IBM WebSphere Business Integration Adapters

Adapter for WebSphere Commerce User Guide

Adapter Version 2.6.x

IBM WebSphere Business Integration Adapters

Adapter for WebSphere Commerce User Guide

Adapter Version 2.6.x

Note!

Before using this information and the product it supports, read the information in “Notices” on page 95.

25June2004

This edition of this document applies to IBM WebSphere Business Integration Adapter forWebSphere Commerce (Product ID 5724-H38), version 2.6.x, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

Preface

The IBM^R WebSphere^R Business Integration Adapter portfolio supplies integration connectivity for leading e-business technologies, enterprise applications, and legacy and mainframe systems. The product set includes tools and templates for customizing, creating, and managing components for business process integration.

This document describes the installation, configuration, troubleshooting, and business object development for the IBM WebSphere Business Integration Adapter for WebSphere Commerce.

Audience

This document is for consultants, developers, and system administrators who use the connector at customer sites.

Prerequisites for this document

Users of this document should be familiar with the IBM WebSphere InterChange Server, with business object and collaboration development, with the WebSphere Commerce application, and with WebSphere MQ. For links, see "Related documents"

Related documents

The complete set of documentation available with this product describes the features and components common to all WebSphere Business Integration Adapters installations, and includes reference material on specific components.

You can install related documentation from the following sites:

- For general adapter information; for using adapters with WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker); and for using adapters with WebSphere Application Server, see the IBM WebSphere Business Integration Adapters InfoCenter:
  http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- For using adapters with WebSphere InterChange Server, see the IBM WebSphere InterChange Server InfoCenters:
  http://www.ibm.com/websphere/integration/wicserver/infocenter
  http://www.ibm.com/websphere/integration/wbicollaborations/infocenter
- For more information about WebSphere message brokers:
  http://www.ibm.com/software/integration/mqfamily/library/manualsa/
- For more information about WebSphere Application Server:
  http://www.ibm.com/software/webservers/appserv/library.html

These sites contain simple directions for downloading, installing, and viewing the documentation.

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

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic,	Indicates a variable name or a cross-reference.
blue text	Blue text, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click any blue text to jump to the object of the reference.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
`[ ]`	In a syntax line, square brackets surround an optional parameter.
`...`	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, `option[,...]` means that you can enter multiple, comma-separated options.
`< >`	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><connector_name>tmp.log`.
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product path names are relative to the directory where the product is installed on your system.
`----UNIX/Windows-----`	Paragraphs beginning with either of these indicate notes listing operating system differences.
`%text%` and `$text`	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is `$text`, indicating the value of the `text` UNIX environment variable.
ProductDir	Represents the directory where the IBM WebSphere Business Integration Adapters product is installed.

Contents

New in This Release

New in release 2.6.x

Two connector-specific properties have been added: EnableMessageProducerCache and SessionPoolSizeForRequests. For further information, see “Connector-specific configuration properties” on page 28.

When processing a response message to a synchronous request, the connector interprets the feedback code MQFB_NONE (the default feedback code if none is set) as VALCHANGE. For further information, see “Synchronous delivery” on page 8.

As of version 2.6.x, the adapter for CORBA is not supported on Solaris 7, so references to that platform version have been deleted from this guide.

New in Release 2.5.x

Adapter installation information has been moved from this guide. See Chapter 2 for the new location of this information.

Beginning with version 2.5, the adapter for WebSPhere Commerce is no longer supported on Microsoft Windows NT.

New in Release 2.3.x

Updated in March, 2003. The ″CrossWorlds″ name is no longer used to describe an entire system or to modify the names of components or tools, which are otherwise mostly the same as before. For example ″CrossWorlds System Manager″ is now ″System Manager,″ and ″CrossWorlds InterChange Server″ is now ″WebSphere InterChange Server.″

You can now associate a data handler with an input queue. For further information, see “Overview of mapping data handlers to input queues” on page 43.

The guaranteed event delivery feature has been enhanced. For further information, see “Enabling guaranteed-event delivery” on page 33.

Chapter 1. Overview

This chapter describes the IBM WebSphere Business Integration Adapter for WebSphere Commerce.

The adapter enables the IBM WebSphere Integration Server adapter to exchange messages with WebSphere Commerce, Business Edition, version 5.4 with Fixpack 2, or or 5.5. The topics included in this chapter include:

WebSphere Commerce software is a flexible platform for fulfilling a variety of commerce integration roles. The adapter for WebSphere Commerce can be used in a solution that uses the integration broker to integrate business data exchanges between WebSphere Commerce and other enterprise information system applications for which appropriate adapters have been installed.

Connectors consist of an application-specific component and the connector framework. The application-specific component contains code tailored to a particular application. The adapter framework, whose code is common to all adapters, acts as an intermediary between the integration broker and the application-specific component. The adapter framework provides the following services between the integration broker and the application-specific component:

- Receives and sends business objects
- Manages the exchange of startup and administrative messages

This document contains information about the adapter framework and the connector. It refers to both of these components as the adapter. For more information about the relationship of the integration broker to the adapter, see the *IBM WebSphere InterChange Server System Administration Guide*.

Note: All WebSphere business integration adapters operate with an integration broker. The adapter for WebSphere Commerce operates with the InterChange Server (ICS) integration broker, which is described in the *Technical Introduction to IBM WebSphere InterChange Server*.

Adapter architecture

The adapter is meta-data-driven. The adapter uses an MQ implementation of the JavaTM Message Service (JMS), an API for accessing enterprise-messaging systems.

The adapter uses WebSphere MQ queues to enable asynchronous data exchange from WebSphere Commerce to ICS and from ICS to WebSphere Commerce. Data is sent between the queues for WebSphere Commerce and the ICS in the form of XML messages. An XML data handler is used to convert the data into business objects that can be processed by ICS collaboration objects.

For information about using the adapter in synchronous exchanges, see “Synchronous request and reply interactions” on page 3.

Asynchronous messages from WebSphere Commerce to integration broker

When an order is placed in WebSphere Commerce, an `OrderCreate` message is generated in XML format and is placed in the WebSphere MQ output queue as shown in the following figure. (In this illustration, it is assumed that WebSphere Commerce and the integration broker are installed on different machines using different queue managers, making it necessary to have a remote queue definition for output from WebSphere Commerce, connecting to an input queue that is local to the integration broker. When WebSphere Commerce and the integration broker are installed on the same machine, a single queue can function as both the output queue from WebSphere Commerce and input queue for the integration broker.)

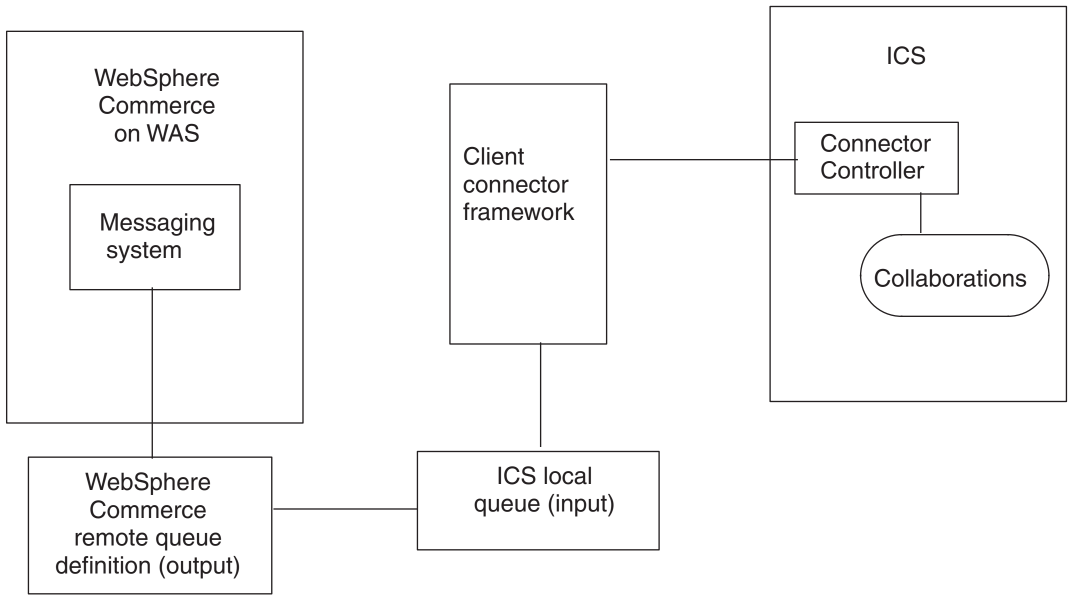

WebSphere Commerce on WAS
Messaging system
Client connector framework
ICS
Connector Controller
Collaborations
WebSphere Commerce remote queue definition (output)
ICS local queue (input)

Figure 1. Adapter architecture

To detect data events in WebSphere Commerce, the adapter polls the WebSphere Commerce output queues for new XML messages. When it discovers a new message, the adapter passes it to an input queue, calls a data handler to convert the message to a business object that is specific to the structure of data originating from WebSphere Commerce, and then passes the business object to the connector in ICS. The connector invokes maps to generate a generic business object from the WebSphere Commerce-specific business object, and then delivers the generic business object to one or more collaboration objects. After the collaboration objects have processed the business object, the generic business object is mapped to an application-specific business object, which is delivered to an adapter (such as a WebSphere Business Integration adapter for SAP) that has been configured for a back-end application.

Asynchronous messages to WebSphere Commerce from ICS

In the opposite direction, the adapter for WebSphere Commerce receives business objects from collaborations, converts them into XML-format messages using the data handler, and then delivers the messages to the WebSphere Commerce WebSphere MQ queue.

Synchronous request and reply interactions

Synchronous request and reply interactions require additions to or customization of the WebSphere Commerce application, as described in the following topics.

Requests from WebSphere Commerce to InterChange server

With the addition of the WebSphere Commerce Enhancement Pack, available at http://www-3.ibm.com/software/webservers/commerce/epacks/v54/, you can use the adapter to set up a synchronous message flow for request and reply interactions from the WebSphere Commerce messaging system to InterChange Server or other external systems. For information about this approach, see the integration documentation for WebSphere Commerce 5.4 and the IBM WebSphere business integration system.

Requests from InterChange server to WebSphere Commerce

Note: This approach requires a customization of the commands that are executed in WebSphere Commerce when a business object comes from ICS. The commands should retrieve the `ReplyTo` queue from the message, and place a reply on the queue within the ResponseTimeout interval. For information about creating and customizing commands in WebSphere Commerce, refer to the *Programmer's Guide for WebSphere Commerce, V. 5.4*.

Without using WebSphere Commerce Enhancement Pack, you can set up a simulated synchronous exchange between WebSphere Commerce and ICS, using `ReplyTo` queues.

See "Synchronous delivery" on page 8, later in this guide, for more information related to this approach.

Event notification

Notification of data events that have occurred in the WebSphere Commerce application is accomplished through the polling mechanism of the adapter. The adapter can poll multiple input queues, polling each in a round-robin manner and retrieving a specified number of messages from each queue. For each message retrieved during polling, the adapter adds a dynamic child meta-object (if specified in the business object). The child meta-object values can direct the adapter to populate attributes with the format of the message as well as with the name of the input queue from which the message was retrieved.

When a message is retrieved from an input queue, the connector looks up the business object name associated with that input queue and with the `FORMAT` field contained in the message header. The message body, along with a new instance of the appropriate business object, is then passed to the data handler. If a business object name that is associated with the input queue and format is not found, the message body alone is passed to the data handler. If a business object is successfully populated with message content, the connector checks to see if it is subscribed, and then delivers it to InterChange Server using the `gotApplEvents()` method.

Business objects and WebSphere MQ message header

The type of business object and verb used in processing a message is based on the `FORMAT` field contained in the WebSphere MQ message header. The adapter uses meta-object entries to determine business object name and verb. You construct a meta-object to store the business object name and verb to associate with the WebSphere MQ message header `FORMAT` field text.

You can optionally construct a dynamic meta-object that is added as a child to the business object passed to the adapter. The child meta-object values override those specified in the static meta-object that is specified for the adapter as a whole. If the child meta-object is not defined or does not define a required conversion property, the adapter, by default, examines the static meta-object for the value. You can specify one or more dynamic child meta-objects instead of, or to supplement, a single static adapter meta-object.

Application-adapter communication

The adapter makes use of IBM's WebSphere MQ implementation of the Java Message Service (JMS). JMS is an open-standard API for accessing enterprise messaging systems. It is designed to allow business applications to asynchronously send and receive business data and events.

Message request

Figure 2 illustrates a message request communication. When the `doVerbFor()` method receives a business object from a collaboration, the adapter passes the business object to the data handler. The data handler converts the business object into XML text and the adapter issues it as a message to a queue. There, the JMS layer makes the appropriate calls to open a queue session and route the message.

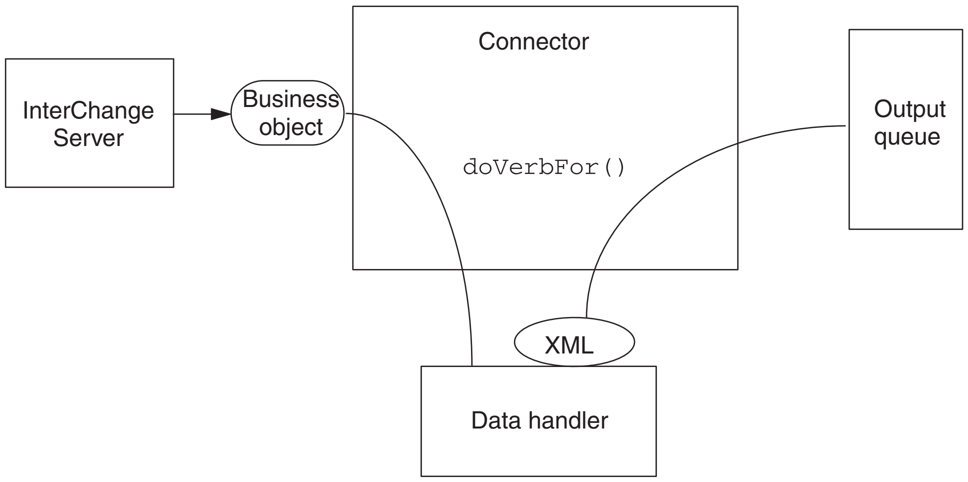


Figure 2. Application-adapter communication method: Message request

Message return

Figure 3 illustrates the message return direction. The `pollForEvents()` method retrieves the next applicable message from the input queue. The message is staged in the in-progress queue where it remains until processing is complete. Using either the static or dynamic meta-objects, the adapter first determines whether the message type is supported. If so, the adapter passes the message to the configured data handler, which converts the message into a business object. The verb that is set reflects the conversion properties established for the message type. The adapter then determines whether the business object is subscribed to by a collaboration. If so, the `gotApplEvents()` method delivers the business object to InterChange Server, and the message is removed from the in-progress queue.

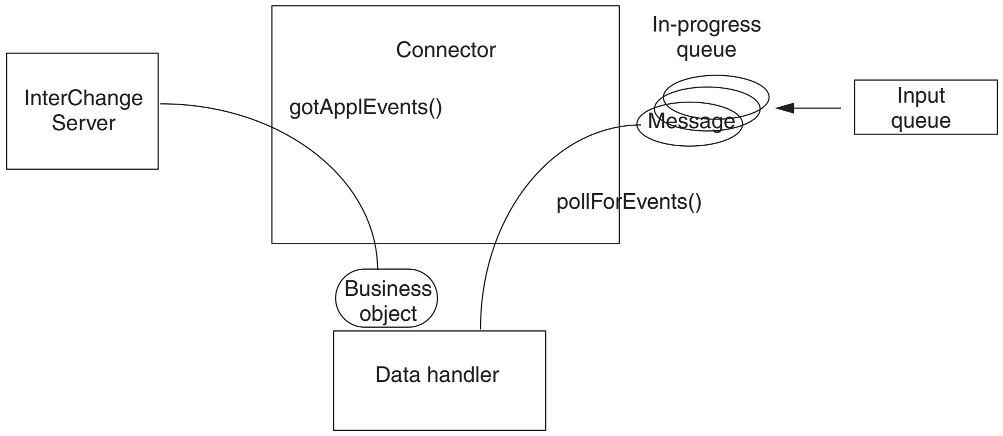


Figure 3. Application-adapter communication method: Message return

Event handling

For event notification, the adapter detects events written to a queue by WebSphere Commerce.

Retrieval

The adapter uses the `pollForEvents()` method to poll the queue at regular intervals for messages. When the adapter finds a message, it retrieves it from the queue and examines it to determine its format. If the format has been defined in the adapter's static meta-object, the adapter passes both the message body and a new instance of the business object associated with the format to the configured data handler; the data handler is expected to populate the business object and specify a verb. If the format is not defined in the static meta-object, the adapter passes only the message body to the data handler; the data handler is expected to determine, create and populate the correct business object for the message. See "Error handling" on page 56 for event failure scenarios.

The adapter processes messages by first opening a transactional session to the input queue. This transactional approach allows for a small chance that a business object could be delivered to a collaboration twice due to the adapter successfully submitting the business object but failing to commit the transaction in the queue. To avoid this problem, the adapter moves all messages to an in-progress queue. There, the message is held until processing is complete. If the adapter shuts down unexpectedly during processing, the message remains in the in-progress queue instead of being reinstated to the original input queue.

Note: Transactional sessions with WebSphere MQ require that every requested action on a queue be performed and committed before events are removed from the queue. Accordingly, when the adapter retrieves a message from the queue, it does not commit to the retrieval until three things occur: 1) The message has been converted to a business object; 2) the business object is delivered to InterChange Server by the `gotApplEvents`() method, and 3) a return value is received.

Recovery

Upon initialization, the adapter checks the in-progress queue for messages that have not been completely processed, presumably due to a connector shutdown. The connector configuration property `InDoubtEvents` allows you to specify one of four options for handling recovery of such messages: fail on startup, reprocess, ignore, or log error.

Fail on startup

With the fail on startup option, if the adapter finds messages in the in-progress queue during initialization, it logs an error and immediately shuts down. It is the responsibility of the user or system administrator to examine the message and take appropriate action, either to delete these messages entirely or move them to a different queue.

Reprocess

With the reprocessing option, if the adapter finds any messages in the in-progress queue during initialization, it processes these messages first during subsequent polls. When all messages in the in-progress queue have been processed, the adapter begins processing messages from the input queue.

Ignore

With the ignore option, if the adapter finds any messages in the in-progress queue during initialization, the adapter ignores them, but does not shut down.

Log error

With the log error option, if the adapter finds any messages in the in-progress queue during initialization, it logs an error but does not shut down.

Archiving

If the connector property `ArchiveQueue` is specified and identifies a valid queue, the adapter places copies of all successfully processed messages in the archive queue. If `ArchiveQueue` is undefined, messages are discarded after processing. For more information on archiving unsubscribed or erroneous messages, see “Error handling” on page 56.

Note: By JMS conventions, a retrieved message cannot be issued immediately to another queue. To enable archiving and re-delivery of messages, the adapter first produces a second message that duplicates the body and the header (as applicable) of the original. To avoid conflicts with the WebSphere Commerce messaging service, only JMS-required fields are duplicated. Accordingly, the format field is the only additional message property that is copied for messages that are archived or re-delivered.

Guaranteed-event-delivery

The guaranteed-event delivery feature enables the connector framework to ensure that events are never lost and never sent twice between the connector’s event store, the JMS event store, and the destination’s JMS queue. To become JMS-enabled, you must configure the connector `DeliveryTransport` standard property to JMS. Thus configured, the connector uses the JMS transport and all subsequent communication between the connector and the integration broker occurs through this transport. The JMS transport ensures that the messages are eventually delivered to their destination. Its role is to ensure that once a transactional queue session starts, the messages are cached there until a commit is issued; if a failure occurs or a rollback is issued, the messages are discarded.

