IBM WebSphere Business Integration Adapters

Adapter for Telcordia User Guide

Version 2.5.x

IBM WebSphere Business Integration Adapters

Adapter for Telcordia User Guide

Version 2.5.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 95.

19December2003

This edition of this document applies to IBM WebSphere Business Integration Adapter for Telcordia, version 2.5.x, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

Contents

New in This release

New in release 2.5.x

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

Adapter installation information has been moved from this guide. See Chapter 2, “Installing and configuring the connector,” on page 19 for the new location of that information.

New in release 2.4

The adapter can now use WebSphere Application Server as an integration broker.

The adapter now supports Telcordia Service Delivery 10.0.

The connector now runs on the following platforms:

- Solaris 7, 8
- AIX 5.x

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 30..

The guaranteed event delivery feature has been enhanced.

New in release 2.2.x

The InProgress queue is no longer required and may be disabled. For more information, see “InProgressQueue” on page 25..

The connector supports interoperability with applications via MQSeries 5.1, 5.2, and 5.3.

The connector now has a UseDefaults property for business object processing. For more information, see “UseDefaults” on page 25..

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 23..

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 35..

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 12..

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 48.

About this document

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

This document describes the installation, configuration, business object development, and troubleshooting for the IBM WebSphere Business Integration 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 complete set of documentation available with this product describes the features and components common to all IBM WebSphere Business Integration Adapters installations, and includes reference material on specific components.

You can install related documentation from the following sites:

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

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

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 outline	A blue outline, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click inside the outline 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 *`$text`*	Text within percent (%) signs indicates the value of the Windows(R) `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the *`text`* UNIX environment variable.

Chapter 1. Overview

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.

Adapter environment

Before installing, configuring, and using the adapter, you must understand its environment requirements.

Broker compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. The 1.0.x version of the adapter for EJB is supported on the following versions of the adapter framework and with the following integration brokers:

- Adapter framework:

1. Telcordia Technologies, Inc.

  - WebSphere Business Integration Adapter Framework, version 2.1.0, 2.2.0, 2.3.0, 2.3.1, 2.4.0
- Integration brokers:
  - WebSphere InterChange Server, version 4.1.1, 4.2.0, 4.2.1, 4.2.2
  - WebSphere MQ Integrator, version 2.1
  - WebSphere MQ Integrator Broker, version 2.1.0
  - WebSphere Business Integration Message Broker, version 5.0
  - WebSphere Application Server Enterprise, version 5.0.2, with WebSphere Studio Application Developer Integration Edition, version 5.0.1

See *Release Notes* for any exceptions.

Note: For instructions on installing the integration broker and its prerequisites, see the following documentation.

For WebSphere InterChange Server (ICS), see the *System Installation Guide for UNIX* or *for Windows*.

For message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker), see *Implementing Adapters with WebSphere Message Brokers*, and the installation documentation for the message broker. Some of this can be found at the following Web site:

http://www.ibm.com/software/integration/mqfamily/library/manualsa/

For WebSphere Application Server, see Implementing Adapters with WebSphere Application Server and the documentation at

http://www.ibm.com/software/webservers/appserv/library.html

Adapter platforms

The adapter runs on the following platforms:

- Windows 2000
- Solaris 7, 8
- HP-UX 11i
- AIX® 5.1, 5.2

Adapter dependencies

- Telcordia Service Delivery Module 9.5.1, 10.0
- The connector supports interoperability with applications via MQSeries 5.1, 5.2,[2] and WebSphere MQ 5.3. Accordingly, you must have one of these software releases installed.
- IBM WebSphere MQ Java client libraries (included in WebSphere MQ 5.3).

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

- XML Schema Object Discovery Agent (ODA). This component is required to generate business object definitions from XML schema documents. The XML ODA is not installed when you install the adapter. You must install the XML ODA separately.

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.

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.

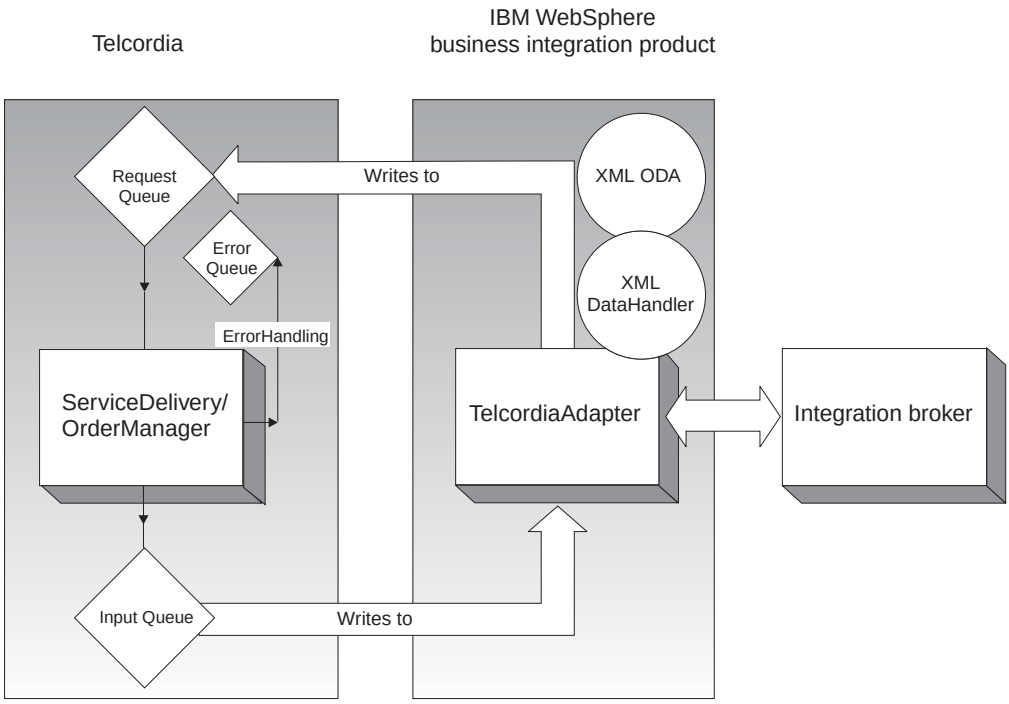

Figure 1. Telcordia-IBM WebSphere business integration architecture

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 *Implementing Adapters with WebSphere MQ Integrator Broker* or Implementing Adapters with WebSphere Application Server.
- 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 metadata-driven. Message routing and format conversion are initiated by an event polling technique. The connector uses an MQ implementation of the Java™ Message Service (JMS), an API for accessing enterprise-messaging systems that also makes possible guaranteed event delivery.

