data Handler, and the data handler for the message execute within the WebSphere InterChange Server.



Message flow from Business Integration Connect to a collaboration through the HTTP transport protocol

1. Business Integration Connect invokes the servlet to communicate with WebSphere InterChange Server. It sends the document to the URL specified as the target gateway. Each URL corresponds to a collaboration to be invoked. (See [“Step 2: Create the properties file”](#) on page 44 for information.) Note that the servlet can be used to invoke multiple collaborations.
2. The servlet creates a Java string from the HTTP request message sent by Business Integration Connect. The HTTP request message contains two parts:
 - HTTP transport protocol headers (the standard headers plus the custom headers that were set by WebSphere Business Integration Connect if Backend integration packaging was specified for the target gateway)
 - The message, which depends on the type of packaging that is used
3. The servlet checks a properties file to determine the collaboration, verb, and MIME type to use.
4. The servlet sends the Java string, along with the information from the properties file, to the Server Access Interface.
5. The Server Access Interface invokes the Wrapper Data Handler.

6. The Wrapper Data Handler extracts the headers and payload from the string and calls the configured data handler to convert the payload into a request business object. It then sets the HTTP headers in the child meta-object of this business object. The Wrapper Data Handler can also be configured to call the Attachment Data Handler, if the message sent from Business Integration Connect includes attachments. The actions of the Attachment Data Handler are described in [“Attachment Data Handler” on page 65](#).
7. The Wrapper Data Handler then creates the top-level business object and sets the request business object in it.
8. The Wrapper Data Handler returns the top-level business object to the Server Access Interface.
9. The Server Access Interface invokes the collaboration with the top-level business object.

The collaboration executes and returns the top-level business object to the Wrapper Data Handler. Note that:

- When interactions between WebSphere InterChange Server and Business Integration Connect are asynchronous, the response business object in the top-level object should not be populated by the collaboration.
- In a synchronous invocation in which a response should be returned in the same HTTP connection, the collaboration should populate the response business object.

If the interaction is successful, an HTTP 200 OK acknowledgment is returned to Business Integration Connect.

Make sure the collaboration port for the collaboration object you will be invoking is configured as the external port. Refer to the WebSphere InterChange Server documentation for details on configuring ports.

An overview of the process for sending documents from the WebSphere InterChange Server to Business Integration Connect is described in [“Overview of sending documents to the Business Integration Connect” on page 60](#).

The next section describes the tasks you perform to set up the environment so that you can send documents to the WebSphere InterChange Server.

Setting up the environment for sending documents

Because the sending and receiving of documents to and from the WebSphere InterChange Server involves the use of Business Integration-supplied components, you have to perform the following setup and configuration tasks.

Note: In this section and throughout this chapter, path names are shown according to Unix syntax. If you are using Business Integration Connect on a Windows platform, use Windows syntax for path names.

1. Deploy the servlet in your Web server or application server.
2. Create a properties file that the servlet uses to find information about a collaboration.
3. Edit the servlet deployment descriptor file to point to the properties file.
4. Add `bcgwbiwrapperdh.jar` to the classpath variable `DATAHANDLER` in the `%CROSSWORLDS%/bin/start_server.bat`.
5. Add the MIME type for the Wrapper Data Handler. Also add the MIME types for the data handler for the request.
6. Create a meta-object for the Wrapper Data Handler.
7. Configure the meta-objects for the Wrapper Data Handler and the data handler for the request message.
8. Define business objects representing the information being sent by the Business Integration Connect request.

These tasks are described in the sections that follow.

Step 1: Deploy the components

The servlet, Wrapper Data Handler, and the repository file for the Wrapper Data Handler are available from the following location on the Business Integration Connect installation medium:

Location of the components

Component	Location
Servlet	<code>integration/wbi/wics/http/bcgwbiservlet.war</code>
Wrapper Data Handler	<code>integration/wbi/wics/http/bcgwbiwrapperdh.jar</code>
Repository file for Wrapper Data Handler	<code>integration/wbi/wics/http/MO_DataHandler_WBIWrapper.in</code>

NOTE: If you are expecting to send and receive documents that include attachments, you can also deploy the Attachment Data Handler and its associated repository file, as described in [“Deploy the Attachment Data Handler” on page 66](#).

1. Deploy the servlet and associated files into the Web server according to the documentation for the Web server.

2. Make sure the following files (which can be found in the lib directory of the WebSphere InterChange Server installation directory) are in the CLASSPATH of the servlet:

- CrossWorlds.jar
- vbjorb.jar

Note: These files must be from the same version of the WebSphere InterChange Server that you will be invoking.

3. Make sure the following files (which can be found in the integration/wbi/wics/http/lib/thirdparty directory of the WebSphere Business Integration Connect installation medium) are in the CLASSPATH of the servlet:

- mail.jar
- log4j-1.2.8.jar

4. Make the IOR file available to the host on which the servlet is deployed.
5. Update the ICS_IORFILE property in servlet.properties with the location of the IOR file.

Step 2: Create the properties file

As mentioned in [“Overview of sending documents to the WebSphere InterChange Server” on page 40](#), the properties file contains information, such as port name and verb, that the servlet needs to invoke a collaboration. You create this properties file, specifying general information about the WebSphere InterChange Server. Then, for any collaboration you want the servlet to invoke, you provide information about that collaboration.

Contents of the property file

Create a properties file containing the information shown in the following table.

Contents of the properties file

Property Name	Example	Description
ICS_SERVERNAME	Server1	The host name where the WebSphere InterChange Server is running.
ICS_VERSION	4.2.0	The version number of the WebSphere InterChange Server. Possible values are 4.1.1, 4.2.0, and 4.2.1.

Contents of the properties file

ICS_IORFILE	c:/myiorlocation/ Server1InterChangeServer.ior	The file name used to access the WebSphere InterChange Server APIs. The example shows how you would specify the path on a Windows ^(R) system. Note that the path should be entered on one line.
ICS_USERNAME	Admin	The User ID for connecting to the server.
ICS_PASSWORD	Null	The password for connecting to the server.
ICS_ENCRYPTED_PASSWORD	False	An indication of whether the ICS_PASSWORD is encrypted. The servlet sets this field to true if the password is encrypted.
ICS_DISABLEENCRIPTION	True	An indication of whether password encryption is disabled (true) or enabled (false). Set this field to false if you want to allow passwords to be encrypted.
The following properties are specified for each collaboration.		
WBIC_SERVLET_COUNT	1	The number of URLs configured in this file. If it is set to 1, the servlet will process the URL and properties for WBIC_URL_1. If it is set for 2, the servlet will process the URL and properties for WBIC_URL_1 and WBIC_URL_2.
WBIC_URL_1	PurchaseOrder	The name of the relative URL.
WBIC_URL_1_COLLAB	PurchaseOrderCollab	The name of the collaboration.
WBIC_URL_1_PORT	From	The port name of the collaboration.

Contents of the properties file

WBIC_URL_1_VERB	Create	The verb subscribed to by the collaboration.
WBIC_URL_1_WRAPPER_MIME	wbic_wrapper	The Wrapper Data Handler MIME type. Note that the example is in lowercase.
WBIC_URL_1_CHARENCODE	UTF-8	The character encoding to use for the HTTP requests. Specify valid Java character encoding.

How the relative URL is used to look up the collaboration

The servlet uses the relative URL to look up the collaboration to execute.

For example, if you deployed the servlet on `http://www.yourcompany.com/here`, and you used the sample values shown in the previous table, the servlet would invoke the collaboration `PurchaseOrderCollab` (if it receives requests on `http://www.yourcompany.com/here`) on `http://www.yourcompany.com/here/PurchaseOrder`.

The `WBIC_URL_1_WRAPPER_MIME` property specifies the MIME type for the Wrapper Data Handler. If you specify more than one MIME type, you need multiple meta-objects. See [“Step 6: Create the child meta-object” on page 47](#) for information.

Sample properties file

An example of a properties file follows:

```
# Example properties file
ICS_SERVERNAME=Server1
ICS_VERSION=4.2
ICS_IORFILE=C:/myiorlocation/Server1InterChangeServer.ior
ICS_USERNAME=admin
ICS_PASSWORD=null
ICS_ENCRYPTED_PASSWORD=false
ICS_DISABLEENCRYPTION=true
WBIC_SERVLET_COUNT=1
WBIC_URL_1=PurchaseOrder
WBIC_URL_1_COLLAB=PurchaseOrderCollab
WBIC_URL_1_CHARENCODE=UTF-8
WBIC_URL_1_PORT=From
WBIC_URL_1_VERB=Create
WBIC_URL_1_WRAPPER_MIME=wbic_wrapper
```

Step 3: Specify the location of the servlet log file

Specify logging properties in the servlet property file. Specify the location of the servlet log file in the properties file by adding the following statement:

```
log4jappender.RollingFile.File=<log file location>
```

A sample Log4J Debug Properties file including an example of this statement follows:

```
#Log4J Debug Properties

#Possible Categories - debug/info/warn/error/fatal
#Default Category "error". Output to: stdout and RollingFile
log4j.rootCategory=debug,RollingFile
log4j.appender.RollingFile=org.apache.log4j.RollingFileAppender
#Log File Name
log4j.appender.RollingFile.File=D:\\_DEV\\servlet.log
log4j.appender.RollingFile.MaxFileSize=1000KB
#Number of backup files to keep
log4j.appender.RollingFile.MaxBackupIndex=10
log4j.appender.RollingFile.layout=org.apache.log4j.PatternLayout
log4j.appender.RollingFile.layout.ConversionPattern=%d{yyyy-MM-dd
HH:mm:ss} %-5p [%c{1}] - %m%n
```

Step 4: Edit the deployment descriptor

To provide an initialization parameter for the servlet, you edit the deployment descriptor for the servlet (web.xml). This file contains a parameter named WBIC_FILENAME, which points to the absolute location of the properties file. Edit the value of this parameter to point to the properties file.

Step 5: Specify the location of the Wrapper Data Handler

The WebSphere InterChange Server needs to know the location of the Wrapper Data Handler, so that it can load it. To specify the location:

1. Edit the start_server.bat file, which is located in the WebSphere InterChange Server installation directory.
2. Add bcgwbwrapperdh.jar to the CLASSPATH.
3. If you have installed the optional Attachment Data Handler, see [“Specify the location of the Attachment Data Handler” on page 66.](#)

Step 6: Create the child meta-object

Create a child meta-object that includes the MIME types for the request business object. The child meta-object structure is as follows:

Child meta-object attributes

Attribute	Description
ClassName	The class name (required), which points to the data handler class (com.ibm.bcg.integration.wbi.datahandlers.WBICWrapperDataHandler)

Child meta-object attributes

Attribute	Description
TopBOPrefix	The prefix is used to determine the name of the top-level object. If the request business object returned by the data handler configured for the request does not have the business-object-level <code>wbic_mainboname</code> application-specific information, the name of the top-level object is obtained by adding the <code>TopBOPrefix</code> to the name of the request business object.
<code>wbic_request_mime</code>	The MIME type of the data handler that the Wrapper Data Handler will invoke to process the payload of the request message. Make sure that this data handler has been configured so that it can be invoked by the Server Access Interface. Refer to the data handler documentation.
<code>wbic_response_mime</code>	The MIME type of the data handler that the Wrapper Data Handler will invoke to process the payload of the response message. The <code>wbic_response_mime</code> does not have to be set if Business Integration Connect is not expecting a response. The <code>wbic_response_mime</code> does not have to be set if Business Integration Connect is not expecting a response.

