

*IBM WebSphere Business Integration Connect  
Enterprise Edition and Advanced Edition*



# Integration Overview

*Version 4.2.0*

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices and Trademarks” on page 61.

First Edition (September 2003)

This edition applies to Version 4, Release 2, Modification 0, of IBM® WebSphere® Business Integration Connect Advanced Edition (5724-E75) and Enterprise Edition (5724-E87), 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 anyway it believes appropriate without incurring any obligation to you.

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

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

# Contents

<b>Chapter 1. About this book</b>	<b>5</b>
<b>Chapter 2. Backend Integration</b>	<b>7</b>
Message transport protocols	8
Supported transport protocols	8
HTTP protocol	9
Process	9
Sending and receiving messages using the HTTP protocol	9
File System protocol for Enterprise and Advanced editions	10
Process	10
File System protocol for Business Integration Connect - Express	11
Process	11
Sending and receiving messages using the file system protocol	12
JMS protocol	12
Sending and receiving messages using the JMS protocol	13
Setting up integration through the JMS transport protocol using WebSphere MQ 5.3	14
Creating queues and channels	14
Creating the JMS bindings file	14
Creating the JMS target	15
Creating the JMS gateway	16
Message handling	16
Queued delivery	16
Communication error handling	17
Duplicate messages	17
Packaging	18
Backend Integration packaging	18
Transport level header content	18
RosettaNet to transport level header fields	20
AS2 to transport level header fields	22
AS1 to transport level header fields	24
Payload-	25
Attachments	25
Example of Backend Integration packaging over HTTP	26
None packaging	27
RosettaNet	27
Event notification	27
Event message structure	28
Event notification message example	30
<b>Chapter 3. Integrating with the WebSphere InterChange Server</b>	<b>-31</b>
Planning for integration	31
Which transport will you use?	31
Which packaging will you use?	32
Is the packaging available for the business protocol?	32
What types of business objects are required for the packaging?	32
Using the HTTP transport protocol	33
Overview of sending documents to the WebSphere InterChange Server	33
Setting up the environment for sending documents	35
Deploy the components	36
Create the properties file	36
Edit the deployment descriptor	38
Specify the location of the wrapper data handler	38
Edit the MO_Server_DataHandler	38
Define the business objects	39
Overview of sending documents to Business Integration Connect	42
Setting up the environment for sending documents	44
Deploy the HTTP Protocol Handler	44
Specify the location of the HTTP Protocol Handler	44
Configure the Adapter for XML	44
Define the business objects	45
Summary of supported platforms and versions	45
Using the JMS transport protocol	45
Overview of sending documents to the WebSphere InterChange Server	46

Overview of sending documents to the Business Integration Connect	48	Configure WebSphere Data Interchange	54
Setting up the environment for sending and receiving documents	49	Set up the JMS environment	54
Creating business objects	50	Configure Business Integration Connect	55
Dynamic meta-object	51		
<b>Chapter 4. Integrating with WebSphere Data Interchange</b>	<b>53</b>	<b>Chapter 5. Routing EDI documents</b>	<b>-57</b>
Introduction	53	Overview of EDI routing	57
Sending documents to WebSphere Data Interchange	53	Special considerations for AS packaging	59
Sending documents to Business Integration Connect	54	Routing the inbound document-	59
Configuring your environment for message exchange	54	Routing the outbound document	59
Configure WebSphere MQ communication	54	Setting both IDs in the participant profile	59
		<b>Notices and Trademarks</b>	<b>-61</b>
		Notices	61
		Programming interface information	63
		Trademarks and service marks	63

---

## Chapter 1. About this book

This document describes the Backend Integration interface, which is the mechanism that backend applications and IBM<sup>®</sup> WebSphere<sup>®</sup> 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.

Unless otherwise noted, the information in this document pertains only to the Business Integration Connect Enterprise or Advanced edition.