Note: Without use of the guaranteed-event delivery feature, a small window of possible failure exists between the time that the connector publishes an event (when the connector calls the gotApplEvent() method within its pollForEvents() method) and the time it updates the event store by deleting the event record (or perhaps updating it with an "event posted" status). If a failure occurs in this window, the event has been sent but its event record remains in the event store with an "in progress" status. When the connector restarts, it finds this event record still in the event store and sends it, resulting in the event being sent twice.

You can configure the guaranteed-event delivery feature for a JMS-enabled connector with, or without, a JMS event store. To configure the connector for guaranteed-event delivery, see "Enabling guaranteed-event delivery" on page 33.

If connector framework cannot deliver the business object to the ICS integration broker, then the object is placed on a FaultQueue (instead of UnsubscribedQueue and ErrorQueue) and generates a status indicator and a description of the problem. FaultQueue messages are written in MQRFH2 format.

Business object requests

Business object requests are processed when InterChange Server sends a business object to the `doVerbFor()` method. Using the configured data handler, the adapter converts the business object to an WebSphere MQ message and issues it. There are no requirements regarding the type of business objects processed except those of the data handler.

Verb processing

The adapter processes business objects passed to it by a collaboration based on the verb for each business object. The adapter uses business object handlers and the `doForVerb()` method to process the business objects that the adapter supports. The adapter supports the following business object verbs:

- Create
- Update
- Delete
- Retrieve
- Exists
- Retrieve by Content

Note: Business objects with Create, Update, and Delete verbs can be issued either asynchronously or synchronously. The default mode is asynchronous. The adapter does not support asynchronous delivery for business objects with the Retrieve, Exists, or Retrieve by Content verbs, Accordingly, for Retrieve, Exists, or Retrieve by Content verbs, the default mode is synchronous.

Create, update, and delete

Processing of business objects with create, update and delete verbs depends on whether the objects are issued asynchronously or synchronously.

Asynchronous delivery

The default delivery mode for business objects with create, update, and delete verbs is asynchronous. A message is created from the business object using a data handler and then written to the output queue. If the message is delivered, the adapter returns BON_SUCCESS, else BON_FAIL.

Note: The adapter has no way of verifying whether the message is received or if action has been taken.

Synchronous delivery

Note: This approach requires a customization of the commands that are executed in WebSphere Commerce when a business object comes from the ICS. The commands should retrieve the `ReplyTo` queue from the message, and place a reply on the queue within the ResponseTimeout interval. For information about creating and customizing commands in WebSphere Commerce, refer to the *Programmer's Guide for WebSphere Commerce, V. 5.4*.

If a `ReplyToQueue` has been defined in the connector properties and a `ResponseTimeout` exists in the conversion properties for the business object, the adapter issues a request in synchronous mode. The adapter then waits for a response to verify that appropriate action was taken by WebSphere Commerce.

The adapter initially issues a message with a header as shown in Table 1.

Table 1. Request Message Descriptor Header (MQMD)

Field	Description	Value
Format	Format name	Output format as defined in the conversion properties and truncated to 8 characters to meet IBM requirements (example: `MQSTR`)
MessageType	Message Type	`MQMT_DATAGRAM`*
Report	Options for report message requested.	When a response message is expected, this field is populated as follows: `MQRO_PAN`* to indicate that a positive-action report is required if processing is successful. `MQRO_NAN`* to indicate that a negative-action report is required if processing fails. `MQRO_COPY_MSG_ID_TO_CORREL_ID`* to indicate that the correlation ID of the report generated should equal the message ID of the request originally issued.
ReplyToQueue	Name of reply queue	When a response message is expected this field is populated with the value of connector property `ReplyToQueue`.
Persistence	Message persistence	`MQPER_PERSISTENT`*
Expiry	Message lifetime	`MQEI_UNLIMITED`*
* Indicates constant defined by IBM.		

The message header described in Table 1 is followed by the message body. The message body is a business object that has been serialized using the data handler.

The Report field is set to indicate that both positive and negative action reports are expected from the WebSphere Commerce. The thread that issued the message waits for a response message that indicates whether WebSphere Commerce was able to process the request.

When WebSphere Commerce receives a synchronous request from the adapter, it processes the data of the business object and issues a report message as described in Table 2, Table 3, and Table 4.

Table 2. Response Message Descriptor Header (MQMD)

Field	Description	Value
Format	Format name	Input format of `busObj` as defined in the conversion properties.
MessageType	Message Type	`MQMT_REPORT`*
* Indicates constant defined by IBM.		

Table 3. Population of response message

Verb	Feedback field	Message body
Create, Update, or Delete	SUCCESS VALCHANGE	(Optional) A serialized business object reflecting changes.
	VALDUPES FAIL	(Optional) An error message.

Table 4. WebSphere MQ feedback codes and ICS response values

WebSphere MQ feedback code	Equivalent ICS response*
`MQFB_PAN` or `MQFB_APPL_FIRST`	SUCCESS
`MQFB_NAN` or `MQFB_APPL_FIRST + 1`	FAIL
`MQFB_APPL_FIRST + 2`	VALCHANGE
`MQFB_APPL_FIRST + 3`	VALDUPES
`MQFB_APPL_FIRST + 4`	MULTIPLE_HITS
`MQFB_APPL_FIRST + 5`	FAIL_RETRIEVE_BY_CONTENT
`MQFB_APPL_FIRST + 6`	BO_DOES_NOT_EXIST
`MQFB_APPL_FIRST + 7`	UNABLE_TO_LOGIN
`MQFB_APPL_FIRST + 8`	APP_RESPONSE_TIMEOUT (results in immediate termination of connector)
`MQFB_NONE (this is the default if no feedback code is specified)`	VALCHANGE
* See the *Connector Development Guide* for details.	

If the business object can be processed, the application creates a report message with the feedback field set to `MQFB_PAN` (or a specific ICS value). Optionally the application populates the message body with a serialized business object containing any changes. If the business object cannot be processed, the application creates a report message with the feedback field set to `MQFB_NAN` (or a specific ICS value) and then optionally includes an error message in the message body. In either case, the application sets the `correlationID` field of the message to the `messageID` of the adapter message and issues it to the queue specified by the `ReplyTo` field.

Upon retrieval of a response message, the adapter matches the `correlationID` of the response to the `messageID` of a request message. The adapter then notifies the thread that issued the request. Depending on the feedback field of the response, the adapter either expects a business object or an error message in the message body. If a business object was expected but the message body is not populated, the adapter simply returns the same business object that was originally issued by InterChange Server for the Request operation. If an error message was expected but the message body is not populated, a generic error message will be returned to InterChange Server along with the response code.

Creating Custom Feedback Codes: You can extend the WebSphere MQ feedback codes to override default interpretations shown in Table 4 by specifying the connector property `FeedCodeMappingMO`. This property allows you to create a meta-object in which all ICS-specific return status values are mapped to the WebSphere MQ feedback codes.

The return status assigned to a feedback code is passed to InterChange Server. For more information, see “FeedbackCodeMappingMO” on page 30.

Retrieve, exists and retrieve by content

Business objects with the retrieve, exists, and retrieve by content verbs support synchronous delivery only. The connector processes business objects with these verbs as it does for the synchronous delivery defined for create, update and delete. However, when using retrieve, exists, and retrieve by content verbs, the `ResponseTimeout` and `ReplyToQueue` are required. Furthermore, for retrieve and retrieve by content verbs, the message body must be populated with a serialized business object to complete the transaction.

Table 5 shows the response messages for these verbs.

Table 5. Population of response message

Verb	Feedback field	Message body
Retrieve or RetrieveByContent	FAIL FAIL_RETRIEVE_BY_CONTENT	(Optional) An error message.
	MULTIPLE_HITS SUCCESS	A serialized business object.
Exist	FAIL	(Optional) An error message.
	SUCCESS	

Common configuration tasks

After installing the adapter, you must configure the connector before starting it. This section provides an overview of some of the configuration and startup tasks that most developers will need to perform.

Installing the adapter

See Chapter 2, “Installing and configuring the connector,” on page 15, for a description of what and where you must install.

Configuring connector properties

Connectors have two types of configuration properties: standard configuration properties and connector-specific configuration properties. Some of these properties have default values that you do not need to change. You may need to set the values of some of these properties before running the connector. For more information, see Chapter 2, “Installing and configuring the connector,” on page 15.

When you configure connector properties for the adapter for WebSphere Commerce, make sure that:

- The value specified for connector property `HostName` matches that of the host of your WebSphere MQ server.
- The value specified for connector property port matches that of the port for the listener of the queue manager used by the adapter.
- The value specified for connector property channel matches the server connection channel for the queue manager used by the adapter.
- The values specified for queues match the names you used in creating the queues.
- The queue URIs for the connector properties `InputQueue`, `InProgressQueue`, `ArchiveQueue`, `ErrorQueue`, and `UnsubscribeQueue` are valid and actually exist. See Table 7.

Sending requests without notification

To configure the adapter to send requests without notification (the default asynchronous mode, also known as “fire and forget”):

- Create a business object that represents the request you want to send and is compatible with the XML data handler.
- Use either a static or a dynamic meta-object to specify the target queue and format. For more on static and dynamic meta-objects, see “Meta-objects configuration” on page 38.
- Set the property ResponseTimeout in the (static or dynamic) meta-object to `-1`. This forces the connector to issue the business object without checking for a return.

Sending requests and get notifications

Note: This approach requires a customization of the commands that are executed in WebSphere Commerce when a business object comes from the ICS. The commands should retrieve the `ReplyTo` queue from the message, and place a reply on the queue within the ResponseTimeout interval. For information about creating and customizing commands in WebSphere Commerce, refer to the *Programmer's Guide for WebSphere Commerce 5.4*.

If you configure the adapter to send requests and get notifications, specify a positive ResponseTimeout value to indicate how long the adapter waits for a reply.

This approach also requires that you define a `ReplyTo` queue in the connector properties. See “Synchronous delivery” on page 8 for more information and details about what the connector expects in a response message. If the requirements listed are not met by the response message, the connector may report errors or fail to recognize the response message. See also sections on “Meta-objects configuration” on page 38, and Chapter 3, “Working with business objects,” on page 55.

Configuring a static meta-object

A static meta-object contains application-specific information that you specify about business objects and how the connector processes them. A static meta-object provides the connector with all the information it needs to process a business object when the connector is started.

If you know at implementation time which queues different business objects must be sent to, use a static meta-object. Make sure the connector subscribes to the static meta-object by specifying the name of the static meta-object in the connector-specific property `DataHandlerConfigMO`. For more information, see "Connector-specific configuration properties" on page 28..

Configuring a dynamic meta-object

If the connector is required to process a business object differently depending on the scenario, use a dynamic meta-object. This is a child object that you add to the business object. The dynamic meta-object tells the connector (at run-time) how to process a request. Unlike the static meta-object, which provides the connector with all of the information it needs to process a business object, a dynamic meta-object provides only those additional pieces of logic required to handle the processing for a specific scenario. To create and configure a dynamic meta-object:

- Create the dynamic meta-object and add it as a child to the request business object
- Program your collaboration with additional logic that populates the dynamic meta-object with information such as the target queue, message format, etc., before issuing it to the connector.

The connector will check for the dynamic meta-object and use its information to determine how to process the business object. For more information, see "Overview of creating dynamic child meta-objects" on page 45.

Configuring MQMD formats

MQMDs are message descriptors. MQMDs contain the control information accompanying application data when a message travels from one application to another. You must specify a value for the MQMD attribute OutputFormat in either your static or dynamic meta-object. For more information, see "Create, update, and delete" on page 8.

Configuring queue URIs

To configure queues for use with the connector:

- Specify all queues as Uniform Resource Identifiers (URIs). The syntax is:

  `queue://<InterChangeServerName.queue.manager>/<actual queue>`
- Specify the host for the queue manager in connector-specific configuration properties (see Table 7).
- If your target application expects an MQMD header only and cannot process the extended MQRFH2 headers used by JMS clients, append `?targetClient=1` to the queue URI. For more information, see "Queue uniform resource identifiers (URIs)" on page 37 and the WebSphere MQ programming guide.

Configuring the XML data handler

The XML data handler is required for using the adapter with WebSphere Commerce. There are two ways to configure the data handler:

- Specify the data handler class name in the connector-specific property `DataHandlerClassName`. For more information, see "Connector-specific configuration properties" on page 28.
- Specify the mime type and the data handler meta-object that defines the configuration for that mime type in the connector-specific properties `DataHandlerMimiType` and `DataHandlerConfigMO`, respectively. For more information, see Table 7 and the *IBM WebSphere InterChange Server Data Handler Guide*.

Modifying the startup script

See Chapter 2, "Installing and configuring the connector," on page 15, for a description of how to start the connector. You must configure connector properties before startup. You must also modify the startup file:

- Make sure you modify the `start_connector` script to point to the location of the client libraries. Do not install multiple versions of the client libraries or versions that are not up-to-date with your WebSphere MQ server. For more information, see "Startup file configuration" on page 52.

Chapter 2. Installing and configuring the connector

This chapter describes how to install and configure the adapter and enable the WebSphere Commerce application to work with the connector.

Adapter environment

Before installing, configuring, and using the adapter, you must understand its environment requirements. They are listed in the following section.

Broker compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. Version 2.6.x of the adapter for WebSphere Commerce is supported on the following adapter framework and integration brokers:

- **Adapter framework**:
  WebSphere Business Integration Adapter Framework versions 2.3.1, and 2.4.
- **Integration brokers:**
  - WebSphere InterChange Server, versions 4.1.1, 4.2, 4.2.1, 4.2.2
  - WebSphere MQ Integrator, version 2.1.0, 5.0
  - WebSphere MQ Integrator Broker, version 2.1.0
  - WebSphere Business Integration Message Broker, version 5.0
  - WebSphere Application Server Enterprise, version 5.0.2, with WebSphere Studio Application Developer Integration Edition, version 5.0.1

See *Release Notes* for any exceptions.

Note: For instructions on installing your integration broker and its prerequisites, see the following guides.
For WebSphere InterChange Server (ICS), see *IBM WebSphere InterChange Server System Installation Guide for UNIX* or *for Windows.*
For WebSphere message brokers, see *Implementing Adapters with WebSphere Message Brokers.*
For WebSphere Application Server, see *Implementing Adapters with WebSphere Application Server.*

Adapter platforms

This adapter is supported on the following software:

Operating systems:

One of the following application platforms:

- AIX 5.1, AIX 5.2
- Solaris 8.0
- HP UX 11.0, HP UX 11i
- Windows 2000
- iSeries
- Linux

Databases:

- DB2

Third-party software:

- WebSphere Commerce versions 5.4 and 5.5

Globalization

This adapter is DBCS (double-byte character set)-enabled but is not translated.

Prerequisite tasks

This section describes the installation and configuration tasks you must perform for WebSphere Commerce and other software before you can install and run the adapter.

These tasks are:

1. Install and configure WebSphere Commerce
2. Install and configure the Commerce Enhancement Pack
3. Publish a WebSphere Commerce store
4. Create and configure WebSphere MQ queues
5. Configure WebSphere Application Server JMS settings
6. Configure JMS ConnectionSpec within WebSphere Commerce
7. Update WebSphere Commerce JVM Settings
8. Enable WebSphere Commerce Adapter

Installing and configuring WebSphere Commerce

Install WebSphere Commerce Version 5.4, Business Edition, with Fix Pack 2 or WebSphere Commerce Version 5.5. Refer to the documentation that comes with the product for the installation steps and the post-install configuration. The WebSphere Commerce messaging system is equipped to handle the messages to interact with back-end systems.

You must update the CMDREG table, which is the command registry table in your WebSphere Commerce database, to use the XML message format.

Installing the Commerce enhancement pack

To install the Commerce Enhancement Pack, download the Commerce Enhancement Pack driver from the following URL and follow the instructions in the readme.txt file:
http://www.ibm.com/software/commerce/epacks

Publishing a store

You can use this adapter with an existing WebSphere Commerce published store, or you can create a new store.

Configuring WebSphere MQ queues

The WebSphere MQ queue configuration required for using the adapter depends in part upon the topology of your WebSphere Commerce and IBM WebSphere InterChange Server installations. You may be using any of the following topologies:

- Single machine

  WebSphere Commerce and IBM WebSphere InterChange Server and the adapter are all installed on the same machine
- Two machines with two queue managers

  WebSphere Commerce is installed on one machine, and IBM WebSphere InterChange Server and the connector are installed on another machine. A different queue manager is used on each machine.
- Two machines with a single queue manager

  WebSphere Commerce is installed on one machine, and IBM WebSphere InterChange Server and the connector are installed on another machine. The same queue manager is used to manage the queues on both machines.

Single-machine topology

In this topology, WebSphere Commerce, InterChange Server and the adapter for WebSphere Commerce are all installed on a single machine. A single queue manager handles all the WebSphere MQ queues used in the solution. It is recommended that you use the queue manager that you set up when you installed ICS.

This topology requires queues that perform the following roles:

- Inbound Queue

  WebSphere Commerce requires that this queue exist. However, the adapter does not make use of it in this solution.
- Parallel Inbound Queue

  WebSphere Commerce requires that this queue exist. However, the adapter does not make use of it in this solution.
- Serial Inbound Queue

  For receiving messages sent from ICS to WebSphere Commerce.
- Outbound Queue

  For sending messages from WebSphere Commerce to ICS.
- InProgress Queue

  The original versions of valid messages sent from WebSphere Commerce to ICS are stored here until the adapter completes processing; when processing is completed, the original message is moved to the local Archive queue.
- Archive_Queue

When a message has been fully processed by the adapter and sent to ICS from WebSphere Commerce, the original version of the message is stored here.

- Unsubscribed_Queue

  If a message is successfully parsed but does not correspond to any business object supported by the adapter, it is stored here.
- ICS_Error_Queue

  If a message is not successfully converted to a business object and sent to the ICS, it is stored here.
- WCS_Error_Queue

  Stores messages that cannot be successfully processed by WebSphere Commerce.
- ReplyTo Queue

  Used only in configurations that are set up for synchronous data exchange.

All of the above queues are local in the single machine topology. If you create the queues manually, you choose the names that you assign for them; if you use the batch file provided with this solution (described below), the batch file will create the queues with pre-assigned names.

If you are using the adapter in a Windows environment, you can use a batch file to generate queues that are appropriate for a single machine topology. The file is installed with your product package in the `\Connector\WebSphereCommerce\Utilities` subdirectory within the root directory you used for your IBM ICS installation. To create the queues using the batch file, run the file `ConfigureWebSphereCommerceAdapter.bat`, as follows:

From the command prompt, enter:

```
ConfigureWebSphereCommerceAdapter
<InterChangeServerName>.queue.manager
```

Where, `<InterChangeServerName>` is the name of your WebSphere InterChange Server.

This will create a queue manager named `InterChange` *`ServerName`*`.queue.manager` and it will create the required WebSphere MQ queues. The names of the created queues, as created by the batch file, are as follows:

WC_MQCONN.IN_PROGRESS In Progress queue for the adapter.

WC_MQCONN.ERROR ICS Error Queue for the adapter.

WC_MQCONN.ARCHIVE Archive Queue for the adapter.

WC_MQCONN.REPLY Reply-To_Queue for the adapter.

WC_MQCONN.UNSUBSCRIBED UnSubscribed Queue for the adapter.

WCS_Serial_Inbound Serial Inbound Queue for WebSphere Commerce. Must match a JMS queue name defined for WebSphere Commerce, as described in “Configuring WebSphere Application Server JMS settings” on page 22

WCS_Outbound Outbound queue for WebSphere Commerce. Must match a JMS queue name defined for WebSphere Commerce, as described in “Configuring WebSphere Application Server JMS settings” on page 22

WCS_Parallel_Inbound Parallel inbound queue for WebSphere Commerce. Must match a JMS queue name defined for WebSphere Commerce, as described in "Configuring WebSphere Application Server JMS settings" on page 22

WCS_Error Error queue for WebSphere Commerce. Must match a JMS queue name defined for WebSphere Commerce, as described in "Configuring WebSphere Application Server JMS settings" on page 22

WCS_Inbound Inbound queue for WebSphere Commerce. Must match a JMS queue name defined for WebSphere Commerce, as described in "Configuring WebSphere Application Server JMS settings" on page 22

Two-machine, two-Queue Manager topology

In this topology, WebSphere Commerce is installed on one machine and IBM WebSphere ICS and the adapter for WebSphere Commerce are installed on another machine.