Step 7: Edit the `MO_Server_DataHandler`

The Server Access Interface uses the `MO_Server_DataHandler` meta-object to identify the data handlers it can use. You add a reference to this meta-object for each Wrapper Data Handler you will use. In other words, if you will be using more than one MIME type, you will need a Wrapper Data Handler reference for each MIME type.

Define a new attribute for each MIME type in the `MO_Server_DataHandler` meta-object and provide the associated child meta-object for that MIME type

For example, if the MIME type is `wbic_wrapper` and the associated child meta-object is `MO_DataHandler_WBICWrapper`, you add the following to the `MO_Server_DataHandler` meta-object:

```
name=wbic_wrapper
type=MO_DataHandler_WBICWrapper
```

Repeat this process for each MIME type.

NOTE: If you are using the Attachment Data Handler to process attachments, you must modify the `MO_Server_DataHandler`, as described in [“Configuring the Attachment Data Handler”](#) on page 66.

Refer to the Data Handler Guide for more information.

Step 8: Define the business objects

You create business object definitions to represent the information being sent from Business Integration Connect to the WebSphere InterChange Server. You must create definitions for:

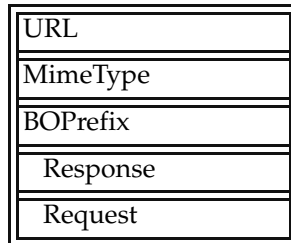
- A top-level business object
- A request business object

- A response business object

Note: If you are defining business objects for cXML documents, see [“Creating business objects for cXML” on page 52](#).

Top-level business object

The top-level business object has the following structure:



When you create the top-level business object definition, note the following:

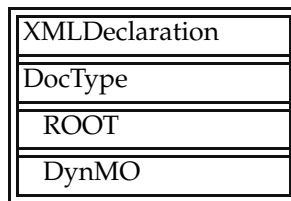
- The first three attributes (URL, MimeType, and BOPrefix) are not used by the Wrapper Data Handler.
- Request and Response are child business objects that correspond to request messages and to response messages (if you are expecting a response).

Note that the same business-object structure is used by the HTTP protocol handler and the Adapter for XML. Refer to the Adapter for XML documentation for more specific information.

If you are using the Attachment Data Handler to process attachments, you must modify your request business object, as described in [“Creating and modifying business objects” on page 69](#).

Request business object

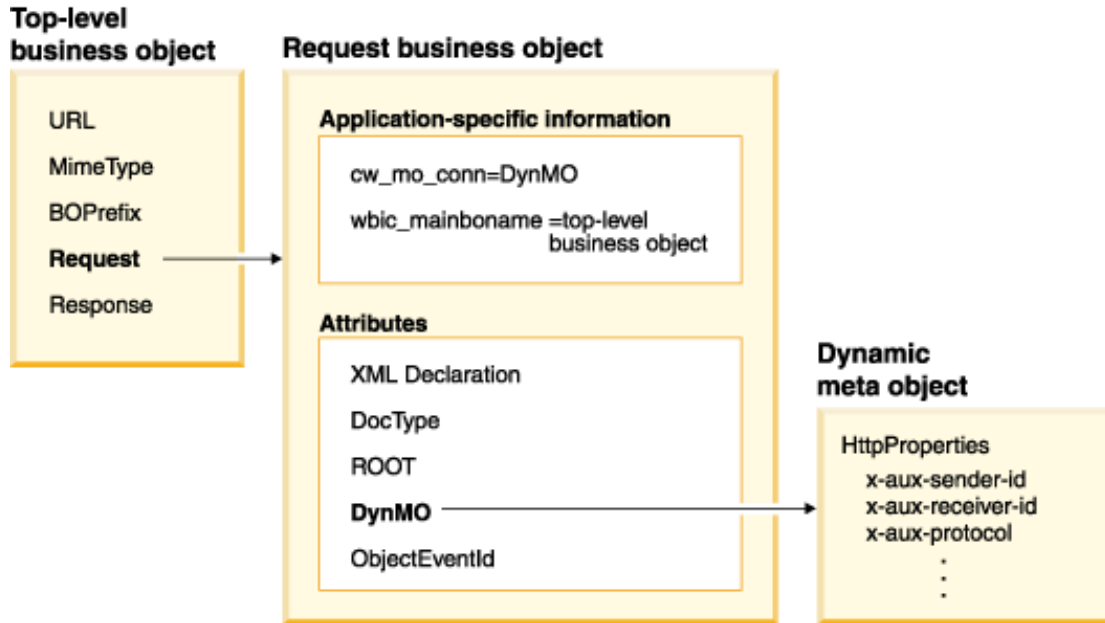
The following is an example of a Request business object containing the DynMO child object:



The request business object has the following business-object level application-specific attributes:

- `wbic_mainboname`, which gives the name of the top-level object.
- `cw_mo_conn`, which specifies the dynamic meta-object that contains an attribute representing HTTPProperties (containing the HTTP headers required when the packaging specified is Backend integration). The dynamic meta-object is described in more detail in [“Dynamic meta-object” on page 50](#).

The following illustration shows how the business objects and meta-object are related:



Relationship of the top-level business object, request business object, and dynamic child meta-object

Dynamic meta-object

As shown in the previous illustration, the dynamic meta-object contains the HTTPProperties business object. If you are using Backend Integration packaging, the HTTPProperties business object contains HTTP headers required by the packaging. It can also contain the Content-Type attribute, which specifies the content-type header to set in the request message.

The attributes of the HTTPProperties business object have attribute-level "name" application-specific information. This information specifies the name of the related protocol header. For example, the x-aux-sender-id attribute has the application-specific information set to name=x-aux-sender-id. The following table shows you the application-specific information for each attribute.

Note that this is not an exhaustive list of the headers required for backend integration. For a complete list and description of the headers, see [“Transport level header content” on page 19](#).

Application-specific information for the HTTPProperties attributes

Name	Application-specific information
x-aux-sender-id	name=x-aux-sender-id;
x-aux-receiver-id	name=x-aux-receiver-id;
x-aux-protocol	name=x-aux-protocol;
x-aux-protocol-version	name=x-aux-protocol-version;
x-aux-process-type	name=x-aux-process-type;
x-aux-process-version	name=x-aux-process-version;
x-aux-create-datetime	name=x-aux-create-datetime;
x-aux-msg-id	name=x-aux-msg-id;
x-aux-production	name=x-aux-production;
x-aux-system-msg	name-x-aux-system-msg;
x-aux-payload-root-tag	name=x-aux-payload-root-tag;
x-aux-process-instance-id	name=x-aux-process-instance-id;
x-aux-event-status-code	name=x-aux-event-status-code;
x-aux-third-party-bus-id	name=x-aux-third-party-bus-id;
x-aux-transport-retry-count	name=x-aux-transport-retry-count;
Content-Type	name=Content-Type;

Response business object

If the exchange between Business Integration Connect and WebSphere InterChange Server is asynchronous, Business Integration Connect does not expect a response, so it is not necessary to create a response business object.

If the exchange is synchronous and a business response is expected, the attribute corresponding to the response in the Top-level object should have the attribute-level application-specific information `wbic_type`, which should be specified as `wbic_type=reply`.

Additionally the `wbic_response_mime` business-object level application-specific information can be specified. This application-specific information is optional. It specifies the MIME type for the data handler to be used for the response Business Object. If this is not specified, the Wrapper Data Handler uses the `wbic_response_mime` child meta-object attribute to determine the data handler to use for the response.

Note that the response business object does not include the DynMO attribute.

Creating business objects for cXML

For cXML documents, you can use the XML Object Discovery Agent (ODA) to create the business objects. The XML ODA can consume the cXML DTD. Note, however, that the XML ODA does not support ENTITY. Therefore, before running the cXML DTD with the XML ODA, you need to remove ENTITY from the DTD.

When generating business objects using the XML ODA, you can select the cXML tag as your root element. This might result in a large business object, capturing the entire cXML DTD, however. If you want to create a smaller business object, you can select a different tag as your root element, which will require that you write a custom name handler for the Data Handler for XML. The data handler will invoke this name handler for the top-level business object name resolution. Refer to the Data Handler for XML documentation for information on writing custom name handlers.

Overview of sending documents to Business Integration Connect

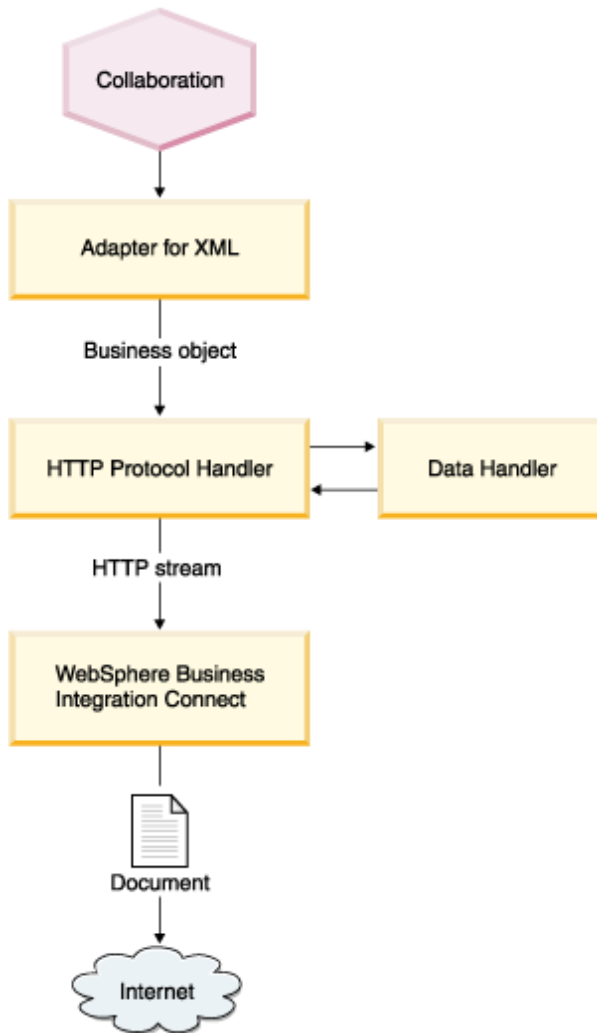
To send a document from the WebSphere InterChange Server using the HTTP transport protocol, you use either the HTTP or HTTPS protocol handler, which is supplied by Business Integration Connect.

