IBM WebSphere Business Integration Adapters

Adapter for Telcordia User Guide

Version 23.x
Integration broker compatibility

Supported on IBM WebSphere Business Integration Adapter Framework version 2.2.0, IBM CrossWorlds Infrastructure version 4.1.1 and 4.2, WebSphere MQ Integrator version 2.1.0, and WebSphere MQ Integrator Broker, version 2.1.0. See Release Notes for any exceptions.
New in This release

New in release 2.3.x

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 “Mapping data handlers to InputQueues“ on page 32.

The guaranteed event delivery feature has been enhanced. For further information, see “Enabling guaranteed event delivery“ on page 24.

New in release 2.2.x

The InProgress queue is no longer required and may be disabled. For more information, see “InProgressQueue“ on page 23.

The connector supports interoperability with applications via MQSeries 5.1, 5.2, and 5.3. For more information, see “Prerequisites“ on page 17.

The connector now has a UseDefaults property for business object processing. For more information, see “UseDefaults“ on page 24.

The connector can now apply a default verb when the data handler does not explicitly assign one to a business object. For more information, see “DefaultVerb“ on page 21.

The ReplyToQueue can now be dictated via the dynamic child meta-object rather than by the ReplyToQueue connector property. For more information see “JMS headers, Telcordia message properties, and dynamic child meta-object attributes“ on page 37.

You can use a message selector to identify, filter and otherwise control how the adapter identifies the response message for a given request. This JMS capability applies to synchronous request processing only. For more information, see “Synchronous delivery“ on page 10.

For information about generating business objects from schemas that include multiple files for element definitions with similar names, see “Generating Telcordia business objects based on schema documents“ on page 70.
About this document

The IBM WebSphere Business Integration Adapters supply integration connectivity for leading e-business technologies and enterprise applications.

This document describes the installation, configuration, and business object development for the adapter for Telcordia.

Audience

This document is for consultants, developers, and system administrators who support and manage the WebSphere business integration system at customer sites.

Prerequisites for this document

Users of this document should be familiar with the WebSphere business integration system, with business object and collaboration development, and with the WebSphere MQ application.

Related documents

The WebSphere business integration system documentation describes the features and components common to all installations, and includes reference material on specific collaborations and connectors.

This document contains many references to two other documents: the System Installation Guide for Windows or for UNIX and the System Implementation Guide for WebSphere InterChange Server. If you choose to print this document, you may want to print these documents as well.

To access the documentation, go to the directory where you installed the product and open the documentation subdirectory. If a welcome.html file is present, open it for hyperlinked access to all documentation. If no documentation is present, you can install it or read it directly online at:


The documentation set consists primarily of Portable Document Format (PDF) files, with some additional files in HTML format. To read it, you need an HTML browser such as Netscape Navigator or Internet Explorer, and Adobe Acrobat Reader 4.0.5 or higher. For the latest version of Adobe Acrobat Reader for your platform, go to the Adobe website (www.adobe.com).

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, 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.

**ProductDir** Represents the directory where the IBM WebSphere Business Integration Adapters product is installed.

/, \ In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the product is installed on your system.

%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.
Chapter 1. Overview

- "Telcordia application-connector overview"
- "Connector architecture" on page 3
- "Application-connector communication method" on page 4
- "Event handling" on page 6
- "Guaranteed event delivery" on page 9
- "Business object requests" on page 9
- "Verb processing" on page 10
- "Processing locale-dependent data" on page 13
- "Configuring connector properties" on page 14

The connector for Telcordia™1 is a runtime component of the IBM WebSphere Business Integration Adapter for Telcordia. The connector allows the WebSphere integration broker to exchange business objects with applications that send or receive data in the form of Telcordia messages. This chapter describes the connector component and the relevant business integration system architecture.

Connectors consist of an application-specific component and the connector framework. The application-specific component contains code tailored to a particular application. The connector framework, whose code is common to all connectors, acts as an intermediary between the integration broker and the application-specific component. The connector 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 application-specific component and connector framework. It refers to both of these components as the connector.

For more information about the relationship of the integration broker to the connector, see the System Administration Guide, or the Implementation Guide for WebSphere MQ Integrator Broker.

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

### Telcordia application-connector overview

Telcordia applications provide telephony, video, and data services to customers over a variety of network access facilities. The Telcordia Service Delivery/Order Manager processes customer orders. The customer orders are uniquely identified by order number and order type. The Telcordia connector allows bi-directional data exchanges between the Telcordia Service Delivery module and the IBM integration broker.

1. Telcordia Technologies, Inc.

© Copyright IBM Corp. 2002, 2003
The Telcordia Service Delivery/Order Manager has two major components:

1. The Service Delivery/Order manger processor, which includes:
   - A generic engine that processes workflows
   - An application task handler that communicates with partner systems, in this case, the IBM WebSphere integration broker and the Telcordia connector.
   - A status task handler that updates the Service Delivery/Order Manager’s Service Request Database (SRDB).

2. The Order Inquiry GUI, which is a component of the Telcordia’s Next Generation Network-Operations Support Systems (NGN-OSS) and Tier 2 Inter-Domain Network Management Solution GUI (also referred to as the Solution GUI).

The Telcordia connector uses Telcordia’s Order Inquiry to monitor the status of customer orders, order requests, and order items. The Order Inquiry allows you to view:

- The status of activities (processing steps) that occur as order requests (the components of the customer order) are processed
- Detailed information about a particular order request
- The messages exchanged between Service Delivery/Order Manager and IBM WebSphere at the customer-order, order-request, or order-item level.

Figure 1 illustrates the architecture of the Telcordia Service Delivery/Order Manager and the IBM WebSphere Telcordia connector.

![Figure 1](image_url)

**Figure 1. Telcordia-IBM WebSphere business integration architecture**

As shown in Figure 1:
- The connector communicates with Telcordia Service Delivery/Order Manager via XML over WebSphere MQ queues.
• The IBM WebSphere Business Integration Data Handler for XML converts XML documents to and from IBM WebSphere business objects. For more information, see the Data Handler Guide.

• The XML ODA is an object discovery agent (ODA) used to generate business object definitions from XML schema documents. The XML ODA can also be used with the IBM WebSphere business object designer. For more information, see the Data Handler Guide.

• The IBM WebSphere integration broker is a process automation server for managing disparate business applications as one. For more information see the System Administration Guide, or the Implementation Guide for WebSphere MQ Integrator Broker.

• The InputQueue, RequestQueue, and ErrorQueue are local WebSphere MQ queues configured on the Telcordia server for event polling, request processing, and error handling, respectively. The InputQueue is named DELV_2_IBM; the RequestQueue is named IBM_2_DELV.

• The Service Delivery/Order Manager allows use of the Order Inquiry GUI to monitor order status.

Service Delivery/Order Manager supports customer order types such as new connects and disconnects. How each customer order is processed depends on order type and action requested. The Telcordia connector supports different message types. Six Service Delivery message types have been tested and are available for the Telcordia connector:

• Generic Order Request
• Generic OrderResponse
• LSR Request
• LSR Response
• Billing Order Completion
• Billing Order Completion Reply

The Service Delivery/Order Manager controls and pro-actively tracks the order request process through its flexible execution sequence flow control capability. The flow involves IBM WebSphere, Telcordia Service Delivery, Network Configuration Manager (NetCon), Work Item Manager (WIM), and three primary, multi-step message exchanges.

### Connector architecture

The connector is meta-data-driven. Message routing and format conversion are initiated by an event polling technique. The connector uses an MQ implementation of the JavaTM Message Service (JMS), an API for accessing enterprise-messaging systems that also makes possible guaranteed event delivery.

The connector allows IBM WebSphere Business Integration Collaborations to asynchronously exchange business objects with applications that issue or receive Telcordia messages when changes to data occur.

The connector retrieves Telcordia messages from queues, calls a data handler for XML to convert messages to their corresponding business objects, and then delivers them to collaboration objects. In the opposite direction, the connector receives business objects from collaboration objects, converts them into Telcordia messages using the same data handler, and then delivers the messages to a Telcordia queue.
The connector comes with two sample business object definitions schemas, and XML messages. You can modify these, or create new ones, using the XML Object Discovery Agent (ODA). For more information, see Chapter 4, “Creating or modifying business objects”, on page 67.

By default, the connector uses the data handler for XML. For more information, see the Data Handler Guide.

The type of business object and verb used in processing a message is based on the FORMAT field contained in the Telcordia message header. The connector 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 Telcordia 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 connector. The child meta-object values override those specified in the static meta-object that is specified for the connector as a whole. If the child meta-object is not defined or does not define a required conversion property, the connector, 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 connector meta-object.

The connector 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 connector adds a dynamic child meta-object (if specified in the business object). The child meta-object values can direct the connector 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 the input queue, the connector looks up the business object name associated 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 is not found associated with the format, 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.

---

**Application-connector communication method**

The connector makes use of IBM’s WebSphere MQ implementation of the Java Message Service (JMS). The 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 an IBM WebSphere InterChange Server business object from a collaboration object, the connector passes the business object to the data handler. The data handler converts the business object into JMS-suitable text and the connector 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.
Event delivery

Figure 3 illustrates the event delivery 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 connector first determines whether the message type is supported. If so, the connector passes the message to the configured data handler, which converts the message into an IBM WebSphere InterChange Server business object. The verb that is set reflects the conversion properties established for the message type. The connector then determines whether the business object is subscribed to by a collaboration object. If so, the gotApplEvents() method delivers the business object to InterChange Server, and the message is removed from the in-progress queue.
Event handling

For event notification, the connector detects events written to a queue by an application rather than a database trigger. An event occurs when the Telcordia Service Delivery application generates messages and stores them on the Telcordia message queue.

Retrieval

The connector uses the `pollForEvents()` method to poll the Telcordia queue at regular intervals for messages. When the connector finds a message, it retrieves it from the Telcordia queue and examines it to determine its format. If the format has been defined in the connector’s static object, the connector 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 connector 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 68 for event failure scenarios.

The connector 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 object twice due to the connector successfully submitting the business object but failing to commit the transaction in the queue. To avoid this problem, the connector moves all messages to an in-progress queue. There, the message is held until processing is complete. If the connector 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 a JMS service provider require that every requested action on a queue be performed and committed before events are removed from the queue. Accordingly, when the connector 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.

Synchronous event handling

Optionally, to support applications that want feedback on the requests they issue, the connector for Telcordia will issue report messages back to the applications detailing the outcome of their requests once they have been processed.

To achieve this, the connector will post the business data for such requests synchronously to InterChange Server. If a collaboration object successfully processes the business object, the connector will send a report back to the requestor including the return code from InterChange Server and any business object changes. If the connector or the collaboration object fails to process the business object, the connector will send a report containing the appropriate error code and error message.

In either case, an application that sends a request to the connector for Telcordia will be notified of its outcome.

Processing: If the connector for Telcordia receives any messages requesting positive or negative acknowledgement reports (PAN or NAN), it will post the
content of the message synchronously to InterChange Server and then incorporate the return code and modified business data in to a report message that will be sent back to the requesting application.

The table below shows the required structure of Telcordia messages sent to the connector to be processed synchronously.

<table>
<thead>
<tr>
<th>MQMD field</th>
<th>Description</th>
<th>Supported values (multiple values should be OR'd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageType</td>
<td>Message Type</td>
<td>DATAGRAM</td>
</tr>
<tr>
<td></td>
<td>Options for report message requested</td>
<td>You can specify one or both of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MQRO_PAN The connector sends a report message if the business object could be successfully processed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• MQRO_NAN The connector sends a report message if an error occurred while processing the business object.</td>
</tr>
<tr>
<td>ReplyToQueue</td>
<td>Name of reply queue</td>
<td>The name of the queue to which the report message should be sent.</td>
</tr>
<tr>
<td>replyToQueueManager</td>
<td>Name of queue manager</td>
<td>The name of the queue manager to which the report message should be sent.</td>
</tr>
<tr>
<td>Message Body</td>
<td>A serialized business object in a format compatible with the data handler configured for the connector.</td>
<td></td>
</tr>
</tbody>
</table>

Upon receipt of a message as described in the table above, the connector will do the following:
1. Reconstruct the business object in the message body using the configured data handler.
2. Look up the collaboration object name specified for the business object and verb in the static meta-data object (but not in the dynamic child meta-object, which cannot convey the collaboration object name).
3. Post the business object synchronously to the specified collaboration object.
4. Generate a report encapsulating the result of the processing and any business object changes or error messages.
5. Send the report to the queue specified in the replyToQueue and replyToQueueManager fields of the request.

The table below shows the structure of the report that is sent back to the requestor from the connector.
<table>
<thead>
<tr>
<th>MQMD field</th>
<th>Description</th>
<th>Supported values (multiple values should be OR’d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageType</td>
<td>Message type</td>
<td>REPORT</td>
</tr>
<tr>
<td>feedback</td>
<td>Type of report</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MQRO_PAN If the collaboration object successfully processed the business object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MQRO_NAN If the connector or the collaboration object encountered an error while processing the request.</td>
</tr>
<tr>
<td>Message Body</td>
<td>If the collaboration object successfully processed the business object, the connector will populate the message body with the business object returned by the collaboration object. This default behavior can be overridden by setting the DoNotReportBusObj property to true in the static meta-data object.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the request could not be processed, the connector will populate the message body with the error message generated by the connector or the collaboration object.</td>
</tr>
</tbody>
</table>

### Recovery

Upon initialization, the connector 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 connector 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 connector 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 connector begins processing messages from the input queue.

**Ignore**

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

**Log error**

With the log error option, if the connector 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 connector 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 68 in Chapter 4, “Creating or modifying business objects”, on page 67

Note: By JMS conventions, a retrieved message cannot be issued immediately to another queue. To enable archiving and re-delivery of messages, the connector first produces a second message that duplicates the body and the header (as applicable) of the original. To avoid conflicts with the JMS service provider, 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$\_delivery$Transport 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 24.

If the 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 connector converts the business object to an Telcordia message and issues it. There are no requirements regarding the type of business objects processed except those of the data handler.
Verb processing

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

Note: The connector supports the business object verb create. Business objects with create verbs can be issued either asynchronously or synchronously. The default mode is asynchronous.

Create

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

Asynchronous delivery

This is the default delivery mode for business objects with create verbs. 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 connector returns SUCCESS, else FAIL.

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

Synchronous delivery

If a `replyToQueue` has been defined in the connector properties and a `responseTimeout` exists in the conversion properties for the business object, the connector issues a request in synchronous mode. The connector then waits for a response to verify that appropriate action was taken by the receiving application.

For Telcordia, the connector initially issues a message with a header as shown in the table below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>Format name</td>
<td>Output format as defined in the conversion properties and truncated to 8 characters to meet IBM requirements (example: MQSTR)</td>
</tr>
<tr>
<td>Message Type</td>
<td>Message type</td>
<td>MQMT_DATAGRAM* if no response is expected from the receiving application. MQMT_REQUEST* if a response is expected</td>
</tr>
<tr>
<td>Report Options</td>
<td>Options for report message requested.</td>
<td>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.</td>
</tr>
<tr>
<td>Reply To Queue</td>
<td>Name of reply queue</td>
<td>When a response message is expected this field is populated with the value of connector property ReplyToQueue.</td>
</tr>
<tr>
<td>Persistence</td>
<td>Message persistence</td>
<td>MQPER_PERSISTENT*</td>
</tr>
<tr>
<td>Expiry</td>
<td>Message lifetime</td>
<td>MQEI_UNLIMITED*</td>
</tr>
</tbody>
</table>

* Indicates constant defined by IBM.