WebSphere MQ must be installed on each machine, and each installation uses a different queue manager. These are the queues you create on each machine.

Note: In this table, the queue names indicate the role of each queue, but you can establish different queue names, as long as you synchronize the queue names with JMS queue names used on your WebSphere Commerce system. In the table, the prefix WCS indicates a queue that is created on the machine on which the WebSphere Commerce system is installed, and that is managed by a queue manager that resides on that machine.
The prefix ICS indicates a queue that is created on the machine on which ICS and the connector are installed, and that is managed by a queue manager that resides on that machine.

Queues on the WebSphere Commerce machine	Queues on the ICS machine
WCS_Outbound Queue	**ICS_Inbound queue**
For sending messages from WebSphere Commerce to ICS. This queue is created as a remote queue definition, pointing to ICS_Inbound on the ICS machine as the remote queue.	For receiving messages sent from WebSphere Commerce to ICS.
WCS_Serial Inbound queue	**ICS_Outbound queue**
For receiving messages sent from ICS to WebSphere Commerce.	For sending messages from ICS to WebSphere Commerce. This queue is created as a remote queue definition, pointing to WCS_Serial Inbound on the WebSphere Commerce machine as the remote queue
To ICS	**To WebSphere Commerce**
Transmission queue from WebSphere Commerce system to ICS.	Transmission queue from ICS to WebSphere Commerce system.
WCS_Error_queue	**ICS_Error_queue**
Stores messages that cannot be successfully processed by WebSphere Commerce.	If a message is not successfully converted to a business object, it is stored here.
WCS_Parallel Inbound	The adapter does not make use of a parallel inbound queue.

Queues on the WebSphere Commerce machine	Queues on the ICS machine
	ICS_InProgress queue The original versions of valid messages sent from WebSphere Commerce to ICS are stored here until the adapter completes processing; when processing is completed, the original message is moved to the local Archive queue.
	ICS_Archive_queue When a message has been fully processed by the adapter and sent to ICS from WebSphere Commerce, the original version of the message is stored here.
	ICS_Unsubscribed_queue If a message is successfully converted to a business object but does not correspond to any business object supported by the adapter, it is stored here.

To enable communication between the two systems, use channels and transmission queues.

Channels that perform the following roles must be created on each machine for this topology.

Channels On the WebSphere Commerce machine	Channels On the ICS machine
Sender_WCS	Sender_ICS
Receiver_ICS	Receiver_WCS

Creating the channels: In the following instructions, specific server and queue manager names are specified so that they correlate the different machines and queues. Channels are used to make sure that the correct queues refer to each other; the "local" versions of the queues are used to hold the actual information.

Perform the following configuration tasks on the WebSphere Commerce machine:

Note: The channel names used here are examples only.

1. Create two channels in the WebSphere Commerce system using WebSphere MQ Explorer. One sender channel named 'WCS' and one receiver channel named 'ICS'.
2. Create a local queue, for example, using the name 'ToICSSystem'.
3. Set the ToICSSystem queue as the transmission queue.
4. Set the following properties for the WCS_Outbound queue.
   a. Remote queue name `ICS_Inbound`Remote queue manager name `ICS_server_name.queue.manager`. For example, `ICS.queue.manager`.
   b. Set the transmission queue name property to 'ToICSSystem' as created in step 2.
5. To configure the sender channel do the following:

   a. Specify the connection name with the IP address and the port, for example, 9.182.12.235(1414). Where, 9.182.12.235 is the IP address of the machine where ICS is running and 1414 is the default listener port.
   b. Specify the transmission queue name as ‘ToICSSystem’.

This completes the configuration tasks for the WebSphere Commerce machine.

Perform the following configuration tasks on the ICS machine:

1. Create two channels using WebSphere MQ Explorer: One sender channel named ‘ICS’ and one receiver channel named ‘WCS’.

   Note: The name of the sender channel in the WebSphere business integration system must be identical to the name of the receiver channel in WebSphere Commerce. The name of the receiver channel in the WebSphere business integration system must be identical to the name of the sender channel in WebSphere Commerce.
2. Create a new local queue, for example ‘ToWCSSystem’. Set the ToWCSSystem queue as the transmission queue.
3. Create a remote definition queue in the WebSphere business integration system. This remote definition queue must be used in the connector component as the output queue. Set the following properties:
   a. Remote queue name `WCS_SerialInbound`
   b. Remote queue manager name `<wcssytems_Q_manager_name>`. For example, QM_wcsfvt3.
   c. Set the transmission queue name property to ‘ToWCSSystem’.
4. To configure the sender channel do the following:
   a. Specify the connection name with the IP address and the port, for example, 9.182.12.18(1414). Where, 9.182.12.18 is the IP address of the machine where WebSphere Commerce is running and 1414 is the default listener port.
   b. Specify the transmission queue name as ‘TOWCSSystem’.

After you finish configuring the WebSphere MQ queues and channels on both the WebSphere Commerce machine and the ICS machine, start the receiver channel and then the sender channel.

Two-machine, one-queue manager topology

In this topology, WebSphere Commerce is installed on one machine and ICS and the adapter for WebSphere Commerce are installed on another machine. Only one instance of WebSphere MQ is running, and the queues used by both machines are managed by a single queue manager. This scenario uses only local queues.

This topology requires queues that perform the following roles:

- Inbound Queue

  WebSphere Commerce requires that this queue exist. However, the adapter does not make use of it in this solution.
- Serial Inbound Queue

  For sending messages from ICS to WebSphere Commerce.
- Outbound Queue

  For receiving messages sent from WebSphere Commerce to ICS.
- InProgress Queue

The original versions of valid messages sent from WebSphere Commerce to ICS are stored here until the adapter completes processing; when processing is completed, the original message is moved to the local Archive queue.

- Archive_Queue

  When a message has been fully processed by the adapter and sent to ICS from WebSphere Commerce, the original version of the message is stored here.
- Unsubscribed_Queue

  If a message is successfully parsed but does not correspond to any business object supported by the adapter, it is stored here.
- WebSphere Commerce Error Queue
- ICS_Error_Queue

  If a message is not successfully converted to a business object and sent to the ICS, it is stored here.
- WCS_Error_Queue

  Stores messages that cannot be successfully processed by WebSphere Commerce.
- ReplyTo Queue

  Used only in configurations that are set up for synchronous data exchange.

Configuring WebSphere Application Server JMS settings

You must configure WebSphere Application Server (WAS), Version 4.x, to create the Java Messaging Service Connection Factory and JMS queues to work with WebSphere MQ. To do so, perform the following steps.

1. From the command prompt:
   a. Update your classpath variable by typing the following command on one line:

   ```
   set classpath= %classpath%;
   MQ_install_path\java\lib\com.ibm.mqjms.jar;
   MQ_install_path\java\lib\com.ibm.mq.jar;
   MQ_install_path\java\lib\com.ibm.mq.iopp.jar;
   MQ_install_path\java\lib\com.ibm.ibmorb.jar;WAS_install_path\lib\ns.jar
   ```

MQ_install_path	The path in which you installed WebSphere MQ
WAS_install_path	The path in which you installed the WebSphere Application Server

   b. Add a new environment variable named MQ_JAVA_INSTALL_PATH by typing the following command:

   ```
   set MQ_JAVA_INSTALL_PATH=MQ_install_path\java
   ```

MQ_install_path	The path in which you installed WebSphere MQ

   c. Update the environment to use the JDK (Java Development Kit) that comes with WebSphere Application Server by typing the following command:

   ```
   set PATH = WAS_Intall_Path\Java\bin;%PATH%
   ```

WAS_install_path	The path in which you installed the WebSphere Application Server

2. Ensure that the WebSphere Application Server is running and the correct classpath and environment variables defined in Step 1 above are added. Also check to make sure the JDK being used is the one in WAS by executing `java -version` and checking the version with the one found in *WAS_Install_Path*`\Java\bin`.

3. In the `MQ_install_path\java\bin directory`, open the `JMSAdmin.config` file and set the following values:

   ```
   INITIAL_CONTEXT_FACTORY=com.ibm.ejs.ns.jndi.CNInitialContextFactory
   PROVIDER_URL=iiop://localhost:900
   ```

   The values above assume that WebSphere Commerce and WebSphere MQ are installed on the same machine.

   ```
   SECURITY_AUTHENTICATION=none
   ```

4. Run the JMSAdmin program by providing the `JMSAdmin.config` file as a command line input:

   CommandPrompt:> `JMSAdmin -cfg JMSAdmin.config -t -v`

   Executing this command should enable you to lookup the JNDI (Java Naming and Directory Interface) service provided by WebSphere Application Server. You will see an InitCtx> prompt that you can use to run the JMS administration commands.

5. Register the QueueConnectionFactory and set the coded character set identifier by typing the following commands:
   - `define qcf (JMS_QueueConnection_Factory) qmanager (Your_QueueManager_Name)`
   - `alter qcf (JMS_QueueConnection_Factory) ccsid(1208)`

     Where `JMS_QueueConnection_Factory` is the name of the MQQueueConnectionFactory JMS object.

   When you execute the above set of commands, an entry for this queue connection factory is created in the WebSphere Application Server database under the BINDINGBEANTBL table. These objects are registered in the WebSphere Application Server database.

6. Define the following JMSQueues to map to the names you have established for your WebSphere MQ queues and the WebSphere MQ queue manager you are using. You can customize the JMSqueue names according to your own requirements, but the WebSphere MQ names to which you define them must match exactly, including casing, queue names you have established in WebSphere MQ. If you are using the batch file provided with this adapter for creating the appropriate WebSphere MQ queues for a single-machine topology, be sure to use those generated WebSphere MQ queue names as the values to which you define the JMS queues, as described in the table below.

   The following syntax defines the JMS Queues:

Note: If you are using a remote queue definition for the outbound queue, as you would in a two-machine, two-queue manager topology, the JMS_Outbound_Queue should not be defined to a local WebSphere MQ queue. If you are using a remote queue definition, the syntax for the outbound queue would be: `define q(`*`JMS_Outbound_Queue`*`)qmanager(`*`Your_Queue_Manager_Name`*`)`

```
define q(JMS_Outbound_Queue)qmanager
 (Your_Queue_Manager_Name)
  queue(Your_Outbound_QueueName)

define q(JMS_Inbound_Queue)qmanager
 (Your_Queue_Manager_Name)
  queue(Your_Inbound_QueueName)

define q(JMS_Parallel_Inbound_Queue)qmanager
  (Your_Queue_Manager_Name)queue
    (Your_Parallel_Inbound_Queue_Name)
```

```
define q(JMS_Serial_Inbound_Queue)qmanager
  (Your_Queue_Manager_Name)queue
   (Your_Serial_Inbound_Queue_Name)
define q(JMS_Error_Queue)qmanager
  (Your_Queue_Manager_Name)
   queue (Your_Error_Queue_Name)
```

Table 6. Defining JMS queue names

Your_Outbound_QueueName	The WebSphere MQ queue created for the outbound queue. By default, this is the queue which the adapter polls to pick up messages from WebSphere Commerce to pass to ICS. In the default WebSphere MQ queue setup created by the batch file, this value should be `WCS_Outbound`
Your_Serial_Inbound_Queue	The WebSphere MQ queue created for the serial inbound queue. This is the queue into which WebSphere Commerce MQ Adapter places messages sent from ICS to Websphere Commerce. In the default WebSphere MQ queue setup created by the batch file, this value should be `WCS_Serial_Inbound`
Your_Parallel_Inbound_Queue_Name	This is the WebSphere MQ queue created for the parallel inbound queue
Your_Error_Queue_Name	The WebSphere MQ queue created for the error queue. This is where WebSphere Commerce MQ Adapter sends messages when it encounters an error with the message. In the default WebSphere MQ queue setup created by the batch file, this value should be `WCS_Error`
Your_Queue_Manager_Name	The name of the queue manager handling WebSphere MQ queues in your set up for the WebSphere Commerce system. In a typical single machine setup, such as the setup created by the batch file ConfigureAdapterQueues.bat (see "Installing and configuring WebSphere Commerce" on page 16), the queue manager that you established for ICS would be used to manage the queues for the WebSphere Commerce system as well. For such a setup, the default would be `<InterchangeServerName>.queue.manager`

After creating the queues, set the following property for the outbound and error queues using the JMSAdmin console. This procedure specifies that JMS is dealing with a native WebSphere MQ application.

- alter q(JMSOutboundQueue) targclient(MQ)
- alter q(JMSErrorQueue) targclient(MQ)

Type `end` to exit the JMSAdmin tool. This completes your task for configuring the Java Messaging Service with WebSphere Application Server running WebSphere Commerce.

Configuring JMS ConnectionSpec within WebSphere Commerce Version 5.4

Note: The JMSQueue names and JMS connection factory must be the same as the values entered in the `connectionSpec` section of Commerce Configuration

Manager, in the instance XML file. You can find the details under the `Transports` section in the WebSphere Commerce Configuration Manager. Also see the instructions below.

Start the WebSphere Commerce Administration Console. Log in as a Site Administrator, go to the Configuration section and choose the Transport option. Select WebSphere MQ as your transport and change the status to active. Log out from the Administration Console.

The WebSphere Commerce solution requires the creation and use of a "store," as described in the *WebSphere Commerce Installation Guide.* When you have completed publishing the store as described in "Publishing a Sample Store" section of that guide, log into the Administration Console, this time as a Store Administrator, and select the store you are using. In the Configuration section, add MQ Transport to the store. An entry for this is made in the STORETRANS table.

To enable the messaging system transport adapter, launch the WebSphere Commerce Configuration Manager and do the following:

1. Select Host name -> Instance, and then open the Components folder.
2. Select TransportAdapter.
3. Select the Enable Component checkbox and click **Apply**.

Configure JMSQueue names and JMS Connection Factory with the values that you are using for the connectionSpec in this instance, as follows:

1. Select the Host Name -> Instance
2. Select Transports, then expand Outbound->JMS
3. Select the ConnectionSpec
4. Input the ConnectionFactory name created when configuring the JMS settings for WebSphere Application Server.
5. Enter the Inbound, Error, and Outbound queue names created above.
6. Click Apply.
7. Expand Inbound->JMSInbound CCF Connector-Serial
8. Select the ConnectionSpec
9. Input the ConnectionFactory Name, SerialInbound, Error and Output JMS queues.
10. Click Apply.
11. Expand Inbound-JMSInbound CCF Connector-Parallel
12. Select the ConnectionSpec
13. Input the ConnectionFactory, ParralelInbound, Error and Output JMS queues.
14. Click Apply.

Exit the Configuration Manager.

Updating WebSphere Commerce JVM settings

You must update the WebSphere Application Server Version 4.x class path for the instance, adding the additional jar file entries. To do so, open the WebSphere Application Server Advanced Administrative Console and complete the following:

1. Select the host on which you are running your WebSphere Commerce instance.
2. Select the WebSphere Administrative Domain.

3. Select Nodes.
4. Select your host name.
5. Select Application Servers.
6. Select the WebSphere Commerce Server instance_name, where instance_name is the name of your WebSphere Commerce instance.
7. Go to the JVM settings of the instance.
8. Select Add a new system property.
9. Type in the following system property:
   ```
   name= ws.ext.dirs value=MQ_INSTALL_PATH/java/lib
   ```
   Where MQ_INSTALL_PATH is the path where you installed WebSphere WebSphere MQ.
10. Restart the WebSphere Application Server service for all the changes to take affect.

Enabling WebSphere MQ for the adapter

In WebSphere Commerce, use the configuration option under the Administration console to configure WebSphere Commerce to communicate with the WebSphere MQ queues for outbound and inbound messaging that you are using in your implementation of WebSphere Commerce and IBM WebSphere ICS. See the *WebSphere Commerce Online Help* guide for additional instructions if needed.

Installing the adapter and related files

For information on installing WebSphere Business Integration adapter products, refer to the *Installation Guide for WebSphere Business Integration Adapters*, located in the WebSphere Business Integration Adapters Infocenter at the following site:

http://www.ibm.com/websphere/integration/wbiadapters/infocenter

Installed file structure

The sections below describe the paths and filenames of the product after installation.

Note: WebSphere Commerce and JMS typically are installed in separate directories in both Windows and UNIX environments. On an AIX system for example, WebSphere Commerce is installed by default in `/var/mqm/` while JMS is installed in `/usr/mqm/java/lib`. You may want to redirect the JMS install to `/var/mqm/java/lib` to avoid deletion by routine `/usr`-related system administration tasks. Likewise on Windows, WebSphere Commerce is generally installed under `\Program Files\WebSphere Commerce` and JMS under `\Program Files\IBM\MQSerires\Java`. Update your classpath accordingly in the WebSphere Commerce connector startup script.

Windows file structure

The Installer copies the standard files associated with the connector into your system.

The utility installs the connector into the *`ProductDir`*`\connectors\WebSphereCommerce` directory, and adds a shortcut for the connector to the Start menu.

The table below describes the Windows file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through Installer.

Subdirectory of *ProductDir*	Description
`connectors\WebSphereCommerce\CWWebSphereCommerce.jar`	Contains classes used by the WebSphere Commerce connector only
`connectors\WebSphereCommerce\start_WebSphereCommerce.bat`	The startup script for the connector (NT/2000)
`connectors\messages\WebSphereCommerceConnector.txt`	Message file for the connector
`bin\Data\App\WebSphereCommerceConnectorTemplate`	Template file for adapter definition
`utilities`	Contains utilities directory

Note: All product path names are relative to the directory where the product is installed on your system.

UNIX file structure

The Installer copies the standard files associated with the connector into your system.

The utility installs the connector into the *ProductDir*`/connectors/WebSphereCommerce` directory.

The table below describes the UNIX file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through Installer.

Subdirectory of *ProductDir*	Description
`connectors/WebSphereCommerce/CWWebSphereCommerce.jar`	Contains classes used by the WebSphere Commerce connector only
`connectors/WebSphereCommerce/start_WebSphereCommerce.sh`	System startup script for the connector. This script is called from the generic connector manager script. When you click the Connector Configuration screen of System Manager, the installer creates a customized wrapper for this connector manager script. Use this customized wrapper to start and stop the connector.
`connectors/messages/WebSphereCommerce/Connector.txt`	Message file for the connector
`bin/Data/App/WebSphereCommerceConnectorTemplate`	Template file for the adapter definition
`utilities`	Contains utilities directory

Note: All product path names are relative to the directory where the product is installed on your system.

Configuring the adapter

Connectors have two types of configuration properties: standard configuration properties and adapter-specific configuration properties. You must set the values of these properties before running the adapter.

You use Connector Configurator to configure connector properties:
- For a description of Connector Configurator and step-by-step procedures, see Appendix B, “Connector Configurator,” on page 79.
- For a description of standard connector properties, see “Standard connector properties” on page 28 and Appendix A, “Standard configuration properties for connectors,” on page 61.

- For a description of connector-specific properties, see "Connector-specific configuration properties."

A connector obtains its configuration values at startup. During a runtime session, you may want to change the values of one or more connector properties. Changes to some connector configuration properties, such as AgentTraceLevel, take effect immediately. Changes to other connector properties require component restart or system restart after a change. To determine whether a property is dynamic (taking effect immediately) or static (requiring either connector component restart or system restart), refer to the Update Method column in the Connector Properties window of Connector Configurator.

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 61, for documentation of these properties.

Because this adapter supports only InterChange Server (ICS) as the integration broker, the only configuration properties relevant to it are for ICS.

Connector-specific configuration properties

Connector-specific configuration properties provide information needed by the connector at runtime. They also provide a way of changing static information or logic within the adapter without having to recode and rebuild the agent.