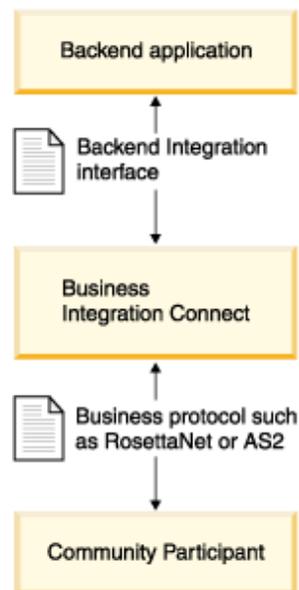


---

## Chapter 2. Backend Integration

With Business Integration Connect, you exchange business documents with your trading partners. The purpose of exchanging these documents is to communicate information, which typically involves processing data and returning a result. Processing of the 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. For example, a purchase order document from a community participant enters Business Integration Connect using a business protocol such as RosettaNet or AS2. The document needs to go to the backend application that processes purchase orders. Business Integration Connect routes the document to the backend application (Community Manager) using the message transport protocol defined for the connection to the application. The Backend Integration interface is the packaging used on the message containing the document. Note that the interaction can also go in the other direction; that is, from the application to Business Integration Connect to a community participant.



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

To implement the Backend Integration interface, a user uses the Community Console to set the packaging on the connection between the Community Participant and the Community Manager (as described in the [User's Guide for the Community Console](#)). The packaging can be None packaging or it can be Backend Integration packaging. 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](#).

---

## 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*

<b>Business protocol and packaging combination</b>	<b>HTTP or HTTPS</b>	<b>JMS</b>	<b>File System</b>
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
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*

<b>Message content and transport packaging combination</b>	<b>HTTP or HTTPS</b>	<b>JMS</b>	<b>File System</b>
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
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 contains 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](#). 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.

With the exception of binary messages, Business Integration Connect supports sending and receiving HTTP 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.

### Process

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

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.

### 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.

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.

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's Guide](#) for information on setting up targets. For information on gateways, see "Managing gateway configurations" in the [Administrator's Guide](#).

## File System protocol for Enterprise and Advanced editions

The File System protocol enables Business Integration Connect to send messages to third party EDI translator applications although its use is not limited to these applications. The File System protocol is based on placing and reading files within a defined 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's 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's 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](#).

## File System protocol for Business Integration Connect - Express

The File System protocol enables Business Integration Connect to send messages to third party EDI translator applications although its use is not limited to these applications. This is the only transport protocol that Business Integration Connect - Express supports.

### Process

For Business Integration Connect - Express, when you create a new community participant, the system creates a corresponding directory structure for that participant. The directory structure contains the participant's name and supported transport protocols, with subdirectories for each state in the processing of a document (for example, sent and received).

```
data
  FileSystemAdapter2
    partners
      partner_name
error
  as2
    binary
    EDI-Consent
    EDIFACT
    EDI-X12
    MDN
    XML
  http
rec_err
  as2
    binary
    EDI-Consent
    EDIFACT
    EDI-X12
    MDN
    XML
  http
received
  as2
    binary
    EDI-Consent
    EDIFACT
    EDI-X12
    MDN
    XML
  http
    send
  as2
    binary
    EDI-Consent
    EDIFACT
    EDI-X12
    MDN
    XML
```

```
http
sent
  as2
    binary
    EDI-Consent
    EDIFACT
    EDI-X12
    MDN
    XML
  http
```

When an application sends a document to Business Integration Connect - Express, it creates a file in the send subdirectory of the intended recipient (community participant). For example, if an application is routing a document with a format of EDIFACT to partner XYZ, it sends the document to the following directory:

```
data/FileSystemAdapter2/partners/XYZ/send/as2/EDIFACT
```

Business Integration Connect - Express polls the send directories. When it detects a document in the send folder, it determines, from the directory structure, the recipient, transport, and protocol, and sends the document over the Internet to the community participant.

Once it has sent the document, Business Integration Connect - Express moves the document to the sent folder.

## 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's 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](#). 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.
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's Guide](#) for information on setting up targets. For information on gateways, see "Managing gateway configurations" in the [Administrator's 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](#) 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's 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](#)). 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.

### Backend Integration packaging

When a participant connection has packaging set to Backend Integration, messages sent through the gateway must have a [transport level header](#), which contains meta information about the message, a [payload](#) which contains the content of the message, and may 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.	