The message header described in the table above 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 receiving application. The thread that issued the message waits for a response message that indicates whether the receiving application was able to process the request.

When an application receives a synchronous request from the connector, it processes the business object and issues a report message as described in the tables below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format</td>
<td>Format name</td>
<td>Input format of busObj as defined in the conversion properties.</td>
</tr>
<tr>
<td>MessageType</td>
<td>Message type</td>
<td>MQMT_REPORT*</td>
</tr>
</tbody>
</table>

*Indicates constant defined by IBM.

<table>
<thead>
<tr>
<th>Verb</th>
<th>Feedback field</th>
<th>Message Body</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>SUCCESS</td>
<td>(Optional) A serialized business object reflecting changes.</td>
</tr>
<tr>
<td></td>
<td>VALCHANGE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VALDUPES</td>
<td>(Optional) An error message.</td>
</tr>
<tr>
<td></td>
<td>FAIL</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Telcordia feedback code</th>
<th>Equivalent InterChange Server response*</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQFB_PAN or MQFB_APPL_FIRST</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>MQFB_NAN or MQFB_APPL_FIRST + 1</td>
<td>FAIL</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 2</td>
<td>VALCHANGE</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 3</td>
<td>VALDUPES</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 4</td>
<td>MULTIPLE_HITS</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 5</td>
<td>FAIL_RETRIEVE_BY.getContent</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 6</td>
<td>BO_DOES_NOT_EXIST</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 7</td>
<td>UNABLE_TO_LOGIN</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 8</td>
<td>APP_RESPONSE_TIMEOUT (results in immediate termination of connector)</td>
</tr>
</tbody>
</table>

*See the Connector Development Guide for Java 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 IBM WebSphere InterChange Server 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 IBM WebSphere InterChange Server 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 connector message and issues it to the queue specified by the replyTo field.

Upon retrieval of a response message, the connector by default matches the correlationID of the response to the messageID of a request message. The connector then notifies the thread that issued the request. Depending on the feedback field of the response, the connector 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 connector 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. However, you can also use a message selector to identify, filter, and otherwise control how the adapter identifies the response message for a given request. This message selector capability is a JMS feature. It applies to synchronous request processing only and is described below.

**Filtering response messages using a message selector:** Upon receiving a business object for synchronous request processing, the connector checks for the presence of a response Selector string in the application-specific information of the verb. If the response Selector is undefined, the connector identifies response messages using the correlation ID as described above.

If response Selector is defined, the connector expects a name-value pair with the following syntax:

```plaintext
response_selector=JMSCorrelationID LIKE 'selectorstring'
```

The message selectorstring must uniquely identify a response and its values be enclosed in single quotes as shown in the example below:

```plaintext
response_selector=JMSCorrelationID LIKE 'Oshkosh'
```

In the above example, after issuing the request message, the adapter would monitor the ReplyToQueue for a response message with a correlationID equal to "Oshkosh." The adapter would retrieve the first message that matches this message selector and then dispatch it as the response.

Optionally, the adapter performs run-time substitutions enabling you to generate unique message selectors for each request. Instead of a message selector, you specify a placeholder in the form of an integer surrounded by curly braces, for example: '{1}'. You then follow with a colon and a list of comma-separated attributes to use for the substitution. The integer in the placeholder acts as an index to the attribute to use for the substitution. For example, the following message selector:

```plaintext
response_selector=JMSCorrelationID LIKE '{1}': MyDynamicMO.CorrelationID
```

would inform the adapter to replace {1} with the value of the first attribute following the selector (in this case the attribute named CorrelationId of the child-object named MyDynamicMO. If attribute CorrelationID had a value of 123ABC, the adapter would generate and use a message selector created with the following criteria:

```plaintext
JMSCorrelation LIKE '123ABC'
```

to identify the response message.

You can also specify multiple substitutions such as the following:

```plaintext
response_selector=PrimaryId LIKE '{1}' AND AddressId LIKE '{2}' : PrimaryId, Address[4].AddressId
```
In this example, the adapter would substitute \{1\} with the value of attribute PrimaryId from the top-level business object and \{2\} with the value of AddressId from the 5th position of child container object Address. With this approach, you can reference any attribute in the business object and meta-object in the response message selector. For more information on how deep retrieval is performed using Address[4].AddressId, see JCDK API manual (getAttribute method).

An error is reported at run-time when any of the following occurs:

- If you specify a non-integer value between the ‘{}’ symbols
- If you specify an index for which no attribute is defined
- If the attribute specified does not exist in the business or meta-object
- If the syntax of the attribute path is incorrect

For example, if you include the literal value ‘{’ or ‘}’ in the message selector, you can use ‘[‘ or ‘]’ respectively. You can also place these characters in the attribute value, in which case the first ‘{‘ is not needed. Consider the following example using the escape character: response_selector=JMSCorrelation LIKE '{1}' and CompanyName='A{P': MyDynamicMO.CorrelationID

The connector would resolve this message selector as follows:

`JMSCorrelationID LIKE '123ABC' and CompanyName='A{P'`

When the connector encounters special characters such as ‘[’, ‘]’, ‘:’ or ‘;’ in attribute values, they are inserted directly into the query string. This allows you to include special characters in a query string that also serve as application-specific information delimiters.

The next example illustrates how a literal string substitution is extracted from the attribute value:

response_selector=JMSCorrelation LIKE '{1}' and CompanyName='A{{P': MyDynamicMO.CorrelationID

If MyDynamicMO.CorrelationID contained the value \{A:B\}C;D, the connector would resolve the message selector as follows: JMSCorrelationID LIKE '{A:B}C;D' and CompanyName='A{P'

For more information on the response selector code, see JMS 1.0.1 specifications.

**Creating custom feedback codes:** You can extend the Telcordia feedback codes to override default interpretations by specifying the connector property FeedbackCodeMappingMO. This property allows you to create a meta-object in which all InterChange Server-specific return status values are mapped to the Telcordia feedback codes. The return status assigned (using the meta-object) to a feedback code is passed to InterChange Server. For more information, see “FeedbackCodeMappingMO” on page 22.

**Processing locale-dependent data**

The connector has been internationalized so that it can support double-byte character sets, and deliver message text in the specified language. When the connector transfers data from a location that uses one character code to a location that uses a different code set, it performs character conversion to preserves the meaning of the data.
The Java runtime environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encodings for characters in most known character code sets (both single-byte and multibyte). Most components in the WebSphere business integration system are written in Java. Therefore, when data is transferred between most integration components, there is no need for character conversion.

To log error and informational messages in the appropriate language and for the appropriate country or territory, configure the Locale standard configuration property for your environment. For more information on configuration properties, see Chapter 3, “Standard configuration properties for connectors”, on page 45.

**Common configuration tasks**

After installation, 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, “Configuring the connector”, on page 17 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, “Configuring the connector”, on page 17.

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

- The value specified for connector property HostName matches that of the host of your Telcordia server.
- The value specified for connector property Port matches that of the port for the listener of your queue manager.
- The value specified for connector property Channel matches the server connection channel for your queue manager.
- The queue URI’s for connector properties InputQueue, InProgressQueue, ArchiveQueue, ErrorQueue, and UnsubscribeQueue are valid and actually exist.
- The CCSID property value must match that of the WebSphere MQ Queue manager on the Telcordia server. For a Windows-based connector, the CCSID value is 819.

**Configuring the connector to send requests without notification**

To configure the connector 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 also compatible with the data handler that you have configured for the connector.
- 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 “Static meta-object” on page 29 and “Dynamic child meta-object” on page 34.
• Set the property \texttt{ResponseTimeout} in the (static or dynamic) meta-object to -1. This forces the connector to issue the business object without checking for a return.

• For more information, see \textit{“Create” on page 10}, \textit{“Meta-object attributes configuration” on page 29}, and \textit{Chapter 4, “Creating or modifying business objects”, on page 67}.

**Configuring the connector to send requests and get notifications**

To configure the connector to send requests and get notifications (synchronous event handling):

• Follow the steps described in \textit{“Configuring the connector to send requests without notification” on page 14} with this exception: you specify a positive \texttt{ResponseTimeout} value to indicate how long the connector waits for a reply.

• See \textit{“Create” on page 10} for details of exactly 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 \textit{“Meta-object attributes configuration” on page 29} and \textit{Chapter 4, “Creating or modifying business objects”, on page 67}.

**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 that different business objects must be sent to, use a static meta-object. To create and configure this object:

• Follow the steps in \textit{“Static meta-object” on page 29}.

• Make sure the connector subscribes to the static meta-object by specifying the name of the static meta-object in the connector-specific property \texttt{DataHandlerConfigMO}. For more information, see \textit{“Connector-specific properties” on page 20}.

• If you use a Default property, be sure that CCSID is specified as part of the URI. If you are running a Windows-based connector, set the value of the CCSID to 819.

**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 object 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 “Dynamic child meta-object” on page 34.

If you use a Default property, be sure that CCSID is specified as part of the URI. If you are running a Windows-based connector, set the value of the CCSID to 819.

**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” on page 10.

**Configuring queue URIs**

To configure queues for use with the adapter for Telcordia:
- Specify all queues as Uniform Resource Identifiers (URIs). The syntax is: `queue://<queue manager name>/<actual queue>`
- Specify the host for the queue manager in connector-specific configuration properties.
- 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 (URI)” on page 28 and the WebSphere MQ programming guide.

**Configuring a data handler**

To configure a data handler:
- Specify the data handler class name (`com.crossworlds.DataHandlers.text.xml`) in the connector-specific property DataHandlerClassName. For more information, see “Connector-specific properties” on page 20.
- Specify the data handler class name (`com.crossworlds.DataHandlers.text.xml`) in the XML data handler meta-object, `MQ_DataHandler_DefaultXMLConfig`.
- Specify both a mime type and the data handler meta-object that defines the configuration for that mime type in the connector-specific properties `DataHandlerMimeType` and `DataHandlerConfigMO`, respectively. For more information, see the Data Handler Guide.
- Specify the full class path of the name handler in the data handler meta-object. The name handler class, `TopElementNameHandler`, is used to extract the root element name from the XML message. The `NameHandlerClass` property value is `com.crossworlds.DataHandlers.xml.TopElementNameHandler`.

**Modifying the startup script**

See Chapter 2, “Configuring the connector”, on page 17 for a description of how to start the connectors. 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 Telcordia server. For more information, see “Startup file configuration” on page 42.
Chapter 2. Configuring the connector

- “Prerequisites”
- “Overview of installation tasks”
- “Installing the connector and related files” on page 18
- “Connector configuration” on page 19
- “Queue uniform resource identifiers (URI)” on page 28
- “Meta-object attributes configuration” on page 29
- “Startup file configuration” on page 42
- “Startup” on page 43

This chapter describes how to install and configure the connector and how to configure the message flows to work with the connector.

Prerequisites

Prerequisite software
- The connector supports interoperability with applications via MQSeries 5.1, 5.2, and WebSphere MQ 5.3. Accordingly, you must have one of these software releases installed.
- In addition, you must have the IBM WebSphere MQ Java client libraries.
- The connector runs on the following platform:
  - Microsoft Windows NT 4.0 Service Pack 6A or Windows 2000
  - Solaris 7 or AIX 4.3.3 Patch Level 9
  - Telcordia Service Delivery Module 9.5.1 running on HP-UX (64-bit, B11.11)

Client setup with NT Server
See the description in WebSphere MQ for Windows: Quick Beginnings Version 5.3.

Overview of installation tasks

To install the connector for Telcordia, you must perform the following tasks:

Install ICS

This task, which includes installing the WebSphere business integration system and starting ICS, is described in the System Installation Guide. You must install release 4.1.1 or 4.2 or later of ICS; to configure the guaranteed event delivery feature, you must install release 4.2 of ICS.

To load files into the repository, consult the System Installation Guide.

Install the connector for Telcordia and related files

This task includes installing the files for the connector from the software package onto your system. See “Installing the connector and related files” on page 18.

2. If your environment implements the convert-on-the-get methodology for character-set conversions you must download the latest MA88 (IMS classes) from IBM. The patch level should be at least 5.2.2 (for MQSeries version 5.2). Doing so may avoid unsupported encoding errors.
Installing the connector and related files

The installation process consists of the following tasks:

- Installing all required files
- Loading the appropriate files into the repository

Installing the files

You install the connector components as an additional component to an existing installation. To do so, run the Installer for IBM WebSphere Business Integration Adapter and select the IBM WebSphere Business Integration Adapter for Telcordia. Then see the System Installation Guide for procedures to load the Telcordia content into the repository.

Windows connector file structure

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

The utility installs the connector into the `ProductDir/connectors/Telcordia` 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.

<table>
<thead>
<tr>
<th>Subdirectory of ProductDir</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>connectors\Telcordia\CWTelcordia.jar</code></td>
<td>Contains classes used by the Telcordia connector only</td>
</tr>
<tr>
<td><code>connectors\Telcordia\CWJMSCommon.jar</code></td>
<td>Contains classes shared by this connector and other WebSphere MQ-based connectors</td>
</tr>
<tr>
<td><code>connectors\Telcordia\start_Telcordia.bat</code></td>
<td>The startup script for the connector (NT/2000)</td>
</tr>
<tr>
<td><code>connectors\messages\TelcordiaConnector.txt</code></td>
<td>Message file for the connector</td>
</tr>
<tr>
<td><code>repository\Telcordia\CN_Telcordia.txt</code></td>
<td>Repository definition for the connector</td>
</tr>
<tr>
<td><code>connectors\Telcordia\Samples\Schema\OrderRequest.xsd</code></td>
<td>Schema for sample OrderRequest business object</td>
</tr>
<tr>
<td><code>connectors\Telcordia\Samples\Schema\OrderResponse.xsd</code></td>
<td>Schema for sample OrderResponse business object</td>
</tr>
<tr>
<td><code>connectors\Telcordia\Samples\BOs\OrderRequest.in</code></td>
<td>Repository definition for sample OrderRequest business object</td>
</tr>
<tr>
<td><code>connectors\Telcordia\Samples\BOs\OrderResponse.in</code></td>
<td>Repository definition for sample OrderResponse business object</td>
</tr>
<tr>
<td><code>connectors\Telcordia\Samples\Message\NewConMsgtoSop.xml</code></td>
<td>XML message for sample OrderRequest business object</td>
</tr>
<tr>
<td><code>connectors\Telcordia\Samples\Message\NewConNGCCinput.xml</code></td>
<td>XML message for sample OrderResponse business object</td>
</tr>
</tbody>
</table>

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

UNIX connector file structure

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

The utility installs the connector into the `ProductDir/connectors/Telcordia` 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.