The following table lists the connector-specific configuration properties for the adapter. See the sections that follow for explanations of the properties.

Note: These properties include default queue name values. You will need to change these values to match the queue names you are actually using in your set up.

Table 7. Connector-specific configuration properties

Name	Possible values	Default value	Required
ApplicationPassword	*Login password*		No
ApplicationUserName	*Login user ID*		No
ArchiveQueue	*Queue to which copies of successfully processed messages are sent*	`queue://<queue_manager_name>/WC_MQCONN.ARCHIVE`	No
CCSID	Character set for the queue manager connection		No
Channel	*MQ server connector channel*		Yes
ConfigurationMetaObject	*Name of configuration meta-object*		Yes
DataHandlerClassName	*Data handler class name*	`com.crossworlds.DataHandlers.text.xml`	No
DataHandlerConfigMO	*Data handler meta-object*	`MO_DataHandler_ Default`	Yes
DataHandlerMimeType	*MIME type of file*	`text/xml`	No
DefaultVerb	Any verb supported by the connector		No
EnableMessageProducerCache	*true or false*	`true`	No
ErrorQueue	*Queue for unprocessed messages*	`queue://<queue_manager_name>/WC_MQCONN.ERROR`	No
FeedbackCodeMappingMO	Feedback code meta-object		No
HostName	*WebSphere MQ server*		Yes

Table 7. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
InDoubtEvents	`FailOnStartup Reprocess Ignore LogError`	`Reprocess`	No
InputQueue	*Poll queues*	`queue://<queue_manager_name>/WC_MQCONN.IN`	No
InProgressQueue	*In-progress event queue*	`queue://<queue_manager_name>/WC_MQCONN.IN_PROGRESS`	Yes
PollQuantity	*Number of messages to retrieve from each queue specified in the `InputQueue` property*	`1`	No
Port	*Port established for the WebSphere MQ listener*		Yes
ReplyToQueue	*Queue to which response messages are delivered when the adapter issues requests*	`queue://<queue_manager_name>/WC_MQCONN.REPLYTO`	No
SessionPoolSizeForRequests	*Maximum pool size for caching the sessions used during request processing*	`10`	No
UnsubscribedQueue	*Queue to which unsubscribed messages are sent*	`queue://<queue_manager_name>/WC_MQCONN.UNSUBSCRIBE`	No
UseDefaults	true or false	false	

ApplicationPassword

Password used with `UserID` to log in to WebSphere MQ.

Default = None.

If the `ApplicationPassword` is left blank or removed, the adapter uses the default password provided by WebSphere MQ.

ApplicationUserName

User ID used with `Password` to log in to WebSphere MQ.

Default=None.

If the `ApplicationUserName` is left blank or removed, the adapter uses the default user ID provided by WebSphere MQ.

ArchiveQueue

Queue to which copies of successfully processed messages are sent.

Default = `queue://<queue_manager_name>/WC_MQCONN.ARCHIVE`

CCSID

The character set for the queue manager connection. The value of this property should match that of the CCSID property in the queue of the URI. For more information, see"Queue uniform resource identifiers (URIs)" on page 37.

Default = none.

Channel

MQ server adapter channel through which the adapter communicates with WebSphere MQ.

Default=none.

If the `Channel` is left blank or removed, the adapter uses the default server channel provided by WebSphere MQ.

ConfigurationMetaObject

Name of static meta-object containing configuration information for the connector.

Default = none.

DataHandlerClassName

Data handler class to use when converting messages to and from business objects.

Default = `com.crossworlds.DataHandlers.text.xml`

DataHandlerConfigMO

Meta-object passed to data handler to provide configuration information.

Default = `MO_DataHandler_Default`

DataHandlerMimeType

Allows you to request a data handler based on a particular MIME type. The XML data handler is required for use with WebSphere Commerce.

Default = `text/xml`

DefaultVerb

Specifies the verb to be set within an incoming business object, if it has not been set by the data handler during polling.

Default = none.

EnableMessageProducerCache

Boolean property to specify that the adapter should enable a message producer cache for sending request messages.

Default = true.

EnableMessageProducerCache

Boolean property to specify that the adapter should enable a message producer cache for sending request messages

Default = `true`

ErrorQueue

Queue to which messages that could not be processed are sent.

Default = `queue://`*`<queue_manager_name>`*`/WC_MQCONN.ERROR`

FeedbackCodeMappingMO

Allows you to override and reassign the default feedback codes used to synchronously acknowledge receipt of messages to InterChange Server. This property enables you to specify a meta-object in which each attribute name is understood to represent a feedback code. The corresponding value of the feedback code is the return status that is passed to InterChange Server. For a listing of the default feedback codes, see "Synchronous delivery" on page 8. The adapter accepts the following attribute values representing WebSphere MQ-specific feedback codes:

- MQFB_APPL_FIRST
- MQFB_APPL_FIRST_OFFSET_*N where N is an integer (interpreted as the value of* MQFB_APPL_FIRST + *N*)

The adapter accepts the following ICS-specific status codes as attribute values in the meta-object:

- SUCCESS
- FAIL
- APP_RESPONSE_TIMEOUT
- MULTIPLE_HITS
- UNABLE_TO_LOGIN
- VALCHANGE
- VALDUPES

The following table shows a sample meta-object.

Table 8. Example feedback code meta-object attributes

Attribute name	**Default value**
`MQFB_APPL_FIRST`	SUCCESS
`MQFB_APPL_FIRST + 1`	FAIL
`MQFB_APPL_FIRST + 2`	UNABLE_TO_LOGIN

Default = none.

HostName

The name of the server hosting WebSphere MQ.

Default=none.

InDoubtEvents

Specifies how to handle in-progress events that are not fully processed due to unexpected adapter shutdown. Choose one of four actions to take if events are found in the in-progress queue during initialization:

- `FailOnStartup.` Log an error and immediately shut down.
- `Reprocess.` Process the remaining events first, then process messages in the input queue.
- `Ignore.` Disregard any messages in the in-progress queue.
- `LogError.` Log an error but do not shut down

Default = `Reprocess`.

InputQueue

Message queues that will be polled by the adapter for new messages. The adapter accepts multiple semi-colon delimited queue names. For example, to poll the following three queues: `MyQueueA`, `MyQueueB`, and `MyQueueC`, the value for connector configuration property *`InputQueue`* would equal: `MyQueueA;MyQueueB;MyQueueC`.

If the `InputQueue` property is not supplied, the connector starts up properly but prints a warning message, and performs request processing only. It will not perform any event processing.

The adapter polls the queues in a round-robin manner and retrieves up to *pollQuantity* number of messages from each queue. For example, if *pollQuantity* equals 2, and MyQueueA contains 2 messages, MyQueueB contains 1 message and MyQueueC contains 5 messages, the adapter retrieves messages in the following manner:

Since we have a pollQuanity of 2, the adapter will retrieve at most two messages from each queue per call to pollForEvents. For the first cycle (1 of 2), the adapter retrieves the first message from each of MyQueueA, MyQueueB, and MyQueueC. That completes the first round of polling and if we had a pollQuantity of 1, the adapter would stop.

Since we have a pollQuanity of 2, the adapter starts a second round of polling (2 of 2) and retrieves one message each from MyQueueA and MyQueueC--it skips MqQueueB since it is now empty. After polling all queues 2x each, the call to the method pollForEvents is complete. Here's the sequence of message retrieval:

1. 1 message from MyQueueA
2. 1 message from MyQueueB
3. 1 message from MyQueueC
4. 1 message from MyQueueA
5. Skip MyQueueB since it's now empty
6. 1 message from MyQueueC

Default = queue://*<queue_manager_name>*/WC_MQCONN.IN

InProgressQueue

Message queue where messages are held during processing.

Default= queue://*<queue_manager_name>*/WC_MQCONN.IN_PROGRESS

PollQuantity

Number of messages to retrieve from each queue specified in the InputQueue property during a pollForEvents scan.

Default =1

Port

Port established for the WebSphere MQ listener.

Default=None.

ReplyToQueue

Queue to which response messages are delivered when the adapter issues requests.

Default = queue://*<queue_manager_name>*/WC_MQCONN.REPLY

SessionPoolSizeForRequests

The maximum pool size for caching the sessions used during request processing.

Default = 10

UnsubscribedQueue

Queue to which messages that are not subscribed are sent.

Default = queue://*<queue_manager_name>*/WC_MQCONN.UNSUBSCRIBED

Note: *Always check the values WebSphere MQ provides since they may be incorrect or unknown. If so, please implicitly specify values.

UseDefaults

On a Create operation, if UseDefaults is set to `true`, the connector checks whether a valid value or a default value is provided for each isRequired business object attribute. If a value is provided, the Create operation succeeds. If the parameter is set to `false`, the connector checks only for a valid value and causes the Create operation to fail if it is not provided. The default is `false`.

Enabling guaranteed-event delivery

You can configure the guaranteed-event -delivery feature for a JMS-enabled connector in one of the following ways:

- If the connector uses a JMS event store (implemented as a JMS source queue), the connector framework can manage the JMS event store. For more information, see "Guaranteed-event-delivery for connectors with JMS event stores."
- If the connector uses a non JMS event store (for example, implemented as a JDBC table, Email mailbox, or flat files), the connector framework can use a JMS monitor queue to ensure that no duplicate events occur. For more information, see "Guaranteed-event-delivery for connectors with non-JMS event stores" on page 35.

Guaranteed-event-delivery for connectors with JMS event stores

If the JMS-enabled connector uses JMS queues to implement its event store, the connector framework can act as a "container" and manage the JMS event store (the JMS source queue). In a single JMS transaction, the connector can remove a message from a source queue and place it on the destination queue. This section provides the following information about use of the guaranteed-event delivery feature for a JMS-enabled connector that has a JMS event store:

- "Enabling the feature for connectors with JMS event stores"
- "Effect on event polling" on page 35

Enabling the feature for connectors with JMS event stores

To enable the guaranteed-event delivery feature for a JMS-enabled connector that has a JMS event store, set the connector configuration properties to values shown in Table 9.

Table 9. Guaranteed-event-delivery connector properties for a connector with a JMS event store

Connector property	Value
`DeliveryTransport`	`JMS`
`ContainerManagedEvents`	`JMS`
`PollQuantity`	The number of events to processing in a single poll of the event store

Table 9. Guaranteed-event-delivery connector properties for a connector with a JMS event store (continued)

Connector property	Value
`SourceQueue`	Name of the JMS source queue (event store) which the connector framework polls and from which it retrieves events for processing **Note:** The source queue and other JMS queues should be part of the same queue manager. If the connector's application generates events that are stored in a different queue manager, you must define a remote queue definition on the remote queue manager. WebSphere MQ can then transfer the events from the remote queue to the queue manager that the JMS-enabled connector uses for transmission to the integration broker. For information on how to configure a remote queue definition, see your IBM WebSphere MQ documentation.

In addition to configuring the connector, you must also configure the data handler that converts between the event in the JMS store and a business object. This data handler information consists of the connector configuration properties that Table 10 summarizes.

Table 10. Data handler properties for guaranteed-event delivery

Data handler property	Value	Required?
`MimeType`	The MIME type that the data handler handles. This MIME type identifies which data handler to call.	Yes
`DHClass`	The full name of the Java class that implements the data handler	Yes
`DataHandlerConfigMOName`	The name of the top-level meta-object that associates MIME types and their data handlers	Optional

Note: The data handler configuration properties reside in the connector configuration file with the other connector configuration properties.

If you configure a connector that has a JMS event store to use guaranteed-event delivery, you must set the connector properties as described in Table 9 *and* Table 10. To set these connector configuration properties, use the Connector Configurator tool. Connector Configurator displays the connector properties in Table 9 on its Standard Properties tab. It displays the connector properties in Table 10 on its Data Handler tab.

Note: Connector Configurator activates the fields on its Data Handler tab only when the `DeliveryTransport` connector configuration property is set to `JMS` and `ContainerManagedEvents` is set to `JMS`.

For information on Connector Configurator, see Appendix B, "Connector Configurator," on page 79.

Effect on event polling

If a connector uses guaranteed-event delivery by setting `ContainedManagedEvents` to `JMS`, it behaves slightly differently from a connector that does not use this feature. To provide container-managed events, the connector framework takes the following steps to poll the event store:

1. Start a JMS transaction.
2. Read a JMS message from the event store.

   The event store is implemented as a JMS source queue. The JMS message contains an event record. The name of the JMS source queue is obtained from the `SourceQueue` connector configuration property.
3. Call the data handler to convert the event to a business object.

   The connector framework calls the data handler that has been configured with the properties in Table 10 on page 34.
4. When WebSphere MQ Integrator Broker is the integration broker, convert the business object to a message based on the configured wire format (XML).
5. Send the resulting message to the JMS destination queue.
   If you are using the WebSphere ICS integration broker, the message sent to the JMS destination queue is the business object. If you are using WebSphere MQ Integrator broker, the message sent to the JMS destination queue is an XML message (which the data handler generated).
6. Commit the JMS transaction.

   When the JMS transaction commits, the message is written to the JMS destination queue and removed from the JMS source queue in the same transaction.
7. Repeat step 1 through 6 in a loop. The `PollQuantity` connector property determines the number of repetitions in this loop.

Important: A connector that sets the `ContainerManagedEvents` property is set to `JMS` does *not* call the `pollForEvents()` method to perform event polling. If the connector's base class includes a `pollForEvents()` method, this method is *not* invoked.

Guaranteed-event-delivery for connectors with non-JMS event stores

If the JMS-enabled connector uses a non-JMS solution to implement its event store (such as a JDBC event table, Email mailbox, or flat files), the connector framework can use duplicate event elimination to ensure that duplicate events do not occur. This section provides the following information about use of the guaranteed-event delivery feature with a JMS-enabled connector that has a non-JMS event store:

- "Enabling the feature for connectors with non-JMS event stores"
- "Effect on event polling"

Enabling the feature for connectors with non-JMS event stores: To enable the guaranteed-event delivery feature for a JMS-enabled connector that has a non-JMS event store, you must set the connector configuration properties to values shown in Table 11.

Table 11. Guaranteed-event-delivery connector properties for a connector with a non-JMS event store

Connector property	Value
`DeliveryTransport`	`JMS`
`DuplicateEventElimination`	`true`

Table 11. Guaranteed-event-delivery connector properties for a connector with a non-JMS event store (continued)

Connector property	Value
`MonitorQueue`	Name of the JMS monitor queue, in which the connector framework stores the ObjectEventId of processed business objects

If you configure a connector to use guaranteed-event delivery, you must set the connector properties as described in Table 11. To set these connector configuration properties, use the Connector Configurator tool. It displays these connector properties on its Standard Properties tab. For information on Connector Configurator, see Appendix B, "Connector Configurator," on page 79.

Effect on event polling: If a connector uses guaranteed-event delivery by setting `DuplicateEventElimination` to `true`, it behaves slightly differently from a connector that does not use this feature. To provide the duplicate event elimination, the connector framework uses a JMS monitor queue to track a business object. The name of the JMS monitor queue is obtained from the `MonitorQueue` connector configuration property.

After the connector framework receives the business object from the application-specific component (through a call to `gotApplEvent()` in the `pollForEvents()` method), it must determine if the current business object (received from `gotApplEvents()`) represents a duplicate event. To make this determination, the connector framework retrieves the business object from the JMS monitor queue and compares its ObjectEventId with the ObjectEventId of the current business object:

- If these two ObjectEventIds are the same, the current business object represents a duplicate event. In this case, the connector framework ignores the event that the current business object represents; it does *not* send this event to the integration broker.
- If these ObjectEventIds are *not* the same, the business object does *not* represent a duplicate event. In this case, the connector framework copies the current business object to the JMS monitor queue and then delivers it to the JMS delivery queue, all as part of the same JMS transaction. The name of the JMS delivery queue is obtained from the `DeliveryQueue` connector configuration property. Control returns to the connector's `pollForEvents()` method, after the call to the `gotApplEvent()` method.

For a JMS-enabled connector to support duplicate event elimination, you must make sure that the connector's `pollForEvents()` method includes the following steps:

- When you create a business object from an event record retrieved from the non-JMS event store, save the event record's unique event identifier as the business object's ObjectEventId attribute.

  The application generates this event identifier to uniquely identify the event record in the event store. If the connector goes down after the event has been sent to the integration broker but before this event record's status can be changed, this event record remains in the event store with an In-Progress status. When the connector comes back up, it should recover any In-Progress events. When the connector resumes polling, it generates a business object for the event record that still remains in the event store. However, because both the business object that was already sent and the new one have the same event record as their ObjectEventIds, the connector framework can recognize the new business object as a duplicate and not send it to the integration broker.

- During connector recovery, make sure that you process In-Progress events *before* the connector begins polling for new events.

  Unless the connector changes any In-Progress events to Ready-for-Poll status when it starts up, the polling method does not pick up the event record for reprocessing.

Queue uniform resource identifiers (URIs)

The URI for a queue begins with the sequence `queue://` followed by:

- The name of the queue manager on which the queue resides
- Another /
- The name of the queue
- Optionally, a list of name-value pairs to set the remaining queue properties.

For example, the following URI connects to queue `IN` on queue manager `<queue.manager.name>` and causes all messages to be sent as WebSphere MQ messages with priority 5.

```
queue://<queue.manager.name>/WC_MQCONN.IN?targetClient=1&priority=5
```

The following table shows property names for queue URIs.

Table 12. WebSphere Commerce-specific connector property names for queue URIs

Property name	Description	Values
`expiry`	Lifetime of the message in milliseconds.	0 = unlimited. positive integers = timeout (in ms).
`priority`	Priority of the message.	0-9, where 1 is the highest priority. A value of -1 means that the property should be determined by the configuration of the queue. A value of -2 specifies that the connector can use its own default value.
`persistence`	Whether the message should be 'hardened' to disk.	1 = non-persistent 2 = persistent A value of -1 means that the property should be determined by the configuration of the queue. A value of -2 specifies that the connector can use its own default value.
`CCSID`	Character set of the destination.	Integers - valid values listed in base WebSphere MQ documentation.
`targetClient`	Whether the receiving application is JMS compliant or not.	0 = JMS (MQRFH2 header) 1 = MQ (MQMD header only)

Table 12. WebSphere Commerce-specific connector property names for queue URIs (continued)

Property name	Description	Values
`encoding`	How to represent numeric fields.	An integer value as described in the base WebSphere MQ documentation.

The adapter has no control of the character set (CCSID) or encoding attributes of data in MQMessages. Because data conversion is applied as the data is retrieved from or delivered to the message buffer, the adapter relies upon the IBM WebSphere MQ implementation of JMS to convert data (see the IBM WebSphere MQ Java client library documentation). Accordingly, these conversions should be bi-directionally equivalent to those performed by the native WebSphere MQ API using option `MQGMO_CONVERT`.

The adapter has no control over differences or failures in the conversion process. The adapter can retrieve message data of any CCSID or encoding supported by WebSphere MQ without additional modifications. To deliver a message of a specific `CCSID` or encoding, the output queue must be a fully-qualified URI and specify values for `CCSID` and `encoding`. The adapter passes this information to WebSphere MQ, which (via the JMS API) uses the information when encoding data for MQMessage delivery.

Often, lack of support for CCSID and encoding can be resolved by downloading the most recent version of the IBM WebSphere MQ Java client library from IBM's web site. If problems specific to CCSID and encoding persist, contact Technical Support to discuss the possibility of using an alternate Java Virtual Machine to run the adapter.

Meta-objects configuration

The connector uses meta-object entries to determine which business object to associate with a message. The type of business object and verb used in processing an event message is based on the `FORMAT` field contained in the WebSphere MQ message header. You construct a meta-object attribute to store the business object name and verb to associate with the WebSphere MQ message header `FORMAT` field text. Meta-object attributes also contain message processing guidelines.