You also use the Adapter for XML. Note the following about the Adapter for XML:

- The adapter is not shipped with Business Integration Connect. It is part of the WebSphere Business Integration Adapters.
- You must use the Adapter for XML version 3.1.x or higher.
- Refer to the adapter documentation to make sure that the level of the adapter is compatible with the version of WebSphere InterChange Server you are using.
- Only the request processing function of the Adapter for XML is used when WebSphere InterChange Server sends a document to Business Integration Connect. The event notification feature of the adapter is not used.

You also use a data handler that has been configured for the payload of the associated business object.

The following illustration provides an overview of how the components interact. Note that all references to the HTTP protocol handler apply to the HTTPS protocol handler as well.



Message flow from a collaboration to Business Integration Connect through the HTTP transport protocol

1. The collaboration makes a service call to the Adapter for XML, sending it a top-level business object that includes request and respond child objects. The request child object contains application-specific information pointing to a dynamic meta-object used to contain the custom HTTP headers expected by Business Integration Connect.
2. The Adapter for XML invokes the HTTP protocol handler.
3. The HTTP protocol handler reads the MIME type and URL from the top-level business object to determine the data handler to use and the address of the recipient.
4. From the top-level business object, the HTTP protocol handler obtains the first populated business object. This is the request business object.
5. The HTTP protocol handler calls the data handler to convert the business object to an HTTP stream. If the Attachment Data Handler is being used, the process described in [“Converting messages to business objects” on page 73](#) is performed.

6. The HTTP protocol handler determines, from the request business object, the dynamic meta-object.
7. The HTTP protocol handler searches for the business object `cw_mo_conn`, which provides the attribute corresponding to the dynamic meta-object. (In the dynamic meta-object, the protocol handler expects an `HTTPProperties` attribute, which is a child business object.)
8. If the `HTTPProperties` attribute is populated, the HTTP protocol handler sets the user-defined headers in the request message. You can also specify the content-type standard HTTP header. If you are using Backend Integration packaging, you need to specify the headers required by Business Integration Connect. See [“Transport level header content” on page 19](#).
9. The HTTP protocol handler invokes Business Integration Connect with the stream returned by the data handler and the protocol headers. Business Integration Connect responds with an HTTP 200 OK.

In the case of a synchronous invocation in which the `ReturnBusObjResponse` property of the Adapter for XML is set to true, the protocol handler converts the response message into a response business object and returns it to the Adapter for XML. The adapter sets the business object in the top-level business object. The top-level business object is then returned to the collaboration.

Setting up the environment for sending documents

Because the sending and receiving of documents to and from the WebSphere InterChange Server involves the use of Business Integration-supplied components, you have to perform the following setup and configuration tasks.

1. Deploy the HTTP protocol handler
2. Specify the location of the HTTP protocol handler
3. Configure the Adapter for XML
4. Define the business objects

These tasks are described in the sections that follow.

Step 1: Deploy the HTTP Protocol Handler

The protocol handler is available from the following location on the Business Integration Connect installation medium:

```
integration/wbi/wics/http/lib/bcgwbiprotocol.jar
```

Deploy the HTTP protocol handler to the Adapter for XML.

Step 2: Specify the location of the HTTP Protocol Handler

The Adapter for XML needs to know the location of the HTTP protocol handler, so that it can load it. To specify the location of the HTTP protocol handler:

1. Edit the `start_xml.bat` file.
2. Add `bcgwbiprotocol.jar` to the `CLASSPATH`.

Step 3: Configure the Adapter for XML

Specify connector properties for the Adapter for XML by setting:

```
JavaProtocolHandlerPkgs=com.ibm.bcg.integration.wbi.utils.protocolhandlers  
ReturnBusObjResponse=False
```

NOTE: If Business Integration Connect supports synchronous interactions for the packaging and business protocol used by the Community Manager, you might want to set the ReturnBusObjResponse to True and provide the response business object in your top-level business object.

Step 4: Define the business objects

The business objects required are the same as those described in the earlier section [“Step 8: Define the business objects” on page 48](#). Refer to the Adapter for XML documentation for information on the business objects.

Summary of supported platforms and versions

The following table summarizes the supported platforms and versions of the components used to send and receive documents through the HTTP transport protocol:

Platforms and versions of the components

Component	Platforms and versions
WebSphere Business Integration Connect Servlet	<p>This servlet can connect to WebSphere InterChange Server versions 4.1.1, 4.2.0, and 4.2.1.</p> <p>The servlet can be deployed on the platforms on which WebSphere InterChange Server is supported. Additionally you need to make sure that the Server Access Interface is supported on that platform. Refer to the WebSphere InterChange Server documentation for a list of the platforms on which the InterChange Server version you are using is supported.</p>
HTTP Protocol Handler for Adapter for XML	<p>The HTTP Protocol Handler can be plugged into the Adapter for XML version 3.1.x or higher.</p> <p>For a list of supported Interchange Server versions and platforms, refer to the documentation for the version of the Adapter for XML you are using.</p>

Using Web Services

The previous section described how you use Business Integration Connect-supplied components to exchange documents with the InterChange Server over HTTP/S. To send these documents, you use the servlet and Wrapper Data Handler, and to receive documents, you use the HTTP Protocol Handler.

SOAP documents, however, 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.

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.

Note the following about the Adapter for Web Services:

- Make sure you are using the Adapter for Web Services 3.1.0 (or higher)
- Refer to the adapter documentation to make sure that the level of the adapter is compatible with the version of WebSphere InterChange Server you are using.

As described in the Administrator Guide, you must have set up a target to receive Web service invocations from a backend application (the Web services target) as well as a target to receive Web service invocations from a Community Participant (the external Web services target).

Overview of how a Community Participant invokes a Web Service provided by the Community Manager

The following steps occur when a Community Participant sends a request for a collaboration that is exposed as a Web service.

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 Business Integration Connect instead of the actual endpoint.
2. Business Integration Connect receives and routes the message to the Adapter for Web services.
3. The adapter 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 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 Business Integration Connect. 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. Business Integration Connect routes the response to the Web service.

Overview of how the Community Manager invokes a Web Service provided by a Community Participant

The Public WSDL provided by Business Integration Connect 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 query string '?to=<Community Participant Web Service Provider's business ID>' (for example, `http://WBICHost/bcgreceiver/Receiver?to=123456789`). This tells Business Integration Connect that the provider of the Web service is the participant with business ID '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 Business Integration Connect.
3. Business Integration Connect 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, Business Integration Connect 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.

Using the JMS transport protocol

This section describes how to send and receive documents through the use of the JMS transport protocol.

To send or receive a document using the JMS transport protocol, you use the Adapter for JMS. If you are expecting to receive attachments, you can also use the Business Integration-supplied Attachment Data Handler.

Note the following about the Adapter for JMS:

- Make sure you are using the Adapter for JMS Version 2.3.1 (or higher), which provides support for custom header properties.
- Refer to the adapter documentation to make sure that the level of the adapter is compatible with the version of WebSphere InterChange Server you are using.

The Adapter for JMS supports JMS Text messages only.

Note: If you intend to use JMS Byte messages, use the Adapter for JMS Version 2.5.0.

The section [“Setting up integration through the JMS transport protocol using WebSphere MQ 5.3” on page 15](#) describes the steps you follow to set up your JMS environment, including setting up queues and channels. When the Adapter for JMS receives a message from a backend application, it places the message in its remote queue to be sent, through the send channel, to the receiver queue of Business Integration Connect.

After the queues and channels are established, you must also define a target and a gateway for the exchange.

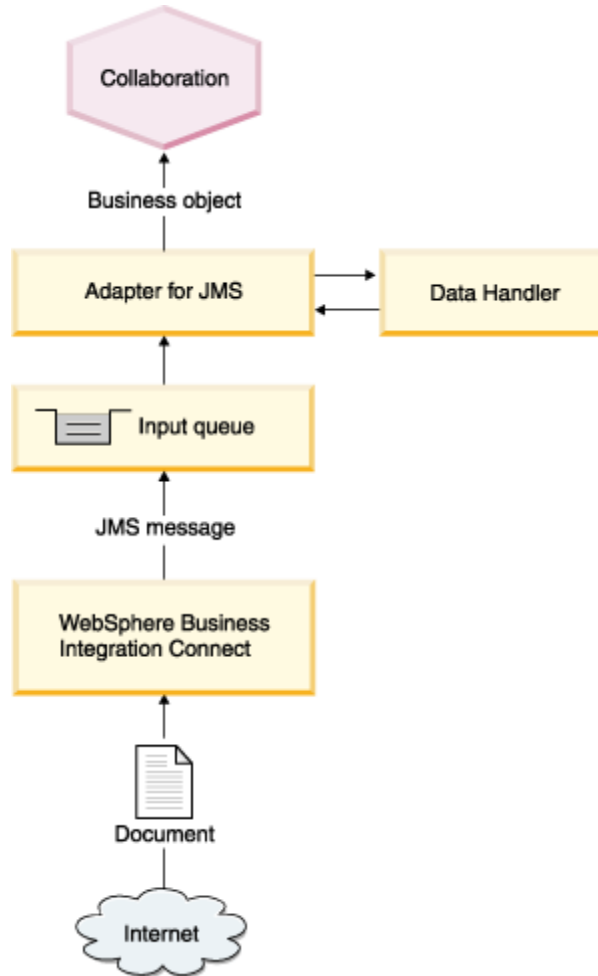
You must configure the target to read from the same queue in which the Adapter for JMS writes the message. The target listens for any incoming messages in this queue and retrieves them.

Similarly, when Business Integration Connect sends a message to a backend application, it puts the message in the queue on which the Adapter for JMS is polling.

You must configure a gateway in Business Integration Connect to write to the queue on which the Adapter polls for messages.

Overview of sending documents to the WebSphere InterChange Server

The following illustration provides an overview of how documents are sent to WebSphere InterChange Server.



Message flow from Business Integration Connect to a collaboration through the JMS transport protocol

1. Business Integration Connect posts a message to the JMS outbound queue. (Remember that you must have configured a gateway to write the message to the queue on which the Adapter for JMS is polling. See the [Administrator Guide](#) for details on configuring a gateway.) The message contains custom properties provided by Business Integration Connect (if the packaging of Backend Integration was selected when the connection was established) in addition to the message. The JMS message header, JMSType, is set with the content type of the payload.
2. As soon as the Adapter for JMS sees a message on one of its input queues, it retrieves the message. For detailed processing of the Adapter for JMS, see the Adapter for JMS documentation.
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.