<table>
<thead>
<tr>
<th>Subdirectory of ProductDir</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectors/Telcordia/CWTelcordia.jar</td>
<td>Contains classes used by the Telcordia connector only</td>
</tr>
<tr>
<td>connectors/Telcordia/CWJMSCommon.jar</td>
<td>Contains classes shared by this connector and other WebSphere MQ-based connectors</td>
</tr>
<tr>
<td>connectors/Telcordia/start_Telcordia.sh</td>
<td>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.</td>
</tr>
<tr>
<td>connectors/messages/TelcordiaConnector.txt</td>
<td>Message file for the connector</td>
</tr>
<tr>
<td>repository/Telcordia/CN_Telcordia.txt</td>
<td>Repository definition for the connector</td>
</tr>
<tr>
<td>connectors/Telcordia/Samples/Schema/OrderRequest.xsd</td>
<td>Schema for sample OrderRequest business object</td>
</tr>
<tr>
<td>connectors/Telcordia/Samples/Schema/OrderResponse.xsd</td>
<td>Schema for sample OrderResponse business object</td>
</tr>
<tr>
<td>connectors/Telcordia/Samples/BOs/OrderRequest.in</td>
<td>Repository definition for sample OrderRequest business object</td>
</tr>
<tr>
<td>connectors/Telcordia/Samples/BOs/OrderResponse.in</td>
<td>Repository definition for sample OrderResponse business object</td>
</tr>
<tr>
<td>connectors/Telcordia/Samples/Message/NewConMsgtoSop.xml</td>
<td>XML message for sample OrderRequest business object</td>
</tr>
<tr>
<td>connectors/Telcordia/Samples/Message/NewConNGCCinput.xml</td>
<td>XML message for sample OrderResponse business object</td>
</tr>
</tbody>
</table>

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

**Connector configuration**

Connectors have two types of configuration properties: standard configuration properties and connector-specific configuration properties. You must set the values of these properties using Connector Configurator before running the connector. For further information, see the Connector Configurator appendix to this guide.

**Standard connector properties**

Standard configuration properties provide information that all connectors use. See Chapter 3, “Standard configuration properties for connectors”, on page 45 for documentation of these properties.

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CharacterEncoding</td>
<td>This connector does not use this property.</td>
</tr>
<tr>
<td>Locale</td>
<td>Because this connector has been internationalized, you can change the value of this property. See release notes for the connector to determine currently supported locales.</td>
</tr>
</tbody>
</table>

You must provide a value for the ApplicationName configuration property before running the connector.
Connector-specific properties

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Possible values</th>
<th>Default value</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplicationPassword</td>
<td>Login password</td>
<td>queue://crossworlds.queuemanager/MQCONN.ARCHIVE</td>
<td>No</td>
</tr>
<tr>
<td>ApplicationUserName</td>
<td>Login user ID</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>ArchiveQueue</td>
<td>Queue to which copies of successfully processed messages are sent</td>
<td>queue://crossworlds.queuemanager/MQCONN.ARCHIVE</td>
<td>No</td>
</tr>
<tr>
<td>CCSID</td>
<td>Character set for queue manager connection</td>
<td>null</td>
<td>No</td>
</tr>
<tr>
<td>Channel</td>
<td>Telcordia server connector channel</td>
<td>CHANNEL1</td>
<td>Yes</td>
</tr>
<tr>
<td>ConfigurationMetaObject</td>
<td>Name of configuration meta-object</td>
<td>Telcordia_MQSeries_MO_Config</td>
<td>Yes</td>
</tr>
<tr>
<td>DataHandlerClassName</td>
<td>Data handler class name</td>
<td>com.crossworlds.DataHandlers.text.xml</td>
<td>No</td>
</tr>
<tr>
<td>DataHandlerConfigMO</td>
<td>Data handler meta-object</td>
<td>MO_DataHandler_Default</td>
<td>Yes</td>
</tr>
<tr>
<td>DataHandlerMimeType</td>
<td>MIME type of file</td>
<td>Text/XML</td>
<td>Yes</td>
</tr>
<tr>
<td>DefaultVerb</td>
<td>Any verb supported by the connector</td>
<td>Create</td>
<td></td>
</tr>
<tr>
<td>ErrorQueue</td>
<td>Queue for unprocessed messages</td>
<td>queue://crossworlds.queuemanager/MQCONN.ERROR</td>
<td>No</td>
</tr>
<tr>
<td>FeedbackCodeMappingMO</td>
<td>Feedback code meta-object</td>
<td></td>
<td>No</td>
</tr>
<tr>
<td>HostName</td>
<td>Telcordia server</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>InDoubtEvents</td>
<td>FailOnStartup Reprocess</td>
<td>Reprocess</td>
<td>No</td>
</tr>
<tr>
<td>InputQueue</td>
<td>Poll queues</td>
<td>queue://crossworlds.queuemanager/DELV_2_IBM</td>
<td>No</td>
</tr>
<tr>
<td>InProgressQueue</td>
<td>In-progress event queue</td>
<td>queue://crossworlds.queuemanager/MQCONN.IN_PROGRESS</td>
<td>No</td>
</tr>
<tr>
<td>PollQuantity</td>
<td>Number of messages to retrieve from each queue specified in the InputQueue property</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>Port</td>
<td>Port established for the WebSphere MQ listener</td>
<td>1414</td>
<td>Yes</td>
</tr>
<tr>
<td>ReplyToQueue</td>
<td>Queue to which response messages are delivered when the connector issues requests</td>
<td>queue://crossworlds.queuemanager/MQCONN.REPLYTO</td>
<td>No</td>
</tr>
<tr>
<td>UnsubscribedQueue</td>
<td>Queue to which unsubscribed messages are sent</td>
<td>queue://crossworlds.queuemanager/MQCONN.UNSUBSCRIBE</td>
<td>No</td>
</tr>
<tr>
<td>UseDefaults</td>
<td>true or false</td>
<td>true or false</td>
<td></td>
</tr>
</tbody>
</table>

**ApplicationPassword**

Password used with UserID to log in to Telcordia.

Default = None.

If the ApplicationPassword is left blank or removed, the connector uses the default password provided by Telcordia.*
**ApplicationUserName**  
User ID used with Password to log in to Telcordia.

Default = None.

If the ApplicationUserName is left blank or removed, the connector uses the default user ID provided by Telcordia.*

**ArchiveQueue**  
Queue to which copies of successfully processed messages are sent.

Default = queue://crossworlds.queue.manager/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 URI; see “Queue uniform resource identifiers (URI)” on page 28. The value, for a Windows-based Telcordia connector, should be set to 819.

Default = null.

**Channel**  
MQ server connector channel through which the connector communicates with Telcordia.

Default = CHANNEL1.

If the Channel is left blank or removed, the connector uses the default server channel provided by Telcordia.*

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

Default = Telcordia_MQSeries_MO_Config.

**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.

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= Create
**ErrorQueue**
Queue to which messages that could not be processed are sent.

Default = queue://crossworlds.queue.manager/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 10. The connector accepts the following attribute values representing Telcordia-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 connector accepts the following InterChange Server-specific status codes as attribute values in the meta-object:

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

*Table 1. The table below shows a sample meta-object*

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQFB_APPL_FIRST</td>
<td>SUCCESS</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 1</td>
<td>FAIL</td>
</tr>
<tr>
<td>MQFB_APPL_FIRST + 2</td>
<td>UNABLE_TO_LOGIN</td>
</tr>
</tbody>
</table>

Default = none.

**HostName**
The name of the server hosting Telcordia.

Default = none.

**InDoubtEvents**
Specifies how to handle in-progress events that are not fully processed due to unexpected connector 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 connector for new messages. The connector 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 will start up properly, print a warning message, and perform request processing only. It will perform no event processing.

The connector 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 connector retrieves messages in the following manner:

Since we have a `pollQuantity` of 2, the connector will retrieve at most 2 messages from each queue per call to pollForEvents. For the first cycle (1 of 2), the connector 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 connector would stop. Since we have a `pollQuantity` of 2, the connector 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://crossworlds.queue.manager/DELV_2_IBM`

**InProgressQueue**
Message queue where messages are held during processing. You can configure the connector to operate without this queue by using System Manager to remove the default `InProgressQueue` name from the connector-specific properties. Doing so prompts a warning at startup that event delivery may be compromised if the connector is shut down while are events pending.

Default = `queue://crossworlds.queue.manager/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 MQ listener.

Default = 1414
**ReplyToQueue**

Queue to which response messages are delivered when the connector issues requests. You can also use attributes in the child dynamic meta-object to ignore a response. For more information on the these attributes, see “JMS headers, Telcordia message properties, and dynamic child meta-object attributes” on page 37.

Default = queue://crossworlds.queue.manager/MQCONN.REPLYTO

**UnsubscribedQueue**

Queue to which messages that are not subscribed are sent.

Default = queue://crossworlds.queue.manager/MQCONN.UNSUBSCRIBED

**Note:** *Always check the values Telcordia 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 26.

**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 26

**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 2.

<table>
<thead>
<tr>
<th>Connector property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeliveryTransport</td>
<td>JMS</td>
</tr>
<tr>
<td>ContainerManagedEvents</td>
<td>JMS</td>
</tr>
</tbody>
</table>

---

Table 2. Guaranteed-event-delivery connector properties for a connector with a JMS event store
Table 2. Guaranteed-event-delivery connector properties for a connector with a JMS event store (continued)

<table>
<thead>
<tr>
<th>Connector property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PollQuantity</td>
<td>The number of events to processing in a single poll of the event store</td>
</tr>
<tr>
<td>SourceQueue</td>
<td>Name of the JMS source queue (event store) which the connector framework polls and from which it retrieves events for processing. <strong>Note:</strong> 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.</td>
</tr>
</tbody>
</table>

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 3 summarizes.

Table 3. Data-handler properties for guaranteed event delivery

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

**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 2 and Table 3. To set these connector configuration properties, use the Connector Configurator tool. Connector Configurator displays the connector properties in Table 2 on its Standard Properties tab. It displays the connector properties in Table 3 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 A, “Connector configurator”, on page 75.
**Effect on event polling:** If a connector uses guaranteed event delivery by setting `ContainerManagedEvents` 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 3 on page 25.
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 4.

**Table 4. Guaranteed-event-delivery connector properties for a connector with a non-JMS event store**

<table>
<thead>
<tr>
<th>Connector property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DeliveryTransport</code></td>
<td>JMS</td>
</tr>
<tr>
<td><code>DuplicateEventElimination</code></td>
<td>true</td>
</tr>
</tbody>
</table>
If you configure a connector to use guaranteed event delivery, you must set the connector properties as described in Table 4. 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 A, “Connector configurator”, on page 75.

**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

---

**Table 4. Guaranteed-event-delivery connector properties for a connector with a non-JMS event store (continued)**

<table>
<thead>
<tr>
<th>Connector property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonitorQueue</td>
<td>Name of the JMS monitor queue, in which the connector framework stores the <code>ObjectEventId</code> of processed business objects</td>
</tr>
</tbody>
</table>
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 (URI)

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 DELV_2_IBM on queue manager crossworlds.queue.manager and causes all messages to be sent as messages with priority 5.

```
queue://crossworlds.queue.manager/DELV_2_IBM?targetClient=1&priority=5
```

The table below shows property names for queue URIs.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>expiry</td>
<td>Lifetime of the message in milliseconds.</td>
<td>0 = unlimited. Positive integers = timeout (in ms).</td>
</tr>
<tr>
<td>priority</td>
<td>Priority of the message.</td>
<td>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.</td>
</tr>
<tr>
<td>persistence</td>
<td>Whether the message should be ‘hardened’ to disk.</td>
<td>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.</td>
</tr>
<tr>
<td>CCSID</td>
<td>Character set encoding of the outbound message.</td>
<td>Integers - valid values listed in base WebSphere MQ documentation. This value should match that of the CCSID connector-specific configuration property; see “CCSID” on page 21</td>
</tr>
<tr>
<td>targetClient</td>
<td>Whether the receiving application is JMS compliant or not.</td>
<td>0 = JMS (MQRFH2 header) 1 = MQ (MQMD header only)</td>
</tr>
<tr>
<td>encoding</td>
<td>How to represent numeric fields.</td>
<td>An integer value as described in the base WebSphere MQ documentation.</td>
</tr>
</tbody>
</table>
Note: The connector 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 connector 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 connector has no control over differences or failures in the conversion process. The connector 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 connector 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 IBM WebSphere InterChange Server Technical Support to discuss the possibility of using an alternate Java Virtual Machine to run the connector.

### Meta-object attributes configuration

The connector for Telcordia 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.

#### Static meta-object

The Telcordia static meta-object consists of a list of conversion properties defined for different business objects. To define the conversion properties for a business object, first create a string attribute and name it using the syntax `busObj_verb`. For example, to define the conversion properties for a Customer object with the verb Create, create an attribute named `Customer_Create`. In the application-specific text of the attribute, you specify the actual conversion properties.

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.

The table below describes the meta-object properties.

<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CorrelationID</td>
<td>This property affects adapter behavior during request processing only and is handled the same as the CorrelationID property in the dynamic meta-object. For more information, see &quot;Asynchronous request processing&quot; on page 41.</td>
</tr>
<tr>
<td>Property name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CollaborationName</td>
<td>The <code>CollaborationName</code> 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 requests for the business object Customer with the Create verb, the static meta-data object must contain an attribute named <code>Customer_Create</code>. The <code>Customer_Create</code> attribute must contain application specific text that includes a name-value pair. For example, <code>CollaborationName=MyCustomerProcessingCollab</code>. See the &quot;Application-specific information&quot; on page 32 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. <strong>Note:</strong> This property is only available for synchronous requests.</td>
</tr>
<tr>
<td>DataEncoding</td>
<td><code>DataEncoding</code> 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. <code>DataEncoding</code> defined in a dynamic child meta-object overrides the value defined in the static meta-object. The default value is <code>Text</code>. The format for the value of this attribute is <code>messageType[:enc]</code>. I.e., <code>Text:ISO8859_1</code>, <code>Text:UnicodeLittle</code>, <code>Text</code>, or <code>Binary</code>. This property is related internally to the <code>InputFormat</code> property: specify one and only one <code>DataEncoding</code> per <code>InputFormat</code>.</td>
</tr>
<tr>
<td>DataHandlerConfigMO</td>
<td>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 <code>DataHandlerConfigMO</code> connector property. Use this static meta-object property when different data handlers are required for processing different business object types. If defined in a dynamic child meta-object, this property will override the connector property and the static meta-object property. 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.</td>
</tr>
<tr>
<td>DataHandlerMimeType</td>
<td>Allows you to request a data handler based on a particular MIME type. If specified in the static meta-object, this will override the value specified in the <code>DataHandlerMimeType</code> connector property. Use this static meta-object property when different data handlers are required for processing different business object types. If defined in a dynamic child meta-object, this property will override the connector property and the static meta-object property. 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 <code>DataHandlerConfigMO</code> should have an attribute that corresponds to the value of this property.</td>
</tr>
<tr>
<td>Property name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DoNotReportBusObj</td>
<td>Optionally, the user can include the DoNotReportBusObj property. By setting this property to true, all PAN report messages issued will have a blank message body. This is recommended when a requestor wants to confirm that a request has been successfully processed and does not need notification of changes to the business object. This does not affect NAN reports. If this property is not found in the static meta-object, the connector will default it to false and populate the message report with the business object. <strong>Note:</strong> This property is only available for synchronous requests.</td>
</tr>
<tr>
<td>InputFormat</td>
<td>The InputFormat is the message format to associate with the given business object. When a message is retrieved and is in this format, it is converted to the given business object if possible. This property is related internally to the DataEncoding property: Specify one and only one DataEncoding per InputFormat.</td>
</tr>
<tr>
<td>OutputFormat</td>
<td>The OutputFormat is set on messages created from the given business object. If the OutputFormat is not specified, the input format is used, if available. An OutputFormat defined in a dynamic child meta-object overrides the value defined in the static meta-object.</td>
</tr>
<tr>
<td>InputQueue</td>
<td>The input queue that the connector polls to detect new messages. You can use connector-specific properties to configure multiple InputQueues and optionally map different data handlers to each queue.</td>
</tr>
<tr>
<td>OutputQueue</td>
<td>The OutputQueue is the output queue to which messages derived from the given business object are delivered. An OutputQueue defined in a dynamic child meta-object overrides the value defined in the static meta-object.</td>
</tr>
<tr>
<td>ResponseTimeout</td>
<td>Indicates the length of time in milliseconds to wait before timing out when waiting for a response. The connector returns SUCCESS immediately without waiting for a response if this is left undefined or with a value less than zero. A ResponseTimeout defined in a dynamic child meta-object overrides the value defined in the static meta-object.</td>
</tr>
<tr>
<td>TimeoutFatal</td>
<td>If this property is defined and has a value of True, the connector returns APP_RESPONSE_TIMEOUT when a response is not received within the time specified by ResponseTimeout. All other threads waiting for response messages immediately return APP_RESPONSE_TIMEOUT to InterChange Server. This causes InterChange Server to terminate the connector. A TimeoutFatal defined in a dynamic child meta-object overrides the value defined in the static meta-object.</td>
</tr>
<tr>
<td>CCSID</td>
<td>The character set for the queue manager connection. Its value should match that of the CCSID property in the queue URI. And the value should be set to 819 for a Windows-based connector to function properly with the Unix-based Telcordia WebSphere MQ application.</td>
</tr>
</tbody>
</table>

Additionally, a reserved attribute named Default can be defined in the meta-object. When this attribute is present, its properties act as default values for all business object conversion properties.