When a message is retrieved from the input queue, the connector looks up the business object name associated with the `FORMAT` text field. The message, along with the business object name, is then passed to the data handler. If a business object is successfully populated with message content, the connector checks to see if it is subscribed, and then delivers it to the integration broker using the `gotApplEvents()` method.

The connector can recognize and read two kinds of meta-objects:
- a static connector meta-object
- a dynamic child meta-object

The attribute values of the dynamic child meta-object duplicate and override those of the static meta-object.

When deciding upon which meta-object will work best for your implementation, consider the following:

- **Static meta-object**
  - Useful if all meta-data for different messages is fixed and can be specified at configuration time.
  - Limits you to specifying values by business-object type. For example, all `Customer`-type objects must be sent to the same destination.
- **Dynamic meta-object**
  - Gives business processes access to information in message headers
  - Allows business processes to change processing of messages at run-time, regardless of business type. For example, a dynamic meta-object would allow you to specify a different destination for every `Customer`-type object sent to the adapter.
  - Requires changes to the structure of supported business objects—such changes may require changes to maps and business processes.
  - Requires changes to custom data handlers.

Meta-object properties

Table 13 provides a complete list of properties supported in meta-objects. Refer to these properties when implementing meta-objects. Your meta object should have one or more of the properties shown in Table 13.

Not all properties are available in both static and dynamic meta-objects. Nor are all properties are readable from or writable to the message header. See the appropriate sections on event and request processing in Chapter 1, "Overview," on page 1, to determine how a specific property is interpreted and used by the connector.

Table 13. WebSphere Commerce adapter meta-object properties

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
`CollaborationName`	Yes	No	The `CollaborationName` must be specified in the application specific text of the attribute for the business object/verb combination. For example, if a user expects to handle synchronous event delivery for the business object Customer with the Create verb, the static metadata object must contain an attribute named `Customer_Create`. The `Customer_Create` attribute must contain application specific text that includes a name-value pair. For example, `CollaborationName=MyCustomerProcessingCollab`. See the "Overview of creating static meta-objects" on page 42 section for syntax details. Failure to do this will result in run-time errors when the connector attempts to synchronously process a request involving the Customer business object. **Note:** This property is only available for synchronous requests.

Table 13. WebSphere Commerce adapter meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
`DataHandlerConfigMO`	Yes	Yes	Meta-object passed to data handler to provide configuration information. If specified in the static meta-object, this will override the value specified in the `DataHandlerConfigMO` connector property. Use this meta-object property when different data handlers are required for processing different business object types. Use the dynamic child meta-object for request processing when the data format may be dependent on the actual business data. The specified business object must be supported by the connector agent. See the description in Appendix B, "Connector Configurator," on page 79.
`DataHandlerMimeType`	Yes	Yes	Allows you to request a data handler based on a particular MIME type. If specified in the meta-object, this will override the value specified in the `DataHandlerMimeType` connector property. Use this meta-object property when different data handlers are required for processing different business object types. Use the dynamic child meta-object for request processing when the data format might be dependent on the actual business data. The business object specified in `DataHandlerConfigMO` should have an attribute that corresponds to the value of this property. See the description in Appendix B, "Connector Configurator," on page 79.
`DataHandlerClassName`	Yes	Yes	See the description in Appendix B, "Connector Configurator," on page 79.
`InputFormat`	Yes	Yes	Format or type of inbound (event) message to associate with the given business object. This value helps identify the message content and is specified by the application that generated the message. When a message is retrieved and is in this format, it is converted to the given business object, if possible. If this format is not specified for a business object, the connector does not handle subscription deliveries for the given business object. Do not set this property using default meta-object conversion properties; its value is used to match incoming messages to business objects. The field that the connector considers as defining the format in the message can be user-defined via the connector-specific property MessageFormatProperty.
`OutputFormat`	Yes	Yes	Format to be populated in outbound messages. If the `OutputFormat` is not specified, the input format is used, if available.

Table 13. WebSphere Commerce adapter meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
InputQueue	Yes	Yes	The input queue that the connector polls to detect new messages. This property is used to match incoming messages to business objects only. Do not set this property using default conversion properties; its value is used to match incoming messages to business objects. **Note:** The InputQueue property in the connector-specific properties defines which queues the adapter polls. This is the only property that the adapter uses to determine which queues to poll. In the static MO, the InputQueue property and the InputFormat property can serve as criteria for the adapter to map a given message to a specific business object. To implement this feature, you would use connector-specific properties to configure multiple input destinations and optionally map different data handlers to each one based on the input formats of incoming messages. For information, see "Overview of mapping data handlers to input queues" on page 43
OutputQueue	Yes	Yes	Queue to which messages derived from the given business object are delivered.
ResponseTimeout	Yes	Yes	Indicates the length of time in milliseconds to wait before timing out when waiting for a response in synchronous request processing. The connector returns SUCCESS immediately without waiting for a response if this is left undefined or with a value less than zero.
TimeoutFatal	Yes	Yes	Used in synchronous request processing to trigger the connector to return an error message if a response is not received. If this property is True, the connector returns APPRESPONSETIMEOUT to the broker when a response is not received within the time specified by ResponseTimeout. If this property is undefined or set to False, then on a response timeout the connector fails the request but does not terminate. Default = False.
DataEncoding	Yes	Yes	DataEncoding is the encoding to be used to read and write messages. If this property is not specified in the static meta-object, the connector tries to read the messages without using any specific encoding. DataEncoding defined in a dynamic child meta-object overrides the value defined in the static meta-object. The default value is Text. The format for the value of this attribute is messageType[:*enc*]. I.e., Text:ISO8859_1, Text:UnicodeLittle, Text, or Binary.This property is related internally to the InputFormat property: specify one and only one DataEncoding per InputFormat.
Below are fields mapping specifically to the JMS message header. For specific explanations, interpretation of values, and more, see the JMS API specification. JMS providers may interpret some fields differently so also check your JMS provider documentation for any deviations.			
ReplyToQueue		Yes	Queue to which a response message for a request is to be sent.

Table 13. WebSphere Commerce adapter meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
Type		Yes	Type of message. Generally user-definable, depending on JMS provider.
MessageID		Yes	Unique ID for message (JMS provider specific).
CorrelationID	Yes	Yes	Used in response messages to indicate the ID of the request message that initiated this response.
Delivery Mode	Yes	Yes	Specifies whether the message is persisted or not in the MOM system. Acceptable values: 1=non-persistent 2=persistent Other values, depending on the JMS provider, may be available.
Priority		Yes	Numeric priority of message. Acceptable values: 0 through 9 inclusive (low to high priority).
Destination		Yes	Current or last (if removed) location of message in MOM system.
Expiration		Yes	Time-to-live of message. If specified as zero, expiration is set to zero. Zero indicates to the JMS provider that the message does not expire.
Redelivered		Yes	Indicates that the JMS provider most likely attempted to deliver the message to the client earlier but receipt was not acknowledged.
Timestamp		Yes	Time message was handed off to JMS provider.
UserID		Yes	Identity of the user sending the message.
AppID		Yes	Identity of the application sending the message.
DeliveryCount		Yes	Number of delivery attempts.
GroupID		Yes	Identity of the message group.
GroupSeq		Yes	Sequence of this message in the message group specified in GroupID.
JMSProperties		Yes	See “JMS properties” on page 49.

Overview of creating static meta-objects

The WebSphere MQ adapter configuration meta-object consists of a list of conversion properties defined for different business objects. The connector supports, at most, one static meta-object at any given time. You implement a static meta-object by specifying its name for connector property ConfigurationMetaObject.

The structure of the static meta-object is such that each attribute represents a single business object and verb combination and all the meta-data associated with processing that object. The name of each attribute should be the name of the business object type and verb separated by an underscore, such as `Customer_Create`. The attribute application-specific information should consist of one or more semicolon-delimited name-value pairs representing the meta-data properties you want to specify for this unique object-verb combination.

Table 14. Static meta-object structure

Attribute name	Application-specific text
`<business object type>_<verb>`	`property=value;property=value;...`
`<business object type>_<verb>`	`property=value;property=value;...`

For example, consider the following meta-object:

Table 15. Sample static meta-object structure

Attribute name	Application-specific information
`Customer_Create`	`OutputFormat=CUST;OutputDestination=QueueA`
`Customer_Update`	`OutputFormat=CUST;OutputDestination=QueueB`
`Order_Create`	`OutputFormat=ORDER;OutputDestination=QueueC`

The meta-object in this example informs the connector that when it receives a request business object of type `Customer` with verb `Create`, to convert it to a message with format `CUST` and then to place it in destination `QueueA`. If the customer object instead had verb `Update`, the message would be placed in `QueueB`. If the object type was `Order` and had verb `Create`, the connector would convert and deliver it with format `ORDER` to `QueueC`. Any other business object passed to the connector would be treated as unsubscribed.

Optionally, you may name one attribute `Default` and assign to it one or more properties in the ASI. For all attributes contained in the meta-object, the properties of the default attribute are combined with those of the specific object-verb attributes. This is useful when you have one or more properties to apply universally (regardless of object-verb combination). In the following example, the connector would consider object-verb combinations of `Customer_Create` and `Order_Create` as having `OutputDestination=QueueA` in addition to their individual meta-data properties:

Table 16. Example static meta-object structure

Attribute name	Application-specific information
`Default`	`OutputDestination=QueueA`
`Customer_Update`	`OutputFormat=CUST`
`Order_Create`	`OutputFormat=ORDER`

Table 13 on page 39 describes the properties that you can specify as application-specific information in the static meta-object.

Note: If a static meta object is not specified, the connector is unable to map a given message format to a specific business object type during polling. When this is the case, the connector passes the message text to the configured data handler without specifying a business object. If the data handler cannot create a business object based on the text alone, the connector reports an error indicating that this message format is unrecognized.

Overview of mapping data handlers to input queues

You can use the InputQueue property in the application-specific information of the static meta-object to associate a data handler with an input queue. This feature is useful when dealing with multiple trading partners who have different formats and conversion requirements.

Steps for mapping data handlers to input queues

To map a data handler to an InputQueue, do the following:

1. Use connector-specific properties (see “InputQueue” on page 31) to configure one or more input queues.

2. For each input queue in the static meta-object, specify the queue manager and input queue name as well as data handler class name and mime type in the application-specific information.

For example, the following attribute in a static meta-object associates a data handler with an InputQueue named CompReceipts:

```
[Attribute]
Name = Cust_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputQueue=://queue.manager/CompReceipts;DataHandlerClassName=
com.crossworlds.DataHandlers.WBIMB.disposition_notification;DataHandlerMimeType=
message/
disposition_notification
IsRequiredServerBound = false
[End]
```

Overloading input formats

When retrieving a message, the connector normally matches the input format to one specific business object and verb combination. The adapter then passes the business object name and the contents of the message to the data handler. This allows the data handler to verify that the message contents correspond to the business object that the user expects.

If, however, the same input format is defined for more than one business object, the connector will be unable to determine which business object the data represents before passing it to the data handler. In such cases, the adapter passes the message contents only to the data handler and then looks up conversion properties based on the business object that is generated. Accordingly, the data handler must determine the business object based on the message content alone.

If the verb on the generated business object is not set, the connector searches for conversion properties defined for this business object with any verb. If only one set of conversion properties is found, the connector assigns the specified verb. If more properties are found, the connector fails the message because it is unable to distinguish among the verbs.

An example meta-object

The meta-object shown below configures the connector to convert Customer business objects using the verb Create.

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

   [Attribute]
   Name = Default
   Type = String
   MaxLength = 1
   IsKey = true
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = OutputQueue=queue://<Queue Manager
    Name>/WCS_Serial_Inbound?targetClient=1;
     OutputFormat=MQSTR
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = WCS_Create_WCS_Customer_Create
   Type = String
   MaxLength = 255
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = OutputQueue=queue://<Queue Manager
    Name>/WCS_Serial_Inbound?targetClient=1;
     OutputFormat=MQSTR
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = WCS_Report_NC_PurchaseOrder_Create
   Type = String
   MaxLength = 255
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = InputFormat=MQSTR
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = ObjectEventId
   Type = String
   MaxLength = 255
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Verb]
   Name = Create
   [End]

   [Verb]
   Name = Delete
   [End]

   [Verb]
   Name = Retrieve
   [End]

   [Verb]
   Name = Update
   [End]
[End]
```

Overview of creating dynamic child meta-objects

If it is difficult or unfeasible to specify the necessary metadata through a static meta-object, the connector can optionally accept meta-data delivered at run-time for each business object instance.

Dynamic meta-objects allow you to change the meta-data used by the connector to process a business object on a per-request basis during request processing, and to retrieve information about an event message during event processing.

The connector recognizes and reads conversion properties from a dynamic meta-object that is added as a child to the top-level business object passed to the connector. The attribute values of the dynamic child meta-object duplicate the conversion properties that you can specify via the static meta-object that is used to configure the connector.

Since dynamic child meta object properties override those found in static meta-objects, if you specify a dynamic child meta-object, you need not include a connector property that specifies the static meta-object. Accordingly, you can use a dynamic child meta-object independently of the static meta-object and vice-versa.

Table 13 on page 39 describes the properties that you can specify as application-specific information in the dynamic meta-object.

The structure of the dynamic meta-object is such that each attribute represents a single metadata property and value: `meta-object property name =meta-object property value`

Note: All standard IBM WebSphere data handlers are designed to ignore this dynamic meta-object attribute by recognizing the `cw_mo_` tag. You must do the same when developing custom data handlers for use with the adapter.

Population of the dynamic child meta-object during polling

In order to provide collaborations with more information regarding messages retrieved during polling, the connector populates specific attributes of the dynamic meta-object, if already defined for the business object created.

Table Table 17 shows how a dynamic child meta-object might be structured for polling.

Table 17. Dynamic child meta-object structure for polling

Property name	Sample value
InputFormat	`CUST_IN`
InputQueue	`MYInputQueue`
OutputFormat	`CxIgnore`
OutputQueue	`CxIgnore`
ResponseTimeout	`CxIgnore`
TimeoutFatal	`CxIgnore`

As shown in Table 17, you can define additional attributes, `Input_Format` and `InputQueue`, in a dynamic child meta-object. The `Input_Format` is populated with the format of the message retrieved, while the `InputQueue` attribute contains the name of the queue from which a given message has been retrieved. If these properties are not defined in the child meta-object, they will not be populated.

Example scenario:

- The connector retrieves a message with the format `CUST_IN` from the queue `MyInputQueue`.
- The connector converts this message to a Customer business object and checks the application-specific text to determine if a meta-object is defined.
- If so, the connector creates an instance of this meta-object and populates the `InputQueue` and `InputFormat` attributes accordingly, then publishes the business object to available collaborations.

Example dynamic child meta-object

```
[BusinessObjectDefinition]
Name = MO_Sample_Config
Version = 1.0.0

   [Attribute]
   Name = OutputFormat
```

```
Type = String
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
DefaultValue = CUST
IsRequiredServerBound = false
[End]
[Attribute]
Name = OutputQueue
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = OUT
IsRequiredServerBound = false
[End]
[Attribute]
Name = ResponseTimeout
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = -1
IsRequiredServerBound = false
[End]
[Attribute]
Name = TimeoutFatal
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = InputFormat
Type = String
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = InputQueue
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Verb]
```

```
   Name = Create
   [End]

   [Verb]
   Name = Delete
   [End]

   [Verb]
   Name = Retrieve
   [End]

   [Verb]
   Name = Update
   [End]
[End]
[BusinessObjectDefinition]
Name = Customer
Version = 1.0.0
AppSpecificInfo = cw_mo_conn=MyConfig

   [Attribute]
   Name = FirstName
   Type = String
   MaxLength = 1
   IsKey = true
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = LastName
   Type = String
   MaxLength = 1
   IsKey = true
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = Telephone
   Type = String
   MaxLength = 1
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = MyConfig
   Type = MO_Sample_Config
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 1
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = ObjectEventId
   Type = String
   MaxLength = 255
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Verb]
   Name = Create
   [End]

   [Verb]
   Name = Delete
   [End]

   [Verb]
   Name = Retrieve
   [End]

   [Verb]
   Name = Update
   [End]
[End]
```

JMS headers and dynamic child meta-object attributes

You can add attributes to a dynamic meta-object to gain more information about, and more control over, the message transport. This section describes these attributes and how they affect event notification and request processing.

JMS properties: Unlike other attributes in the dynamic meta-object, `JMSProperties` must define a single-cardinality child object. Every attribute in this child object must define a single property to be read/written in the variable portion of the JMS message header as follows:

1. The name of the attribute has no semantic value.
2. The type of the attribute should always be `String` regardless of the JMS property type.
3. The application-specific information of the attribute must contain two name-value pairs defining the name and format of the JMS message property to which the attribute maps. The name is user-definable. The value type must be one of the following:
   - Boolean
   - String
   - Int
   - Float
   - Double
   - Long
   - Short
   - Byte

The table below shows application-specific information properties that you must define for attributes in the `JMSProperties` object.

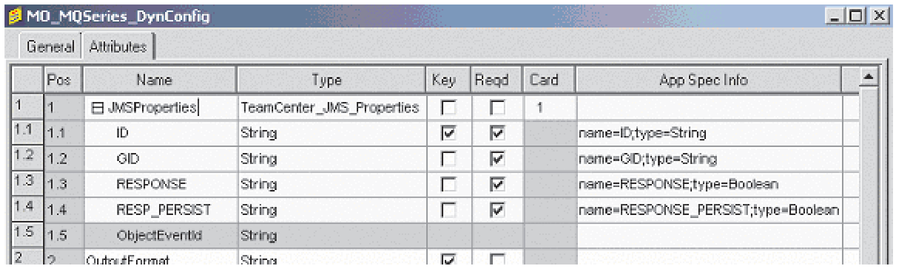
Table 18. Application-specific information for JMS property attributes

Attribute	Possible values	ASI	Comments
Name	Any valid JMS property name (valid = compatible with type defined in ASI)	`name=<JMS property name>;type=<JMS property type>`	Some vendors reserve certain properties to provide extended functionality. In general, users should not define custom properties that begin with `JMS` unless they are seeking access to these vendor-specific features.
Type	`String`	type=<*see comments*>	This is the type of the JMS property. The JMS API provides a number of methods for setting values in the JMS Message: `setIntProperty`, `setLongProperty`, `setStringProperty`, etc. The type of the JMS property specified here dictates which of these methods is used for setting the property value in the message.

In the example below, a JMSProperties child object is defined for the Customer object to allow access to the user-defined fields of the message header:

```
Customer (ASI = cw_mo_conn=MetaData)
   |-- Id
   |-- FirstName
   |-- LastName
   |-- ContactInfo
   |-- MetaData
          |-- OutputFormat = CUST
          |-- OutputDestination = QueueA
          |-- JMSProperties
               |-- RoutingCode = 123 (ASI= name=RoutingCode;type=Int)
               |-- Dept = FD (ASI= name=RoutingDept;type=String)
```

To illustrate another example, Figure 4 shows attribute `JMSProperties` in the dynamic meta-object and definitions for four properties in the JMS message header: ID, GID, RESPONSE and RESPONSE_PERSIST. The application-specific information of the attributes defines the name and type of each. For example, attribute ID maps to JMS property `ID` of type `String`).



MO_MQSeries_DynConfig

General | Attributes

	Pos	Name	Type	Key	Reqd	Card	App Spec Info
1	1	⊟ JMSProperties	TeamCenter_JMS_Properties	☐	☐	1	
1.1	1.1	ID	String	☑	☑		name=ID;type=String
1.2	1.2	GID	String	☐	☑		name=GID;type=String
1.3	1.3	RESPONSE	String	☐	☑		name=RESPONSE;type=Boolean
1.4	1.4	RESP_PERSIST	String	☐	☑		name=RESPONSE_PERSIST;type=Boolean
1.5	1.5	ObjectEventId	String				
2	2	OutputFormat	String	☑	☐		

Figure 4. JMS properties attribute in a dynamic meta-object

Creating multiple instances of the adapter

Creating multiple instances of a connector is in many ways the same as creating a custom connector. You can set your system up to create and run multiple instances of a connector by following the steps below. You must:

- Create a new directory for the connector instance
- Make sure you have the requisite business object definitions
- Create a new connector definition file
- Create a new start-up script

Create a new directory

You must create a connector directory for each connector instance. This connector directory should be named:

```
ProductDir\connectors\connectorInstance
```

where `connectorInstance` uniquely identifies the connector instance.

If the connector has any connector-specific meta-objects, you must create a meta-object for the connector instance. If you save the meta-object as a file, create this directory and store the file here:

```
ProductDir\repository\connectorInstance
```

Create business object definitions

If the business object definitions for each connector instance do not already exist within the project, you must create them.

1. If you need to modify business object definitions that are associated with the initial connector, copy the appropriate files and use Business Object Designer to import them. You can copy any of the files for the initial connector. Just rename them if you make changes to them.
2. Files for the initial connector should reside in the following directory:

   ```
   ProductDir\repository\initialConnectorInstance
   ```

   Any additional files you create should be in the appropriate `connectorInstance` subdirectory of `ProductDir\repository`.

Create a connector definition

You create a configuration file (connector definition) for the connector instance in Connector Configurator. To do so:

1. Copy the initial connector's configuration file (connector definition) and rename it.

2. Make sure each connector instance correctly lists its supported business objects (and any associated meta-objects).
3. Customize any connector properties as appropriate.

Create a start-up script

To create a startup script:

1. Copy the initial connector's startup script and name it to include the name of the connector directory:

   dirname
2. Put this startup script in the connector directory you created in "Create a new directory" on page 51.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

You can now run both instances of the connector on your integration server at the same time.

For more information on creating custom connectors, refer to the *Connector Development Guide for C++ or for Java*.

Startup file configuration

Before you start the adapter for WebSphere Commerce, you must configure the startup file.

Windows

To complete the configuration of the adapter for Windows platforms, you must modify the startup file (either `start_WebSphereCommerceAdapter.bat` or `start_WebSphereCommerce.bat`,whichever is provided with your adapter):

1. Open the `start_WebSphereCommerceAdapter.bat` file.
2. Scroll to the section beginning with "`Set the directory containing your WebSphere MQ Java client libraries`," and specify the location of your WebSphere MQ Java client libraries.

UNIX

To complete the configuration of the adapter for UNIX platforms, you must modify the startup file (either `start_WebSphereCommerceAdapter.sh` or `start_WebSphereCommerce.sh`,whichever is provided with your adapter):

1. Open the `start_WebSphereCommerceAdapter.sh` file.
2. Scroll to the section beginning with "`Set the directory containing your WebSphere MQ Java client libraries`," and specify the location of your WebSphere MQ Java client libraries.

Starting the connector

A connector must be explicitly started using its **connector start-up script**. The startup script should reside in the connector's runtime directory:

```
ProductDir\connectors\connName
```

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 19 shows.

Table 19. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	`connector_manager_connName`
Windows	`start_connName.bat`

You can invoke the connector startup script in any of the following ways:

- On Windows systems, from the **Start** menu

  Select **Programs>IBM WebSphere Business Integration Adapters>Adapters>Connectors**. By default, the program name is "IBM WebSphere Business Integration Adapters". However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.
- From the command line
  - On Windows systems:
    ```
    start_connName connName brokerName [-cconfigFile ]
    ```
  - On UNIX-based systems:
    ```
    connector_manager_connName -start
    ```

  where *connName* is the name of the connector and *brokerName* identifies your integration broker, as follows:
  - For WebSphere InterChange Server, specify for *brokerName* the name of the ICS instance.
  - For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for *brokerName* a string that identifies the broker.

  Note: For a WebSphere message broker or WebSphere Application Server on a Windows system, you *must* include the `-c` option followed by the name of the connector configuration file. For ICS, the `-c` is optional.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

For more information on how to start a connector, including the command-line startup options, refer to one of the following documents:

- For WebSphere InterChange Server, refer to the *System Administration Guide*.
- For WebSphere message brokers, refer to *Implementing Adapters with WebSphere Message Brokers*.
- For WebSphere Application Server, refer to *Implementing Adapters with WebSphere Application Server*.

Stopping the connector

The way to stop a connector depends on the way that the connector was started, as follows:

- If you started the connector from the command line, with its connector startup script:
  - On Windows systems, invoking the startup script creates a separate "console" window for the connector. In this window, type "Q" and press Enter to stop the connector.
  - On UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:

    ```
    connector_manager_connName -stop
    ```

    where *connName* is the name of the connector.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.

Chapter 3. Working with business objects

This chapter describes how the adapter processes business objects and describes the assumptions that the connector makes. You can use this information as a guide to implement new business objects.

This chapter also describes how the adapter processes business objects and describes the assumptions the connector makes. You can use this information as a guide to implementing new business objects.

The adapter comes with some example business objects only. The systems integrator, consultant, or customer must build your specific business objects.

The sample business objects are contained under the /samples directory in your delivered adapter package.

The adapter is metadata-driven. In this context, metadata is data about the application, which is stored in a business object definition and which helps the adapter interact with an application. A metadata-driven connector handles each business object that it supports based on metadata encoded in the business object definition rather than on instructions hard-coded in the connector.

Business object metadata includes the structure of a business object, the settings of its attribute properties, and the content of its application-specific text. Because the connector is metadata-driven, it can handle new or modified business objects without requiring modifications to the adapter code. However, the adapter's configured data handler makes assumptions about the structure of its business objects, object cardinality, the format of the application-specific text, and the database representation of the business object. Therefore, when you create or modify a business object for the adapter, your modifications must conform to the rules the adapter is designed to follow, or it cannot process new or modified business objects correctly.

Business object structure

After installing the adapter, you must create business objects.

The adapter for WebSphere Commerce retrieves WebSphere MQ messages from a queue and attempts to populate business objects (defined by the meta-object) with the business data contained in the message.

The business data in the MQ Series messages exchanged by the adapter is contained within XML documents. The adapter uses an XML data handler to convert the data from the XML documents into business objects, and from business objects into XML documents.

The structure of the business object definitions must conform to the requirements of the XML data handler. For information about these requirements, and for instructions on how to use XML DTDs and the Edifecs SpecBuilder utility to translate a DTD into a business object definition, refer to the IBM WebSphere Business Integration *Data Handler for XML*.

To see the properties and structure used in a typical Customer_Create business object definition for WebSphere Commerce, open the file /connector/WebSphereCommerce/samples/WC_BODefinition.in.

Error handling

All error messages generated by the adapter are stored in a message file named `WebSphereCommerce.txt`. (The name of the file is determined by the `LogFileName` standard connector configuration property.) Each error has an error number followed by the error message:

```
Message number
Message text
```

The adapter handles specific errors as described in the following sections.

Application timeout

The error message BON_APPRESPONSETIMEOUT is returned when:

- The adapter cannot establish a connection to WebSphere MQ during message retrieval.
- The connector successfully converts a business object to a message but cannot deliver it the outgoing queue due to connection loss.
- The adapter issues a message but times out waiting for a response for a business object with conversion property `TimeoutFatal` equal to `True`.
- The adapter receives a response message with a return code equal to APP_RESPONSE_TIMEOUT or UNABLE_TO_LOGIN.

Unsubscribed business object

If the adapter retrieves a message that is associated with an unsubscribed business object, the adapter delivers a message to the queue specified by the `UnsubscribedQueue` property.

Note: If the `UnsubscribedQueue` is not defined, unsubscribed messages will be discarded.

Data handler conversion

If the data handler fails to convert a message to a business object, or if a processing error occurs that is specific to the business object (as opposed to WebSphere MQ), the message is delivered to the queue specified by `ErrorQueue`. If the `ErrorQueue` is not defined, messages that cannot be processed due to errors will be discarded.

If the data handler fails to convert a business object to a message, BON_FAIL is returned.

Tracing

Tracing is an optional debugging feature you can turn on to closely follow adapter behavior. Trace messages, by default, are written to STDOUT. See the connector configuration properties for more on configuring trace messages. For more information on tracing, including how to enable and set it, see the *Connector Development Guide*.

What follows is recommended content for trace messages.

Level 0	This level is used for trace messages that identify the adapter version.
Level 1	Use this level for trace messages that provide key information on each business object processed or record each time a polling thread detects a new message in an input queue.
Level 2	Use this level for trace messages that log each time a business object is posted to InterChange Server, either from `gotApplEvent()` or `executeCollaboration()`.
Level 3	Use this level for trace messages that provide information regarding message-to-business-object and business-object-to-message conversions or provide information about the delivery of the message to the output queue.
Level 4	Use this level for trace messages that identify when the adapter enters or exits a function.
Level 5	Use this level for trace messages that indicate adapter initialization, represent statements executed in the application, indicate whenever a message is taken off of or put onto a queue, or record business object dumps.

Chapter 4. Troubleshooting

This chapter describes problems that you may encounter when starting up or running the adapter.

Startup problems

Problem	Potential solution / explanation
The adapter shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/jms/JMSException...`	Adapter cannot find file `jms.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The adapter shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: com/ibm/mq/jms/MQConnectionFactory...`	Adapter cannot find file `com.ibm.mqjms.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The adapter shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/naming/Referenceable...`	Adapter cannot find file `jndi.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The adapter shuts down unexpectedly during initialization and the following exception is reported: `java.lang.UnsatisfiedLinkError: no mqjbnd01 in shared library path`	Adapter cannot find a required run-time library (`mqjbnd01.dll` [NT] or `libmqjbnd01.so` [Solaris]) from the IBM WebSphere MQ Java client libraries. Ensure that your path includes the IBM WebSphere MQ Java client library folder.
The adapter reports `MQJMS2005: failed to create MQQueueManager for ':'`	Explicitly set values for the following properties: `HostName`, `Channel`, and `Port`.
The adapter reports that classes could not be found.	Verify that the following directories are included in the paths for the classes: `<MQSeries Install>\java]lib` `<install_path>\bin`
The adapter hangs on startup	Verify that the WebSphere MQ listener is running.

Event processing

Problem	Potential solution / explanation
The adapter delivers all messages with an MQRFH2 header.	To deliver messages with only the MQMD WebSphere MQ header, append `?targetClient=1` to the name of output queue URI. For example, if you output messages to queue `queue://my.queue.manager/OUT`, change the URI to `queue://my.queue.manager/OUT?targetClient=1`. See Chapter 2, "Installing and configuring the connector," on page 15, for more information.

Problem	Potential solution / explanation
The adapter truncates all message formats to 8-characters upon delivery regardless of how the format has been defined in the adapter meta-object.	This is a limitation of the WebSphere MQ MQMD message header and not the adapter.

Business object size limitation

There are fundamental limitations to the Java Virtual Machine upon which the InterChange Server runs.

The reasons for these recommendations are as follows:

- As business objects propagate through the ICS additional references are added to it, expanding the size of the initial business object as it progresses from ASBO to GBO.
- Additional overhead is required when transactional behavior is required, also adding to business object size. This overhead is required for persistence and flexibility of handling the varied business logic that is performed upon business objects within the ICS.

Note: Large objects for the JVM (> 20 MB) lead to excessive heap fragmentation and the inability to allocate enough memory, leading to `java.lang.OutOfMemory` errors, which cause the ICS to shut itself down.

Appendix A. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of WebSphere Business Integration adapters. The information covers connectors running on the following integration brokers:

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker, collectively referred to as the WebSphere Message Brokers (WMQI).
- WebSphere Application Server (WAS)

Not every connector makes use of all these standard properties. When you select an integration broker from Connector Configurator, you will see a list of the standard properties that you need to configure for your adapter running with that broker.

For information about properties specific to the connector, see the relevant adapter user guide.

Note: In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes and follow the conventions for each operating system.

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNameSpaceFormat

Deleted properties

- RestartCount

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

This section describes the standard configuration properties. For information on configuration properties specific to a connector, see its adapter user guide.

Using Connector Configurator

You configure connector properties from Connector Configurator, which you access from System Manager. For more information on using Connector Configurator, refer to the sections on Connector Configurator in this guide.

Note: Connector Configurator and System Manager run only on the Windows system. If you are running the connector on a UNIX system, you must have a Windows machine with these tools installed. To set connector properties for a connector that runs on UNIX, you must start up System Manager on the Windows machine, connect to the UNIX integration broker, and bring up Connector Configurator for the connector.

Setting and updating property values

The default length of a property field is 255 characters.

The connector uses the following order to determine a property's value (where the highest number overrides other values):

1. Default
2. Repository (only if WebSphere InterChange Server is the integration broker)
3. Local configuration file
4. Command line

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a run-time session, the property's **Update Method** determines how the change takes effect. There are four different update methods for standard connector properties:

- **Dynamic**
  The change takes effect immediately after it is saved in System Manager. If the connector is working in stand-alone mode (independently of System Manager), for example with one of the WebSphere message brokers, you can only change properties through the configuration file. In this case, a dynamic update is not possible.
- **Agent restart (ICS only)**
  The change takes effect only after you stop and restart the application-specific component.
- **Component restart**
  The change takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the application-specific component or the integration broker.
- **Server restart**
  The change takes effect only after you stop and restart the application-specific component and the integration broker.

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in Table 20 on page 63 below.

Summary of standard properties

Table 20 on page 63 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

You must set the values of some of these properties before running the connector. See the following section for an explanation of each property.

Note: In the ″Notes″column in Table 20 on page 63, the phrase ″Repository directory is REMOTE″ indicates that the broker is the InterChange Server. When the broker is WMQI or WAS, the repository directory is set to LOCAL

Table 20. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdminInQueue	Valid JMS queue name	*CONNECTORNAME* /ADMININQUEUE	Component restart	Delivery Transport is JMS
AdminOutQueue	Valid JMS queue name	*CONNECTORNAME*/ADMINOUTQUEUE	Component restart	Delivery Transport is JMS
AgentConnections	1-4	1	Component restart	Delivery Transport is MQ or IDL: Repository directory is <REMOTE> (broker is ICS)
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS, WMQI, WAS		Component restart	
CharacterEncoding	ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 **Note:** This is a subset of supported values.	ascii7	Component restart	
ConcurrentEventTriggeredFlows	1 to 32,767	1	Component restart	Repository directory is <REMOTE> (broker is ICS)
ContainerManagedEvents	No value or JMS	No value	Component restart	Delivery Transport is JMS
ControllerStoreAndForwardMode	true or false	true	Dynamic	Repository directory is <REMOTE> (broker is ICS)
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE> (broker is ICS)
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	MQ, IDL, or JMS	JMS	Component restart	If Repository directory is local, then value is JMS only

Table 20. Summary of standard configuration properties (continued)

Property name	**Possible values**	**Default value**	**Update method**	**Notes**
DuplicateEventElimination	`true or false`	`false`	Component restart	JMS transport only: Container Managed Events must be <NONE>
FaultQueue		*`CONNECTORNAME`*`/FAULTQUEUE`	Component restart	JMS transport only
jms.FactoryClassName	`CxCommon.Messaging.jms .IBMMQSeriesFactory or CxCommon.Messaging .jms.SonicMQFactory` or any Java class name	`CxCommon.Messaging. jms.IBMMQSeriesFactory`	Component restart	JMS transport only
jms.MessageBrokerName	If FactoryClassName is `IBM`, use `crossworlds.queue. manager`. If FactoryClassName is `Sonic`, use `localhost:2506`.	`crossworlds.queue.manager`	Component restart	JMS transport only
jms.NumConcurrentRequests	Positive integer	`10`	Component restart	JMS transport only
jms.Password	Any valid password		Component restart	JMS transport only
jms.UserName	Any valid name		Component restart	JMS transport only
JvmMaxHeapSize	Heap size in megabytes	`128m`	Component restart	Repository directory is <REMOTE> (broker is ICS)
JvmMaxNativeStackSize	Size of stack in kilobytes	`128k`	Component restart	Repository directory is <REMOTE> (broker is ICS)
JvmMinHeapSize	Heap size in megabytes	`1m`	Component restart	Repository directory is <REMOTE> (broker is ICS)
ListenerConcurrency	`1- 100`	`1`	Component restart	Delivery Transport must be `MQ`
Locale	`en_US`, `ja_JP`, `ko_KR`, `zh_CN`, `zh_TW`, `fr_FR`, `de_DE`, `it_IT`, `es_ES`, `pt_BR` **Note:** This is a subset of the supported locales.	`en_US`	Component restart	

Table 20. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
LogAtInterchangeEnd	`true` or `false`	`false`	Component restart	Repository Directory must be <REMOTE> (broker is ICS)
MaxEventCapacity	`1-2147483647`	`2147483647`	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
MessageFileName	Path or filename	`CONNECTORNAMEConnector.txt`	Component restart	
MonitorQueue	Any valid queue name	*`CONNECTORNAME`*`/MONITORQUEUE`	Component restart	JMS transport only: DuplicateEvent Elimination must be true
OADAutoRestartAgent	`true` or `false`	`false`	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
OADMaxNumRetry	A positive number	`1000`	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
OADRetryTimeInterval	A positive number in minutes	`10`	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
PollEndTime	`HH:MM`	`HH:MM`	Component restart	
PollFrequency	A positive integer in milliseconds `no` (to disable polling) `key` (to poll only when the letter `p` is entered in the connector's Command Prompt window)	`10000`	Dynamic	
PollQuantity	`1-500`	`1`	Agent restart	JMS transport only: Container Managed Events is specified
PollStartTime	`HH:MM`(HH is 0-23, MM is 0-59)	`HH:MM`	Component restart	

Table 20. Summary of standard configuration properties (continued)

Property name	**Possible values**	**Default value**	**Update method**	**Notes**
RepositoryDirectory	Location of metadata repository		Agent restart	For ICS: set to <REMOTE> For WebSphere MQ message brokers and WAS: set to C:\crossworlds\repository
RequestQueue	Valid JMS queue name	`CONNECTORNAME/REQUESTQUEUE`	Component restart	Delivery Transport is JMS
ResponseQueue	Valid JMS queue name	`CONNECTORNAME/RESPONSEQUEUE`	Component restart	Delivery Transport is JMS: required only if Repository directory is <REMOTE>
RestartRetryCount	`0-99`	`3`	Dynamic	
RestartRetryInterval	A sensible positive value in minutes: 1 - 2147483547	`1`	Dynamic	
RHF2MessageDomain	`mrm, xml`	`mrm`	Component restart	Only if Delivery Transport is JMS and WireFormat is CwXML.
SourceQueue	Valid WebSphere MQ name	`CONNECTORNAME/SOURCEQUEUE`	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue		`CONNECTORNAME/ SYNCHRONOUSREQUESTQUEUE`	Component restart	Delivery Transport is JMS
SynchronousRequestTimeout	0 - any number (millisecs)	`0`	Component restart	Delivery Transport is JMS
SynchronousResponseQueue		`CONNECTORNAME/ SYNCHRONOUSRESPONSEQUEUE`	Component restart	Delivery Transport is JMS
WireFormat	`CwXML, CwBO`	`CwXML`	Agent restart	CwXML if Repository Directory is not <REMOTE>: CwBO if Repository Directory is <REMOTE>
WsifSynchronousRequestTimeout	0 - any number (millisecs)	`0`	Component restart	WAS only
XMLNameSpaceFormat	`short, long`	`short`	Agent restart	WebSphere MQ message brokers and WAS only

Standard configuration properties

This section lists and defines each of the standard connector configuration properties.

AdminInQueue

The queue that is used by the integration broker to send administrative messages to the connector.

The default value is `CONNECTORNAME/ADMININQUEUE`.

AdminOutQueue

The queue that is used by the connector to send administrative messages to the integration broker.

The default value is `CONNECTORNAME/ADMINOUTQUEUE`.

AgentConnections

Applicable only if `RepositoryDirectory` is <REMOTE>.

The AgentConnections property controls the number of ORB (Object Request Broker) connections opened by `orb.init[]`.

The default value of this property is set to 1. You can change it as required.

AgentTraceLevel

Level of trace messages for the application-specific component. The default is `0`. The connector delivers all trace messages applicable at the tracing level set or lower.

ApplicationName

Name that uniquely identifies the connector's application. This name is used by the system administrator to monitor the WebSphere business integration system environment. This property must have a value before you can run the connector.

BrokerType

Identifies the integration broker type that you are using. The options are ICS, WebSphere message brokers (WMQI, WMQIB or WBIMB) or WAS.

CharacterEncoding

Specifies the character code set used to map from a character (such as a letter of the alphabet, a numeric representation, or a punctuation mark) to a numeric value.

Note: Java-based connectors do not use this property. A C++ connector currently uses the value `ascii7` for this property.

By default, a subset of supported character encodings only is displayed in the drop-down list. To add other supported values to the drop-down list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, see the sections on Connector Configurator in this guide.

ConcurrentEventTriggeredFlows

Applicable only if RepositoryDirectory is <REMOTE>.

Determines how many business objects can be concurrently processed by the connector for event delivery. Set the value of this attribute to the number of business objects you want concurrently mapped and delivered. For example, set the value of this property to 5 to cause five business objects to be concurrently processed. The default value is 1.

Setting this property to a value greater than 1 allows a connector for a source application to map multiple event business objects at the same time and deliver them to multiple collaboration instances simultaneously. This speeds delivery of business objects to the integration broker, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

To implement concurrent processing for an entire flow (from a source application to a destination application), you must:

- Configure the collaboration to use multiple threads by setting its Maximum number of concurrent events property high enough to use multiple threads.
- Ensure that the destination application's application-specific component can process requests concurrently. That is, it must be multi-threaded, or be able to use connector agent parallelism and be configured for multiple processes. Set the Parallel Process Degree configuration property to a value greater than 1.

The ConcurrentEventTriggeredFlows property has no effect on connector polling, which is single-threaded and performed serially.

ContainerManagedEvents

This property allows a JMS-enabled connector with a JMS event store to provide guaranteed event delivery, in which an event is removed from the source queue and placed on the destination queue as a single JMS transaction.

There is no default value.

When ContainerManagedEvents is set to JMS, you must configure the following properties to enable guaranteed event delivery:

- PollQuantity = 1 to 500
- SourceQueue = /SOURCEQUEUE

You must also configure a data handler with the MimeType, DHClass (data handler class), and DataHandlerConfigMOName (the meta-object name, which is optional) properties. To set those values, use the **Data Handler** tab in Connector Configurator.

Thes properties are adapter-specific, but **example** values are:

- MimeType = text\xml
- DHClass = com.crossworlds.DataHandlers.text.xml
- DataHandlerConfigMOName = MO_DataHandler_Default

The fields for these values in the Data Handler tab will be displayed only if you have set ContainerManagedEvents to JMS.

Note: When ContainerManagedEvents is set to `JMS`, the connector does *not* call its `pollForEvents()` method, thereby disabling that method's functionality.

This property only appears if the `DeliveryTransport` property is set to the value `JMS`.

ControllerStoreAndForwardMode

Applicable only if `RepositoryDirectory` is <REMOTE>.

Sets the behavior of the connector controller after it detects that the destination application-specific component is unavailable.

If this property is set to `true` and the destination application-specific component is unavailable when an event reaches ICS, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forwards the request to it.

However, if the destination application's application-specific component becomes unavailable **after** the connector controller forwards a service call request to it, the connector controller fails the request.

If this property is set to `false`, the connector controller begins failing all service call requests as soon as it detects that the destination application-specific component is unavailable.

The default is `true`.

ControllerTraceLevel

Applicable only if `RepositoryDirectory` is <REMOTE>.

Level of trace messages for the connector controller. The default is `0`.

DeliveryQueue

Applicable only if `DeliveryTransport` is JMS.

The queue that is used by the connector to send business objects to the integration broker.

The default value is `CONNECTORNAME/DELIVERYQUEUE`.

DeliveryTransport

Specifies the transport mechanism for the delivery of events. Possible values are `MQ` for WebSphere MQ, `IDL` for CORBA IIOP, or `JMS` for Java Messaging Service.

- If the `RepositoryDirectory` is remote, the value of the `DeliveryTransport` property can be MQ, IDL, or JMS, and the default is IDL.
- If the `RepositoryDirectory` is a local directory, the value may only be JMS.

The connector sends service call requests and administrative messages over CORBA IIOP if the value configured for the `DeliveryTransport` property is `MQ` or `IDL`.

WebSphere MQ and IDL

Use WebSphere MQ rather than IDL for event delivery transport, unless you must have only one product. WebSphere MQ offers the following advantages over IDL:

- Asynchronous communication:
  WebSphere MQ allows the application-specific component to poll and persistently store events even when the server is not available.
- Server side performance:
  WebSphere MQ provides faster performance on the server side. In optimized mode, WebSphere MQ stores only the pointer to an event in the repository database, while the actual event remains in the WebSphere MQ queue. This saves having to write potentially large events to the repository database.
- Agent side performance:
  WebSphere MQ provides faster performance on the application-specific component side. Using WebSphere MQ, the connector's polling thread picks up an event, places it in the connector's queue, then picks up the next event. This is faster than IDL, which requires the connector's polling thread to pick up an event, go over the network into the server process, store the event persistently in the repository database, then pick up the next event.

JMS

Enables communication between the connector and client connector framework using Java Messaging Service (JMS).

If you select JMS as the delivery transport, additional JMS properties such as `jms.MessageBrokerName`, `jms.FactoryClassName`, `jms.Password`, and `jms.UserName`, appear in Connector Configurator. The first two of these properties are required for this transport.

Important: There may be a memory limitation if you use the JMS transport mechanism for a connector in the following environment:

- AIX 5.0
- WebSphere MQ 5.3.0.1
- When ICS is the integration broker

In this environment, you may experience difficulty starting both the connector controller (on the server side) and the connector (on the client side) due to memory use within the WebSphere MQ client. If your installation uses less than 768M of process heap size, IBM recommends that you set:

- The `LDR_CNTRL` environment variable in the `CWSharedEnv.sh` script.

  This script resides in the `\bin` directory below the product directory. With a text editor, add the following line as the first line in the `CWSharedEnv.sh` script:

  ```
  export LDR_CNTRL=MAXDATA=0x30000000
  ```

  This line restricts heap memory usage to a maximum of 768 MB (3 segments * 256 MB). If the process memory grows more than this limit, page swapping can occur, which can adversely affect the performance of your system.
- The `IPCCBaseAddress` property to a value of 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

When you set this property to `true`, a JMS-enabled connector can ensure that duplicate events are not delivered to the delivery queue. To use this feature, the connector must have a unique event identifier set as the business object's **ObjectEventId** attribute in the application-specific code. This is done during connector development.

This property can also be set to `false`.

Note: When `DuplicateEventElimination` is set to `true`, you must also configure the `MonitorQueue` property to enable guaranteed event delivery.

FaultQueue

If the connector experiences an error while processing a message then the connector moves the message to the queue specified in this property, along with a status indicator and a description of the problem.

The default value is `CONNECTORNAME/FAULTQUEUE`.

JvmMaxHeapSize

The maximum heap size for the agent (in megabytes). This property is applicable only if the `RepositoryDirectory` value is <REMOTE>.

The default value is 128m.

JvmMaxNativeStackSize

The maximum native stack size for the agent (in kilobytes). This property is applicable only if the `RepositoryDirectory` value is <REMOTE>.

The default value is 128k.

JvmMinHeapSize

The minimum heap size for the agent (in megabytes). This property is applicable only if the `RepositoryDirectory` value is <REMOTE>.

The default value is 1m.

jms.FactoryClassName

Specifies the class name to instantiate for a JMS provider. You *must* set this connector property when you choose `JMS` as your delivery transport mechanism (DeliveryTransport).

The default is `CxCommon.Messaging.jms.IBMMQSeriesFactory`.

jms.MessageBrokerName

Specifies the broker name to use for the JMS provider. You *must* set this connector property when you choose `JMS` as your delivery transport mechanism (DeliveryTransport).

The default is `crossworlds.queue.manager`. Use the default when connecting to a local message broker.

When you connect to a remote message broker, this property takes the following (mandatory) values:
`QueueMgrName:<Channel>:<HostName>:<PortNumber>`,
where the variables are:
`QueueMgrName`: The name of the queue manager.
`Channel`: The channel used by the client.
`HostName`: The name of the machine where the queue manager is to reside.
`PortNumber`: The port number to be used by the queue manager for listening.

For example:

```
jms.MessageBrokerName = WBIMB.Queue.Manager:CHANNEL1:RemoteMachine:1456
```

jms.NumConcurrentRequests

Specifies the maximum number of concurrent service call requests that can be sent to a connector at the same time. Once that maximum is reached, new service calls block and wait for another request to complete before proceeding.

The default value is 10.

jms.Password

Specifies the password for the JMS provider. A value for this property is optional.

There is no default.

jms.UserName

Specifies the user name for the JMS provider. A value for this property is optional.

There is no default.

ListenerConcurrency

This property supports multi-threading in MQ Listener when ICS is the integration broker. It enables batch writing of multiple events to the database, thus improving system performance. The default value is 1.

This property applies only to connectors using MQ transport. The `DeliveryTransport` property must be set to `MQ`.

Locale

Specifies the language code, country or territory, and, optionally, the associated character code set. The value of this property determines such cultural conventions as collation and sort order of data, date and time formats, and the symbols used in monetary specifications.

A locale name has the following format:

`ll_TT.codeset`

where:

`ll`	a two-character language code (usually in lower case)
`TT`	a two-letter country or territory code (usually in upper case)
`codeset`	the name of the associated character code set; this portion of the name is often optional.

By default, only a subset of supported locales appears in the drop-down list. To add other supported values to the drop-down list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, refer to the sections on Connector Configurator in this guide.

The default value is `en_US`. If the connector has not been globalized, the only valid value for this property is `en_US`. To determine whether a specific connector has been globalized, see the connector version list on these websites:

http://www.ibm.com/software/websphere/wbiadapters/infocenter, or
http://www.ibm.com/websphere/integration/wicserver/infocenter

LogAtInterchangeEnd

Applicable only if RespositoryDirectory is <REMOTE>.

Specifies whether to log errors to the integration broker's log destination. Logging to the broker's log destination also turns on e-mail notification, which generates e-mail messages for the `MESSAGE_RECIPIENT` specified in the `InterchangeSystem.cfg` file when errors or fatal errors occur.

For example, when a connector loses its connection to its application, if `LogAtInterChangeEnd` is set to `true`, an e-mail message is sent to the specified message recipient. The default is `false`.

MaxEventCapacity

The maximum number of events in the controller buffer. This property is used by flow control and is applicable only if the value of the `RepositoryDirectory` property is <REMOTE>.

The value can be a positive integer between 1 and 2147483647. The default value is 2147483647.

MessageFileName

The name of the connector message file. The standard location for the message file is `\connectors\messages` in the product directory. Specify the message filename in an absolute path if the message file is not located in the standard location.

If a connector message file does not exist, the connector uses `InterchangeSystem.txt` as the message file. This file is located in the product directory.

Note: To determine whether a specific connector has its own message file, see the individual adapter user guide.

MonitorQueue

The logical queue that the connector uses to monitor duplicate events. It is used only if the `DeliveryTransport` property value is JMS and `DuplicateEventElimination` is set to `TRUE`.

The default value is `CONNECTORNAME/MONITORQUEUE`

OADAutoRestartAgent

Valid only when the `RepositoryDirectory` is <REMOTE>.

Specifies whether the connector uses the automatic and remote restart feature. This feature uses the MQ-triggered Object Activation Daemon (OAD) to restart the connector after an abnormal shutdown, or to start a remote connector from System Monitor.

This property must be set to `true` to enable the automatic and remote restart feature. For information on how to configure the MQ-triggered OAD feature. see the *Installation Guide for Windows* or *for UNIX*.

The default value is `false`.

OADMaxNumRetry

Valid only when the `RepositoryDirectory` is <REMOTE>.

Specifies the maximum number of times that the MQ-triggered OAD automatically attempts to restart the connector after an abnormal shutdown. The `OADAutoRestartAgent` property must be set to `true` for this property to take effect.

The default value is `1000`.

OADRetryTimeInterval

Valid only when the `RepositoryDirectory` is <REMOTE>.

Specifies the number of minutes in the retry-time interval for the MQ-triggered OAD. If the connector agent does not restart within this retry-time interval, the connector controller asks the OAD to restart the connector agent again. The OAD repeats this retry process as many times as specified by the `OADMaxNumRetry` property. The `OADAutoRestartAgent` property must be set to `true` for this property to take effect.

The default is `10`.

PollEndTime

Time to stop polling the event queue. The format is `HH:MM`, where *`HH`* represents 0-23 hours, and *`MM`* represents 0-59 seconds.

You must provide a valid value for this property. The default value is `HH:MM`, but must be changed.

PollFrequency

This is the interval between the end of the last poll and the start of the next poll. `PollFrequency` specifies the amount of time (in milliseconds) between the end of one polling action, and the start of the next polling action. This is not the interval between polling actions. Rather, the logic is as follows:

- Poll to obtain the number of objects specified by the value of `PollQuantity`.
- Process these objects. For some adapters, this may be partly done on separate threads, which execute asynchronously to the next polling action.
- Delay for the interval specified by `PollFrequency`.
- Repeat the cycle.

Set `PollFrequency` to one of the following values:

- The number of milliseconds between polling actions (an integer).
- The word `key`, which causes the connector to poll only when you type the letter `p` in the connector's Command Prompt window. Enter the word in lowercase.
- The word `no`, which causes the connector not to poll. Enter the word in lowercase.

The default is `10000`.

Important: Some connectors have restrictions on the use of this property. Where they exist, these restrictions are documented in the chapter on installing and configuring the adapter.

PollQuantity

Designates the number of items from the application that the connector should poll for. If the adapter has a connector-specific property for setting the poll quantity, the value set in the connector-specific property will override the standard property value.

FIX

An email message is also considerd an event. The connector behaves as follows when it is polled for email.

Polled once - connector goes to pick 1. the body of the message as it is also considered an attachment also. Since no DH was specified for this mime type, it it will ignore the body. 2. conector process first PO attachment. DH is avaiable for this mime type so it sends the business object to the Visual Test Connector. If the 3. accept in VTC again no BO should come thru Polled second time 1. conector process second PO attachment. DH is avaiable for this mime type so it sends teh BO to VTC2. accept in VTC again now the third PO attachment should come through. This is the correct behaviour.

PollStartTime

The time to start polling the event queue. The format is *`HH:MM`*, where *`HH`* represents 0-23 hours, and *`MM`* represents 0-59 seconds.

You must provide a valid value for this property. The default value is `HH:MM`, but must be changed.

RequestQueue

The queue that is used by the integration broker to send business objects to the connector.

The default value is `CONNECTOR/REQUESTQUEUE`.

RepositoryDirectory

The location of the repository from which the connector reads the XML schema documents that store the meta-data for business object definitions.

When the integration broker is ICS, this value must be set to <REMOTE> because the connector obtains this information from the InterChange Server repository.

When the integration broker is a WebSphere message broker or WAS, this value must be set to <*local directory*>.

ResponseQueue

Applicable only if `DeliveryTransport` is JMS and required only if `RepositoryDirectory` is <REMOTE>.

Designates the JMS response queue, which delivers a response message from the connector framework to the integration broker. When the integration broker is ICS, the server sends the request and waits for a response message in the JMS response queue.

RestartRetryCount

Specifies the number of times the connector attempts to restart itself. When used for a parallel connector, specifies the number of times the master connector application-specific component attempts to restart the slave connector application-specific component.

The default is `3`.

RestartRetryInterval

Specifies the interval in minutes at which the connector attempts to restart itself. When used for a parallel connector, specifies the interval at which the master connector application-specific component attempts to restart the slave connector application-specific component. Possible values ranges from 1 to 2147483647.

The default is `1`.

RHF2MessageDomain

WebSphere message brokers and WAS only.

This property allows you to configure the value of the field domain name in the JMS header. When data is sent to WMQI over JMS transport, the adapter framework writes JMS header information, with a domain name and a fixed value of `mrm`. A connfigurable domain name enables users to track how the WMQI broker processes the message data.

A sample header would look like this:

```
<mcd><Msd>mrm</Msd><Set>3</Set><Type>
Retek_POPhyDesc</Type><Fmt>CwXML</Fmt></mcd>
```

The default value is `mrm`, but it may also be set to `xml`. This property only appears when DeliveryTransport is set to `JMS`and WireFormat is set to `CwXML`.

SourceQueue

Applicable only if `DeliveryTransport` is JMS and ContainerManagedEvents is specified.

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 68.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is JMS.

Delivers request messages that require a synchronous response from the connector framework to the broker. This queue is necessary only if the connector uses synchronous execution. With synchronous execution, the connector framework sends a message to the SynchronousRequestQueue and waits for a response back from the broker on the SynchronousResponseQueue. The response message sent to the connector bears a correlation ID that matches the ID of the original message.

The default is CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE

SynchronousResponseQueue

Applicable only if DeliveryTransport is JMS.

Delivers response messages sent in reply to a synchronous request from the broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

Specifies the time in minutes that the connector waits for a response to a synchronous request. If the response is not received within the specified time, then the connector moves the original synchronous request message into the fault queue along with an error message.

The default value is 0.

WireFormat

Message format on the transport.
- If the RepositoryDirectory is a local directory, the setting is CwXML.
- If the value of RepositoryDirectory is <REMOTE>, the setting isCwBO.

WsifSynchronousRequestTimeout

WAS integration broker only.

Specifies the time in minutes that the connector waits for a response to a synchronous request. If the response is not received within the specified, time then the connector moves the original synchronous request message into the fault queue along with an error message.

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

A strong property that allows the user to specify short and long name spaces in the XML format of business object definitions.

The default value is short.

Appendix B. Connector Configurator

This appendix describes how to use Connector Configurator to set configuration property values for your adapter.

You use Connector Configurator to:
- Create a connector-specific property template for configuring your connector
- Create a configuration file
- Set properties in a configuration file

Note:
In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes and follow the conventions for each operating system.

The topics covered in this appendix are:
- "Overview of Connector Configurator" on page 79
- "Starting Connector Configurator" on page 80
- "Creating a connector-specific property template" on page 81
- "Creating a new configuration file" on page 83
- "Setting the configuration file properties" on page 86
- "Using Connector Configurator in a globalized environment" on page 92

Overview of Connector Configurator

Connector Configurator allows you to configure the connector component of your adapter for use with these integration brokers:
- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator, WebSphere MQ Integrator Broker, and WebSphere Business Integration Message Broker, collectively referred to as the WebSphere Message Brokers (WMQI)
- WebSphere Application Server (WAS)

You use Connector Configurator to:
- Create a **connector-specific property template** for configuring your connector.
- Create a **connector configuration file**; you must create one configuration file for each connector you install.
- Set properties in a configuration file.
  You may need to modify the default values that are set for properties in the connector templates. You must also designate supported business object definitions and, with ICS, maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 80).

Connector configuration properties include both standard configuration properties (the properties that all connectors have) and connector-specific properties (properties that are needed by the connector for a specific application or technology).

Because **standard properties** are used by all connectors, you do not need to define those properties from scratch; Connector Configurator incorporates them into your configuration file as soon as you create the file. However, you do need to set the value of each standard property in Connector Configurator.

The range of standard properties may not be the same for all brokers and all configurations. Some properties are available only if other properties are given a specific value. The Standard Properties window in Connector Configurator will show the properties available for your particular configuration.

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 81 to set up a new one.

Note: Connector Configurator runs only in a Windows environment. If you are running the connector in a UNIX environment, use Connector Configurator in Windows to modify the configuration file and then copy the file to your UNIX environment.

Starting Connector Configurator

You can start and run Connector Configurator in either of two modes:

- Independently, in stand-alone mode
- From System Manager

Running Configurator in stand-alone mode

You can run Connector Configurator independently and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere InterChange Server>IBM WebSphere Business Integration Tools>Connector Configurator**.
- Select **File>New>Connector Configuration**.
- When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select ICS, WebSphere Message Brokers or WAS, depending on your broker.

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 85.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

1. Open the System Manager.

2. In the System Manager window, expand the **Integration Component Libraries** icon and highlight **Connectors**.
3. From the System Manager menu bar, click **Tools>Connector Configurator**. The Connector Configurator window opens and displays a **New Connector** dialog box.
4. When you click the pull-down menu next to **System Connectivity Integration Broker**, you can select ICS, WebSphere Message Brokers or WAS, depending on your broker.

To edit an existing configuration file:
- In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it. Connector Configurator opens and displays the configuration file with the integration broker type and file name at the top.
- From Connector Configurator, select **File>Open**. Select the name of the connector configuration file from a project or from the directory in which it is stored.
- Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

To create a configuration file for your connector, you need a connector-specific property template as well as the system-supplied standard properties.

You can create a brand-new template for the connector-specific properties of your connector, or you can use an existing connector definition as the template.
- To create a new template, see "Creating a new template" on page 81.
- To use an existing file, simply modify an existing template and save it under the new name. You can find existing templates in your `\WebSphereAdapters\bin\Data\App` directory.

Creating a new template

This section describes how you create properties in the template, define general characteristics and values for those properties, and specify any dependencies between the properties. Then you save the template and use it as the base for creating a new connector configuration file.

To create a template in Connector Configurator:
1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears.
   - Enter a name for the new template in the **Name** field below **Input a New Template Name.** You will see this name again when you open the dialog box for creating a new configuration file from a template.
   - To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template appears in the **Template Preview** display.
3. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template. If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

- If you are planning to modify an existing template, select the name of the template from the list in the **Template Name** table below **Select the Existing Template to Modify: Find Template.**
- This table displays the names of all currently available templates. You can also search for a template.

Specifying general characteristics

When you click **Next** to select a template, the **Properties - Connector-Specific Property Template** dialog box appears. The dialog box has tabs for General characteristics of the defined properties and for Value restrictions. The General display has the following fields:

- **General:**
  Property Type
  Updated Method
  Description
- **Flags**
  Standard flags
- **Custom Flag**
  Flag

After you have made selections for the general characteristics of the property, click the **Value** tab.

Specifying values

The **Value** tab enables you to set the maximum length, the maximum multiple values, a default value, or a value range for the property. It also allows editable values. To do so:

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.
2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, enter your values.

To create a new property value:

1. Select the property in the **Edit properties** list and right-click on it.
2. From the dialog box, select **Add**.
3. Enter the name of the new property value and click OK. The value appears in the **Value** panel on the right.

The **Value** panel displays a table with three columns:

The **Value** column shows the value that you entered in the **Property Value** dialog box, and any previous values that you created.

The **Default Value** column allows you to designate any of the values as the default.

The **Value Range** shows the range that you entered in the **Property Value** dialog box.

After a value has been created and appears in the grid, it can be edited from within the table display.

To make a change in an existing value in the table, select an entire row by clicking on the row number. Then right-click in the **Value** field and click **Edit Value**.

Setting dependencies

When you have made your changes to the **General** and **Value** tabs, click **Next**. The **Dependencies - Connector-Specific Property Template** dialog box appears.

A dependent property is a property that is included in the template and used in the configuration file *only if* the value of another property meets a specific condition. For example, `PollQuantity` appears in the template only if JMS is the transport mechanism and `DuplicateEventElimination` is set to `True`.
To designate a property as dependent and to set the condition upon which it depends, do this:

1. In the **Available Properties** display, select the property that will be made dependent.
2. In the **Select Property** field, use the drop-down menu to select the property that will hold the conditional value.
3. In the **Condition Operator** field, select one of the following:

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

   <=(less than or equal to)
4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator stores the information you have entered as an XML document, under `\data\app` in the`\bin` directory where you have installed Connector Configurator.

Creating a new configuration file

When you create a new configuration file, you must name it and select an integration broker.

- In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator opens and displays the **New Connector** dialog box.
- In stand-alone mode: from Connector Configurator, select **File>New>Connector Configuration**. In the New Connector window, enter the name of the new connector.

You also need to select an integration broker. The broker you select determines the properties that will appear in the configuration file. To select a broker:

- In the **Integration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
- drop-downte the remaining fields in the **New Connector** window, as described later in this chapter.

Creating a configuration file from a connector-specific template

Once a connector-specific template has been created, you can use it to create a configuration file:

1. Click **File>New>Connector Configuration**.
2. The **New Connector** dialog box appears, with the following fields:
   - **Name**

     Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, and must be consistent with the file name for a connector that is installed on the system.

     Important: Connector Configurator does not check the spelling of the name that you enter. You must ensure that the name is correct.
   - **System Connectivity**

     Click ICS or WebSphere Message Brokers or WAS.
   - **Select Connector-Specific Property Template**

     Type the name of the template that has been designed for your connector. The available templates are shown in the **Template Name** display. When you select a name in the Template Name display, the **Property Template Preview** display shows the connector-specific properties that have been defined in that template.

     Select the template you want to use and click **OK**.
3. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector name. You can fill in all the field values to drop-downte the definition now, or you can save the file and complete the fields later.
4. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.
   If you save as a file, the **Save File Connector** dialog box appears. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator indicates that the configuration file was successfully created.

   Important: The directory path and name that you establish here must match the connector configuration file path and name that you supply in the startup file for the connector.
5. To complete the connector definition, enter values in the fields for each of the tabs of the Connector Configurator window, as described later in this chapter.

Using an existing file

You may have an existing file available in one or more of the following formats:

- A connector definition file.
  This is a text file that lists properties and applicable default values for a specific connector. Some connectors include such a file in a `\repository` directory in their delivery package (the file typically has the extension `.txt`; for example, CN_XML.txt for the XML connector).
- An ICS repository file.
  Definitions used in a previous ICS implementation of the connector may be available to you in a repository file that was used in the configuration of that connector. Such a file typically has the extension `.in` or `.out`.
- A previous configuration file for the connector.
  Such a file typically has the extension *.cfg.

Although any of these file sources may contain most or all of the connector-specific properties for your connector, the connector configuration file will not be complete until you have opened the file and set properties, as described later in this chapter.

To use an existing file to configure a connector, you must open the file in Connector Configurator, revise the configuration, and then resave the file.

Follow these steps to open a *.txt, *.cfg, or *.in file from a directory:

1. In Connector Configurator, click **File>Open>From File**.
2. In the **Open File Connector** dialog box, select one of the following file types to see the available files:
   - Configuration (`*.cfg`)
   - ICS Repository (`*.in`, `*.out`)

     Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will appear when you open the file.
   - All files (*.*)

     Choose this option if a `*.txt` file was delivered in the adapter package for the connector, or if a definition file is available under another extension.
3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

Follow these steps to open a connector configuration from a System Manager project:

1. Start System Manager. A configuration can be opened from or saved to System Manager only if System Manager has been started.
2. Start Connector Configurator.
3. Click **File>Open>From Project**.

Completing a configuration file

When you open a configuration file or a connector from a project, the Connector Configurator window displays the configuration screen, with the current attributes and values.

The title of the configuration screen displays the integration broker and connector name as specified in the file. Make sure you have the correct broker. If not, change the broker value before you configure the connector. To do so:

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 88..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

   If you are saving to file, select `*.cfg` as the extension, select the correct location for the file and click **Save**.

   If multiple connector configurations are open, click **Save All to File** to save all of the configurations to file, or click **Save All to Project** to save all connector configurations to a System Manager project.

Before it saves the file, Connector Configurator checks that values have been set for all required standard properties. If a required standard property is missing a value, Connector Configurator displays a message that the validation failed. You must supply a value for the property in order to save the configuration file.

Setting the configuration file properties

When you create and name a new connector configuration file, or when you open an existing connector configuration file, Connector Configurator displays a configuration screen with tabs for the categories of required configuration values.

Connector Configurator requires values for properties in these categories for connectors running on all brokers:

- Standard Properties
- Connector-specific Properties
- Supported Business Objects
- Trace/Log File values
- Data Handler (applicable for connectors that use JMS messaging with guaranteed event delivery)

Note: For connectors that use JMS messaging, an additional category may display, for configuration of data handlers that convert the data to business objects.

For connectors running on **ICS**, values for these properties are also required:

- Associated Maps
- Resources
- Messaging (where applicable)

Important: Connector Configurator accepts property values in either English or non-English character sets. However, the names of both standard and connector-specific properties, and the names of supported business objects, must use the English character set only.

Standard properties differ from connector-specific properties as follows:

- Standard properties of a connector are shared by both the application-specific component of a connector and its broker component. All connectors have the same set of standard properties. These properties are described in Appendix A of each adapter guide. You can change some but not all of these values.
- Application-specific properties apply only to the application-specific component of a connector, that is, the component that interacts directly with the application. Each connector has application-specific properties that are unique to its application. Some of these properties provide default values and some do not; you can modify some of the default values. The installation and configuration chapters of each adapter guide describe the application-specific properties and the recommended values.

The fields for **Standard Properties** and **Connector-Specific Properties** are color-coded to show which are configurable:

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.

- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.
- The **Update Method** field is displayed for each property. It indicates whether a component or agent restart is necessary to activate changed values. You cannot configure this setting.

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.
3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
   - To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

Setting application-specific configuration properties

For application-specific configuration properties, you can add or change property names, configure values, delete a property, and encrypt a property. The default property length is 255 characters.

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. To add a child property, right-click on the parent row number and click **Add child**.
2. Enter a value for the property or child property.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for “Setting standard connector properties.”

The Update Method displayed for each property indicates whether a component or agent restart is necessary to activate changed values.

Important: Changing a preset application-specific connector property name may cause a connector to fail. Certain property names may be needed by the connector to connect to an application or to run properly.

Encryption for connector properties

Application-specific properties can be encrypted by selecting the **Encrypt** check box in the Connector-specific Properties window. To decrypt a value, click to clear the **Encrypt** check box, enter the correct value in the **Verification** dialog box, and click **OK**. If the entered value is correct, the value is decrypted and displays.

The adapter user guide for each connector contains a list and description of each property and its default value.

If a property has multiple values, the **Encrypt** check box will appear for the first value of the property. When you select **Encrypt**, all values of the property will be encrypted. To decrypt multiple values of a property, click to clear the **Encrypt** check box for the first value of the property, and then enter the new value in the **Verification** dialog box. If the input value is a match, all multiple values will decrypt.

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 62.

Specifying supported business object definitions

Use the **Supported Business Objects** tab in Connector Configurator to specify the business objects that the connector will use. You must specify both generic business objects and application-specific business objects, and you must specify associations for the maps between the business objects.

Note: Some connectors require that certain business objects be specified as supported in order to perform event notification or additional configuration (using meta-objects) with their applications. For more information, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

If ICS is your broker

To specify that a business object definition is supported by the connector, or to change the support settings for an existing business object definition, click the **Supported Business Objects** tab and use the following fields.

Business object name: To designate that a business object definition is supported by the connector, with System Manager running:

1. Click an empty field in the **Business Object Name** list. A drop-down list displays, showing all the business object definitions that exist in the System Manager project.
2. Click on a business object to add it.
3. Set the **Agent Support** (described below) for the business object.
4. In the File menu of the Connector Configurator window, click **Save to Project**. The revised connector definition, including designated support for the added business object definition, is saved to an ICL (Integration Component Library) project in System Manager.

To delete a business object from the supported list:

1. To select a business object field, click the number to the left of the business object.
2. From the **Edit** menu of the Connector Configurator window, click **Delete Row**. The business object is removed from the list display.
3. From the **File** menu, click **Save to Project**.

Deleting a business object from the supported list changes the connector definition and makes the deleted business object unavailable for use in this implementation of this connector. It does not affect the connector code, nor does it remove the business object definition itself from System Manager.

Agent support: If a business object has Agent Support, the system will attempt to use that business object for delivering data to an application via the connector agent.

Typically, application-specific business objects for a connector are supported by that connector's agent, but generic business objects are not.

To indicate that the business object is supported by the connector agent, check the **Agent Support** box. The Connector Configurator window does not validate your Agent Support selections.

Maximum transaction level: The maximum transaction level for a connector is the highest transaction level that the connector supports.

For most connectors, `Best Effort` is the only possible choice.

You must restart the server for changes in transaction level to take effect.

If a WebSphere Message Broker is your broker

If you are working in stand-alone mode (not connected to System Manager), you must enter the business object name manually.

If you have System Manager running, you can select the empty box under the **Business Object Name** column in the **Supported Business Objects** tab. A combo box appears with a list of the business object available from the Integration Component Library project to which the connector belongs. Select the business object you want from the list.

The **Message Set ID** is an optional field for WebSphere Business Integration Message Broker 5.0, and need not be unique if supplied. However, for WebSphere MQ Integrator and Integrator Broker 2.1, you must supply a unique **ID**.

If WAS is your broker

When WebSphere Application Server is selected as your broker type, Connector Configurator does not require message set IDs. The **Supported Business Objects** tab shows a **Business Object Name** column only for supported business objects.

If you are working in stand-alone mode (not connected to System Manager), you must enter the business object name manually.

If you have System Manager running, you can select the empty box under the Business Object Name column in the Supported Business Objects tab. A combo box appears with a list of the business objects available from the Integration Component Library project to which the connector belongs. Select the business object you want from this list.

Associated maps (ICS only)

Each connector supports a list of business object definitions and their associated maps that are currently active in WebSphere InterChange Server. This list appears when you select the **Associated Maps** tab.

The list of business objects contains the application-specific business object which the agent supports and the corresponding generic object that the controller sends to the subscribing collaboration. The association of a map determines which map will be used to transform the application-specific business object to the generic business object or the generic business object to the application-specific business object.

If you are using maps that are uniquely defined for specific source and destination business objects, the maps will already be associated with their appropriate business objects when you open the display, and you will not need (or be able) to change them.

If more than one map is available for use by a supported business object, you will need to explicitly bind the business object with the map that it should use.

The **Associated Maps** tab displays the following fields:

- **Business Object Name**

  These are the business objects supported by this connector, as designated in the **Supported Business Objects** tab. If you designate additional business objects under the Supported Business Objects tab, they will be reflected in this list after you save the changes by choosing **Save to Project** from the **File** menu of the Connector Configurator window.
- **Associated Maps**

  The display shows all the maps that have been installed to the system for use with the supported business objects of the connector. The source business object for each map is shown to the left of the map name, in the **Business Object Name** display.
- **Explicit**

  In some cases, you may need to explicitly bind an associated map.

  Explicit binding is required only when more than one map exists for a particular supported business object. When ICS boots, it tries to automatically bind a map to each supported business object for each connector. If more than one map takes as its input the same business object, the server attempts to locate and bind one map that is the superset of the others.

  If there is no map that is the superset of the others, the server will not be able to bind the business object to a single map, and you will need to set the binding explicitly.

  To explicitly bind a map:

  1. In the **Explicit** column, place a check in the check box for the map you want to bind.
  2. Select the map that you intend to associate with the business object.
  3. In the **File** menu of the Connector Configurator window, click **Save to Project**.
  4. Deploy the project to ICS.
  5. Reboot the server for the changes to take effect.

Resources (ICS)

The **Resource** tab allows you to set a value that determines whether and to what extent the connector agent will handle multiple processes concurrently, using connector agent parallelism.

Not all connectors support this feature. If you are running a connector agent that was designed in Java to be multi-threaded, you are advised not to use this feature, since it is usually more efficient to use multiple threads than multiple processes.

Messaging (ICS)

The messaging properties are available only if you have set MQ as the value of the `DeliveryTransport` standard property and ICS as the broker type. These properties affect how your connector will use queues.

Setting trace/log file values

When you open a connector configuration file or a connector definition file, Connector Configurator uses the logging and tracing values of that file as default values. You can change those values in Connector Configurator.

To change the logging and tracing values:

1. Click the **Trace/Log Files** tab.
2. For either logging or tracing, you can choose to write messages to one or both of the following:
   - To console (STDOUT):
     Writes logging or tracing messages to the STDOUT display.

     Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.
   - To File:
     Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

     Note: Both logging and tracing files are simple text files. You can use the file extension that you prefer when you set their file names. For tracing files, however, it is advisable to use the extension `.trace` rather than `.trc`, to avoid confusion with other files that might reside on the system. For logging files, `.log` and `.txt` are typical file extensions.

Data handlers

The data handlers section is available for configuration only if you have designated a value of JMS for DeliveryTransport and a value of JMS for ContainerManagedEvents. Not all adapters make use of data handlers.

See the descriptions under `ContainerManagedEvents` in Appendix A, Standard Properties, for values to use for these properties. For additional details, see the *Connector Development Guide for* C++ or the *Connector Development Guide for Java.*

Saving your configuration file

When you have finished configuring your connector, save the connector configuration file. Connector Configurator saves the file in the broker mode that you selected during configuration. The title bar of Connector Configurator always displays the broker mode (ICS, WMQI or WAS) that it is currently using.

The file is saved as an XML document. You can save the XML document in three ways:

- From System Manager, as a file with a `*.con` extension in an Integration Component Library, or
- In a directory that you specify.

- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder. By default, the file is saved to `\WebSphereAdapters\bin\Data\App`.
- You can also save it to a WebSphere Application Server project if you have set one up.

For details about using projects in System Manager, and for further information about deployment, see the following implementation guides:

- For ICS: *Implementation Guide for WebSphere InterChange Server*
- For WebSphere Message Brokers: *Implementing Adapters with WebSphere Message Brokers*
- For WAS: *Implementing Adapters with WebSphere Application Server*

Changing a configuration file

You can change the integration broker setting for an existing configuration file. This enables you to use the file as a template for creating a new configuration file, which can be used with a different broker.

Note: You will need to change other configuration properties as well as the broker mode property if you switch integration brokers.

To change your broker selection within an existing configuration file (optional):

- Open the existing configuration file in Connector Configurator.
- Select the **Standard Properties** tab.
- In the **BrokerType** field of the Standard Properties tab, select the value that is appropriate for your broker.
  When you change the current value, the available tabs and field selections on the properties screen will immediately change, to show only those tabs and fields that pertain to the new broker you have selected.

Completing the configuration

After you have created a configuration file for a connector and modified it, make sure that the connector can locate the configuration file when the connector starts up.

To do so, open the startup file used for the connector, and verify that the location and file name used for the connector configuration file match exactly the name you have given the file and the directory or path where you have placed it.

Using Connector Configurator in a globalized environment

Connector Configurator is globalized and can handle character conversion between the configuration file and the integration broker. Connector Configurator uses native encoding. When it writes to the configuration file, it uses UTF-8 encoding.

Connector Configurator supports non-English characters in:

- All value fields
- Log file and trace file path (specified in the **Trace/Log files** tab)

The drop list for the `CharacterEncoding` and `Locale` standard configuration properties displays only a subset of supported values. To add other values to the drop list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory.

For example, to add the locale `en_GB` to the list of values for the `Locale` property, open the `stdConnProps.xml` file and add the line in boldface type below:

```
<Property name="Locale"
isRequired="true"
updateMethod="component restart">
                <ValidType>String</ValidType>
            <ValidValues>
                                <Value>ja_JP</Value>
                                <Value>ko_KR</Value>
                                <Value>zh_CN</Value>
                                <Value>zh_TW</Value>
                                <Value>fr_FR</Value>
                                <Value>de_DE</Value>
                                <Value>it_IT</Value>
                                <Value>es_ES</Value>
                                <Value>pt_BR</Value>
                                <Value>en_US</Value>
                                <Value>en_GB</Value>
                    <DefaultValue>en_US</DefaultValue>
            </ValidValues>
    </Property>
```

Notices

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

IBM WebSphere Business Integration Adapter Framework V2.4.

Index

Printed in USA