When configured with the InterChange Server integration broker, 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 3, “Creating or modifying business objects,” on page 45.

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.

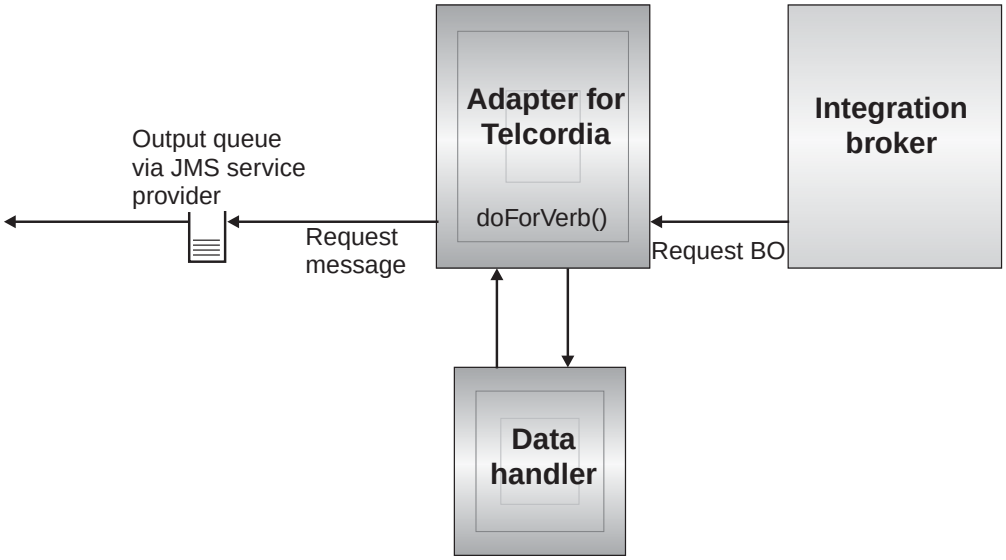

Figure 2. Application-connector communication method: Message request

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.

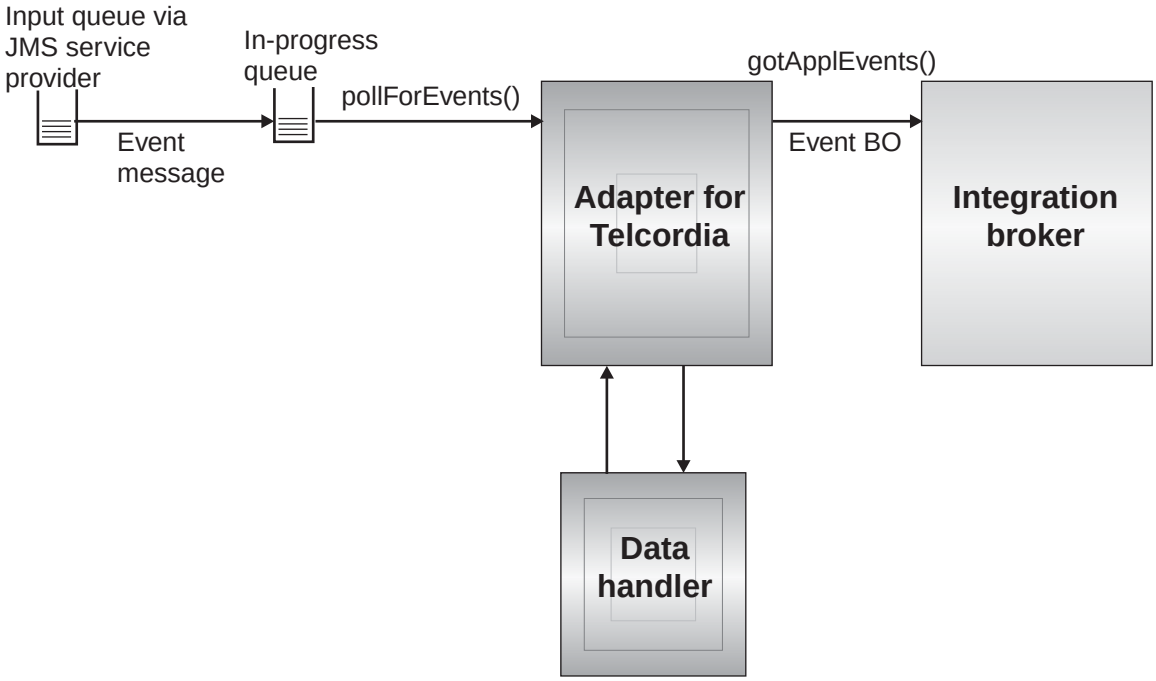

Figure 3. Application-connector communication method: Event delivery

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 46 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.

MQMD field	Description	Supported values (use the logical OR operator to support multiple values)
`MessageType`	Message Type	`DATAGRAM`

MQMD field	Description	Supported values (use the logical OR operator to support multiple values)
report	Options for report message requested	You can specify one or both of the following: • MQRO_PAN The connector sends a report message if the business object could be successfully processed. • MQRO_NAN The connector sends a report message if an error occurred while processing the business object. You can specify one of the following to control how the correlation ID of the report message is to be set: • MQRO_COPY_MSG_ID_TO_CORREL_ID The connector copies the message ID of the request message to the correlation ID of the report. This is the default action. • MQRO_PASS_CORREL_ID The connector copies the correlation ID of the request message to the correlation ID of the report.
ReplyToQueue	Name of reply queue	The name of the queue to which the report message should be sent.
replyToQueue Manager	Name of queue manager	The name of the queue manager to which the report message should be sent.
Message Body		A serialized business object in a format compatible with the data handler configured for the connector.

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 metadata 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.

MQMD field	Description	Supported values (use the logical OR operator to support multiple values)
MessageType	Message type	REPORT
feedback	Type of report	One of the following: • MQRO_PAN If the collaboration object successfully processed the business object. • MQRO_NAN If the connector or the collaboration object encountered an error while processing the request.

MQMD field	Description	Supported values (use the logical OR operator to support multiple values)
Message Body		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 metadata object. 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.

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 46 in Chapter 3, "Creating or modifying business objects," on page 45.

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`DeliveryTransport` standard property to JMS. Thus configured, the connector uses the JMS transport and all subsequent communication between the connector and the integration broker occurs through this transport. The JMS transport ensures that the messages are eventually delivered to their destination. Its role is to ensure that once a transactional queue session starts, the messages are cached there until a commit is issued; if a failure occurs or a rollback is issued, the messages are discarded.