*Transport header values*

x-aux-production	<p>Routing of the message. Valid values are:</p> <p>Production</p> <p>Test</p> <p>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.</p>	
x-aux-system-msg-id	<p>Global Unique Identifier (GUID) for the message, which is used for duplicate checking.</p>	Yes
x-aux-payload-root-tag	<p>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.</p>	
x-aux-process-instance-id	<p>Identifier that links documents in a multiple message business process to a unique process instance.</p> <p>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.</p>	
x-aux-event-status-code	<p>Status code for the event notification. See the StatusCode field in <a href="#">Event message structure</a>.</p>	
x-aux-third-party-bus-id	<p>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.</p>	
x-aux-transport-retry-count	<p>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.</p>	

**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	<p>For RosettaNet 2.0:</p> <p>&lt;(DeliveryHeader)&gt;</p> <p>&lt;messageSenderIdIdentification&gt;</p> <p>&lt;PartnerIdentification&gt;</p> <p>&lt;GlobalBusinessIdentifier&gt;</p> <p>For RosettaNet 1.1:</p> <p>&lt;ServiceHeader&gt;</p> <p>&lt;ProcessControl&gt;</p> <p>&lt;TransactionControl&gt;</p> <p>&lt;ActionControl&gt; or &lt;SignalControl&gt;</p> <p>&lt;PartnerRouter&gt;</p> <p>&lt;fromPartner&gt;</p> <p>&lt;PartnerDescription&gt;</p> <p>&lt;BusinessDescription&gt;</p> <p>&lt;GlobalBusinessIdentifier&gt;</p>
x-aux-receiver-id	<p>For RosettaNet 2.0:</p> <p>&lt;(DeliveryHeader)&gt;</p> <p>&lt;messageReceiverIdentification&gt;</p> <p>&lt;PartnerIdentification&gt;</p> <p>&lt;GlobalBusinessIdentifier&gt;</p> <p>For RosettaNet 1.1:</p> <p>&lt;ServiceHeader&gt;</p> <p>&lt;ProcessControl&gt;</p> <p>&lt;TransactionControl&gt;</p> <p>&lt;ActionControl&gt; or &lt;SignalControl&gt;</p> <p>&lt;PartnerRouter&gt;</p> <p>&lt;toPartner&gt;</p> <p>&lt;PartnerDescription&gt;</p> <p>&lt;BusinessDescription&gt;</p> <p>&lt;GlobalBusinessIdentifier&gt;</p>
x-aux-protocol	<p>Set value for RosettaNet:</p> <p>RNSC</p>
x-aux-protocol-version	<p>Set value:</p> <p>1.0</p>

*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	<p>Name of the PIP such as Pip3A4PurchaseOrderRequest</p>
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>&lt;ServiceHeader&gt;   &lt;ProcessControl&gt;     &lt;pipInstanceid&gt;       &lt;InstanceIdentifier&gt;</pre> <p>For RosettaNet 1.1:</p> <pre>&lt;ServiceHeader&gt;   &lt;ProcessControl&gt;     &lt;ProcessIdentity&gt;       &lt;InstanceIdentifier&gt;</pre>
x-aux-msg-id	<pre>&lt;(RNPipServiceContent)&gt;   &lt;thisDocumentIdentifier&gt;     &lt;ProprietaryDocumentIdentifier&gt;</pre>
x-aux-production	<p>For RosettaNet 2.0:</p> <pre>&lt;ServiceHeader&gt;   &lt;ProcessIndicator&gt;     &lt;GlobalUsageCode&gt;</pre> <p>For RosettaNet 1.1:</p> <pre>&lt;Preamble&gt;   &lt;GlobalUsageCode&gt;</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 &lt;FromID&gt; in the "Subject: &lt;ToID&gt;;&lt;FromID&gt;" 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 &lt;FromID&gt; in the "Subject: &lt;ToID&gt;;&lt;FromID&gt;" 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, &lt;ToID&gt; in the "Subject: &lt;ToID&gt;;&lt;FromID&gt;" 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 &lt;ToID&gt; in the "Subject: &lt;ToID&gt;;&lt;FromID&gt;" 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	This field is not used for AS1.
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	This field is not used for AS1.

## 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](#).

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>
```

## 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</ProprietaryDocumentIdentifier>
</thisDocumentIdentifier>
```

```
<GlobalDocumentFunctionCode>Request</GlobalDocumentFunctionCode>
</Pip3A4PurchaseOrderRequest>
```

## None packaging

If the participant connection has packaging set to None, Business Integration Connect does not add a transport-level header when it sends a message and does not expect one when it receives a message.