Consider the following sample meta-object.
### Application-specific information

The application-specific information is structured in name-value pair format, separated by semicolons. For example:

\[
\text{InputFormat=CUST\_IN;OutputFormat=CUST\_OUT}
\]

### Mapping data handlers to InputQueues

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. To do so you must:

1. Use connector-specific properties (see "InputQueue" on page 23) to configure one or more input queues.
2. For each input queue, 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.MQ.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 connector then passes the
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 connector 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.

A sample meta-object
The static meta-object shown below configures the connector to convert Customer business objects using verbs Create, Update, Delete, and Retrieve. Note that attribute `Default` is defined in the meta-object. The connector uses the conversion properties of this attribute:

```
OutputQueue=queue://queue.manager/IBM_2_DELV;ResponseTimeout=5000;TimeoutFatal=true
```

as default values for all other conversion properties. Thus, unless specified otherwise by an attribute or overridden by a dynamic child meta-object value, the connector will issue all business objects to queue `queue://queue.manager/IBM_2_DELV`; and then wait for a response message. If a response does not arrive within 5000 milliseconds, the connector terminates immediately.

**Customer object with verb create:** Attribute `LSRRequest_Create` indicates to the connector that any messages of format `NEW` should be converted to a loop service request business object with the verb Create. Since an output format is not defined, the connector will send messages representing this object-verb combination using the format defined for input (in this case `NEW`).

Here is the sample:

```
[ReposCopy]
Version = 3.1.0
Repositories = 1cHyILNuPTc=
[End]

[BusinessObjectDefinition]
Name = Sample_MO
Version = 1.0.0

[Attribute]
Name = Default
Type = String
Cardinality = 1
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
AppSpecificInfo = OutputQueue=queue://queue.manager/IBM_2_DELV;
ResponseTimeout=5000;TimeoutFatal=true
IsRequiredServerBound = false
[End]
[Attribute]
```
Dynamic child meta-object

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

The connector recognizes and reads conversion properties from a dynamic meta-object 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.

**Note:** The connector does not support use of a dynamic child meta-object to supply a collaboration object name during synchronous event delivery.

The table in the previous section and the table below show sample static and dynamic child meta-objects, respectively, for a business object. Note that the application-specific information consists of semi-colon delimited name-value pairs.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Application-specific information</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderResponse_Create</td>
<td>OutputQueue=queue://queue.manager/IBM_2_DELVX;DataEncoding=Text:UnicodeLittle;</td>
<td>The OutputQueue and DataEncoding properties specified here override those in the static meta-object.</td>
</tr>
</tbody>
</table>
The connector checks the application-specific information of top-level business object received to determine whether tag cw_mo_conn specifies a child meta-object. If so, the dynamic child meta-object values override those specified in the static meta-object.

Population of the dynamic child meta-object during polling
In order to provide collaboration objects 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.

The table below shows how a dynamic child meta-object might be structured for polling.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Sample value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputFormat</td>
<td>CUST_IN</td>
</tr>
<tr>
<td>InputQueue</td>
<td>MYInputQueue</td>
</tr>
<tr>
<td>OutputFormat</td>
<td>CxIgnore</td>
</tr>
<tr>
<td>OutputQueue</td>
<td>CxIgnore</td>
</tr>
<tr>
<td>ResponseTimeout</td>
<td>CxIgnore</td>
</tr>
<tr>
<td>TimeoutFatal</td>
<td>CxIgnore</td>
</tr>
</tbody>
</table>

As shown in the table above, you can define an additional attribute, InputQueue, in a dynamic child meta-object. This attribute contains the name of the queue from which a given message has been retrieved. If this property is not defined in the child meta-object, it 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 collaboration objects.

Sample 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

[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
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]

[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
The following attributes, which reflect JMS and Telcordia header properties, are recognized in the dynamic meta-object.
Table 5. Dynamic meta-object header attributes

<table>
<thead>
<tr>
<th>Header attribute name</th>
<th>Mode</th>
<th>Corresponding JMS header</th>
</tr>
</thead>
<tbody>
<tr>
<td>CorrelationID</td>
<td>Read/Write</td>
<td>JMSCorrelationID</td>
</tr>
<tr>
<td>ReplyToQueue</td>
<td>Read/Write</td>
<td>JMSReplyTo</td>
</tr>
<tr>
<td>DeliveryMode</td>
<td>Read</td>
<td>JMSDeliveryMode</td>
</tr>
<tr>
<td>Priority</td>
<td>Read</td>
<td>JMSPriority</td>
</tr>
<tr>
<td>Destination</td>
<td>Read</td>
<td>JMSDestination</td>
</tr>
<tr>
<td>Expiration</td>
<td>Read</td>
<td>JMSExpiration</td>
</tr>
<tr>
<td>MessageID</td>
<td>Read</td>
<td>JMSMessageID</td>
</tr>
<tr>
<td>Redelivered</td>
<td>Read</td>
<td>JMSRedelivered</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Read</td>
<td>JMSTimeStamp</td>
</tr>
<tr>
<td>Type</td>
<td>Read</td>
<td>JMSType</td>
</tr>
<tr>
<td>UserID</td>
<td>Read</td>
<td>JMSXUserID</td>
</tr>
<tr>
<td>AppID</td>
<td>Read</td>
<td>JMSXAppID</td>
</tr>
<tr>
<td>DeliveryCount</td>
<td>Read</td>
<td>JMSXDeliveryCount</td>
</tr>
<tr>
<td>GroupID</td>
<td>Read</td>
<td>JMSXGroupID</td>
</tr>
<tr>
<td>GroupSeq</td>
<td>Read</td>
<td>JMSXGroupSeq</td>
</tr>
<tr>
<td>JMSProperties</td>
<td>Read/Write</td>
<td></td>
</tr>
</tbody>
</table>

Read-only attributes are read from a message header during event notification and written to the dynamic meta-object. These properties also populate the dynamic MO when a response message is issued during request processing. Read/write attributes are set on message headers created during request processing. During event notification, read/write attributes are read from message headers to populate the dynamic meta-object.

The interpretation and use of these attributes are described in the sections below.

Note: None of the above attributes are required. You may add any attributes to the dynamic meta-object that relate to your business process.

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 table below shows application-specific information properties that you must define for attributes in the JMSProperties object.
Table 6. Application-specific information for JMS property attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Possible values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Any valid JMS property name</td>
<td>This is the name of the JMS property. 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.</td>
</tr>
<tr>
<td>Type</td>
<td>String, Int, Boolean, Float, Double, Long, Short</td>
<td>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.</td>
</tr>
</tbody>
</table>

The figure below 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.

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

Asynchronous event notification: If a dynamic meta-object with header attributes is present in the event business object, the connector performs the following steps (in addition to populating the meta-object with transport-related data):

1. Populates the CorrelationId attribute of the meta-object with the value specified in the JMSCorrelationId header field of the message.
2. Populates the ReplyToQueue attribute of the meta-object with the queue specified in the JMSReplyTo header field of the message. Since this header field is represented by a Java object in the message, the attribute is populated with the name of the queue (often a URI).
3. Populates the DeliveryMode attribute of the meta-object with the value specified in the JMSDeliveryMode header field of the message.
4. Populates the Priority attribute of the meta-object with the JMSPriority header field of the message.

5. Populates the Destination attribute of the meta-object with the name of the JMSDestination header field of the message. Since the Destination is represented by an object, the attribute is populated with the name of the Destination object.

6. Populates the Expiration attribute of the meta-object with the value of the JMSExpiration header field of the message.

7. Populates the MessageID attribute of the meta-object with the value of the JMSMessageID header field of the message.

8. Populates the Redelivered attribute of the meta-object with the value of the JMSRedelivered header field of the message.

9. Populates the TimeStamp attribute of the meta-object with the value of the JMSTimeStamp header field of the message.

10. Populates the Type attribute of the meta-object with the value of the JMSType header field of the message.

11. Populates the UserID attribute of the meta-object with the value of the JMSXUserID property field of the message.

12. Populates the AppID attribute of the meta-object with the value of the JMSXAppID property field of the message.

13. Populates the DeliveryCount attribute of the meta-object with the value of the JMSXDeliveryCount property field of the message.

14. Populates the GroupID attribute of the meta-object with the value of the JMSXGroupID property field of the message.

15. Populates the GroupSeq attribute of the meta-object with the value of the JMSXGroupSeq property field of the message.

16. Examines the object defined for the JMSProperties attribute of the meta-object. The adapter populates each attribute of this object with the value of the corresponding property in the message. If a specific property is undefined in the message, the adapter sets the value of the attribute to CxBlank.

Synchronous event notification: For synchronous event processing, the adapter posts an event and waits for a response from the integration broker before sending a response message back to the application. Any changes to the business data are reflected in the response message returned. Before posting the event, the adapter populates the dynamic meta-object just as described for asynchronous event notification. The values set in the dynamic meta-object are reflected in the response-issued header as described below (all other read-only header attributes in the dynamic meta-object are ignored.):

- **CorrelationID** If the dynamic meta-object includes the attribute CorrelationId, you must set it to the value expected by the originating application. The application uses the CorrelationID to match a message returned from the connector to the original request. Unexpected or invalid values for a CorrelationID will cause problems. It is helpful to determine how the application handles correlating request and response messages before using this attribute. You have four options for populating the CorrelationID in a synchronous request.

  1. Leave the value unchanged. The CorrelationID of the response message will be the same as the CorrelationID of the request message. This is equivalent to the Telcordia option MQRO_PASS_CORREL_ID.
2. Change the value to CxIgnore. The connector by default copies the message ID of the request to the CorrelationID of the response. This is equivalent to the Telcordia option MQRO_COPY_MSG_ID_TO_CORREL_ID.

3. Change the value to CxBlank. The connector will not set the CorrelationID on the response message.

4. Change the value to a custom value. This requires that the application processing the response recognize the custom value.

If you do not define attribute CorrelationID in the meta-object, the connector handles the CorrelationID automatically.

- **ReplyToQueue** If you update the dynamic meta-object by specifying a different queue for attribute ReplyToQueue, the connector sends the response message to the queue you specify. This is not recommended. Having the connector send response messages to different queues may interfere with communication because an application that sets a specific reply queue in a request message is assumed to be waiting for a response on that queue.

- **JMS properties** The values set for the JMS Properties attribute in the dynamic meta-object when the updated business object is returned to the connector are set in the response message.

**Asynchronous request processing:** The connector uses the dynamic meta-object, if present, to populate the request message prior to issuing it. The connector performs the following steps before sending a request message:

1. If attribute CorrelationID is present in the dynamic meta-object, the connector sets the CorrelationID of the outbound request message to this value.

2. If attribute ReplyToQueue is specified in the dynamic meta-object, the connector passes this queue via the request message and waits on this queue for a response. This allows you to override the ReplyToQueue value specified in the connector configuration properties. If you additionally specify a negative ResponseTimeout (meaning that the connector should not wait for a response), the ReplyToQueue is set in the response message, even though the connector does not actually wait for a response.

3. If attribute JMSProperties is specified in the dynamic meta-object, the corresponding JMS properties specified in the child dynamic meta-object are set in the outbound message sent by the connector.

**Note:** If header attributes in the dynamic meta-object are undefined or specify CxIgnore, the connector follows its default settings.

**Synchronous request processing:** The connector uses the dynamic meta-object, if present, to populate the request message prior to issuing it. If the dynamic meta-object contains header attributes, the connector populates it with corresponding new values found in the response message. The connector performs the following steps (in addition to populating the meta-object with transport-related data) after receiving a response message:

1. If attribute CorrelationID is present in the dynamic meta-object, the adapter updates this attribute with the JMSCorrelationID specified in the response message.

2. If attribute ReplyToQueue is defined in the dynamic meta-object, the adapter updates this attribute with the name of the JMSReplyTo specified in the response message.

3. If attribute DeliveryMode is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSDeliveryMode header field of the message.
4. If attribute Priority is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSPriority header field of the message.

5. If attribute Destination is defined in the dynamic meta-object, the adapter updates this attribute with the name of the JMSDestination specified in the response message.

6. If attribute Expiration is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSExpiration header field of the message.

7. If attribute MessageID is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSMessageID header field of the message.

8. If attribute Redelivered is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSRedelivered header field of the message.

9. If attribute TimeStamp is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSTimeStamp header field of the message.

10. If attribute Type is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSType header field of the message.

11. If attribute UserID is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXUserID header field of the message.

12. If attribute AppID is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXAppID property field of the message.

13. If attribute DeliveryCount is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXDeliveryCount header field of the message.

14. If attribute GroupID is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXGroupID header field of the message.

15. If attribute GroupSeq is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXGroupSeq header field of the message.

16. If attribute JMSProperties is defined in the dynamic meta-object, the adapter updates any properties defined in the child object with the values found in the response message. If a property defined in the child object does not exist in the message, the value is set to CxBlank.

Note: Using the dynamic meta-object to change the CorrelationID set in the request message does not affect the way the adapter identifies the response message—the adapter by default expects that the CorrelationID of any response message equals the message ID of the request sent by the adapter.