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

You can configure the guaranteed-event-delivery feature for a JMS-enabled connector with, or without, a JMS event store. To configure the connector for guaranteed event delivery, see the *Connector Development Guide for Java*.

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.

Field	Description	Value
Format	Format name	Output format as defined in the conversion properties and truncated to 8 characters to meet IBM requirements (example: `MQSTR`)
MessageType	Message type	`MQMT_DATAGRAM*`if no response is expected from the receiving application. `MQMT_REQUEST*` if a response is expected
Report	Options for report message requested.	When a response message is expected, this field is populated as follows:`MQRO_PAN*` to indicate that a positive-action report is required if processing is successful.`MQRO_NAN*` to indicate that a negative-action report is required if processing fails.`MQRO_COPY_MSG_ID_TO_CORREL_ID*` to indicate that the correlation ID of the report generated should equal the message ID of the request originally issued.
ReplyToQueue	Name of reply queue	When a response message is expected this field is populated with the value of connector property `ReplyToQueue`.
Persistence	Message persistence	`MQPER_PERSISTENT*`
Expiry	Message lifetime	`MQEI_UNLIMITED*`

* Indicates constant defined by IBM.

The message header described in 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.

Field	Description	Value
Format	Format name	Input format of busObj as defined in the conversion properties.
MessageType	Message type	MQMT_REPORT*

*Indicates constant defined by IBM.

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

Telcordia feedback code	Equivalent InterChange Server response*
MQFB_PAN orMQFB_APPL_FIRST	SUCCESS
MQFB_NAN orMQFB_APPL_FIRST + 1	FAIL
MQFB_APPL_FIRST + 2	VALCHANGE
MQFB_APPL_FIRST + 3	VALDUPES
MQFB_APPL_FIRST + 4	MULTIPLE_HITS
MQFB_APPL_FIRST + 5	FAIL_RETRIEVE_BY_CONTENT
MQFB_APPL_FIRST + 6	BO_DOES_NOT_EXIST
MQFB_APPL_FIRST + 7	UNABLE_TO_LOGIN
MQFB_APPL_FIRST + 8	APP_RESPONSE_TIMEOUT (results in immediate termination of connector)

*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:

```
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:

```
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:

```
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:

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

to identify the response message.

You can also specify multiple substitutions such as the following:

```
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 23..

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 Appendix A, "Standard configuration properties for connectors," on page 53.

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, "Installing and configuring the connector," on page 19 for a description of what and where you must install.

Configuring connector properties

Connectors have two types of configuration properties: standard configuration properties and connector-specific configuration properties. Some of these properties have default values that you do not need to change. You may need to set the values of some of these properties before running the connector. For more information, see Chapter 2, "Installing and configuring the connector," on page 19.

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 27 and "Dynamic child meta-object" on page 32.
- Set the property ResponseTimeout in the (static or dynamic) meta-object to `-1`. This forces the connector to issue the business object without checking for a return.
- For more information, see "Create" on page 12, "Meta-object attributes configuration" on page 27, and Chapter 3, "Creating or modifying business objects," on page 45.

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 "Configuring the connector to send requests without notification" on page 16 with this exception: you specify a positive ResponseTimeout value to indicate how long the connector waits for a reply.
- See "Create" on page 12 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 "Meta-object attributes configuration" on page 27, and Chapter 3, "Creating or modifying business objects," on page 45.

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 "Static meta-object" on page 27.
- Make sure the connector subscribes to the static meta-object by specifying the name of the static meta-object in the connector-specific property ConfigurationMetaObject. For more information, see "Connector-specific properties" on page 21..
- 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 32..

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 12.

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 26 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 21..
- Specify the data handler class name (`com.crossworlds.DataHandlers.text.xml`)in the XML data handler meta-object, `MO_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, “Installing and configuring the connector,” on page 19 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 41..

Chapter 2. Installing and configuring the connector

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

Overview of installation tasks

To install the adapter, perform the following tasks.

Confirm adapter prerequisites

Before you install the adapter, confirm that all the environment prerequisites for installing and running the adapter are on your system. For details, see "Adapter environment" on page 1.

Install the integration broker

Installing the integration broker, a task that includes installing the WebSphere business integration system and starting the broker, is described in the documentation for your broker. For details about the brokers that the connector for Telcordia supports, see "Broker compatibility" on page 1.

For details about installing the broker, see the appropriate implementation documentation of the broker you are using.

Install the adapter for Telcordia and related files

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

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

Installed file structure

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.

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.

Subdirectory of *ProductDir*	Description
connectors\Telcordia\CWTelcordia.jar	Contains classes used by the Telcordia connector
connectors\Telcordia\start_Telcordia.bat	The startup script for the connector (Windows 2000)
connectors\messages\TelcordiaConnector.txt	Message file for the connector
repository\Telcordia\CN_Telcordia.txt	Repository definition for the connector

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.

Subdirectory of *ProductDir*	Description
connectors/Telcordia/CWTelcordia.jar	Contains classes used by the Telcordia connector
connectors/Telcordia/start_Telcordia.sh	System startup script for the connector. This script is called from the generic connector manager script. When you click the Connector Configuration screen of System Manager, the installer creates a customized wrapper for this connector manager script. Use this customized wrapper to start and stop the connector.
connectors/messages/TelcordiaConnector.txt	Message file for the connector
repository/Telcordia/CN_Telcordia.txt	Repository definition for the connector

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 adapter-specific configuration properties. You must set the values of these properties before running the adapter.

You use Connector Configurator to configure connector properties:

- For a description of Connector Configurator and step-by-step procedures, see Appendix B, "Connector configurator," on page 69.
- For a description of standard connector properties, see "Standard connector properties" and then Appendix A, "Standard configuration properties for connectors," on page 53.
- For a description of connector-specific properties, see "Connector-specific properties."

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

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 53 for documentation of these properties.

Note: When you set configuration properties in Connector Configurator, you specify your broker using the BrokerType property. Once this is set, the properties relevant to your broker will appear in the Connector Configurator window.

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.