---

## 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.

## 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 when the following occurs:
- A RosettaNet document has been delivered to a community participant
- A community participant sends an 0A1 message to cancel a PIP
- A community participant sends an exception or receipt acknowledgment exception

Business Integration Connect can send 0A1 message to the destination application as it would do with any other PIP depending on whether it has been configured to send these messages using Exclusion List Management. See "Managing Exclusion Lists" in the [Administrator's Guide](#).

An application can send a 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/xmleventnotification"

xmlns:evntf="http://www.ibm.com/websphere/bcg/2003/v1.0/xmleventnotification"

elementFormDefault="qualified">

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

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

<!-- 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: <b>100</b> - Business Integration Connect has delivered the document and received a receipt acknowledgment. <b>800</b> - The application cancelled the PIP. <b>900</b> - 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: evtntf: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"?>
<evtntf:EventNotification
xmlns:evtntf="http://www.ibm.com/websphere/bcg/2003/v1.0/xmleventnotification">
  <evtntf:StatusCode>100</evtntf:StatusCode>
  <evtntf:StatusMessage>The message was delivered</evtntf:StatusMessage>
  <evtntf:EventMessageID>12345</evtntf:EventMessageID>
  <evtntf:BusinessObjectID>34234</evtntf:BusinessObjectID>
  <evtntf:GlobalMessageID>98732</evtntf:GlobalMessageID>
  <evtntf:Timestamp>2001-01-31T13:20:00Z</evtntf:Timestamp>
</evtntf:EventNotification>
```

---

## Chapter 3. Integrating with the WebSphere InterChange Server

[Backend Integration](#) 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
- 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](#).
- Sending 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 interactions between Business Integration Connect and backend application 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 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's 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](#) 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. Refer to the WebSphere InterChange Server Data Handler documentation for information on creating a data handler.

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 [Define the business objects](#) 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.

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.

### Overview of sending documents to the WebSphere InterChange Server

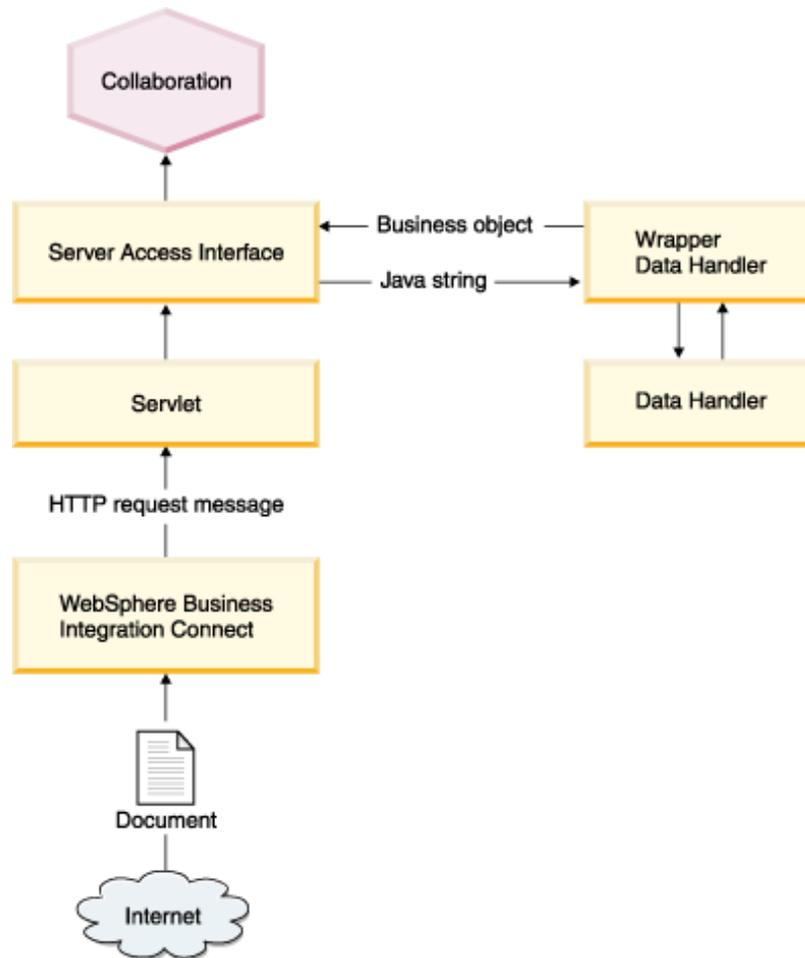
To send a document from Business Integration Connect to the WebSphere InterChange Server using the HTTP transport protocol, you interact with the following components, which are provided with Business Integration Connect:

- The WebSphere Business Integration Connect Servlet
- A wrapper data handler

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 and 4.2.

The following illustration provides an overview of how the components interact. Note that the wrapper 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 [Create the properties file](#) 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.
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. Because all 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. 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 Business Integration Connect](#).

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.

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. Also configure the meta-objects for the wrapper data handler and the data handler for the request message.
7. Define business objects representing the information being sent by the Business Integration Connect request.

These tasks are described in the sections that follow.

## 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	integration/wbi/wics/http/lib/bcgwbiservlet.war
Wrapper Data Handler	integration/wbi/wics/http/lib/bcgwbiwrapperdh.jar
Repository file for Wrapper Data Handler	integration/wbi/wics/http/lib/MO_DataHandler_WBIWrapper.in

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 the IOR file available to the host on which the servlet is deployed.
4. Update the ICS\_IORFILE property in servlet.properties with the location of the IOR file.

## Create the properties file

As mentioned in [Overview of sending documents to the WebSphere InterChange Server](#), 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.

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	The version number of the WebSphere InterChange Server. Possible values are 4.1.1 and 4.2.

*Contents of the properties file*

ICS_IORFILE	c:/myiorlocation/ Server1InterChangeServer.i or	The file name used to access the WebSphere InterChange Server APIs. The example shows how you would specify the path on a Windows <sup>(R)</sup> 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.
TRACING_ENABLED	True	Whether to enabling tracing in the servlet. Possible values are True and False.
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.
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.

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 [Edit the MO\\_Server\\_DataHandler](#) for information.

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
TRACING_ENABLED=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
```