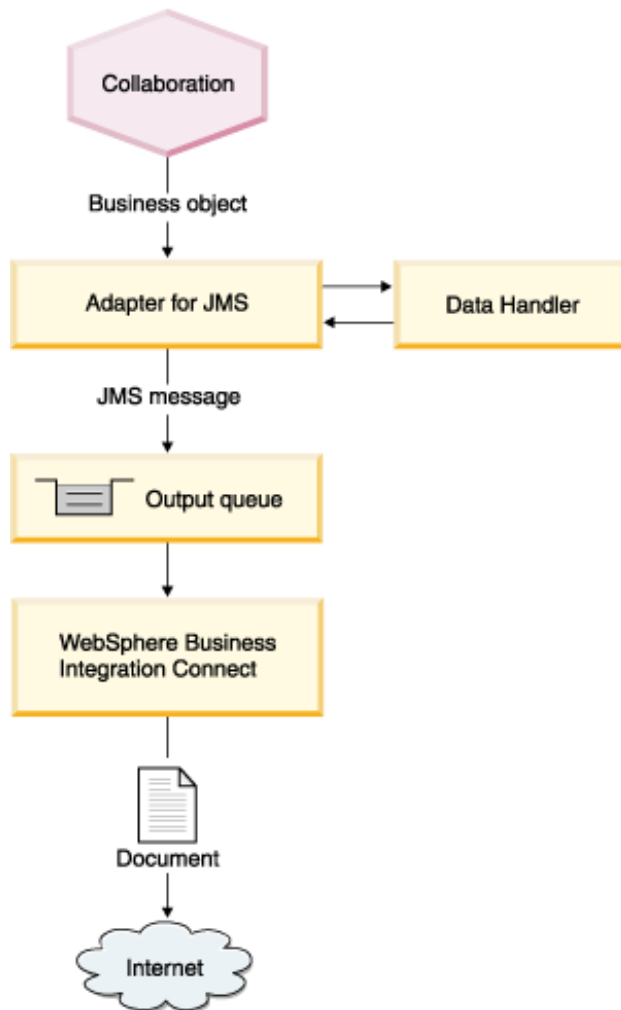
If you have installed the Attachment Data Handler and defined business objects used by the Attachment Data Handler, the Adapter for JMS creates an instance of the Attachment Data Handler. The process is described in [“Attachment Data Handler” on page 65](#).

5. The data handler returns the business object. If the Attachment Data Handler was used, the 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.

Note that, when Backend Integration packaging has been specified and the document contains attachments, the configured data handler is responsible for handling the payload and attachments.

Overview of sending documents to the Business Integration Connect

The following illustration provides an overview of how documents are sent from the WebSphere InterChange Server to Business Integration Connect:



Message flow from a collaboration to Business Integration Connect through the JMS transport protocol

1. The collaboration makes a service call to the Adapter for JMS.
2. The Adapter for JMS uses a data handler to convert the business object sent by the collaboration into a JMS message.

If the Attachment Data Handler is used, the Adapter for JMS creates an instance of the Attachment Data Handler and passes the business object to the Attachment Data Handler. This process is described in [“Attachment Data Handler” on page 65](#).

3. The data handler converts the business object to a string and returns it to the Adapter for JMS. The Adapter for JMS then processes the dynamic meta-object for custom JMS properties. If you are using Backend Integration packaging, you can specify JMS properties in the dynamic meta-object. The dynamic meta-object has an attribute called JMSProperties, which is of type child business object. This child business object contains the custom header properties. Refer to the Adapter for JMS documentation for information on the structure of this business object.

4. The Adapter for JMS creates a JMS message, using the string returned by the data handler. It then sets custom properties as defined in the dynamic meta-object.
5. The Adapter for JMS sends the message to a queue. The queue can be specified in the static meta-object or the dynamic meta-object. Refer to the Adapter for JMS documentation for information on specifying queues.
6. Business Integration Connect receives the message from the queue.

Note: You must have configured a target for this queue.

The Adapter for JMS can write only JMS text messages.

Business Integration Connect supports only asynchronous interaction with backend applications over JMS. Therefore, you might not want to wait for the response. The response from the community participant or Business Integration Connect 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 the InterChange Server as part of the event delivery.

The next section describes the tasks you perform to set up the environment so that you can send documents to the WebSphere InterChange Server.

Setting up the environment for sending and receiving documents

The sending and receiving of documents to and from the WebSphere InterChange Server involves the use of the Adapter for JMS.

The section [“Setting up integration through the JMS transport protocol using WebSphere MQ 5.3” on page 15](#) provides the general steps you perform before exchanging messages through the JMS transport protocol. The steps are summarized in this section.

To use the JMS transport protocol, you must create a JMS bindings file and configure a gateway and a target for JMS before sending or receiving documents from the WebSphere InterChange Server. See [“Creating the JMS bindings file” on page 15](#).

Before Business Integration Connect can send messages to the Adapter for JMS, you must perform the following steps:

- You must configure a gateway in Business Integration Connect to write to the queue. [“Creating the JMS gateway” on page 17](#) provides an overview of the steps for creating the gateway. A complete description can be found in the [Administrator Guide](#). Note that the Adapter for JMS supports JMS text messages only. The gateway should therefore be configured to write only JMS text messages.
- You must configure the Adapter for JMS to poll on the same queue.

The adapter can poll multiple queues.

Before Business Integration Connect can receive messages from the Adapter for JMS, you must perform the following steps:

- You must configure a target in Business Integration Connect for JMS. [“Creating the JMS target” on page 17](#) provides an overview of the steps for creating the target. A complete description can be found in the [Administrator Guide](#).
- You must make sure that the static or dynamic meta-objects are configured so that they can write to the queue on which the Business Integration Connect target is listening.

Creating business objects

The Adapter for JMS documentation provides information about the required business object structure. Refer to that information when defining your business objects.

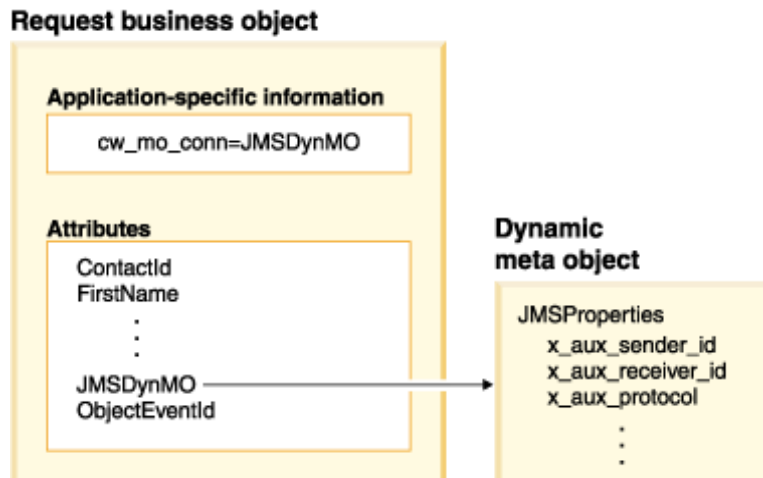
In addition, if you are using the Attachment Data Handler, see [“Attachment Data Handler” on page 65](#) for modifications you must make to the business object and for information on additional business objects that are required to handle attachments.

If you are using JMS with the Backend Integration packaging, make sure your business objects have a dynamic child meta-object and that you specify the `cw_mo_conn` application-specific information to point to that dynamic child meta-object.

1. Modify your business object to include the following application-specific attributes:

```
cw_mo_conn=JMSDynMO
```

The new attribute, `JMSDynMo`, contains the `JMSProperties` business object. If you are using Backend Integration packaging, the `JMSProperties` business object contains JMS properties required by the 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. The following illustration shows how the business objects and meta-object are related:



Relationship of the request business object to the dynamic child meta-object

2. Create the JMSProperties business object.

The attributes of the JMSProperties business object have attribute-level "name" application-specific information. This information specifies the name of the related protocol header. For example, the x_aux_sender_id attribute has the application-specific information set to name=x_aux_sender_id. The following table shows you the application-specific information for each attribute.

Note that this is not an exhaustive list of the headers required for backend integration. For a complete list and description of the headers, see ["Transport level header content" on page 19](#).

Application-specific information for the JMSProperties attributes

Name	Application-specific information
x_aux_sender_id	name=x_aux_sender_id;type=string
x_aux_receiver_id	name=x_aux_receiver_id;type=string
x_aux_protocol	name=x_aux_protocol;type=string
x_aux_protocol_version	name=x_aux_protocol_version;type=string
x_aux_process_type	name=x_aux_process_type;type=string
x_aux_process_version	name=x_aux_process_version;type=string
x_aux_create_datetime	name=x_aux_create_datetime;type=string
x_aux_msg_id	name=x_aux_msg_id;type=string
x_aux_production	name=x_aux_production;type=string
x_aux_system_msg	name_x_aux_system_msg;type=string
x_aux_payload_root_tag	name=x_aux_payload_root_tag;type=string
x_aux_process_instance_id	name=x_aux_process_instance_id;type=string
x_aux_event_status_code	name=x_aux_event_status_code;type=string
x_aux_third_party_bus_id	name=x_aux_third_party_bus_id;type=string
x_aux_transport_retry_count	name=x_aux_transport_retry_count;type=string
content_type	name=content_type;type=string
content_length	name=content_length;type=string

Attachment Data Handler

This section describes how the Attachment Data Handler processes messages and business objects that are sent from Business Integration Connect to the InterChange Server and from the InterChange Server to Business Integration Connection.

It also describes how to modify the default data handler meta-object and modify your business objects to support the use of the Attachment Data Handler.

Overview

As described in [“Attachments” on page 28](#), messages that contain attachments are sent by Business Integration Connect to an adapter or the Server Access Interface enclosed in an XML wrapper. The messages and attachments are contained in a defined structure (a transport envelope). The content type of the payload is specified in the <payload> XML tag, just as the content type for each attachment is specified in the <attachment> XML tag.

When the adapter or Server Access Interface calls the Attachment Data Handler to process the message, the Attachment Data Handler extracts the payload and attachments and then looks up the content type in its child meta-object to determine whether a content-type map exists for that type. For example, if the content type of the payload is application/xml, the Attachment Data Handler looks for a content-type map that matches application/xml. If it finds such a map, it uses the information in the map to determine which data handler to call.

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. See [“Attributes” on page 66](#) for a description of the child meta-object attributes and [“Example” on page 68](#) for an example of a meta-object.

When the Attachment Data Handler processes the payload and attachments, it sends the resulting business object back to the adapter or Server Access Interface that invoked it. (In the case of the Server Access Interface, it is actually the Wrapper Data Handler that calls the Attachment Data Handler.)

Similarly, when an adapter or the HTTP Transport Protocol calls the Attachment Data Handler to process a business object with attachments, the Attachment Data Handler calls the appropriate data handlers to convert the payload and attachments. It then produces an XML-wrapped message that is sent to Business Integration Connect.

These processes are described in detail in [“Converting messages to business objects” on page 73](#) and [“Converting business objects to messages” on page 74](#).

Setting up the environment for the Attachment Data Handler

To use the Business Integration Connect-supplied Attachment Data Handler, you must deploy it and specify its location, as described in the following sections.

Deploy the Attachment Data Handler

The Attachment Data Handler and associated repository file are located on the installation medium in the following directories:

Location of the components

Component	Location
Attachment Data Handler	integration/wbi/wics/attachment/bcgwbiattachmentdh.jar
Repository file	integration/wbi/wics/attachment/MO_DataHandler_DefaultAttachmentConfig.in

Deploy the files into the Web server according to the documentation for the Web server.