Name	Possible values	Default value	Required
ApplicationPassword	*Login password*		Yes
ApplicationUserName	*Login user ID*		Yes
ArchiveQueue	*Queue to which copies of successfully processed messages are sent*	`queue://crossworlds. queuemanager/MQCONN.ARCHIVE`	No
CCSID	*Character set for queue manager connection*	null	No
Channel	*Telcordia server connector channel*	`CHANNEL1`	Yes
ConfigurationMetaObject	*Name of configuration meta-object*	`Telcordia_MQSeries_MO_Config`	Yes
DataHandlerClassName	*Data handler class name*	`com.crossworlds.DataHandlers. text.xml`	No
DataHandlerConfigMO	*Data handler meta-object*	`MO_DataHandler_ Default`	Yes
DataHandlerMimeType	*MIME type of file*	`Text/XML`	Yes
DefaultVerb	*Any verb supported by the connector.*	`Create`	
ErrorQueue	*Queue for unprocessed messages*	`queue://crossworlds. queuemanager/MQCONN.ERROR`	No

Name	Possible values	Default value	Required
FeedbackCodeMappingMO	Feedback code meta-object		No
HostName	*Telcordia server*		Yes
InDoubtEvents	`FailOnStartup Reprocess IgnoreLogError`	`Reprocess`	No
InputQueue	*Poll queues*	`queue://crossworlds.queuemanager/DELV_2_IBM`	No
InProgressQueue	*In-progress event queue*	`queue://crossworlds.queuemanager/MQCONN.IN_PROGRESS`	No
PollQuantity	*Number of messages to retrieve from each queue specified in the `InputQueue` property*	1	No
Port	*Port established for the WebSphere MQ listener*	1414	Yes
ReplyToQueue	*Queue to which response messages are delivered when the connector issues requests*	`queue://crossworlds.queuemanager/MQCONN.REPLYTO`	No
UnsubscribedQueue	*Queue to which unsubscribed messages are sent*	`queue://crossworlds.queuemanager/MQCONN.UNSUBSCRIBE`	No
UseDefaults	true or false	false	

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 26. 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 12. 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

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

Default = none.

HostName

The name of the server hosting 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 pollQuanity 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 pollQuanity 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 35.

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.

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.

Property name	Description	Values
`expiry`	Lifetime of the message in milliseconds.	0 = unlimited. positive integers = timeout (in ms).
`priority`	Priority of the message.	0-9, where 1 is the highest priority. A value of -1 means that the property should be determined by the configuration of the queue. A value of -2 specifies that the connector can use its own default value.
persistence	Whether the message should be 'hardened' to disk.	1 = non-persistent 2 = persistent A value of -1 means that the property should be determined by the configuration of the queue. A value of -2 specifies that the connector can use its own default value.
CCSID	Character set encoding of the outbound message.	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 22
targetClient	Whether the receiving application is JMS compliant or not.	0 = JMS (MQRFH2 header) 1 = MQ (MQMD header only)
`encoding`	How to represent numeric fields.	An integer value as described in the base WebSphere MQ documentation.

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.

Property name	Description
`CorrelationID`	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 "Asynchronous request processing" on page 39.

Property name	Description
CollaborationName	The CollaborationName must be specified in the application specific text of the attribute for the business object/verb combination. For example, if a user expects to handle synchronous requests for the business object Customer with the Create verb, the static metadata object must contain an attribute named Customer_Create. The Customer_Create attribute must contain application specific text that includes a name-value pair. For example, CollaborationName=MyCustomerProcessingCollab. See the "Application-specific information" on page 30 section for syntax details. Failure to do this will result in run-time errors when the connector attempts to synchronously process a request involving the Customer business object. **Note:** This property is only available for synchronous requests.
DataEncoding	DataEncoding is the encoding to be used to read and write messages. If this property is not specified in the static meta-object, the connector tries to read the messages without using any specific encoding. DataEncoding defined in a dynamic child meta-object overrides the value defined in the static meta-object. The default value is Text. The format for the value of this attribute is messageType[:*enc*]. I.e., Text:ISO8859_1, Text:UnicodeLittle, Text, or Binary.This property is related internally to the InputFormat property: specify one and only one DataEncoding per InputFormat.
DataHandlerConfigMO	Meta-object passed to data handler to provide configuration information. If specified in the static meta-object, this will override the value specified in the DataHandlerConfigMO connector property. Use this 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.
DataHandlerMimeType	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 DataHandlerMimeType 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 DataHandlerConfigMO should have an attribute that corresponds to the value of this property.

Property name	Description
DoNotReportBusObj	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 effect 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. **Note:** This property is only available for synchronous requests.
InputFormat	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 oneDataEncoding per InputFormat. Do not set this property using default conversion properties; its value is used to match incoming messages to business objects that will store message content.
OutputFormat	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.
InputQueue	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. Do not set this property using default conversion properties; its value is used to match incoming messages to business objects.
OutputQueue	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.
ResponseTimeout	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.
TimeoutFatal	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.
CCSID	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.

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.

Attribute name	Application-specific information	Description
Default	OutputQueue=queue://queue.manager/IBM_2_DELV; DataEncoding=Text; ResponseTimeout=10000; TimeoutFatal=False; CCSID=819	Default values for all conversion properties
OrderResponse_Create	OutputFormat=NEW;	Indicates to connector that any (outgoing?) messages of format NEW should be converted to an OrderResponse business object with the verb Create.

Application-specific information

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

```
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 24) 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 business object name and the contents of the message to the data handler. This allows the data handler to verify that the message contents correspond to the business object that the user expects.

If, however, the same input format is defined for more than one business object, the connector will be unable to determine which business object the data represents before passing it to the data handler. In such cases, the 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]
Name = LSRRequest_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputFormat=NEW
IsRequiredServerBound = false
[End]

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

[Verb]
Name = Create
[End]
[End]
```

Dynamic child meta-object

If it is difficult or unfeasible to specify the necessary metadata through a static meta-object, the connector can optionally accept metadata 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.

Attribute name	Application-specific information	Description
OrderResponse_Create	`OutputQueue=queue://queue.manager/IBM_2_DELVX;DataEncoding=Text:UnicodeLittle;`	The OutputQueue and DataEncoding properties specified here override those in the static meta-object.

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.