## 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.

## 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 of the wrapper data handler:

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

## 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.

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

```
ClassName=com.ibm.bcg.integration.wbi.datahandlers.WBICWrapperDataHandler
    wbic_request_mime
    wbic_response_mime
```

- In the meta-object, specify the attribute `wbic_request_mime`, which should be set to the MIME type of the data handler that the wrapper data handler will invoke to process the payload of a 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.

**Note:** The `wbic_response_mime` does not have to be set, because Business Integration Connect is not expecting a response.

- 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.

Refer to the Data Handler Guide for more information.

## 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

### Top-level business object

The top-level business object has the following structure:

URL
MimeType
BOPrefix
Response
Request

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.

**Request business object**

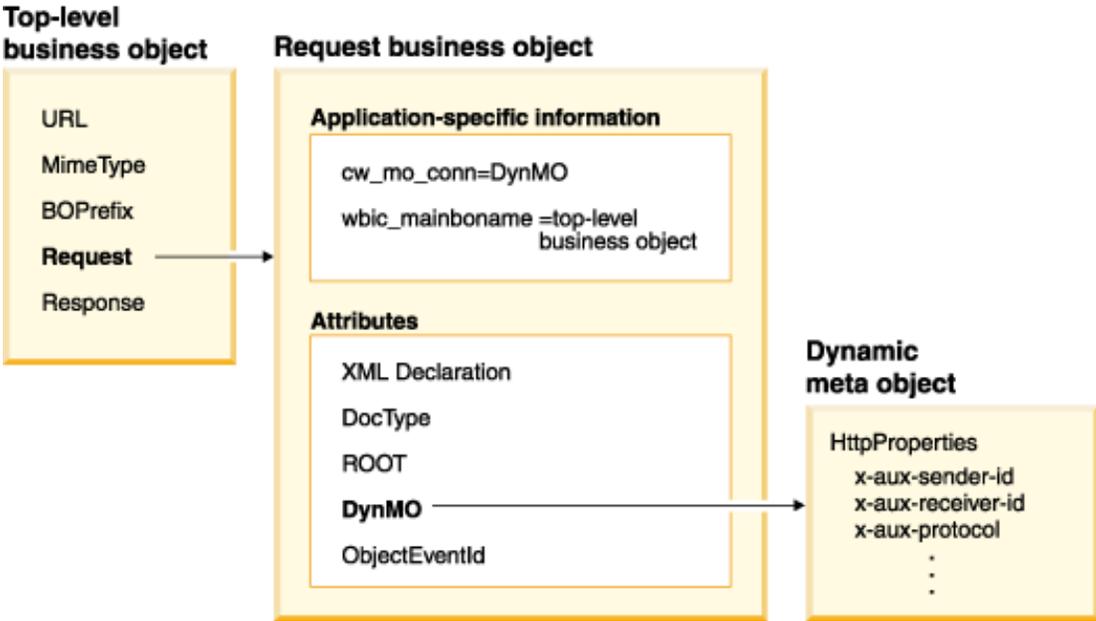
The following is an example of a Request business object containing the DynMO child object:

XMLDeclaration
DocType
ROOT
DynMO

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](#).