Specify the location of the Attachment Data Handler

The WebSphere InterChange Server needs to know the location of the Attachment Data Handler, so that it can load it. To specify the location of the Attachment Data Handler:

1. Edit the start_server.bat file, which is located in the WebSphere InterChange Server installation directory.
2. Add bcgwbiattachmentdh.jar to the CLASSPATH.

Configuring the Attachment Data Handler

Configuring the Attachment Data Handler consists of creating a child meta-object for the data handler, editing the top-level data-handler meta-object to include the new child meta-object, and setting up the other business objects required by the data handler.

Create the child meta-object

The child meta-object provides the class name and configuration properties that are needed by the Attachment Data Handler. Create a child meta-object that includes MIME types for the payload and for the types of attachments you expect to receive.

Attributes

The attributes of the child meta-object are shown in the following table. An example of a child meta-object for the Attachment Data Handler is shown in [“Example” on page 68](#). Note that the sample business objects shown in this chapter do not include the standard attributes (such as ObjectEventId) required by the WebSphere InterChange Server but not used by the Attachment Data Handler.

Child meta-object attributes

Attribute Name	Description
ClassName	The class name (required), which points to the data handler class (com.ibm.bcg.DataHandlers.AttachmentDataHandler).
ContentTypeMap_x	<p>The content-type map for the payload and each type of attachment you expect to receive, which determines the data handler to call. The ContentTypeMap_x attribute can be defined for both the payload and the attachments because, when a message is received in an XML wrapper, the payload might have a content type of application/xml that should be mapped to the MIME type text/xml.</p> <p>Note that you must order the ContentTypeMap_x attributes in sequence. For example, if you have three content type maps, they must be named ContentType_1, ContentType_2, and ContentType_3.</p> <p>The four values you can set in the ContentTypeMap_x attribute are described in the following list. Two samples of their use are shown in "Example" on page 68.</p> <ul style="list-style-type: none"> • ContentType - Required The actual content type (for example, text/xml). • MimeType - Optional The MIME type used to instantiate a data handler to convert the payload or attachment to a business object. If you do not specify MimeType, the value of ContentType is used. • CharSet - Optional The character set (for example, UTF-8) used to convert bytes to a string or a string to bytes. If you do not specify CharSet, 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). • ConvertAttachment - Optional An indicator of whether the attachment should be converted to a business object. The default is False.
Payload DataHandlerMimeType	MIME type used to create an instance of a data handler to process a payload that does not have associated attachments.

Example

The following sample shows an example of the attributes and values described in the previous section:

```
[BusinessObjectDefinition]
Name = MO_DataHandler_DefaultAttachmentConfig
Version = 3.0.0

  [Attribute]
  Name = ClassName
  Type = String
  MaxLength = 255
  IsKey = false
  IsForeignKey = false
  IsRequired = false
  DefaultValue = com.ibm.bcg.DataHandlers.AttachmentDataHandler
  IsRequiredServerBound = false
  [End]

  [Attribute]
  Name = ContentTypeMap_1
  Type = String
  MaxLength = 255
  IsKey = false
  IsForeignKey = false
  IsRequired = false
  DefaultValue = ContentType=application/xml;MimeType=text/xml;
                CharSet=UTF-8;ConvertAttachment=true
  IsRequiredServerBound = false
  [End]

  [Attribute]
  Name = ContentTypeMap_2
  Type = String
  MaxLength = 255
  IsKey = false
  IsForeignKey = false
  IsRequired = false
  DefaultValue = ContentType=text/xml;MimeType=text/xml;CharSet=UTF-8
  IsRequiredServerBound = false
  [End]

  [Attribute]
  Name = PayloadDataHandlerMimeType
  Type = String
  MaxLength = 255
  IsKey = false
  IsForeignKey = false
  IsRequired = false
  DefaultValue = text/xml
  IsRequiredServerBound = false
  [End]
```

Update 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. The Server Access Interface uses the MO_Server_DataHandler for the same purpose. Add a reference to the Attachment Data Handler in one of these meta-objects.

For example, if the MIME type is wbic_attachment and the associated child meta-object is MO_DataHandler_DefaultAttachmentConfig, add the following to the MO_DataHandler_Default meta-object or the data handler configuration meta-object:

```
name=wbic_attachment
type=MO_DataHandler_DefaultAttachmentConfig
```

Creating and modifying business objects

This section describes business objects that you will create or modify to handle attachments. A brief overview of these business objects is provided, and then the steps for creating and modifying the objects are listed.

Description of the business objects

If you are dealing with attachments, you modify your payload business object to include attributes for attachments.

In any document flow, there is one payload and, optionally, multiple attachments. All the attachments are contained in the Attachment Container business object. If there are attachments, the payload business object has an attribute corresponding to the Attachment Container business object:

Payload Business Object example

Application-specific information	
cw_mo_bcg_attachment=attachments	
Attributes	
Payload	<payloadBO type>
attachments	AttachmentContainerBO

Each attribute in the Attachment Container business object represents an attachment, if determined by the attribute-level application-specific information wbic_type=Attachment of that attribute. The Attachment Container business object can optionally have an attribute corresponding to the Default Attachment business object.

Attachment Container Business Object example

Application-specific information		
cw_mo_bcg_default_attribute=defaultAttachment		
Attributes		
defaultAttachment	Default_Attachment_BO	
attachmentOne	AttachmentBO	wbic_type=Attachment
attachmentTwo	AttachmentBO	wbic_type=Attachment

Each attachment attribute in the Attachment Container business object should be of type Attachment Business Object. The Attachment business object can contain a Content Type Encoding business object.

Attachment Business Object example

Application-specific information	
cw_mo_bcg_content_info=contentTypeEncoding	
Attributes	
attachment	
contentTypeEncoding	contentTypeEncodingBO

Steps for creating and modifying the business objects

This section describes how to create or modify the business objects described in the previous section.

1. Create the Content Type Encoding business object.

This business object stores the content type and encoding of the associated payload or attachment. The following is an example of creating the Content Type Encoding business object. Note that "ContentTypeEncoding" is just an example of a name that can be assigned to this business object. The name can be anything. The application-specific information of the attachment business object determines if this is a Content Type Encoding business object type.

```
[BusinessObjectDefinition]
Name = ContentTypeEncodingBO
Version = 3.0.0

[Attribute]
Name = contentType
Type = String
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
```

```

[End]

[Attribute]
Name = encoding
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

```

2. Define the structure of the Default Attachment business object. This business object has the following business-object level application-specific information:

```

cw_mo_bcg_content_info=contentTypeEncoding

```

An example of the business object follows:

```

[BusinessObjectDefinition]
Name = Default_Attachment_BO
Version = 3.0.0
AppSpecificInfo =
cw_mo_bcg_content_info=contentTypeEncoding

```

```

[Attribute]
Name = attachment
Type = String
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

```

```

[Attribute]
Name = contentTypeEncoding
Type = ContentTypeEncodingBO
ContainedObjectVersion = 3.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

```

3. Add Content Type Encoding attributes to the Attachment Business Object. Modify the business object to be used as an attachment by adding the business-object-level application-specific information `cw_mo_bcg_content_info`. The value of this application-specific information gives the name of the attribute, which is of type Content Type Encoding Business Object.

```
cw_mo_bcg_content_info=contentTypeEncoding
```

4. Define the Attachment Container business object. If you are using default attachment processing, add the Default Attachment Business Object as a child of the Attachment Container Business Object. If you are adding the Default Attachment Business Object, you need to add the business-object-level application-specific information `cw_mo_bcg_default_attribute` in the Attachment Container Business Object. The value of this application-specific information gives the name of the attribute that is of type Default Attachment Business Object. There can be only one such attribute in the Attachment Container Business Object. This attribute can have multiple cardinality.

Add all of your Attachment Business Objects as children of the Attachment Container Business Object. All the attributes corresponding to type Attachment Business Object should have the attribute-level application-specific information `wbic_type = Attachment`. Note that these attributes can have multiple cardinality.

An example of the business object follows:

```
[BusinessObjectDefinition]
Name = AttachmentContainerBO
Version = 3.0.0
AppSpecificInfo =
cw_mo_bcg_default_attribute=defaultAttachment
```

```
[Attribute]
Name = defaultAttachment
Type = Default_Attachment_BO
ContainedObjectVersion = 3.0.0
Relationship = Containment
Cardinality = N
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
```

```
[Attribute]
Name = attachmentOne
Type = <attachment-type BO>
ContainedObjectVersion = 3.0.0
Relationship = Containment
Cardinality = N
MaxLength = 255
```



```
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = wbic_type=Attachment
IsRequiredServerBound = false
[End]
```

5. Modify the payload business object by adding the `cw_mo_bcg_content_info` and `cw_mo_bcg_attachment` attribute-level application-specific information.
 - The value of the `cw_mo_bcg_content_info` application-specific information gives the name of the attribute in the payload business object that is of type Content Type Encoding Business Object.
 - The value of `cw_mo_bcg_attachment` attribute gives the name of the attribute in the payload business object that is of type Attachment Container Business Object.
6. Add the AttachmentContainer business object as a child of the payload business object.

Converting messages to business objects

A WebSphere Business Integration adapter (for example, the Adapter for JMS) or the Wrapper Data Handler can be configured to call the Attachment Data Handler to handle the payload and attachments of an XML-wrapped message.

The Attachment Data Handler does the following:

1. Loads the content-type maps defined in its configuration meta-object.
2. Checks the message to see whether it is included in an XML Wrapper.

If the Attachment Data Handler does not detect the wrapper, it uses the `PayloadDataHandlerMimeType` configuration property defined in the meta-object to create an instance of a data handler for the message payload. It then returns the resulting business object to the adapter or to the Wrapper Data Handler.

If the Attachment Data Handler does detect the wrapper, it does the following:

- a. Extracts the payload and attachments from the wrapper and decodes the payload data
- b. Uses the MIME type specified in the `ContentTypeMap_x` for the payload to create an instance of a data handler to process the payload data, and receives the resulting business object
- c. Examines the payload business object and retrieves the attribute name for `cw_mo_bcg_content_info` and its type, creates an instance of the Content Type Encoding Business object, and sets the values for the content type and encoding
- d. Examines the payload business object, retrieves the attribute name for `cw_mo_bcg_attachment` and its type, and creates an instance of the Attachment Container Business Object.

- e. Examines the attachment container business object, and, using the business object level attribute `cw_mo_bcg_default_attribute`, retrieves the attribute name for the Default Attachment Business Object and its type
- f. Gets the content type and character-set encoding for the attachment and checks to see whether there is a corresponding entry in the content-type map.
 - If no corresponding content-type map is found, the Attachment Data Handler creates an instance of the Default Attachment Business Object, sets the values for the content type and encoding within the `contentTypeEncoding` attribute, and sets the base64-encoded attachment data (as a string) in the attachment attribute.