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

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
[End]
[Attribute]
Name = OutputQueue
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = OUT
```

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

[Verb]
Name = Create
[End]

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

[Attribute]
Name = FirstName
Type = String
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
```

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

[Verb]
Name = Create
[End]

[End]
```

JMS headers, Telcordia message properties, and dynamic child meta-object attributes

You can add attributes to a dynamic meta-object to gain more information about, and more control over, the message transport. Adding such attributes allows you to modify JMS properties, to control the ReplyToQueue on a per-request basis (rather than using the default ReplyToQueue specified in the adapter properties), and to re-target a message CorrelationID. This section describes these attributes and how they affect event notification and request processing in both synchronous and asynchronous modes.

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

Table 2. Dynamic meta-object header attributes

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

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 3. Application-specific information for JMS property attributes

Name	Possible values	Comments
Name	Any valid JMS property name	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.
Type	`String`, `Int`, `Boolean`, `Float`, `Double`, `Long`, `Short`	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.

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`).

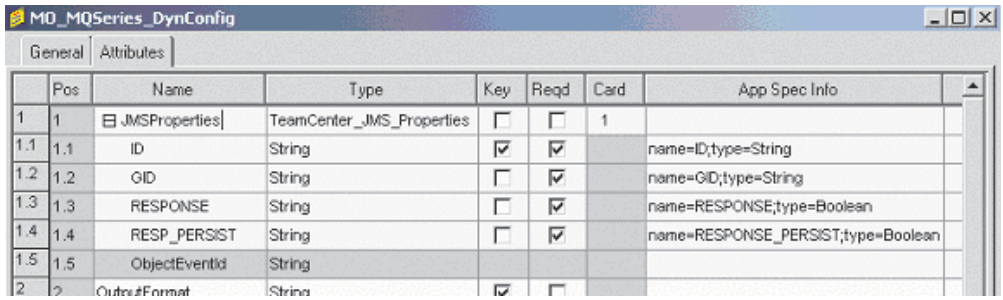

Figure 4. JMS properties attribute in a dynamic meta-object

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 `DeliveryMode` is set to 2, the message is sent persistently. If `DeliveryMode` is set to 1, the message is not sent persistently. Any other value may fail the connector. If `DeliveryMode` is not specified in the MO, then the JMS provider establishes the persistence setting.
4. If attribute `Priority` is specified, the connector sets the value in the outgoing request. The `Priority` attribute can take values 0 through 9; any other value may cause the connector to terminate.
5. 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, 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.

Creating multiple connector instances

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

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

Create a new directory

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

```
ProductDir\connectors\connectorInstance
```

where `connectorInstance` uniquely identifies the connector instance.

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

```
ProductDir\repository\connectorInstance
```

Create business object definitions

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

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

   `ProductDir\repository\initialConnectorInstance`

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

Create a connector definition

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

1. Copy the initial connector's configuration file (connector definition) and rename it.
2. Make sure each connector instance correctly lists its supported business objects (and any associated meta-objects).
3. Customize any connector properties as appropriate.

Create a start-up script

To create a startup script:

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

   dirname
2. Put this startup script in the connector directory you created in "Create a new directory" on page 41.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

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

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

Starting the connector

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

ProductDir`\connectors\`*connName*

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 4 shows.

Table 4. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	`connector_manager_`*connName*
Windows	`start_`*connName*`.bat`

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

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

  Select **Programs>IBM WebSphere Business Integration Adapters>Adapters>Connectors**. By default, the program name is "IBM WebSphere Business Integration Adapters". However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.
- From the command line
  - On Windows systems:

    `start_`*`connName connName brokerName`* `[-c`*`configFile`* `]`
  - On UNIX-based systems:

    `connector_manager_`*`connName`* `-start`

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

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

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

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

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

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

Stopping the connector

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

- If you started the connector from the command line, with its connector startup script:
  - On Windows systems, invoking the startup script creates a separate "console" window for the connector. In this window, type "Q" and press Enter to stop the connector.

– On UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:

```
connector_manager_connName -stop
```

where *connName* is the name of the connector.

- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.

- From System Monitor (WebSphere InterChange Server product only)

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.

- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.

Chapter 3. Creating or modifying business objects

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

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

Business object metadata includes the structure of a business object, the settings of its attribute properties, and the content of its application-specific information. Because the connector is metadata-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 48.

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:

  <MSG> <CODE> 0173 </CODE> <TEXT> LSO NOT FOUND IN WC MAPING REFERENCE TABLE </TEXT> <ERRTAGPATH> DELV.SR.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, "Installing and configuring the connector," on page 19 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 `BillingOrderCompletionContract` 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 4. 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

Problem	Potential solution / explanation
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/jms/JMSException...`	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.
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: com/ibm/mq/jms/MQConnectionFactory...`	Connector cannot find file `com.ibm.mqjms.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/naming/Referenceable...`	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.
The connector shuts down unexpectedly during initialization and the following exception is reported: `java.lang.UnsatisfiedLinkError: no mqjbnd01 in shared library path`	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.
The connector reports `MQJMS2005: failed to create MQQueueManager for ':'`	Explicitly set values for the following properties: `HostName`, `Channel`, and `Port`.

Event processing problems

Problem	Potential solution / explanation
The connector delivers all messages with an MQRFH2 header.	To deliver messages with only the MQMD WebSphere MQ header, append `?targetClient=1` to the name of output queue URI. For example, if you output messages to queue `queue://my.queue.manager/OUT`, change the URI to `queue://my.queue.manager/OUT?targetClient=1`. See Chapter 2, "Installing and configuring the connector," on page 19 for more information.
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. Standard configuration properties for connectors

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

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

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

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

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

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNameSpaceFormat

Deleted properties

- RestartCount
- RHF2MessageDomain

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

Using Connector Configurator

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

Note: Connector Configurator and System Manager run only on the Windows system. If you are running the connector on a UNIX system, you must have

a Windows machine with these tools installed. To set connector properties for a connector that runs on UNIX, you must start up System Manager on the Windows machine, connect to the UNIX integration broker, and bring up Connector Configurator for the connector.

Setting and updating property values

The default length of a property field is 255 characters.

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

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

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

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

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in the Property Summary table below.

Summary of standard properties

Table 5 on page 55 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

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

Table 5. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdminInQueue	Valid JMS queue name	*CONNECTORNAME* /ADMININQUEUE	Component restart	Delivery Transport is JMS
AdminOutQueue	Valid JMS queue name	*CONNECTORNAME*/ADMINOUTQUEUE	Component restart	Delivery Transport is JMS
AgentConnections	1-4	1	Component restart	Delivery Transport is MQ or IDL: Repository directory is <REMOTE>
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS, WMQI, WAS			
CharacterEncoding	ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 **Note:** This is a subset of supported values.	ascii7	Component restart	
ConcurrentEventTriggeredFlows	1 to 32,767	1	Component restart	Repository directory is <REMOTE>
ContainerManagedEvents	No value or JMS	No value	Component restart	Delivery Transport is JMS
ControllerStoreAndForwardMode	true or false	True	Dynamic	Repository directory is <REMOTE>
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE>
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	MQ, IDL, or JMS	JMS	Component restart	If Repository directory is local, then value is JMS only
DuplicateEventElimination	True or False	False	Component restart	JMS transport only: Container Managed Events must be <NONE>
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	JMS transport only

Table 5. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
jms.FactoryClassName	CxCommon.Messaging.jms .IBMMQSeriesFactory or CxCommon.Messaging .jms.SonicMQFactory or any Java class name	CxCommon.Messaging. jms.IBMMQSeriesFactory	Component restart	JMS transport only
jms.MessageBrokerName	If FactoryClassName is IBM, use crossworlds.queue. manager. If FactoryClassName is Sonic, use localhost:2506.	crossworlds.queue.manager	Component restart	JMS transport only
jms.NumConcurrentRequests	Positive integer	10	Component restart	JMS transport only
jms.Password	Any valid password		Component restart	JMS transport only
jms.UserName	Any valid name		Component restart	JMS transport only
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	Repository directory is <REMOTE>
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	Repository directory is <REMOTE>
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	Repository directory is <REMOTE>
ListenerConcurrency	1- 100	1	Component restart	Delivery Transport must be MQ
Locale	en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR **Note:** This is a subset of the supported locales.	en_US	Component restart	
LogAtInterchangeEnd	True or False	False	Component restart	Repository Directory must be <REMOTE>
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory must be <REMOTE>
MessageFileName	Path or filename	InterchangeSystem.txt	Component restart	
MonitorQueue	Any valid queue name	*CONNECTORNAME*/MONITORQUEUE	Component restart	JMS transport only: DuplicateEvent Elimination must be True
OADAutoRestartAgent	True or False	False	Dynamic	Repository Directory must be <REMOTE>

Table 5. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
OADMaxNumRetry	A positive number	1000	Dynamic	Repository Directory must be <REMOTE>
OADRetryTimeInterval	A positive number in minutes	10	Dynamic	Repository Directory must be <REMOTE>
PollEndTime	HH:MM	HH:MM	Component restart	
PollFrequency	A positive integer in milliseconds no (to disable polling) key (to poll only when the letter p is entered in the connector's Command Prompt window)	10000	Dynamic	
PollQuantity	1-500	1	Agent restart	JMS transport only: Container Managed Events is specified
PollStartTime	HH:MM(HH is 0-23, MM is 0-59)	HH:MM	Component restart	
RepositoryDirectory	Location of metadata repository		Agent restart	For ICS: set to <REMOTE> For WebSphere MQ message brokers and WAS: set to C:\crossworlds\ repository
RequestQueue	Valid JMS queue name	*CONNECTORNAME*/REQUESTQUEUE	Component restart	Delivery Transport is JMS
ResponseQueue	Valid JMS queue name	*CONNECTORNAME*/RESPONSEQUEUE	Component restart	Delivery Transport is JMS: required only if Repository directory is <REMOTE>
RestartRetryCount	0-99	3	Dynamic	
RestartRetryInterval	A sensible positive value in minutes: 1 - 2147483547	1	Dynamic	
SourceQueue	Valid WebSphere MQ name	*CONNECTORNAME*/SOURCEQUEUE	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue		*CONNECTORNAME*/ SYNCHRONOUSREQUESTQUEUE	Component restart	Delivery Transport is JMS

Table 5. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery Transport is JMS
SynchronousResponseQueue		*CONNECTORNAME*/SYNCHRONOUSRESPONSEQUEUE	Component restart	Delivery Transport is JMS
WireFormat	CwXML, CwBO	CwXML	Agent restart	CwXML if Repository Directory is not <REMOTE>: CwBO if Repository Directory is <REMOTE>
WsifSynchronousRequest Timeout	0 - any number (millisecs)	0	Component restart	WAS only
XMLNameSpaceFormat	short, long	short	Agent restart	WebSphere MQ message brokers and WAS only

Standard configuration properties

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

AdminInQueue

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

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

Applicable only if RepositoryDirectory is <REMOTE>.

The AgentConnections property controls the number of ORB connections opened by orb.init[].

By default, the value of this property is set to 1. There is no need to change this default.

AgentTraceLevel

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

ApplicationName

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

BrokerType

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

CharacterEncoding

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

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

By default, a subset of supported character encodings only is displayed in the drop 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.

ConcurrentEventTriggeredFlows

Applicable only if `RepositoryDirectory` is <REMOTE>.

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

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

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

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

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

ContainerManagedEvents

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

The default value is No value.

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

- PollQuantity = 1 to 500
- SourceQueue = CONNECTORNAME/SOURCEQUEUE

You must also configure a data handler with the MimeType, DHClass, and DataHandlerConfigMOName (optional) properties. To set those values, use the **Data Handler** tab in Connector Configurator. The fields for the values under the Data Handler tab will be displayed only if you have set ContainerManagedEvents to JMS.

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

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

ControllerStoreAndForwardMode

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

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

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

The default is true.

ControllerTraceLevel

Applicable only if RepositoryDirectory is <REMOTE>.

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

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

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

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

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

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

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

WebSphere MQ and IDL

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

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

JMS

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

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

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

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

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

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

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

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

This line restricts heap memory usage to a maximum of 768 MB (3 segments * 256 MB). If the process memory grows more than this limit, page swapping can occur, which can adversely affect the performance of your system.

- The `IPCCBaseAddress` property to a value of 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

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

This property can also be set to `false`.

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

FaultQueue

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

The default value is `CONNECTORNAME/FAULTQUEUE`.

JvmMaxHeapSize

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

The default value is 128m.

JvmMaxNativeStackSize

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

The default value is 128k.

JvmMinHeapSize

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

The default value is 1m.

jms.FactoryClassName

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

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

jms.MessageBrokerName

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

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

ListenerConcurrency

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

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

Locale

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

A locale name has the following format:

ll_TT.codeset

where:

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

By default, only a subset of supported locales appears in the drop 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.

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

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

LogAtInterchangeEnd

Applicable only if RespositoryDirectory is <REMOTE>.

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

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

MaxEventCapacity

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

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

MessageFileName

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

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

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

MonitorQueue

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

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

Valid only when the RepositoryDirectory is <REMOTE>.

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

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

The default value is `false`.

OADMaxNumRetry

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

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

The default value is `1000`.

OADRetryTimeInterval

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

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

The default is `10`.

PollEndTime

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

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

PollFrequency

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.

PollQuantity

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

PollStartTime

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

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

RequestQueue

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

The default value is `CONNECTOR/REQUESTQUEUE`.

RepositoryDirectory

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

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

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

ResponseQueue

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

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

RestartRetryCount

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

The default is `3`.

RestartRetryInterval

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

The default is 1.

SourceQueue

Applicable only if DeliveryTransport is JMS and ContainerManagedEvents is specified.

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 59.

The default value is CONNECTOR/SOURCEQUEUE.

SynchronousRequestQueue

Applicable only if DeliveryTransport is JMS.

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

The default is CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE

SynchronousResponseQueue

Applicable only if DeliveryTransport is JMS.

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

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

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

The default value is 0.

WireFormat

Message format on the transport.

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

WsifSynchronousRequest Timeout

WAS integration broker only.

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

The default value is 0.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

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

The default value is `short`.

Appendix B. Connector configurator

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

You use Connector Configurator to:

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

Note:

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

The topics covered in this appendix are:

Overview of Connector Configurator

Connector Configurator allows you to configure the connector component of your adapter for use with these integration brokers:

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

You use Connector Configurator to:

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

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 70).

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

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

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

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 71 to set up a new one.

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