Error handling: If a JMS property cannot be read from or written to a message, the connector logs an error and the request or event fails. If a user-specified ReplyToQueue does not exist or cannot be accessed, the connector logs an error and the request fails. If a CorrelationID is invalid or cannot be set, the connector logs an error and the request fails. In all cases, the message logged is from the connector message file.

Startup file configuration

Before you start the connector for Telcordia, you must configure the startup file.
Windows

To complete the configuration of the connector for Windows platforms, you must modify the start_Telcordia.bat file:

1. Open the start_Telcordia.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 connector for UNIX platforms, you must modify the start_Telcordia.sh file:

1. Open the start_Telcordia.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.

Startup

For information on starting a connector, stopping a connector, and the connector’s temporary startup log file, see the startup chapter in the System Installation Guide for your platform.
Chapter 3. Standard configuration properties for connectors

Connectors have two types of configuration properties:
- Standard configuration properties
- Connector-specific configuration properties

This chapter describes standard configuration properties, applicable to all connectors. For information about properties specific to the connector, see the installing and configuring chapter of its adapter guide.

The connector uses the following order to determine a property’s value (where the highest numbers override the value of those that precede):
1. Default
2. Repository (relevant only if InterChange Server is the integration broker)
3. Local configuration file
4. Command line

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

New and deleted properties

The following are the standard properties that have been either added or deleted in the 2.2 release of the adapters.
- **New properties**
  - CharacterEncoding
  - Local
  - JVMMinHeapSize
  - JVMMaxHeapSize
  - JVMMaxStackSize
  - WireFormat
  - MaxEventCapacity
  - DuplicateEventElimination
  - jms.NumConcurrentRequests
  - ContainerManagedEvents
  - jms.Messagebrokername (replaces jms.BrokerName)
- **Deleted properties**
  - RequestTransport
  - PingFrequency
  - TraceLevel
  - AgentProxyType
  - MaxThreadPoolSize
  - Anonymous Connections
  - GW Name
  - Agent URL
Configuring standard connector properties for WebSphere InterChange Server

This section describes standard configuration properties applicable to connectors whose integration broker is WebSphere InterChange Server (ICS). Standard configuration properties provide information that is used by a configurable component of InterChange Server called the connector controller. Like the connector framework, the code for the connector controller is common to all connectors. However, you configure a separate instance of the controller for each connector.

A connector, which consists of the connector framework and the application-specific component, has been referred to historically as the connector agent. When a standard configuration property refers to the agent, it is referring to both the connector framework and the application-specific component.

For general information about how connectors work with InterChange Server, see the Technical Introduction to IBM WebSphere InterChange Server.

Important: Not all properties are applicable to all connectors that use InterChange Server. For information specific to an connector, see its adapter guide.

You configure connector properties from Connector Configurator, which you access from System Manager.

Note: Connector Configurator and System Manager run only on the Windows system. Even if you are running the connector on a UNIX system, you must still have a Windows machine with these tools installed. Therefore, 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 InterChange Server, and bring up Connector Configurator for the connector.

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a runtime session, the property’s update semantics determine how and when the change takes effect. There are four different types of update semantics for standard connector properties:

- Dynamic—The change takes effect immediately after it is saved.
- Component restart—The change takes effect only after the connector is stopped and then restarted in System Manager. This does not require stopping and restarting the application-specific component or InterChange Server.
- Server restart—The change takes effect only after you stop and restart the application-specific component and InterChange Server.
- Agent restart—The change takes effect only after you stop and restart the application-specific component.
To determine the update semantics for a specific property, refer to the Update Method column in the Connector Configurator window, or see the Update Method column of the table below.

The following table provides a quick reference to the standard connector configuration properties. You must set the values of some of these properties before running the connector. See the sections that follow for explanations of the properties.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Possible values</th>
<th>Default value</th>
<th>Update method</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminInQueue</td>
<td>valid JMS queue name</td>
<td>CONNECTORNAME/ADMININQUEUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AdminOutQueue</td>
<td>valid JMS queue name</td>
<td>CONNECTORNAME/ADMINOUTQUEUE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AgentConnections</td>
<td>1-4</td>
<td>1</td>
<td>server restart</td>
<td>multi-threaded connector only</td>
</tr>
<tr>
<td>AgentTraceLevel</td>
<td>0-5</td>
<td>0</td>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>ApplicationName</td>
<td>application name</td>
<td>the value that is specified for the connector name</td>
<td>component restart</td>
<td>value required</td>
</tr>
<tr>
<td>BrokerType</td>
<td>ICS, WMQI</td>
<td></td>
<td></td>
<td>ICS is required if your broker is ICS</td>
</tr>
<tr>
<td>CharacterEncoding</td>
<td>ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437</td>
<td>ascii7</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>ConcurrentEventTriggeredFlows</td>
<td>1 to 32,767</td>
<td>no value</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>ContainerManagedEvents</td>
<td>JMS or no value</td>
<td>JMS</td>
<td>guaranteed event delivery</td>
<td></td>
</tr>
<tr>
<td>ControllerStoreAndForwardMode</td>
<td>true or false</td>
<td>true</td>
<td>dynamic</td>
<td>JMS transport only</td>
</tr>
<tr>
<td>ControllerTraceLevel</td>
<td>0-5</td>
<td>0</td>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>DeliveryQueue</td>
<td></td>
<td>CONNECTORNAME/DELIVERYQUEUE</td>
<td>component restart</td>
<td>JMS transport only</td>
</tr>
<tr>
<td>DeliveryTransport</td>
<td>MQ, IDL, or JMS</td>
<td>IDL</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>FaultQueue</td>
<td></td>
<td>CONNECTORNAME/FAULTQUEUE</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>DuplicateEventElimination</td>
<td>True/False</td>
<td>False</td>
<td>component restart</td>
<td>JMS transport only</td>
</tr>
<tr>
<td>JvmMaxHeapSize</td>
<td>heap size in megabytes</td>
<td>128m</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>JvmMaxNativeHeapSize</td>
<td>size of stack in kilobytes</td>
<td>128k</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>JvmMinHeapSize</td>
<td>heap size in megabytes</td>
<td>1m</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>jms.MessageBrokerName</td>
<td>crossworlds.queue.manager</td>
<td></td>
<td>server restart</td>
<td>JMS transport only</td>
</tr>
<tr>
<td>jms.FactoryClassName</td>
<td>CxCommon.Messaging. CxCommon.Messaging. jms.IBMMQSeriesFactory jms.IBMMQSeriesFactory or CxCommon.Messaging. jms.SonicMQFactory or any Java class name</td>
<td>server restart</td>
<td>JMS transport only</td>
<td></td>
</tr>
<tr>
<td>Property Name</td>
<td>Possible values</td>
<td>Default value</td>
<td>Update method</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------------</td>
<td>---------------</td>
<td>---------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>jms.NumConcurrentRequests</td>
<td>positive integer</td>
<td>10</td>
<td>component</td>
<td>JMS transport only</td>
</tr>
<tr>
<td>jms.Password</td>
<td>Any valid password</td>
<td></td>
<td>restart</td>
<td>JMS transport only</td>
</tr>
<tr>
<td>jms.UserName</td>
<td>Any valid name</td>
<td></td>
<td>server restart</td>
<td>JMS transport only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>restart</td>
<td>Notes: These are only a subset of supported locales.</td>
</tr>
<tr>
<td>LogAtInterchangeEnd</td>
<td>true or false</td>
<td>false</td>
<td>component</td>
<td>Repository Directory value must be &lt;REMOTE&gt;</td>
</tr>
<tr>
<td>MaxEventCapacity</td>
<td>1-2147483647</td>
<td>2147483647</td>
<td>dynamic</td>
<td>JMS transport only, DuplicateEvent Elimination must be True</td>
</tr>
<tr>
<td>MessageFileName</td>
<td>path/filename</td>
<td>Connectorname.text or InterchangeSystem.txt</td>
<td>component</td>
<td>restart</td>
</tr>
<tr>
<td>MonitorQueue</td>
<td>any valid queue name</td>
<td></td>
<td>component</td>
<td>restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CONNECTORNAME/MONITORQUEUE</td>
<td>JMS transport only, DuplicateEvent Elimination must be True</td>
<td></td>
</tr>
<tr>
<td>OADAutoRestartAgent</td>
<td>true or false</td>
<td>false</td>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>OADMaxNumRetry</td>
<td>a positive number</td>
<td>1000</td>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>OADRetryTimeInterval</td>
<td>a positive number in minutes</td>
<td>10</td>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>PollEndTime</td>
<td>HH:MM</td>
<td>HH:MM</td>
<td>component</td>
<td></td>
</tr>
<tr>
<td>PollFrequency</td>
<td>a positive integer in milliseconds</td>
<td>10000</td>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>PollQuantity</td>
<td>1-500</td>
<td>1</td>
<td>component</td>
<td>Number of items to poll from application</td>
</tr>
<tr>
<td>PollStartTime</td>
<td>HH:MM</td>
<td>HH:MM</td>
<td>component</td>
<td></td>
</tr>
<tr>
<td>RepositoryDirectory</td>
<td>location where repository is located</td>
<td>&lt;REMOTE&gt;</td>
<td>component</td>
<td>restart, &lt;REMOTE&gt; for ICS broker</td>
</tr>
<tr>
<td>RequestQueue</td>
<td>valid JMS queue name</td>
<td>CONNECTORNAME/REQUESTQUEUE</td>
<td>component</td>
<td>restart</td>
</tr>
<tr>
<td>ResponseQueue</td>
<td>valid JMS queue name</td>
<td>CONNECTORNAME/RESPONSEQUEUE</td>
<td>component</td>
<td>restart</td>
</tr>
<tr>
<td>RestartRetryCount</td>
<td>0-99</td>
<td>3</td>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>RestartRetryInterval</td>
<td>a sensible positive value in minutes</td>
<td>1</td>
<td>dynamic</td>
<td></td>
</tr>
<tr>
<td>SourceQueue</td>
<td>valid MQSeries queue name</td>
<td>CONNECTORNAME/SOURCEQUEUE</td>
<td>component</td>
<td>restart</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Valid only if delivery transport is JMS and Container Managed Events is specified.</td>
<td></td>
</tr>
<tr>
<td>Property Name</td>
<td>Possible values</td>
<td>Default value</td>
<td>Update method</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>--------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>---------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>SynchronousRequestQueue</td>
<td>CONNECTORNAME/ SYNCHRONOUSREQUESTQUEUE</td>
<td>CONNECTORNAME/ SYNCHRONOUSREQUESTQUEUE</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>SynchronousResponseQueue</td>
<td>CONNECTORNAME/ SYNCHRONOUSRESPONSEQUEUE</td>
<td>CONNECTORNAME/ SYNCHRONOUSRESPONSEQUEUE</td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>SynchronousRequestTimeout</td>
<td>0</td>
<td></td>
<td>component restart</td>
<td></td>
</tr>
<tr>
<td>WireFormat</td>
<td>CwXML, CwBO</td>
<td>cwxml</td>
<td>agent restart</td>
<td>CwXML for non-ICS broker; CwBO if Repository Directory is &lt;REMOTE&gt;</td>
</tr>
</tbody>
</table>

### 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

The AgentConnections property controls the number of IIOP connections opened for request transport between an application-specific component and its connector controller. By default, the value of this property is set to 1, which causes InterChange Server to open a single IIOP connection.

This property enhances performance for a multi-threaded connector by allowing multiple connections between the connector controller and application-specific component. When there is a large request/response workload for a particular connection, the IBM WebSphere administrator can increase this value to enhance performance. Recommended values are in the range of 2 to 4. Increasing the value of this property increases the scalability of the Visigenic software, which establishes the IIOP connections. You must restart the application-specific component and the server for a change in property value to take effect.

**Important:** If a connector is single-threaded, it cannot take advantage of the multiple connections. Increasing the value of this property causes the request transport to bottleneck at the application-specific component.

To determine whether a specific connector is single- or multi-threaded, see the installing and configuring chapter of its adapter guide.

### 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. If you are using an ICS connector, this setting must be ICS.

**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 ASCII for this property. If you previously configured the value of this property to ascii7 or ascii8, you must reconfigure the connector to use either ASCII or one of the other supported values. To determine whether a specific connector is written in Java or C++, see the installing and configuring chapter of its adapter guide.

*Important:* By default only a subset of supported character encodings display in the drop list. To add other supported values to the drop list, you must manually modify the `Data\Std\stdConnProps.xml` file in the product directory. For more information, see the appendix on Connector Configurator.

*Attention:* Do not run a non-internationalized connector against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed.

The default value is `ascii`.

**ConcurrentEventTriggeredFlows**

Determines how many business objects can be concurrently processed by the connector controller 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 controller for a source application to simultaneously map multiple event business objects, and to simultaneously deliver them to multiple collaboration instances. Setting this property to enable concurrent mapping of multiple business objects can speed delivery of business objects to a collaboration, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

*Note:* To implement concurrent processing for an entire flow (from a source application to a destination application) also requires that the collaboration be configured to use multiple threads and that the destination application's application-specific component be able to process requests concurrently. To configure the collaboration, set its Maximum number of concurrent events property high enough to use multiple threads. For an application-specific component to process requests concurrently, it must be either
multi-threaded, or be capable of using Connector Agent Parallelism and be configured for multiple processes (setting the Parallel Process Degree configuration property greater than 1).

**Important:** To determine whether a specific connector is single- or multi-threaded, see the installing and configuring chapter of its adapter guide.

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

**ContainerManagedEvents**

Setting this property to JMS 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. This property can also be set to no value.

**Notes:**

1. When ContainerManagedEvents is set to JMS, you must also configure the following properties to enable guaranteed event delivery: PollQuantity = 1 to 500, SourceQueue = SOURCEQUEUE. In addition, you must configure a data handler with the MimeType, DHClass, and DataHandlerConfigMOName (optional) properties. To set those values, use the Data Handler tab of Connector Configurator. The fields for the values under the Data Handler tab will be displayed only if you have set ContainerManagedEvents to JMS.

2. When ContainerManagedEvents is set to JMS, the connector does not call its pollForEvents() method, thereby disabling that method’s functionality.

The default value is JMS.

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

**ControllerStoreAndForwardMode**

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 InterChange Server, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forward the request to it.

**Important:** 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**

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

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

The default value is `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 ICS is the broker type, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.

If WMQI is the broker type, JMS is the only possible Delivery Transport value.

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 have compelling reasons not to license and maintain two separate products. 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 the overhead of 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 controller 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," display 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
- InterChange Server (ICS) as the Integration broker
In this environment, you may experience difficulty starting the 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.

Notes:

- If your installation uses more than 768M of process heap size, this resolution would adversely affect product performance.

- If you run on AIX 4.3.3, you do not need to set the LDR_CNTRL environment variable. However, you must set IPCCBaseAddress to a value of 11 or 12.

**DuplicateEventElimination**

Setting this property to true enables a JMS-enabled connector with a non-JMS event store to ensure that duplicate events are not delivered to the delivery queue. To make use of this feature, during connector development a unique event identifier must be set as the business object’s ObjectEventId attribute in the application specific code.

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.