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](#).

*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**

As mentioned earlier, Business Integration Connect is not expecting a response, so it is not necessary to create a response business object.

## Overview of sending documents to Business Integration Connect

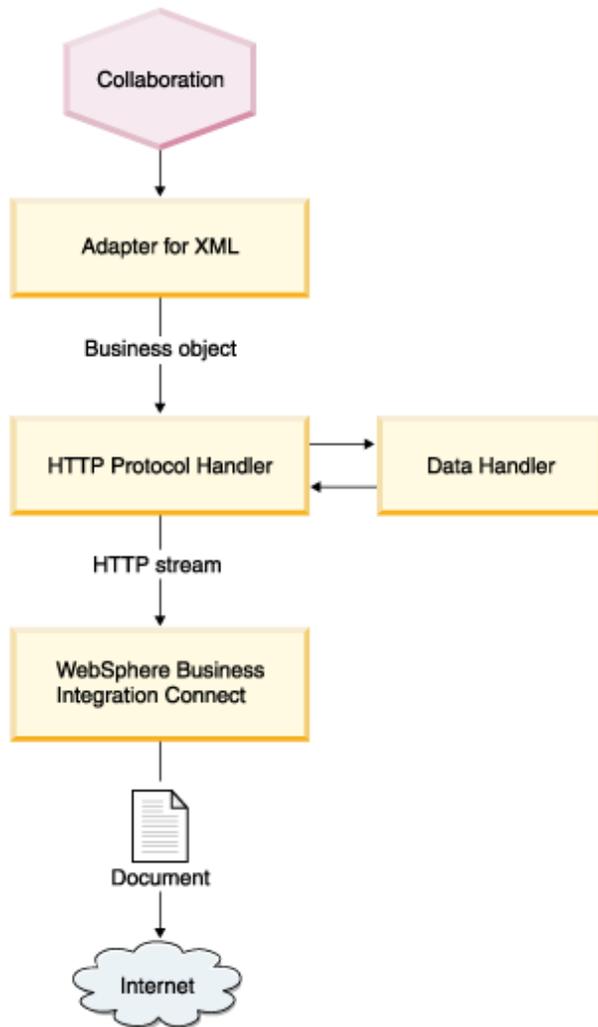
To send a document from the WebSphere InterChange Server using the HTTP transport protocol, you use either the HTTP or HTTPS protocol handler, which is supplied by Business Integration Connect.

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.
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](#).
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.