Starting Connector Configurator

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

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

Running Configurator in stand-alone mode

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

To do so:

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

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 75.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

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

To edit an existing configuration file:

1. In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it. Connector Configurator opens and displays the configuration file with the integration broker type and file name at the top.
2. Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

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

You can create a brand-new template for the connector-specific properties of your connector, or you can use an existing file as the template.

- To create a new template, see "Creating a new template" on page 71.
- To use an existing file, simply modify an existing template and save it under the new name.

Creating a new template

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

To create a template:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears, with the following fields:
   - **Template**, and **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 box for creating a new configuration file from a template.
   - **Old Template**, and **Select the Existing Template to Modify**

     The names of all currently available templates are displayed in the Template Name display.

- To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template will appear in the **Template Preview** display. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template.

3. Select a template from the **Template Name** display, enter that template name in the **Find Name** field (or highlight your selection in **Template Name**), and click **Next**.

If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

Specifying general characteristics

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

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

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

Specifying values

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

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.
2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, make any changes. The changes will not be accepted unless you also open the **Property Value** dialog box for the property, described in the next step.
4. Right-click the box in the top left-hand corner of the value table and click **Add**. A **Property Value** dialog box appears. Depending on the property type, the dialog box 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**. It displays a table with three columns:

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

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

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

After a value has been created and appears in the grid, it can be edited from within the table display. To make a change in an existing value in the table, select an entire row by clicking on the row number. Then right-click in the **Value** field and click **Edit Value**.

Setting dependencies

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

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

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

Creating a new configuration file

When you create a new configuration file, your first step is to select an integration broker. The broker you select determines the properties that will appear in the configuration file.

To select a broker:

- In the Connector Configurator home menu, click **File>New>Connector Configuration**. The **New Connector** dialog box appears.
- In the I**ntegration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
- Complete the remaining fields in the **New Connector** window, as described later in this chapter.

You can also do this:

- In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator opens and displays the **New Connector** dialog box.

Creating a configuration file from a connector-specific template

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

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

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

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

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

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

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

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

Using an existing file

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

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

- A previous configuration file for the connector.
  Such a file typically has the extension *.cfg.

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

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

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

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

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

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

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

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

Completing a configuration file

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

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

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 78..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

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

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

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

Setting the configuration file properties

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

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

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

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

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

- Associated Maps
- Resources
- Messaging (where applicable)

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

Standard properties differ from connector-specific properties as follows:

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

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

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.
- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.
- The **Update Method** field is 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 select one from the drop-down menu if it appears.
3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
   - To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

Setting application-specific configuration properties

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

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

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 54.

Specifying supported business object definitions

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

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

If ICS is your broker

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

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

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

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

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

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

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

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

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

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

If a WebSphere Message Broker is your broker

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

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

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

If WAS is your broker

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

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

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

Associated maps (ICS only)

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

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

will be used to transform the application-specific business object to the generic business object or the generic business object to the application-specific business object.

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

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

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

- **Business Object Name**

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

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

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

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

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

  To explicitly bind a map:

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

Resources (ICS)

The **Resource** tab allows you to set a value that determines whether and to what extent the connector agent will handle multiple processes concurrently, using connector agent parallelism.
Not all connectors support this feature. If you are running a connector agent that was designed in Java to be multi-threaded, you are advised not to use this feature, since it is usually more efficient to use multiple threads than multiple processes.

Messaging (ICS)

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

Setting trace/log file values

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

To change the logging and tracing values:

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

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

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

Data handlers

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

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

Saving your configuration file

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

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

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

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

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

Changing a configuration file

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

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

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

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

Completing the configuration

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

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

Using Connector Configurator in a globalized environment

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

Connector Configurator supports non-English characters in:

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

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

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

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

Appendix C. Adapter for Telcordia samples

About the samples

The samples in this appendix illustrate a Telcordia order response in the following formats:

- business object definition
- xml schema
- xml

Sample business object definition

The following is a business object definition `Telcordia_OrderResponse`:

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

        [Attribute]
        Name = XMLDeclaration
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = type=pi
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ROOT
        Type = Telcordia_ROOT_OrderResponse
        ContainedObjectVersion = 1.0.0
        Relationship = containment
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = OrderResponse
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]

        [Verb]
        Name = Create
```

```
        [End]
[End]
[BusinessObjectDefinition]
Name = Telcordia_ROOT_OrderResponse
Version = 1.0.0
AppSpecificInfo = OrderResponse

        [Attribute]
        Name = OrderResponse_Wrapper1
        Type = Telcordia_OrderResponse_OrderResponse_Wrapper1
        ContainedObjectVersion = 1.0.0
        Relationship = containment
        Cardinality = n
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = true
        AppSpecificInfo = (C0|CUR)
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]

        [Verb]
        Name = Create
        [End]
[End]
[BusinessObjectDefinition]
Name = Telcordia_OrderResponse_OrderResponse_Wrapper1
Version = 1.0.0

        [Attribute]
        Name = C0
        Type = Telcordia_OrderResponse_C0
        ContainedObjectVersion = 1.0.0
        Relationship = containment
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = true
        AppSpecificInfo = C0
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = CUR
        Type = Telcordia_OrderResponse_CUR
        ContainedObjectVersion = 1.0.0
        Relationship = containment
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = true
        AppSpecificInfo = CUR
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
```

```
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]

        [Verb]
        Name = Create
        [End]
[End]
[BusinessObjectDefinition]
Name = Telcordia_OrderResponse_C0
Version = 1.0.0
AppSpecificInfo = C0

        [Attribute]
        Name = C0_Wrapper1
        Type = Telcordia_OrderResponse_C0_Wrapper1
        ContainedObjectVersion = 1.0.0
        Relationship = containment
        Cardinality = n
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = true
        AppSpecificInfo = (CORS|DD|FT|ORDNO|OT|RSYS|TSYS|TT|WC)
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]

        [Verb]
        Name = Create
        [End]
[End]
[BusinessObjectDefinition]
Name = Telcordia_OrderResponse_C0_Wrapper1
Version = 1.0.0

        [Attribute]
        Name = CORS
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = CORS;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = DD
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
```

```
IsForeignKey = false
IsRequired = true
AppSpecificInfo = DD;type=pcdata
IsRequiredServerBound = false
[End]
[Attribute]
Name = FT
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = FT;type=pcdata
IsRequiredServerBound = false
[End]
[Attribute]
Name = ORDNO
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = ORDNO;type=pcdata
IsRequiredServerBound = false
[End]
[Attribute]
Name = OT
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = OT;type=pcdata
IsRequiredServerBound = false
[End]
[Attribute]
Name = RSYS
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = RSYS;type=pcdata
IsRequiredServerBound = false
[End]
[Attribute]
Name = TSYS
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = TSYS;type=pcdata
IsRequiredServerBound = false
[End]
[Attribute]
Name = TT
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
```

```
        IsRequired = true
        AppSpecificInfo = TT;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = WC
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = true
        AppSpecificInfo = WC;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]

        [Verb]
        Name = Create
        [End]
[End]
[BusinessObjectDefinition]
Name = Telcordia_OrderResponse_CUR
Version = 1.0.0
AppSpecificInfo = CUR

        [Attribute]
        Name = CUR_Wrapper1
        Type = Telcordia_OrderResponse_CUR_Wrapper1
        ContainedObjectVersion = 1.0.0
        Relationship = containment
        Cardinality = n
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = true
        AppSpecificInfo = (STAT|MSG)
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]

        [Verb]
        Name = Create
        [End]
[End]
[BusinessObjectDefinition]
Name = Telcordia_OrderResponse_CUR_Wrapper1
Version = 1.0.0
```

```
        [Attribute]
        Name = STAT
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = STAT;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = MSG
        Type = Telcordia_OrderResponse_MSG
        ContainedObjectVersion = 1.0.0
        Relationship = containment
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = MSG
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]

        [Verb]
        Name = Create
        [End]
[End]
[BusinessObjectDefinition]
Name = Telcordia_OrderResponse_MSG
Version = 1.0.0
AppSpecificInfo = MSG

        [Attribute]
        Name = MSG_Wrapper1
        Type = Telcordia_OrderResponse_MSG_Wrapper1
        ContainedObjectVersion = 1.0.0
        Relationship = containment
        Cardinality = n
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = true
        AppSpecificInfo = (CODE|TEXT|ERRTAGPATH|LDEST|CATEGORY)
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]
```

```
        [Verb]
        Name = Create
        [End]
[End]
[BusinessObjectDefinition]
Name = Telcordia_OrderResponse_MSG_Wrapper1
Version = 1.0.0

        [Attribute]
        Name = CODE
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = CODE;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = TEXT
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = TEXT;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ERRTAGPATH
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = ERRTAGPATH;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = LDEST
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = LDEST;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = CATEGORY
        Type = String
        Cardinality = 1
        MaxLength = 255
        IsKey = false
        IsForeignKey = false
        IsRequired = false
        AppSpecificInfo = CATEGORY;type=pcdata
        IsRequiredServerBound = false
        [End]
        [Attribute]
        Name = ObjectEventId
        Type = String
```

```
        Cardinality = 1
        MaxLength = 255
        IsKey = true
        IsForeignKey = false
        IsRequired = false
        IsRequiredServerBound = false
        [End]

        [Verb]
        Name = Create
        [End]
[End]
```