The Attachment Data Handler then populates the `defaultAttachment` object of the Attachment Container 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 it does, the Attachment Data Handler decodes the payload data and creates an instance of a data handler to process the payload data. The data handler processes the decoded bytes and returns a business object.

If the attribute for the Content Type Encoding Business Object is not found, the attachment is not processed.

3. The Attachment Data Handler sets the resulting attachment business object in the Attachment Container business object, and returns the business object to the adapter or Wrapper Data Handler.

Converting business objects to messages

A WebSphere Business Integration adapter (for example, the Adapter for JMS or the Adapter for XML) can be configured to call the Attachment Data Handler to convert a business object sent by the collaboration into a JMS message or HTTP stream.

The Attachment Data Handler then does the following:

1. Loads the content-type maps defined in its configuration meta-object.
2. Checks the business object to determine whether it contains attachments.

If it does not contain attachments, the Attachment Data Handler uses the `PayloadDataHandlerMimeType` (which is defined in `MO_DataHandler_DefaultAttachmentConfig`) to create an instance of that data handler, passes the business object to the data handler, and receives the resulting string.

If it does contain attachments, the Attachment Data Handler:

- a. Gets the content type for the payload from the attribute for the Content-Type EncodingBO. If the Content Type Encoding Business Object is not found for the payload, the default data handler MIME type is used for the conversion of the payload.
- b. Gets the character set encoding, checks to see if there is an entry for the content type in the content-type map, and creates an instance of a data handler. From the string that is returned by the data handler, the Attachment Data Handler encodes the bytes using Base64 and stores the result.
- c. Checks to see whether the attachment is of type Default Attachment Business Object. If it is, the Attachment Data Handler retrieves the attribute for the business object, extracts the Base64-encoded data, and stores the result.

If the attachment is not of type Default Attachment Business Object, the Attachment Data Handler:

1. Retrieves the attribute for the business object and gets the content type and encoding for the attachment
2. Gets the character-set encoding from the content type and checks to see whether there is a corresponding content type in the content-type map.
3. Uses the MIME type to create an instance of a data handler for the MIME type, passes the attachment business object to the data handler, and gets the bytes from the string that is returned (using the character set, if one was present)
4. Encodes the bytes using Base64, and stores the results.

If the business object contains attachments, the Attachment Data Handler wraps the payload and attachment data in the XML Wrapper.

Chapter 3. Integrating with WebSphere Data Interchange

“Backend Integration ” on page 7 described the general process used to integrate Business Integration Connect with a backend application. This chapter describes a specific implementation of that process—how to integrate Business Integration Connect with the WebSphere Data Interchange.

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.

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 Business Integration Connect 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 that 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.

Sending documents to WebSphere Data Interchange

This section describes the process by which an EDI document is sent from Business Integration Connect to WebSphere Data Interchange:

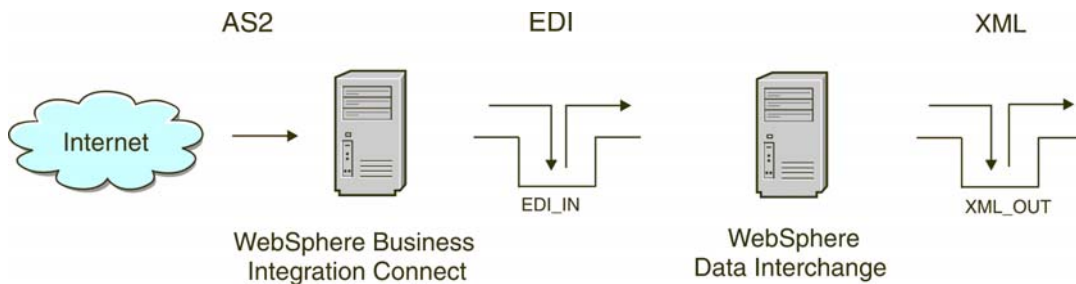
1. An EDI document is sent from a Community Participant to Business Integration Connect. The document is sent through the AS2 over HTTP transport protocol. Business Integration Connect strips off the AS2 packaging from the EDI document.
2. Business Integration Connect places the EDI document on a queue.

Note: WebSphere Business Integration Connect 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 105](#) for details.

3. WebSphere Data Interchange reads the message from the queue. It performs the tasks of deenveloping, validating, and translating the EDI document.

Note: WebSphere Data Interchange must be configured for user profiles and the desired mappings.

4. WebSphere Data Interchange distributes the document to a back-end application, or it uses the Adapter for MQ to interact with the WebSphere InterChange Server to create a business object and invoke a collaboration within the WebSphere InterChange Server.



In the illustration, a Community Participant sends an AS2 document to Business Integration Connect, 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 Business Integration Connect 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.

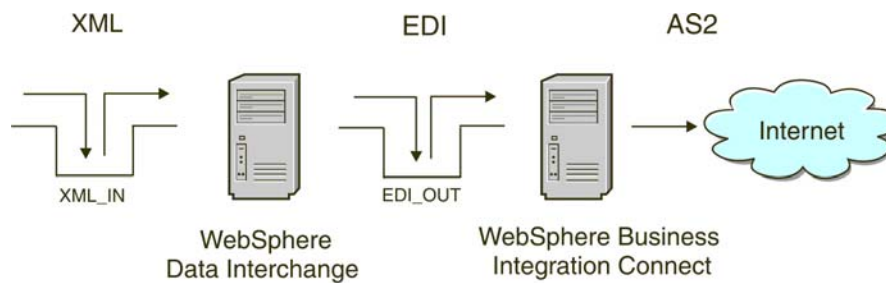
Sending documents to Business Integration Connect

This section describes the process by which an EDI document is sent from WebSphere Data Interchange to Business Integration Connect:

1. WebSphere Data Interchange places the EDI document on a queue.
2. Business Integration Connect reads the message from the queue.

Note: Business Integration Connect determines how to route the document as described in [“Overview of EDI routing” on page 105](#).

3. Business Integration Connect routes the document to the Community Participant.

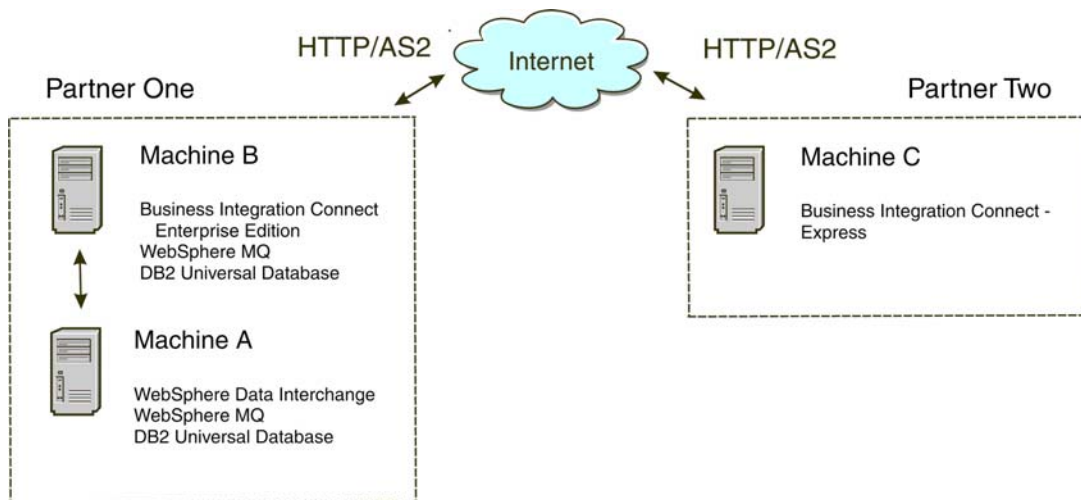


In the above illustration, an EDI document is placed into the XML_IN queue for WebSphere Data Interchange to translate. It is assumed that the user profiles, mappings, and so on, are already performed. Upon receiving a valid XML document, WebSphere Data Interchange converts it into EDI format and places the output in the EDI_OUT queue (a remote queue). It is assumed that the transmission queue, sender and receiver channels, and receiver queue on the Business Integration Connect side are set up. Upon receiving the document, Business Integration Connect routes it to the Community Participant.

Sample scenario used in this chapter

Throughout this chapter, you will see the steps you would take 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.

The trading partners involved in the sample are partnerOne and partnerTwo. The following figure illustrates the configurations of the two partners:



The following software is used to implement this sample scenario. Refer to the Business Integration Connect Installation Guide and to the WebSphere Data Interchange documentation for a complete list of software prerequisites.

- On Machine A (Partner One):
 - Operating System: Microsoft Windows 2000 Professional
 - WebSphere Data Interchange Server V3.2 with Fix Pack 7 (or higher)
 - WebSphere Data Interchange Client V3.2 with CSD 07 (or higher)
 - WebSphere MQ V5.3 with CSD 04
 - IBM DB2 V7.2 with Fix Pack 10
- On Machine B (Partner One):
 - Operating System: Red Hat Linux Advanced Server v2.1
 - WebSphere Business Integration Connect Enterprise Edition v4.2.0 (or higher)
 - WebSphere MQ V5.3 with CSD 04
 - IBM DB2 V8.1 with Fix Pack 2
- On Machine C (Partner Two):
 - Operating System: Windows 2000 Professional
 - WebSphere Business Integration Connect Express v4.2.0 (or higher)

In this example, partnerOne is operating two machines. Machine A has both WebSphere MQ and WebSphere Data Interchange Server installed. Machine B has WebSphere MQ as well as WebSphere Business Integration Connect Enterprise Edition installed. Machine B supports the communications between the two trading partners.

WebSphere Data Interchange supports integration with WebSphere MQ, enabling interoperability with a wide range of enterprise applications and business process engines. WebSphere Business Integration Connect employs WebSphere MQ as a JMS provider. As such, integration between WebSphere Data Interchange and WebSphere Business Integration Connect is through MQ messages destined for JMS API clients.

WebSphere Business Integration Connect is used to communicate EDI transactions over the Internet using the AS2 protocol.

Note that, in this example, partnerTwo is using WebSphere Business Integration Connect - 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 machines 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

The sections that follow list the setup and configuration tasks you perform to enable communication between WebSphere Data Interchange and Business Integration Connect.

Configure WebSphere MQ communication

The first step in setting up the environment is to configure WebSphere MQ communication.

Overview

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 Business Integration Connect 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.

Refer to the WebSphere MQ documentation for additional details on defining queue managers.

Example

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 Machine A and Machine B. The first step, then, is to create a queue manager on both Machine A and Machine B for use with WebSphere Data Interchange and WebSphere Business Integration Connect 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 Machine 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 Machine B, you would use the queue manager created during the initial installation and configuration of WebSphere Business Integration Connect Enterprise Edition. For the remainder of this chapter, this queue manager is referred to as WBIC42_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

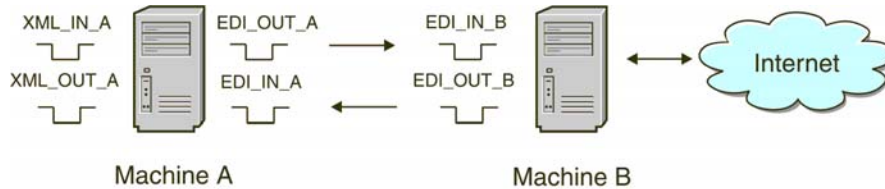
In the sample scenario, both Machine A and Machine B act as sender and receiver. Therefore, you would have to define a number of objects on each machine.