**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.

**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. For more information, see the overview chapter of the connector guide for an internationalized connector.

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.

The default is en_US.

**Important:** By default only a subset of supported locales display in the drop list. To add other supported values to the drop list, you must manually modify the \Data\Std\stdConnProps.xml file in the product directory. For more information, see the appendix on Connector Configurator.

**Attention:** If the connector has not been internationalized, the only valid value for this property is en_US. Do not run a non-internationalized C++ connector against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed. To determine whether a specific connector has been internationalized, see the installing and configuring chapter of its connector guide.

**LogAtInterchangeEnd**

Specifies whether to log errors to InterChange Server’s log destination, in addition to the location specified in the LogFileName property. Logging to the server’s log destination also turns on email notification, which generates email messages for the MESSAGE_RECIPIENT specified in the InterchangeSystem.cfg file when errors or fatal errors occur. As an example, when a connector loses its connection to its application, if LogAtInterChangeEnd is set to true, an email 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. 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.

**Important:** To determine whether a specific connector has its own message file, see the installing and configuring chapter of its adapter guide.

**OADAutoRestartAgent**

Specifies whether the Object Activation Daemon (OAD) automatically attempts to restart the application-specific component after an abnormal shutdown. The properties “OADMaxNumRetry” on page 56 and “OADRetryTimeInterval” on page 56 are related to this property. This property is required for automatic restart.
The default value is "false.

**OADMaxNumRetry**

Specifies the maximum number of times that the OAD automatically attempts to restart the application-specific component after an abnormal shutdown.

The default value is 1000.

**OADRetryTimeInterval**

Specifies the number of minutes of the retry time interval that the OAD automatically attempts to restart the application-specific component after an abnormal shutdown. If the application-specific component does not start within the specified interval, the OAD repeats the attempt as many times as specified in "OADMaxNumRetry".

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**

The amount of time between polling actions. Set PollFrequency to one of the following values:

- The number of milliseconds between polling actions.
- 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. To determine whether a specific connector does, see the installing and configuring chapter of its adapter guide.

**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 REQUESTQUEUE.
RepositoryDirectory

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

When the integration broker is ICS, this value must be set to <REMOTE> because the connector uses the InterChange Server repository to obtain its connector-definition information.

ResponseQueue

Designates the JMS response queue, which delivers a response message from the connector framework to the integration broker. When the integration broker is InterChange Server, InterChange 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.

The default is 1.

SourceQueue

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 51.

The default value is SOURCEQUEUE.

SynchronousRequestQueue

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.

SynchronousResponseQueue

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.
**SynchronousRequestTimeout**

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.

**TraceFileName**

The name of the file where the application-specific component writes trace messages. Specify the filename in an absolute path. The default is STDOUT.

**WireFormat**

Message format on the transport.

Possible values are:
- CWXML if the broker is not ICS.
- CWB0 if the value of RepositoryDirectory is <REMOTE>.

---

## Configuring standard connector properties for WebSphere MQ Integrator

This section describes standard configuration properties applicable to adapters whose integration broker is WebSphere MQ Integrator Broker. For information on using WebSphere Integrator Broker, see the Implementation Guide for WebSphere MQ Integrator Broker.

**Important:** Not all properties are applicable to all connectors that use WebSphere MQ Integrator Broker. For information specific to a connector, see its adapter user guide.

You configure connector properties from Connector Configurator.

**Note:** Connector Configurator runs only on the Windows system. Even if you are running the connector on a UNIX system, you must still have a Windows machine with this tool installed. Therefore, to set connector properties for a connector that runs on UNIX, you must run Connector Configurator on the Windows computer and copy the configuration files to the UNIX computer using FTP or some other file transfer mechanism. For more information about Connector Configurator, see Appendix B, "Connector Configurator."

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a runtime session, you must restart the connector. Standard configuration properties provide information that is used by the adapter framework and connector framework, and is common to all connectors.

### Standard connector properties

The following table provides a quick reference for standard connector configuration properties. See the sections that follow for explanations of the properties.
<table>
<thead>
<tr>
<th>Name</th>
<th>Possible values</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdminInQueue</td>
<td>valid JMS queue name</td>
<td>CONNECTORNAME/ADMININQUEUE</td>
</tr>
<tr>
<td>AdminOutQueue</td>
<td>valid WebSphere MQ queue name</td>
<td>CONNECTORNAME/ADMINOUTQUEUE</td>
</tr>
<tr>
<td>AgentTraceLevel</td>
<td>0-5</td>
<td>0</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>application name</td>
<td>AppNameConnector</td>
</tr>
<tr>
<td>BrokerType</td>
<td>WMQI</td>
<td>WMQI</td>
</tr>
<tr>
<td>CharacterEncoding</td>
<td>ASCII, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437</td>
<td>ASCII</td>
</tr>
<tr>
<td></td>
<td>Note: These are only a subset of supported values.</td>
<td></td>
</tr>
<tr>
<td>ContainerManagedEvents</td>
<td>JMS or no value</td>
<td>JMS</td>
</tr>
<tr>
<td>DeliveryQueue</td>
<td>valid WebSphere MQ queue name</td>
<td>CONNECTORNAME/DELIVERYQUEUE</td>
</tr>
<tr>
<td>DeliveryTransport</td>
<td>JMS</td>
<td>JMS</td>
</tr>
<tr>
<td>DuplicateEventElimination</td>
<td>true, false</td>
<td></td>
</tr>
<tr>
<td>FaultQueue</td>
<td>valid WebSphere MQ queue name</td>
<td>CONNECTORNAME/FAULTQUEUE</td>
</tr>
<tr>
<td>jms.FactoryClassName</td>
<td>If FactoryClassName is IBM, use crossworlds.queue.manager. If FactoryClassName is Sonic, use localhost:2506.</td>
<td>CxCommon.Messaging.jms. IBMMQSeriesFactory</td>
</tr>
<tr>
<td>jms.MessageBrokerName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jms.NumConcurrentRequests</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>jms.Password</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jms.UserName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MessageFileName</td>
<td>path/filename</td>
<td>InterchangeSystem.txt</td>
</tr>
<tr>
<td>PollEndTime</td>
<td>HH:MM</td>
<td>HH:MM</td>
</tr>
<tr>
<td>PollFrequency</td>
<td>milliseconds/key/no</td>
<td>10000</td>
</tr>
<tr>
<td>PollStartTime</td>
<td>HH:MM</td>
<td>HH:MM</td>
</tr>
<tr>
<td>RepositoryDirectory</td>
<td>path/directory name Note: Typically you must change this value from the default to whatever path and directory name was actually used when you installed the the connector files.</td>
<td>C:\crossworlds\Repository</td>
</tr>
<tr>
<td>RequestQueue</td>
<td>valid WebSphere MQ queue name</td>
<td>CONNECTORNAME/REQUESTQUEUE</td>
</tr>
<tr>
<td>ResponseQueue</td>
<td></td>
<td>RESPONSEQUEUE</td>
</tr>
<tr>
<td>RestartRetryCount</td>
<td>0-99</td>
<td>3</td>
</tr>
<tr>
<td>RestartRetryInterval</td>
<td>an appropriate integer indicating the number of minutes between restart attempts</td>
<td>1</td>
</tr>
<tr>
<td>Name</td>
<td>Possible values</td>
<td>Default value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SourceQueue</td>
<td>valid WebSphere MQ queue name</td>
<td>CONNECTORNAME/SOURCEQUEUE</td>
</tr>
<tr>
<td>SynchronousRequestQueue</td>
<td>valid WebSphere MQ queue name</td>
<td></td>
</tr>
<tr>
<td>SynchronousResponseQueue</td>
<td>valid WebSphere MQ queue name</td>
<td></td>
</tr>
<tr>
<td>SynchronousTimeout</td>
<td>an appropriate integer indicating the number of minutes the connector waits for a response to a synchronous request</td>
<td>0</td>
</tr>
<tr>
<td>WireFormat</td>
<td>CwXML</td>
<td>CwXML</td>
</tr>
</tbody>
</table>

**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.

**AgentTraceLevel**

Level of trace messages for the connector’s 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 connection to the application. This name is used by the system administrator to monitor the connector’s environment. When you create a new connector definition, this property defaults to the name of the connector; when you work with the definition for an IBM WebSphere-delivered connector, the property is also likely to be set to the name of the connector. Set the property to a value that suggests the program with which the connector is interfacing, such as the name of an application, or something that identifies a file system or website in the case of technology connectors.

**BrokerType**

This property is set to the value WMQI for connectors that are configured to use WebSphere MQ Integrator Broker as the integration broker.

**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 ASCII for this property. If you previously configured the value of this property to ascii7 or ascii8, you must reconfigure the connector to use either ASCII or one of the other supported values. To determine whether a specific connector is written in Java or C++, see the installing and configuring chapter of its adapter guide.

**Important:** By default only a subset of supported character encodings display in the drop list. To add other supported values to the drop list, you must
manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, see the appendix on Connector Configurator.

**Attention:** Do not run a non-internationalized connector against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed.

The default value is `ascii`.

**ContainerManagedEvents**
Setting this property to `JMS` enables 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. This property can also be set to no value.

**Notes:**
1. When `ContainerManagedEvents` is set to `JMS`, you must also configure the following properties to enable guaranteed event delivery: `PollQuantity = 1` to `500`, `SourceQueue = SOURCEQUEUE`. In addition, you must configure a data handler with the `MimeType`, `DHClass`, and `DataHandlerConfigMOMName` (optional) properties.
2. When `ContainerManagedEvents` is set to `JMS`, the connector does not call its `pollForEvents()` method, thereby disabling that method’s functionality.

The default value is `JMS`.

**DeliveryQueue**
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. The property defaults to the value `JMS`, indicating that the Java Messaging Service (JMS) is used for communication with WebSphere MQ Integrator. This property must be set to `JMS` when WebSphere MQ Integrator Broker is the integration broker. Otherwise, the connector cannot start.

**DuplicateEventElimination**
Setting this property to `true` enables a JMS-enabled connector with a non-JMS event store to ensure that duplicate events are not delivered to the delivery queue. To make use of this feature, during connector development a unique event identifier must be set as the business object’s `ObjectEventId` attribute in the application specific code.

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.

**jms.FactoryClassName**
Specifies the class name to instantiate for a JMS provider.

The default is CxCommon.Messaging.jms.IBMMQSeriesFactory.

**jms.MessageBrokerName**
Specifies the broker name to use for the JMS provider.

The default is crossworlds.queue.manager.

**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.

**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. For more information, see the overview chapter of the connector guide for an internationalized connector.

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.

The default is en_US.

**Important**: By default only a subset of supported locales display in the drop list. To add other supported values to the drop list, you must manually modify the \Data\Std\stdConnProps.xml file in the product directory.
Attention:

- WebSphere MQ Integrator supports only one locale at a time. Ensure that every component of the installation (for example, all adapters, applications, and the integration broker itself) is set to the same locale.
- If the connector has not been internationalized, the only valid value for this property is en_US. Do not run a non-internationalized C++ connector against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed. To determine whether a specific connector has been internationalized, see the installing and configuring chapter of its connector guide.

**MessageFileName**
The name of the connector message file. The standard location for the message file is \connectors\messages. Specify the message filename in an absolute path if the message file is not located in the standard location. This property defaults to the value InterchangeSystem.txt for new connector definitions and should be changed to the name of the message file for the specific connector.

**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**
The amount of time between polling actions. Set the PollFrequency to one of the following values:
- The number of milliseconds between polling actions.
- 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.

**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.

**RepositoryDirectory**
The path and name of the directory from which the connector reads the XML schema documents that store the meta-data of business object definitions.

The default value is C:\crossworlds\repository. You must change this to the directory path that you are using for the \repository directory for your connector. Typically that path is established when you install the adapter product; for
example, C:\WebSphereAdapters\repository. The value must be a directory path. Do not use <REMOTE> as the RepositoryDirectory value for a connector that is not using ICS as the broker.

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

The default value is CONNECTORNAME/REQUESTQUEUE.

**ResponseQueue**
Designates the JMS response queue, which delivers a response message from the connector framework to the integration broker.

**RestartRetryCount**
Specifies the number of times the connector attempts to restart itself. The default value is 3, indicating that the connector tries to restart 3 times. For instance, if a connector is unable to log in to an application it fails to start, but with this property set to the value 3 the connector tries a total of three times to start. When used in conjunction with the "RestartRetryInterval" property, this behavior enables a connector to make several attempts at communicating with an application that might not reliably have a connection available all the time.

**RestartRetryInterval**
Specifies the interval in minutes at which the connector attempts to restart itself. The default value is 1, indicating that the connector waits 1 minute in between its restart attempts.

**SourceQueue**
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 51.

The default is CONNECTORNAME/SOURCEQUEUE.

**SynchronousRequestQueue**
Delivers request messages that require a synchronous response from the connector framework to WebSphere MQ Integrator 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 WebSphere MQ Integrator Broker on the SynchronousResponseQueue. The response message sent to the connector bears a correlation ID that matches the ID of the original message.

**SynchronousResponseQueue**
Delivers response messages sent in reply to a synchronous request from WebSphere MQ Integrator Broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

**SynchronousTimeout**
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**
The data format for messages exchanged by the connector. The default value `CwXML` is the only valid value, and directs the connector to compose the messages in XML.
Chapter 4. Creating or modifying business objects

- “Connector business object structure”
- “Error handling” on page 68
- “Generating Telcordia business objects based on schema documents” on page 70

The connector comes with sample business objects only. The systems integrator, consultant, or customer must build business objects.

The connector is a meta-data-driven connector. In IBM WebSphere InterChange Server business objects, meta-data is data about the application, which is stored in a business object definition and which helps the connector interact with an application. A meta-data-driven connector handles each business object that it supports based on meta-data encoded in the business object definition rather than on instructions hard-coded in the connector.

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

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

Connector business object structure

After installing the connector, you must create business objects or use (or modify) those shipped with the connector.

The Telcordia connector can processes Telcordia schema files. Each schema file has a corresponding IBM WebSphere InterChange Server business object. The business objects are generated using the XML Object Discovery Agent (ODA). You must specify a prefix, such as Telcordia_, for these objects; follow the naming conventions specified in the TelcordiaPrefix_SchemaFileType. The six tested adapter-specific business objects are:

- Telcordia_OrderRequest
- Telcordia_OrderResponse
- Telcordia_LSRRequest
- Telcordia_LSRResponse
- Telcordia_BillingOrderCompletionResponse
- Telcordia_BillingOrderCompletionContract

For more on generating business objects using the XML ODA, see “Generating Telcordia business objects based on schema documents” on page 70.
There are no additional requirements regarding the structure of the business objects other than those imposed by the configured data handler. For more on naming conventions see the *Naming Components Guide*.

The connector retrieves messages from a queue and attempts to populate a business object (defined by the meta-object) with the message contents. Strictly speaking, the connector neither controls nor influences business object structure. Those are functions of meta-object definitions as well as the connector’s data handler requirements. In fact, there is no business-object level application information. Rather, the connector’s main role when retrieving and passing business objects is to monitor the message-to-business-object (and vice versa) process for errors.

**Error handling**

All error messages generated by the connector are stored in a message file named `TelcordiaConnector.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 connector handles specific errors as described in the following sections.