## 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.

### 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.

### 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`.

### 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
```

## Define the business objects

The business objects required are the same as those described in the earlier [Define the business objects](#) section. 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	This servlet can connect to WebSphere InterChange Server versions 4.1.1 and 4.2.  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.
HTTP Protocol Handler for Adapter for XML	The HTTP Protocol Handler can be plugged into the Adapter for XML version 3.1.x or higher.  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.

---

## 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.

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.

The section [Setting up integration through the JMS transport protocol using WebSphere MQ 5.3](#) describes the steps you follow to set up your JMS environment, including setting up queues and channels. The following is an illustration of how the Adapter for JMS, having received a message from a backend application, 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.

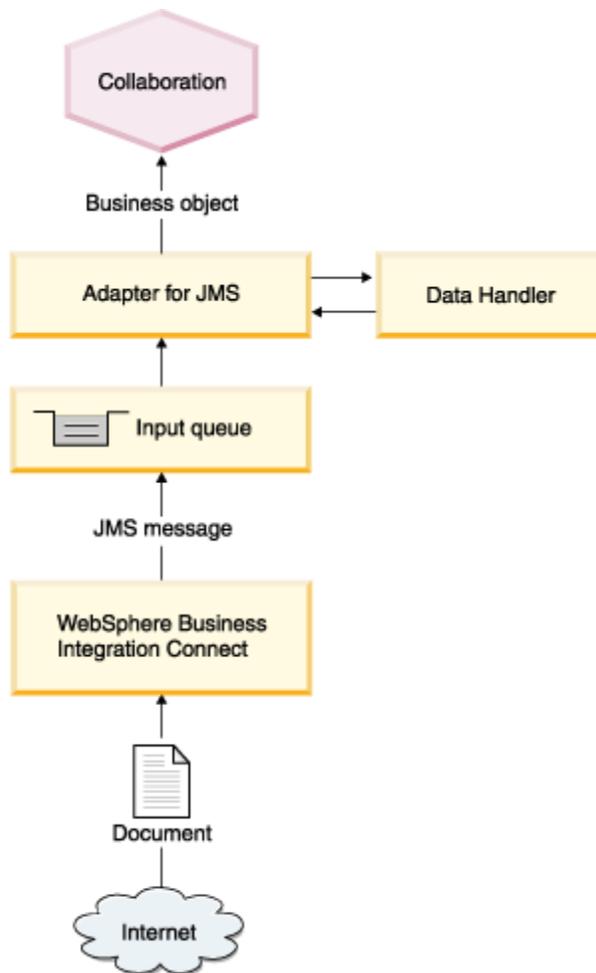
You must configure the target to read from the same queue in which the Adapter for JMS writes the message. The target listens for any incoming messages in this queue and retrieves them.

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's 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.
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.
 

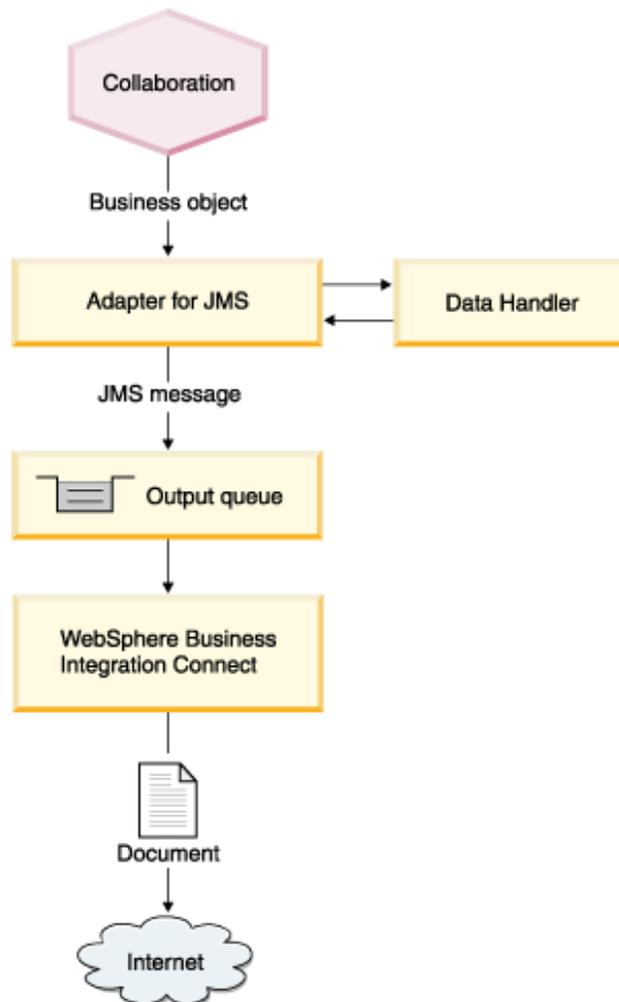
**Note:** The Adapter for JMS provides various techniques for configuring data handlers. For example, you can configure data handlers per queues. Refer to the Adapter for JMS documentation for information on configuring a data handler.
5. The data handler returns a business object.

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 converts the business object sent by the collaboration into a JMS message, using a data handler. The data handler can be specified in the static meta-object or in a dynamic meta-object. The dynamic meta-object is a child of the business object. The dynamic meta-object attribute is specified with the business-level `cw_mo_conn` application-specific information. Refer to the Adapter for JMS documentation for more details.
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. No Business Integration-supplied components are involved (as in the case of the HTTP transport protocol).

The section [Setting up integration through the JMS transport protocol using WebSphere MQ 5.3](#) 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](#).

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](#) provides an overview of the steps for creating the gateway. A complete description can be found in the [Administrator's 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:

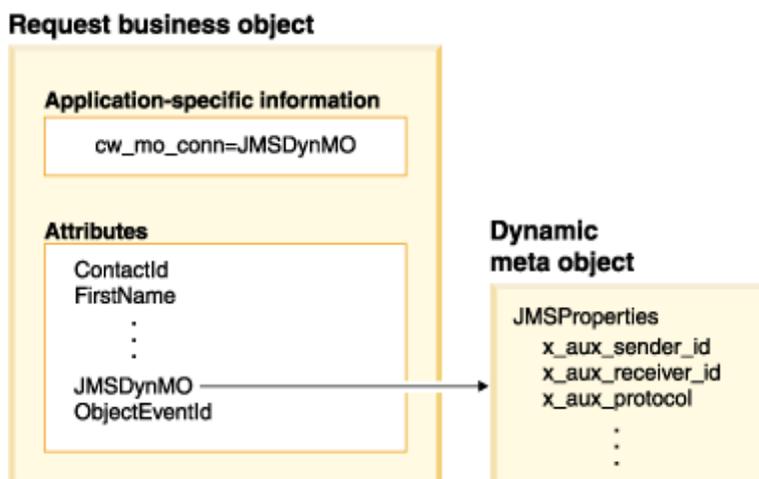
- You must configure a target in Business Integration Connect for JMS. [Creating the JMS target](#) provides an overview of the steps for creating the target. A complete description can be found in the [Administrator's Guide](#).
- You must make sure that the static or dynamic meta-objects are configured 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.

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. The dynamic child meta-object contains an attribute representing `JMSProperties`. The dynamic meta-object is described in more detail in [Dynamic meta-object](#).

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*

## Dynamic meta-object

As shown in the previous illustration, the dynamic meta-object contains the JMSProperties business object. If you are using Backend Integration packaging, the JMSProperties business object contains JMS properties required by the packaging.