Table 1 below lists the objects you would create to set Machine A and Machine B as sender and receiver:

Table 1: WebSphere MQ Objects to create

	Machine A	Machine B
Queue Manager	WDI32_QM	WBIC42_QM
Sender Channel	TO.WBIC42	TO.WDI32
Receiver Channel	TO.WDI32	TO.WBIC42
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

The following figure is a graphical illustration of the message flow between Machine A and Machine B, indicating the role of the WebSphere MQ objects listed above.



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.

Configure 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.

Overview of the profiles

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

MQSeries queue profile

An MQSeries Queue profile contains information about a WebSphere MQ message queue.

The properties to configure for each profile are:

- Queue Profile ID, which is the unique identifier to name the profile (logical name).
- Full Queue Name, which is the actual name of the WebSphere MQ queue.
- Queue Manager Name, which is the actual name of the WebSphere MQ queue manager.
- Description, which is any string to identify the purpose of the profile.
- Maximum Length, which is the largest possible message for the queue as configured in WebSphere MQ.
- Destructive Reads, which, if selected, cause WebSphere Data Interchange to remove the message from the WebSphere MQ queue when reading.
- Syncpoint Control, which, when checked, means that 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.

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.

The properties to configure for the Network Profile are:

- Network ID, which is a unique identifier to name the profile
- Communication Routine, which is the name of the program that builds network commands and invokes the network program to process the commands
- Network Program, which is the program invoked by the communication routine to process requests
- Network Parameters, which are parameters required by the network program

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.

The properties to configure for each Mailbox Profile are:

- Mailbox ID, which is a unique identifier to name the profile
- Network ID, which is the network ID of the network profile created earlier

Service profile

The purpose of Service Profiles is to allow you to enter a utility command and define all the files that will be used during execution of that command.

Example

In the sample scenario, WebSphere Data Interchange will receive XML messages from the WebSphere MQ queue XML_IN_A and will write the result of translation to WebSphere MQ queue EDI_OUT_A. WebSphere Data Interchange will also receive EDI from WebSphere Business Integration Connect Enterprise Edition on the WebSphere MQ queue EDI_IN_A and will write the result of translation to XML_OUT_A.

Configure MQSeries Queue profiles

Because you're 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. The actual parameters specified in each MQSeries Queue profile created are listed in [Table 2](#) below. The queues represented here are used with XML-to-EDI translation.

Table 2: MQSeries Queue profiles for XML_IN_A and EDI_OU_A

	XML_IN_A	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 eight characters only (hence the name 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_OUT_A. [Table 3](#) below defines the properties of each queue used in EDI-to-XML translation.

Table 3: MQSeries Queue profiles for EDI_IN_A and XML_OU_A

	EDI_IN_A	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

Configure Network Profiles

For this scenario, you would create and configure a Network Profile that communicates with the WebSphere MQ queues created earlier. You would:

1. Create a new Network Profile called WBIC_IN. This network profile is used in the XML-to-EDI scenario. The actual parameters specified for WBIC are listed in [Table 4](#) below:

Table 4: Network Profile for WBIC_IN

Network ID	WBIC_IN
Communication Routine	VANIMQ
Network Program	EDIMQSR
Network Parameters	SENDMQ=EDI_OU_A RECEIVEMQ=XML_IN_A

2. Create a second Network Profile called WBIC_OUT. This network profile is used in the translation of EDI received from WebSphere Business Integration Connect Enterprise Edition. A second Network Profile is required, because WebSphere Business Integration Connect Enterprise Edition places messages on the WebSphere MQ queues that include RFH2 headers. The properties of WBIC_OUT are listed in [Table 5](#) below:

Table 5: Network Profile for WBIC_OUT

Network ID	WBIC_OUT
Communication Routine	VANIMQ
Network Program	EDIRFH2
Network Parameters	SENDMQ=XML_OU_A RECEIVEMQ=EDI_IN_A

Configure Mailbox Profiles

You create mailbox profiles for each of the WebSphere MQ queues, to identify the individuals and groups in the organization. You would:

1. Create a Mailbox Profile for each WebSphere MQ queue used. The actual parameters used in each of the Mailbox Profiles are shown in [Table 6](#):

Table 6: Mailbox Profiles for XML_IN_A and EDI_OU_A

	XML_IN_A	EDI_OU_A
Mailbox ID	XML_IN_A	EDI_OU_A
Network ID	WBIC_IN	WBIC_IN
Receive File	XML_IN_A	EDI_OU_A

2. Create a second pair of mailboxes. The properties for each are shown in [Table 7](#) below:

Table 7: Mailbox Profiles for EDI_IN_A and XML_OU_A

	EDI_IN_A	XML_OU_A
Mailbox ID	EDI_IN_A	XML_OU_A
Network ID	WBIC_OUT	WBIC_OUT
Receive File	EDI_IN_A	XML_OU_A

Configure Service Profile

You use the Service Profile to be able to enter a utility command and define all the files that will be used during execution of that command. For the sample scenario, you would:

1. Create a new Service Profile for XML_IN_A. The properties to be defined under the **General** tab are as follows:

- Continue Command Chaining: **On Success**
- PERFORM Command:
PERFORM TRANSFORM WHERE INFILE (XML_IN_A) SYNTAX (X)
OUTTYPE (MQ) OUTFILE (EDI_OU_A)

Common Files properties are defined in [Table 8](#):

Table 8: Common Files for XML_IN_A

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. The properties to be defined under the **General** tab are as follows:

- Continue Command Chaining: **On Success**
- PERFORM Command:
PERFORM TRANSFORM WHERE INFILE(EDI_IN_A) SYNTAX(E)
OUTTYPE(MQ) OUTFILE(XML_OU_A)

The Common Files properties are as shown in [Table 9](#):

Table 9: Common files for EDI_IN_A

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 sections, 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.

Set up the JMS environment

As mentioned earlier in this chapter, WebSphere Business Integration Connect Enterprise Edition uses the WebSphere MQ implementation of the Java Message Service for integration with WebSphere Data Interchange. This section outlines the steps involved in creating a JMS environment on Machine B.

WebSphere MQ classes for Java and WebSphere MQ classes for Java Message Service (JMS) are built in to WebSphere MQ for Windows V5.3.

Configure JMSAdmin

Use the JMSAdmin tool available in WebSphere MQ V5.3 to create the JMS objects in JNDI.

The JMSAdmin tool by default uses a configuration file called JMSAdmin.config that resides in the bin directory of WebSphere MQ Java (/opt/mqm/java/bin).

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. Create the JNDI directory under /opt/mqm/java if it doesn't exist already.

Note: It is possible to use LDAP or WebSphere Application Server as a JNDI provider too.

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, 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 connecton factory:

```
DEF QCF(WBIC42_QM_QCF) TRAN(CLIENT) HOST(IP_MACHINE_B)  
PORT(9999) CHAN(java.channel) QMANAGER(WBIC42_QM)
```

4. Define the EDI_IN_B queue:

```
DEF Q(EDI_IN_B) QMANAGER(WBIC42_QM) QUEUE(EDI_IN_B)
```

5. Define the EDI_OUT_B queue:

```
DEF Q(EDI_OUT_B) QMANAGER(WBIC42_QM) QUEUE(EDI_OUT_B)
```

6. End the JMSAdmin session

```
END
```

Configure Business Integration Connect

WebSphere Business Integration Connect is the communication layer between disparate Community Participants and internal processes. When setting up Business Integration Connect to work with EDI documents, you can configure it to:

- Send and receive EDI to and from WebSphere Data Interchange
- Communicate EDI transactions with external trading partners using AS2

Overview

The *Administrator Guide* provides complete information on configuring WebSphere Business Integration Connect Enterprise and Advanced editions. The following overview summarizes the steps you take to configure Business Integration Connect. [“Example of configuring Business Integration Connect Enterprise Edition” on page 94](#) provides an example of a setup for the sample scenario described throughout this chapter.

To configure Business Integration Connect, you set up:

1. Participants

A participant profile identifies companies to the system. You create participants in the WebSphere Business Integration Connect Enterprise Edition Community Console.

2. B2B Capabilities

You define the B2B capabilities for each participant in WebSphere Business Integration Connect 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.

3. Gateways

A gateway in Business Integration Connect defines a network point that acts as the entrance to another network. The gateway contains the information that tells WebSphere Business Integration Connect how to deliver documents to the Enterprise Application Integration (EAI) layer.

4. Document Flow Definitions

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.

5. 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.

6. Targets

The Target List screen provides location information that enables the 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.

The remainder of this chapter provides you with the steps for configuring Business Integration Connect for the sample scenario. It also provides you with the steps you would use to configure the Community Participant's environment (in this case, a WebSphere Business Integration Connect - Express system).

Example of configuring Business Integration Connect Enterprise Edition

This section provides you with an example of configuring the WebSphere Business Integration Connect Enterprise Edition that is described in the sample scenario.

Create a Participant for Partner One

You would first create a participant profile to represent Machine A and Machine B, which are the two systems owned by Partner One.

To create a new participant profile, you would:

1. Open the WebSphere Business Integration Connect Community Console.
2. Log in as the **Hub Operator**.
3. Verify that **Profiles** is already selected from the Account Admin menu.
4. Click **Create** and enter the details as listed in [Table 10](#) below:

Table 10: Partner One's properties

Participant Login Name	partnerOne
Participant Name	Partner One
Participant Type	Community Manager
Status	Enabled
Vendor Type	Other
Web Site	http://IP_MACHINE_A
Business ID Type	Freeform
Business ID Identifier	123456789
IP Address Gateway Type	Production
IP Address	IP_MACHINE_A

Note: To create the Business ID Type and Business ID Identifier, you first click on the **New** button below Business ID. The Business ID must be unique. Similarly, to create details relating to the IP Address, you click on the **New** button below the IP Address header.

IP_MACHINE_A refers to the internet protocol (IP) address of Machine A.

5. Click **Save**.

The Business ID Identifier as defined above is used by WebSphere Business Integration Connect Enterprise Edition as a means of identifying 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.

You would need to make a note of the Administrator's Password. When you logged on to the Community Console as Partner One, you would be asked to enter the password and then to change it.

Create a participant for Partner Two

You would next create a Community Participant to represent partnerTwo. To create the participant, you would:

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 [Table 11](#) below:

Table 11: Partner Two's properties

Participant Login Name	partnerTwo
Participant Name	Partner Two
Participant Type	Community Participant
Status	Enabled
Vendor Type	Other
Web Site	http://IP_MACHINE_C
Business ID Type	Freeform
Business ID Identifier	987654321
IP Address Gateway Type	Production
IP Address	IP_MACHINE_C


Note: IP_MACHINE_C refers to the IP address of Machine C.