**Ill-formed and invalid requests**

There are two ways to handle requests that cannot be processed:

- When Telcordia Service Delivery polls an ill-formed message from the Request Queue (IBM_2_DELV) and the message cannot be processed, the message is sent to the predefined error queue, DELV_ERROR_QUEUE. The ICS integration broker will not receive a response.

- Upon receipt of an invalid request from the connector, Telcordia Service Delivery generates a response message that contains a `<MSG>` (message) section:
  ```xml
  <MSG> <CODE> 0173 </CODE> <TEXT> LSO NOT FOUND IN WC MAPING REFERENCE TABLE </TEXT> <ERRTAGPATH> DELVSR.RU.DATA.CKT.NEWACL.LSO </ERRTAGPATH> </MSG>
  ```

  Such responses or order state changes must trigger a collaboration object to return to the ICS integration broker.

**Application timeout**

The error message `ABON_APPRESPONSETIMEOUT` is returned when:

- The connector cannot establish a connection to the JMS service provider 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 connector issues a message but times out waiting for a response for a business object with conversion property `TimeoutFatal` equal to `True`.

- The connector receives a response message with a return code equal to `APP_RESPONSE_TIMEOUT` or `UNABLE_TO_LOGIN`. 
Unsubscribed business object

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

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

When a NO_SUBSCRIPTION_FOUND code is returned by the gotApplEvent() method, the connector sends the message to the queue specified by the UnsubscribedQueue property and continues processing other events.

Connector not active

When the gotApplEvent() method returns a CONNECTOR_NOT_ACTIVE code, the pollForEvents() method returns an APP_RESPONSE_TIMEOUT code and the event remains in the InProgress queue.

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 the JMS provider), 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 connector behavior. Trace messages, by default, are written to STDOUT. See the connector configuration properties in Chapter 2, "Configuring the connector", on page 17 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 connector trace messages.

Level 0 This level is used for trace messages that identify the connector 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 connector enters or exits a function.

Level 5 Use this level for trace messages that indicate connector
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.

Generating Telcordia business objects based on schema documents

If you are creating request and response business objects based on an XML schema document, you must create a business object definition for each type of XML document to be processed. The business object definition contains structure information that is contained in the XML document’s schema. For example, if there is one request stream (a single schema document), but four possible response stream types (four separate schema documents), you must define five business object definitions. On the other hand, if the request and response stream use the same schema, you need only one business object definition. You can use the XML Object Discovery Agent (ODA) to generate business object definitions based on schema documents.

For information about how to define business object definitions for XML documents, either using the XML ODA or manually, see the Data Handler Guide.

ODA limitation

The XML ODA does not support schemas that include multiple files for element definitions with similar names. Accordingly, the XML ODA will not correctly generate IBM WebSphere business object definitions for such schemas. To work around this problem:

1. Identify elements that are being used in multiple included files with different definitions for the same name.
2. Rename the element name in one of the included file by appending “1” at the end of the name and save the modified file.
3. Use XML ODA to generate CW Business Object definition for the schema and save the file to a .in file.
4. Open the .in file and replace the element name back to the original name by removing all “1”s and re-save the .in file.
5. Load in the business object definition file into your repository and things should work now.

For example, schema file BilingOrderCompletionContract.xsd, includes two files, CRS.xsd and ORDERINP.xsd. And ORDERINP.xsd includes DELV.xsd which also includes DELVTAGS.xsd. The workaround for this example is as follows:

1. Identify element definitions with similar names: both CRS.xsd and DELVTAGS.xsd have an element called SLOT
2. Rename element SLOT to SLOT1 in CRS.xsd and save the file.
3. Use the XML ODA to generate business object definition for BilingOrderCompletionContract schema and save the definition to BillingContract.in.
4. Open BillingContract.in in a word processing application, replace SLOT1 with SLOT, and re-save the file.
5. Repos_Copy BillingContract.in into your repository.

Note: With release 2.3 of the adapter for Telcordia, the XML ODA cannot automatically select a key attribute for the top-level business object. For business objects at all other levels, the XML ODA sets the first attribute as the key. Accordingly, when you save XML ODA-generated objects in
Business Object Designer, an error message informs you that the top-level object is missing a key attribute. Assign a key attribute that reflects your business data and business object requirements, then re-save the objects.
Chapter 5. Troubleshooting

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

- “Start-up problems”
- “Event processing problems”

Start-up problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Potential solution / explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The connector shuts down unexpectedly during initialization and the following message is reported: Exception in thread &quot;main&quot; java.lang.NoClassDefFoundError: javax/jms/JMSException...</td>
<td>Connector 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.</td>
</tr>
<tr>
<td>The connector shuts down unexpectedly during initialization and the following message is reported: Exception in thread &quot;main&quot; java.lang.NoClassDefFoundError: com/ibm/mq/jms/MQConnectionFactory...</td>
<td>Connector cannot find file com.ibm.mq.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.</td>
</tr>
<tr>
<td>The connector shuts down unexpectedly during initialization and the following message is reported: Exception in thread &quot;main&quot; java.lang.NoClassDefFoundError: javax/naming/Referenceable...</td>
<td>Connector 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.</td>
</tr>
<tr>
<td>The connector shutdowns unexpectedly during initialization and the following exception is reported: java.lang.UnsatisfiedLinkError: no mqjbnd01 in shared library path</td>
<td>Connector 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. Explicitly set values for the following properties: HostName, Channel, and Port.</td>
</tr>
</tbody>
</table>

The connector reports MQJMS2005: failed to create MQQueueManager for ':'

Event processing problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Potential solution / explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>The connector delivers all messages with an MQRFH2 header.</td>
<td>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, “Configuring the connector”, on page 17 for more information.</td>
</tr>
</tbody>
</table>

The connector truncates all message formats to 8-characters upon delivery regardless of how the format has been defined in the connector meta-object. This is a limitation of the WebSphere MQ MQMD message header and not the connector.
Appendix A. Connector configurator

Before you can use a connector, you must create a connector configuration file that sets the properties for the connector, designates the business objects and any meta-objects that it supports, and sets logging and tracing values that the connector will use at runtime. The configuration file may also contain properties for the use of messaging and data handlers required by your connector.

Use Connector Configurator to create and modify the configuration file for your connector. If a configuration file has previously been created for your connector, you can use Connector Configurator to open the file and modify its settings. If no configuration file has yet been created for your connector, you can use Connector Configurator to both create the file and set its properties.

When you complete a connector configuration file, the file is saved as an XML document. You will save the XML document either as a project in System Manager (if ICS is your broker) or as a file with a *.cfg extension in a directory folder (if WebSphere MQ Integrator Broker is your broker, or if you are using the file as a local configuration file for ICS).

This appendix describes how to use Connector Configurator to:

• Create a connector-specific property template for configuring your connector
• Create a configuration file
• Set properties in a configuration file

Connector Configurator runs only in a Windows environment. If you are running the connector itself in a UNIX environment, use Connector Configurator in the Windows system in the network to modify the configuration file. Then copy the file to your UNIX environment.

Note: Some properties in the connector configuration file use directory paths, and these paths default to the Windows convention for directory paths. If you use the connector configuration file in a UNIX environment, revise any directory path constructs in the configuration properties to match the UNIX convention for directory paths.

Using Connector Configurator in an internationalized environment

Connector Configurator is internationalized and handles 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:

```xml
<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>
```

**Starting Connector Configurator**

Connector Configurator can be started and run in either of two modes:

- Launched from System Manager
- Independent of System Manager (stand-alone mode)

**Running Configurator from System Manager**

When you run Connector Configurator in conjunction with System Manager, you can

- Save connector configuration files (XML documents with the extension *.cfg) to a directory that you specify, and
- Save connector configuration files as components of System Manager projects. If you are using ICS as your broker, this is a mandatory step before you deploy your configuration into the ICS.

**Note:** When you save a configuration file as a component of a System Manager project, the file is stored in the designated project as an XML document file with the extension *.con. It is not advisable to open the *.con file and edit it directly; instead, make any changes by opening the component in System Manager.

To run Connector Configurator with System Manager, do any of the following:

- In System Manager, right-click on the Connector folder of the Integration Components Library (to create a new configuration), or right-click on a connector configuration component within the Connector folder (to edit an existing configuration), or
- From the System Manager menu, choose Tools>Connector Configurator, or
- With System Manager already running, from Start>Programs choose IBM WebSphere InterChange Server>IBM WebSphere Business Integration Toolset>Development>Connector Configurator.

For details about using projects in System Manager and deploying to InterChange Server, see the Implementation Guide for WebSphere InterChange Server.
Running Configurator independently of System Manager

When you run Connector Configurator without connecting to System Manager, you can save a connector configuration file (an XML document with the extension *.cfg) to a directory that you specify, but you cannot save or open a System Manager project.

When you are creating a connector for use with a broker other than ICS, you do not need to connect to System Manager at any point in order to use the file. If you are creating a connector configuration for use with ICS as the broker, you may still find it useful on occasion to run Connector Configurator independently, and then connect to System Manager when you are ready to save the configuration file as a component of a System Manager project.

Choosing your broker

Connector Configurator can be used to configure connectors either for use with ICS as the broker, or with WebSphere MQ Integrator Broker (also referred to as WMQI) as the broker.

Before you begin to configure the connector, you must choose the mode of Connector Configurator that is appropriate for your broker. The mode that you choose determines the properties that Connector Configurator will include in the configuration file. Choosing a broker is a mandatory step when you begin the process of creating a completely new configuration file. After a configuration file has been created, you can optionally change the designated broker mode, using a standard configuration property. (This makes it possible to use an existing configuration file as a starting point for creating a configuration file that will be used with a different broker. However, be aware that revising a configuration file for use with a different broker typically involves changing other configuration properties as well, and not just the broker mode property.)

To choose a broker when you create a new configuration file (mandatory):

- In the Connector Configurator home menu, choose File>New>Connector Configuration. The New Connector Dialog displays.
- In the Integration Broker field, choose either WMQI connectivity (for WebSphere Integrator Broker) or ICS connectivity, according to the broker you are using.
- Complete the remaining fields of the New Connector dialog, as described later in this chapter for your specific broker.

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 Broker Type field of the Standard Properties tab, choose the value that is appropriate for your broker. If you change the existing value, the available tabs and field selections of the properties screen will immediately refresh, to show only those tabs and fields that appropriate for a configuration using the broker you have selected.

After you have chosen your broker type, you can complete the remaining Connector Configurator tasks for configuring your connector. When you save the connector configuration file, Connector Configurator will save it in the broker mode that you have already selected. The title bar of Connector Configurator always displays the broker mode (such as ICS or WMQI) that Connector Configurator is currently using.
After you have completed the configuration file and set its properties, it will need to be deployed to the appropriate location for your connector.

- If you are using ICS as your broker, save the configuration in a System Manager project, and use System Manager to load the file into InterChange Server.
- If you are using WebSphere MQ Integrator Broker as your broker, manually copy the configuration file to its appropriate location, which must match exactly the configuration file location specified in the startup file for your connector.

For further information about deployment, see the Implementation Guide for WebSphere InterChange Server (for using the connector with ICS as the broker), or the Implementation Guide for WebSphere MQ Integrator Broker (for using the connector with MQ Integrator as the broker).

---

### Using a connector-specific property template

To create a configuration file for your connector, you can start with a previously created connector configuration file (*.cfg), a connector definition file (*.txt) or a repository file (*.in or *.out), if any of these already exist for your connector. For instructions on using such existing files, see “Using an existing file” on page 82.

If none of those files exist, or if they are too dissimilar to the configuration requirements of your connector, you can start instead by creating a template for the connector-specific properties of your connector. You’ll create properties in the template, define general characteristics and values for those properties, and specify any dependencies between the properties. Then you’ll save the template and use it as the base for creating a new connector configuration file.

### Creating a template of connector-specific properties

To create a template:

2. The Connector-Specific Property Template dialog appears, with the following fields
   - **Name**
     Enter a unique name that identifies the connector, or type of connector, for which this template will be used. You will see this name again when you open the dialog for creating a new configuration file from a template.
   - **Find Template, and Template Name**
     The names of all currently available templates are displayed in the Template Name display. Look for an existing template that would make a good starting point for your new connector template (such as a template whose property definitions are a subset of the properties used by your connector). To see the connector-specific property definitions that are contained in any template, select that template’s name in the Template Name display. A list of the property definitions contained in that template will appear in the Template Preview display.
     If you do not see any template that displays the connector-specific properties that are used by your connector, you will need to create one. Connector Configurator provides a template named None, containing no property definitions, as a default choice.
     Choose a template from the Template Name display, enter that template name in the Find Name field (or highlight your choice in Template Name), and choose Next.
Specifying general characteristics

The Properties - Connector-Specific Property Template dialog appears. The dialog has tabs for General characteristics of the defined properties and for Value restrictions. The General display has the following fields:

- **Edit properties**
  Use the buttons provided (or right-click within the Edit properties display) to add a new property to the template, to edit or delete an existing property, or to add a child property to an existing property.

A child property is a property that is an attribute of another property— the ‘parent’ property. The parent property can obtain values, or child properties, or both. These property relationships are commonly referred to as 'hierarchical' properties. Later, when you create a configuration file from these properties, Connector Configurator will identify hierarchical property sets with a plus sign in a box at the left of any parent property.

- **Property type**
  Choose one of these property types: Boolean, String, Integer, or Time.

- **Flags**
  You can set Standard Flags (IsRequired, IsDepracated, IsOverridden) or Custom Flags (for Boolean operators) to apply to this property.

After you have made selections for the general characteristics of the property, choose 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. To do so:

1. Choose 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, make any necessary changes. Note that the changes will not be accepted until and unless you also open the Property Value dialog for the property, described in the next step.
4. Right-click the box in the left-hand corner of the Value display panel. A Property Value dialog displays. Depending on the type of the property, the dialog allows you to enter either a value, or both a value and range. Enter the appropriate value or range, and click OK.
5. The Value panel refreshes to display any changes you made in Max Length and Max Multiple Values, and it displays a table with three columns:
   - **Value column** shows the value that you entered in the Property Value dialog, and any previous values that you created.
   - **Default Value column** allows you to designate any of the values as the default.
   - **Value Range** shows the range that you entered in the Property Value dialog.

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 choose EditValue.
Setting dependencies

After you have finished making changes in both the General and the Value tabs, choose Next. The Dependencies dialog 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. To designate a property as being dependent and 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, choose 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 configuration file from a connector-specific template

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

2. The New Connector dialog appears, with the following fields:
   - Name
     Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, must end with the word “connector”, and must be consistent with the file name for a connector that is installed on the system; for example, enter PeopleSoftConnector if the connector file name is PeopleSoft.jar.
   - System Connectivity
     Choose ICS or choose WMQI (for WebSphere MQ Integrator Broker) connectivity.
   - Select Connector-Specific Property Template
     Type the name of the template that has been designed for your connector. The names of all available templates are displayed 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.

After you have chosen the template you want to use, choose OK.

3. A configuration screen will display for the connector that you are configuring. The title bar of the configuration screen shows the broker that you are using and the name that you have given to the connector. You can fill in all the field values to complete the definition now, or you can save the file and complete the fields later.