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](#).

*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

*Application-specific information for the JMSProperties attributes*

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

---

## Chapter 4. Integrating with WebSphere Data Interchange

[Backend Integration](#) 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.

It is assumed that you are familiar with using WebSphere Data Interchange. See the WebSphere Data Interchange documentation for additional information.

---

### 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. 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 based. See [Overview of EDI routing](#) for details.
2. WebSphere Data Interchange reads the message from the queue. It performs the tasks of deenveloping, validating, and translating the EDI document.
3. 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.

## 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](#).

---

## 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

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 details on defining queue managers.

### 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 the following profiles in the WebSphere Data Interchange Client:

- MQSeries queue profile
- Network profile
- Mailbox profile

Refer to the WebSphere Data Interchange documentation for details on configuring these profiles.

### Set up the JMS environment

You set up the JMS environment and then create a JMS target and a JMS gateway, based on the information you specify in the bindings file. See [Creating the JMS bindings file](#).

## Configure Business Integration Connect

A target in Business Integration Connect provides location information to the Document Manager component so that it can retrieve documents from the appropriate system location based on the transport type of the incoming document (including JMS queues). [Creating the JMS target](#) provides an overview of the steps for creating the target. A complete description can be found in the [Administrator's Guide](#).

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 Business Integration Connect how to deliver documents to WebSphere Data Interchange. [Creating the JMS gateway](#) provides an overview of the steps for creating the gateway. A complete description can be found in the [Administrator's Guide](#).



---

## Chapter 5. Routing EDI documents

This section describes the process by which Business Connection 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](#))

You can find additional information in the following sections:

- How file-based integration can be used when routing EDI documents (see [File System protocol for Enterprise and Advanced editions](#))
- How file-based integration can be used when routing EDI documents in Business Integration Connect - Express (see [File System protocol for Business Integration Connect Express](#))

---

### 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

2. 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*

<b>Code</b>	<b>From Qualifier</b>	<b>From ID</b>	<b>To Qualifier</b>	<b>To ID</b>
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

3. 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.
4. 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.
5. To look up the receiving partner, Business Integration Connect determines the receiver qualifier and ID from the EDI message.
6. Business Integration Connect concatenates the qualifier and ID with a dash (-) character to look up the receiver ID in the profile repository.
7. 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](#).

### 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](#)) 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's 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  
Passport Advantage  
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.