4. Click **Save**.

Again, you would make a note of the Administrator's Password, which would be required when you logged on to the Community Console as Partner Two.






Set the B2B capabilities for Partner One

To define the B2B Capabilities for Partner One, you would:

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  icon next to **Partner One**, and then click **B2B Capabilities**.


B2B Capabilities are set to active by clicking on the  icon. For the purposes of this sample, only the B2B Capabilities required to implement the scenario will be configured.

To set the source and target packaging for Partner One to None, you would:






1. Click the  icon underneath **Set Source for Package: None** to enable it. Repeat this step for **Set Target**.
2. Click the  icon to drill down.
3. Click the  icon for **Protocol: EDI-X12 (ALL)** for both source and target.
4. Click .
5. Click the  icon for **Document Flow: All** for both source and target.

Set the B2B capabilities for Partner Two


To define the B2B capabilities for Partner Two, you would

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  icon next to **Partner Two**, and then click **B2B Capabilities**.

To set the source and target packaging for Partner Two to AS, you would:

1. Click the  icon underneath **Set Source for Package: AS** to enable it. Repeat this step for **Set Target**.
2. Click the  icon to drill down.
3. Click the  icon for **Protocol: EDI-X12 (ALL)** for both source and target.
4. Click .
5. Click the  icon for **Document Flow: All** for both source and target.

Next, you would 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. You would:

1. Click on the **Edit** icon ().
2. Enter an AS MDN E-mail address.

This is the address used to receive MDNs for AS1.
3. Enter an AS MDN HTTP URL:

```
http://IP_MACHINE_B:PORT/bcgreceiver/submit
```

Note that the URL defined for AS2 uses the same parameters that will be defined for the AS2 Target later in this chapter.

Create a Gateway for Partner One

Partner Two sends EDI to Partner One using AS2. Partner One's gateway is used to send the EDI received via AS2 to a JMS queue and ultimately to WebSphere Data Interchange for translation.

To create a new gateway for Partner One, you would:


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  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 12](#) below:

Table 12: JMS Gateway properties for Partner One

Gateway Name	JMStoPartnerOne
Transport	JMS
Target URI	file:///opt/mqm/java/JNDI/WdiJms
JMS Factory Name	WBIC42_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**.

To make JMStoPartnerOne the default gateway for Partner One, you would:

1. Click **View Default Gateways**.
2. From the **Production** list, select **JMS2toPartnerOne**.
3. Click **Save**.

Note: A JMS Gateway can be defined only for the Community Manager (Partner One, in the sample scenario).

Create a Gateway for Partner Two

Partner One sends EDI to WebSphere Business Integration Connect Enterprise Edition over a JMS queue. Partner Two's gateway is used to send the received EDI to Partner Two via AS2.

To create a new gateway for Partner Two, you would:


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  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 13](#):

Table 13: Properties for Partner Two Gateway

Gateway Name	AS2toPartnerTwo
Transport	HTTP/1.1
Target URI	http://IP_MACHINE_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 Business Integration Connect - Express.

An example of setting these properties in WebSphere Business Integration Connect - Express is shown in [“Example of configuring WebSphere Business Integration Connect - Express”](#) on page 102.

Notice that AS2toPartnerTwo is displayed as Online with a Status of **Enabled**.

To make AS2toPartnerTwo the default gateway for PartnerTwo, you would:

1. Click **View Default Gateways**.
2. From the **Production** list, select **AS2toPartnerTwo**.
3. Click **Save**.

Configure a Document Flow Definition and Interaction

To create a Document Flow Definition and Valid Interaction between Partner One and Partner Two, you would:

1. Click **Hub Admin** from the main menu and **Document Flow Definition** from the horizontal navigation bar.
2. Click **Manage Interactions** and then **Create a Valid Interaction**.
3. From the Source column, select:
 - a. Package: **None**
 - b. Protocol: **EDI-X12**
 - c. Document Flow: **All**
4. From the Target column, select:
 - a. Package: **AS**
 - b. Protocol: **EDI-X12**
 - c. Document Flow: **ALL**
5. Set the Action as **Pass Through**.
6. Click **Save**.
7. Click **Create a Valid Interaction** again.
8. From the Source column, select:
 - a. Package: **AS**
 - b. Protocol: **EDI-X12**
 - c. Document Flow: **ALL**
9. From the Target column select:
 - a. Package: **None**
 - b. Protocol: **EDI-X12**
 - c. Document Flow: **All**
10. Set the Action as **Pass Through**.
11. Click **Save**.

Create Participant Connections

To create a participant connection between Partner One and Partner Two, you would:

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's displayed below by clicking on the **Activate** button. This should display the following B2B Capabilities:

Table 14: Activate participant connection

	Source	Target
Package	None (N/A)	AS (N/A)
Protocol	EDI-X12 (ALL)	EDI-X12 (ALL)
Document Flow	ALL (ALL)	ALL (ALL)

To create a participant connection where Partner Two is the source and Partner One is the target, you would:

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 following details:

Table 15: Activate participant connection

	Source	Target
Package	AS (N/A)	None (N/A)
Protocol	EDI-X12 (ALL)	EDI-X12 (ALL)
Document Flow	ALL (ALL)	ALL (ALL)

Create and configure targets

To receive an EDI transaction from WebSphere Data Interchange, you would 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 from [Table 16](#) below:

Table 16: WdiJmsListener

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	WBIC42_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. You would:

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 17](#) below:

Table 17: Target properties for receipt of AS2

Target Name	WbicAS2Listener
Transport	HTTP/S
Gateway Type	Production
URI	/bcgreceiver/submit

Note: The URI for receipt of HTTP/S must always begin with /bcgreceiver

4. Click **Save**.

This completes the steps for configuring Business Integration Connect Enterprise Edition for the sample scenario. In the next section, you will see the steps for configuring the Community Participant (in this case, WebSphere Business Integration Connect - Express) for the sample scenario.

Example of configuring WebSphere Business Integration Connect - Express

In the sample scenario presented in this chapter, partnerTwo is using WebSphere Business Integration Connect - Express to send and receive EDI using HTTP AS2.

To successfully receive EDI via HTTP AS2, you would first create a profile for Partner Two in WebSphere Business Integration Connect - Express.

Configure My Profile

To create a profile for Partner Two, you would:

1. Click on **Configuration** from the main menu.
2. Click **My Profile** from the horizontal navigation bar.
3. Enter the details as outlined in [Table 18](#) below:

Table 18: My Profile

Receipt Address Unsecure Domain	IP_MACHINE_C
Receipt Address Unsecure Port	80
AS2 Sender ID	987654321
Business ID Type	DUNS
Business Identifier	987654321

Note: IP_MACHINE_C is the internet protocol (IP) address of the machine on which WebSphere Business Integration Connect - Express is running, and port 80 is the port assigned for use by WebSphere Business Integration Connect - Express during installation.

4. Click **Save**.

Create and configure a Participant for Partner One.

Partner One must be identified as a participant to WebSphere Business Integration Connect - Express. To create Partner One as a participant, you would:

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.

Next, you would configure partnerOne for AS2 and HTTP. This identifies the parameters required by WebSphere Business Integration Connect - Express for both sending and receiving HTTP and AS2 to partnerOne.

To configure partnerOne for HTTP and AS2, you would:

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_MACHINE_B:7080/bcgreceiver/submit`
Where IP_MACHINE_B is the IP address of Machine 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 Business Integration Connect Enterprise Edition on Machine B.
9. Set the Outbound Destination Address to:
`http://IP_MACHINE_B:7080/bcgreceiver/submit`
10. Click **Save**.

Note: After making these changes in WebSphere Business Integration Connect - 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 Business Integration Connect interacts with WebSphere Data Interchange. It also provided you with procedures to set up the sample scenario described in [“Sample scenario used in this chapter” on page 79](#).

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.

Chapter 4. Routing EDI documents

This section describes the process by which Business Integration Connect determines the routing information for 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 107](#))

You can find additional information on how file-based integration can be used when routing EDI documents in [“File System protocol for Enterprise and Advanced editions” on page 12](#).

Overview of EDI routing

An EDI document contains information, within the document, about the sender and the recipient of the document. Business Integration Connect uses this information when it routes the EDI document. The general flow is as follows:

1. Business Integration Connect determines the protocol used by examining the first three characters of the document. The following table shows the document-type protocol associated with each code.

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

- Business Integration Connect extracts, from the EDI document, the sender information, based on the element and position for that particular document type, as described in the following table:

EDI codes and the location of the sender and receiver information

Code	From Qualifier	From ID	To Qualifier	To ID
ISA	Element 105 at position 5	Element 107 at position 6	Element 105 at position 7	Element 106 at position 8
GS	N/A	Element 142 at position 2	N/A	Element 124 at position 3
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
ICS	Element X05 at position 4	Element X06 at position 5	Element X05 at position 6	Element X08 at position 7
STX	Element FROM1 at position 3	Element FROM2 at position 3	Element UNT1 at position 4	Element UNT2 at position 4
BG	N/A	Element BG03 at position 3	N/A	Element BG04 at position 4
UCS	N/A	Element 142 at position 3	N/A	Element 124 at position 4

- Business Integration Connect 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, Business Integration Connect uses only the ID.
- Business Integration Connect concatenates the qualifier and ID with a dash (-) character to look up the sender ID from the Business Integration Connect profile repository. For example, if, in the EDI message for the sender, the qualifier is AB and the identifier is 1234567, Business Integration Connect expects to find a community participant with an identifier of AB-1234567 in the profile repository. If Business Integration Connect cannot find this ID, the EDI document is not routed.
- To look up the receiving partner, Business Integration Connect determines the receiver qualifier and ID from the EDI message.
- Business Integration Connect concatenates the qualifier and ID with a dash (-) character to look up the receiver ID in the profile repository.
- Business Integration Connect routes the document to the intended recipient.

Special considerations for AS packaging

When the packaging of the document is specified as AS, Business Integration Connect performs some additional processing.

Routing the inbound document

When an EDI document is received from a community participant:

1. Business Integration Connect 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, Business Integration Connect does not route the document.

2. Business Integration Connect then performs the steps described in [“Overview of EDI routing” on page 105](#).

Routing the outbound document

When an EDI document is received from a backend application, Business Integration Connect 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, Business Integration Connect uses this information to generate the From and To IDs in the AS1 or AS2 header.
- If the attribute has not been specified, Business Integration Connect 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 105](#)) and then populates the header information.

Setting both IDs in the participant profile

Because Business Integration Connect 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 [Administrator Guide](#) for information.

Notices and Trademarks

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800
Burlingame, CA 94010
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Programming interface information

Programming interface information 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 of International Business Machines Corporation in the United States, other countries, or both:

IBM
the IBM logo
CrossWorlds
DB2
DB2 Universal Database
MQSeries
Tivoli
WebSphere

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel and Xeon are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Solaris, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.