When you are using the configuration screen, you can, if you wish, add additional connector-specific properties, as described under “Setting application-configuration properties (ICS)” on page 84. Any such additions become part of the configuration file that you are creating, but do not affect the template that you used in creating the file.

4. To save the file, choose File > Save > to File or File > Save > Save to the project. To save to a project, you must be using ICS as the broker, and System Manager must be running. If you save as a file, the Save File Connector dialog displays. 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 choose 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 for your broker later in this chapter.

Using Connector Configurator with ICS as the broker

To use Connector Configurator to configure a connector that will be used with ICS, first select ICS as the broker mode in which you are running Connector Configurator, as described under “Choosing your broker” on page 77.

In a typical ICS implementation, the configuration file that you create with Connector Configurator is not put into use until after you have deployed it to the ICS server. You will perform that deployment (described in the Implementation Guide for WebSphere InterChange Server) after you have finished using Connector Configurator to complete the connector configuration file.

Completing a configuration file

This topic assumes that you already have a starting point for your connector configuration, either from an existing file (a connector definitions file, a repository file, or a *.cfg file) or from an existing project in System Manager. If you do not, see “Creating a template of connector-specific properties” on page 78.

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

The title of the configuration screen displays the type of the broker and the name of the connector as specified in the file. Make sure the title indicates the
appropriate type for your broker—either ICS or WebSphere MQ Integrator Broker (for WMQI). If it does 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 WMQI or ICS.
2. The Standard Properties tab refreshes to display properties associated with the selected broker. When you save the file, you retain this broker selection. You can save the file now or proceed to complete the remaining configuration fields, as described in “Setting the configuration file properties (WebSphere MQ Integrator Broker)” on page 88.
3. When you have finished making entries in the configuration fields, choose View > To File or File > Save To File.

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

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

Before it saves the file, Connector Configurator validates 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.

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 save the file as a configuration file (*.cfg file).

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

1. In Connector Configurator, choose File > Open > From File.
2. In the Open File Connector dialog, choose 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 display 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 choose Open.

Using an existing System Manager project

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. Choose File > Open > From Project.

Setting the configuration file properties (ICS)

The topics in this section apply if you are using InterChange Server as the
integration broker. If you are using WebSphere MQ Integrator Broker as the
integration broker, see "Setting the configuration file properties (WebSphere MQ
Integrator Broker)" on page 88. 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 all of these categories:
1. Standard Properties
2. Connector-Specific Properties
3. Supported Business Objects
4. Associated Maps
5. Resources
6. Trace/Log File values
7. Messaging (where applicable)
8. Data handlers (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.

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-configuration (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 chapter of each adapter guide describes 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 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 informational and not configurable. This field specifies the action required to activate a property whose value has changed.

**Setting standard connector properties (ICS)**

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 choose from the drop-down menu if one appears.
3. After entering all values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator, choose File > Exit (or close the window), and choose No when prompted to save changes.
   - To enter values for other categories in Connector Configurator, choose the tab for the category. The values you enter for Standard Properties (or 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 of the categories as a whole.
   - To save the revised values, choose File > Exit (or close the window) and choose Yes when prompted to save changes. Alternatively, choose Save > To File from either the File menu or the toolbar.

**Setting application-configuration properties (ICS)**

For application-specific configuration properties, you can add or change property names, configure values, delete a property, and encrypt a property:

1. Right click in the top-left portion of the grid. A pop-up menu bar will appear. Select Add to add a property or Add Child to add a child property for a property.
2. Enter a value for the property or child property.
3. To encrypt a property, click 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 (ICS)
Application-specific properties can be encrypted by clicking the Encrypt check box in the Edit Property window. To decrypt a value, click to clear the Encrypt check box, enter the correct value in the Verification dialog box, and choose OK. If the entered value is correct, the value is decrypted and displays. The adapter 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 click the Encrypt check box, all values of the property will encrypted. To decrypt multiple values of a property, click to clear the Encrypt check box of the first value of the property, and then enter the correct value of the first value in the Verification dialog box. If the input value is a match, all multiple values will decrypt.

Update method (ICS)
When WebSphere MQ Integrator Broker is the integration broker, connector properties are static. The Update Method is always Connector Restart. In other words, for changes to take effect, you must restart the connector after saving the revised connector configuration file.

Specifying supported business object definitions (ICS)
This topic assumes that you have already created or acquired the intended business objects, created or acquired maps for them, and have saved both the business object definitions and map definitions into System Manager projects.

Before you can make use of a connector (and before you can bind the connector with a collaboration’s ports), you must make selections under the Supported Business Objects tab to specify the business objects that the connector will use. You must specify both generic business objects and corresponding 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.

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

**Business object name**
These instructions assume that you started Business Object Designer with System Manager running.

To designate that a business object definition is supported by the connector:
1. Click in an empty field of 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, choose Save to Project. The revised connector definition, including designated support for the added business object definition, is saved to the 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, choose Delete Row. The business object is removed from the list display.
3. From the File menu, choose Save to Project.

Note that deleting a business object from the supported list does not affect the code of the connector, nor does it remove the business object definition itself from System Manager. It does, however, change the connector definition and make the deleted business object unavailable for use in this implementation of this connector.

**Agent support**
Indicating Agent Support for a business object means that 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, put a check in the Agent Support box. Note that 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, because most application APIs do not support the Stringent level.

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

**Note:** For this release, maximum transaction level of a connector is always Best Effort.

**Associated maps (ICS)**
Each connector supports a list of business object definitions and their associated maps that are currently active in InterChange Server. This list displays 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 InterChange Server 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 not a 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, choose Save to Project.
  4. Deploy the project to InterChange Server.
  5. Reboot the InterChange 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, and use of this feature is not usually advised for connector agents that were designed in Java to be multi-threaded, since it is usually more efficient to use multiple threads than multiple processes.

**Setting trace/log file values (ICS)**

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. Choose 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.
   • To File: Writes logging or tracing messages to a file that you specify. To specify the file, choose the directory button (ellipsis), navigate to the preferred location, provide a file name, and choose 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.

### Configuring messaging
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.

### 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. 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**.

### Setting the configuration file properties (WebSphere MQ Integrator Broker)

The topics in this section apply if you are using WebSphere MQ Integrator (also referred to as WMQI) as the integration broker.

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 all of these categories:
1. Standard Properties
2. Connector-Specific Properties
3. Supported Business Objects
4. Trace/Log File values
5. Data Handlers (where applicable)

**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 information about the values to use in the Data Handlers category, see the **Connector Development Guide for C++** or the **Connector Development Guide for Java**.
**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-configuration (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 chapter of each adapter guide describes 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 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 informational and not configurable. This field specifies the action required to activate a property whose value has changed.

### 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 choose from the drop-down menu if one appears.
3. After entering all values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator, choose File > Exit (or close the window), and choose No when prompted to save changes.
   - To enter values for other categories in Connector Configurator, choose the tab for the category. The values you enter for Standard Properties (or 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 of the categories as a whole.
   - To save the revised values, choose File > Exit (or close the window) and choose Yes when prompted to save changes. Alternatively, choose Save > To File from either the File menu or the toolbar.

### Setting application-configuration properties

For application-specific configuration properties, you can add or change property names, configure values, delete a property, and encrypt a property:

1. Click in the field whose name or value you want to set.
2. Enter a name or value.
3. To encrypt a property, click 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 clicking the Encrypt check box in the Edit Property window. To decrypt a value, click to clear the Encrypt check box, enter the correct value in the Verification dialog box, and choose OK. If the entered value is correct, the value is decrypted and displays. The adapter guide for each connector contains a list and description of each property and its default value.

**Update method**
When WebSphere MQ Integrator Broker is the integration broker, connector properties are static. The Update Method is always Agent Restart. In other words, for changes to take effect, you must restart the connector agent after saving the revised connector configuration file.

**Specifying supported business object definitions**
The procedures in this section assume that you have already created:
- Business object definitions
- MQ message set files (*.set files)

The *.set files contain message set IDs that Connector Configurator requires for designating the connector’s supported business objects. See the *Implementation Guide for WebSphere MQ Integrator Broker* for information about creating the MQ message set files.

Each time that you add business object definitions to the system, you must use Connector Configurator to designate those business objects as supported by the connector.

**Important:** If the connector requires meta-objects, you must create message set files for each of them and load them into Connector Configurator, in the same manner as for business objects.

To specify supported business objects:
1. Select the Supported Business Objects tab and choose Load. The Open Message Set ID File(s) dialog displays.
2. Navigate to the directory where you have placed the message set file for the connector and select the appropriate message set file (*.set) or files.
3. Choose Open. The Business Object Name field displays the business object names contained in the *.set file; the numeric message set ID for each business object is listed in its corresponding Message Set ID field. Do not change the message set IDs. These names and numeric IDs are saved when you save the configuration file.
4. When you add business objects to the configuration, you must load their message set files. If you attempt to load a message set that contains a business object name that already exists in the configuration, or if you attempt to load a message set file that contains a duplicate business object name, Connector Configurator detects the duplicate and displays the Load Results dialog. The dialog shows the business object name or names for which there are duplicates. For each duplicate name shown, click in the Message Set ID field, and choose the Message Set ID that you wish to use.

**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. Choose 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.
   - To File: Writes logging or tracing messages to a file that you specify. To specify the file, choose the directory button (ellipsis), navigate to the preferred location, provide a file name, and choose 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.

**Configuring 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. 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.

---

**Using standard and connector-specific properties with Connector Configurator**

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 within your configuration file; Connector Configurator already has those definitions, and it incorporates them into your configuration file as soon as you create the file. For standard properties, your only task is to use Connector Configurator to set the values of the properties.
For connector-specific properties, however, you will need to both define the properties and set their values. Connector Configurator provides the interface for performing both of these tasks.

## 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.
Appendix B. Connector feature list

This appendix provides information about the features supported by the connector, and presents the following topics:

- “Business object request handling features”
- “Event notification features”
- “General features” on page 94

For descriptions of these features, see “Appendix A: Connector Feature Checklist” in the Connector Development Guide.

Business object request handling features

The table below details the business object request handling features supported by the connector.

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature</th>
<th>Support</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>Create verb</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td>Delete</td>
<td>Delete verb</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td></td>
<td>Logical delete</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td>Exist</td>
<td>Exist verb</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td>Misc</td>
<td>Attribute names</td>
<td>Full</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Business object names</td>
<td>Full</td>
<td></td>
</tr>
<tr>
<td>Retrieve</td>
<td>Ignore missing child object</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td>RetrieveByContent</td>
<td>Ignore missing child object</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td></td>
<td>Multiple results</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td></td>
<td>RetrieveByContent verb</td>
<td>N/A</td>
<td>The connector supports RetrieveByContent verb in full when using synchronous request/response.</td>
</tr>
<tr>
<td>Update</td>
<td>After-image support</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td></td>
<td>Delta support</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td></td>
<td>KeepRelations</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td>Verbs</td>
<td>Retrieve verb</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td></td>
<td>Subverb support</td>
<td>Partial</td>
<td>Depends on data handler chosen for the connector.</td>
</tr>
<tr>
<td></td>
<td>Verb stability</td>
<td>Full</td>
<td></td>
</tr>
</tbody>
</table>

Event notification features

The table below details the event notification features supported by the connector.
### General features

The table below details the general features supported by the connector.

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature</th>
<th>Support</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Object Attributes</td>
<td>Foreign key</td>
<td>No</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td></td>
<td>Foreign key attribute property</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td></td>
<td>Key</td>
<td>No</td>
<td>May be used by the data handler chosen for the connector.</td>
</tr>
<tr>
<td></td>
<td>Max length</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Meta-data-driven design</td>
<td>Full</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Required</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Connection Lost</td>
<td>Connection lost on poll</td>
<td>No</td>
<td>Connector may report an error or may continue polling and not retrieve events. The adapter listens for errors reported by WebSphere MQ—whether such errors are generated depends solely on WebSphere MQ.</td>
</tr>
<tr>
<td></td>
<td>Connection lost on request processing</td>
<td>Full</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connection lost while idle</td>
<td>No</td>
<td>WebSphere MQ does not use password for authentication.</td>
</tr>
<tr>
<td>Connector Properties</td>
<td>ApplicationPassword</td>
<td>No</td>
<td>Connector sends message under authority of specified local user.</td>
</tr>
<tr>
<td></td>
<td>ApplicationUserName</td>
<td>Partial</td>
<td>Depends on the data handler established for the connector.</td>
</tr>
<tr>
<td></td>
<td>UseDefaults</td>
<td>Partial</td>
<td></td>
</tr>
<tr>
<td>Message Tracing</td>
<td>General messaging</td>
<td>Full</td>
<td></td>
</tr>
</tbody>
</table>

### Connector Properties

<table>
<thead>
<tr>
<th>Feature</th>
<th>Support</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event distribution</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>PollQuantity</td>
<td>Full</td>
<td></td>
</tr>
<tr>
<td>Event status values</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Object key</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Object name</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Priority</td>
<td>Full</td>
<td></td>
</tr>
</tbody>
</table>

Connector retrieves messages based on the priority specified in their message header (range 0-9).

### Misc.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Support</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archiving</td>
<td>Full</td>
<td>Connector can deliver copies of messages to different queues depending on whether the message was unsubscribed, was successfully processed, or resulted in errors.</td>
</tr>
<tr>
<td>CDK method</td>
<td>Full</td>
<td></td>
</tr>
<tr>
<td>gotApplEvent</td>
<td>Full</td>
<td></td>
</tr>
<tr>
<td>Delta event notification</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Event sequence</td>
<td>Full</td>
<td></td>
</tr>
<tr>
<td>Future event processing</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>In-Progress event recovery</td>
<td>Full</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td>Physical delete event</td>
<td>N/A</td>
<td>Support is entirely dependent on application receiving connector request.</td>
</tr>
<tr>
<td>RetrieveAll</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Smart filtering</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Verb stability</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Feature</td>
<td>Support</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>generateMsg()</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Trace level 0</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Trace level 1</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Trace level 2</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Trace level 3</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Trace level 4</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Trace level 5</td>
<td>Full</td>
</tr>
<tr>
<td>Misc.</td>
<td>CDK method LogMsg</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Java Package Names</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Logging messages</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>NT service compliance</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Transaction support</td>
<td>Full</td>
</tr>
<tr>
<td>Special Value</td>
<td>CxBlank processing</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>CxIgnore processing</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Notices

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

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

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

IBM RTP Laboratory
3039 Cornwallis Road
P.O. BOX 12195
Programming interface information

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

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

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

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

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:
IBM
the IBM logo
AIX
CrossWorlds 
DB2
DB2 Universal Database
MQIntegrator
MQSeries
Tivoli
WebSphere

Lotus, Domino, Lotus Notes, and Notes Mail are trademarks of the Lotus
Development Corporation in the United States, other countries, or both.Microsoft,
Windows, Windows NT, and the Windows logo are trademarks of Microsoft
Corporation in the United States, other countries, or both.

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

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

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

IBM WebSphere InterChange Server V4.2, IBM WebSphere Business Integration
Toolset V4.2, IBM WebSphere Business Integration Adapters V2.2, IBM WebSphere
Business Integration Collaborations V4.2