Sample xml schema

The following is the xml schema for that corresponds to the business object definition Telcordia_OrderResponse:

```
<?xml version = "1.0" encoding = "UTF-8"?>

<!-- Schema Identifier: BD-DELV-SPEC-040, Issue 2A, July 2002, Release 9.5.1-->
<!-- Last Modified: November 3, 2002-->

<xsd:schema xml:lang = "en" xmlns:xsd = "http://www.w3.org/2001/XMLSchema">

   <xsd:include schemaLocation = "./CURTags.xsd"/>

   <xsd:element name = "C0">
      <xsd:annotation>
         <xsd:documentation>
FullName=[*C0 Route Control Header Section]
Desc=[This section contains routing data for a Customer Order. The *C0
section must be the first]
         </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:all>
            <xsd:element ref = "CORS" minOccurs = "0"/>
            <xsd:element ref = "DD"/>
            <xsd:element ref = "FT"/>
            <xsd:element ref = "ORDNO"/>
            <xsd:element ref = "OT"/>
            <xsd:element ref = "RSYS"/>
            <xsd:element ref = "TSYS"/>
            <xsd:element ref = "TT"/>
            <xsd:element ref = "WC"/>
         </xsd:all>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name = "CUR">
      <xsd:annotation>
         <xsd:documentation>
FullName=[Customer Order Response Section]
Desc=[This Customer Order Response section contains status information
and exception information to be returned to the order originator.]
         </xsd:documentation>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:all>
            <xsd:element ref = "STAT" minOccurs = "0"/>
            <xsd:element name = "MSG" minOccurs = "0">
               <xsd:annotation>
                  <xsd:documentation>
FullName=[Message Aggregate]
Desc=[The MSG aggregate contains exception information to be returned
to the order originator.]
```

```
Format=[ Conditional (required for error and non-involvement statuses;
not present if STAT is ACTV). This aggregate will appear once for each
exception message]
                  </xsd:documentation>
               </xsd:annotation>
               <xsd:complexType>
                  <xsd:all>
                     <xsd:element ref = "CODE" minOccurs = "0"/>
                     <xsd:element ref = "TEXT" minOccurs = "0"/>
                     <xsd:element ref = "ERRTAGPATH" minOccurs = "0"/>
                     <xsd:element ref = "LDEST" minOccurs = "0"/>
                     <xsd:element ref = "CATEGORY" minOccurs = "0"/>
                  </xsd:all>
               </xsd:complexType>
            </xsd:element>
         </xsd:all>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name = "OrderResponse">
      <xsd:complexType>
         <xsd:all>
            <xsd:element ref = "C0"/>
            <xsd:element ref = "CUR"/>
         </xsd:all>
      </xsd:complexType>
   </xsd:element>

</xsd:schema>
```

Sample xml

The following is the xml file that corresponds to the xml schema presented in "Sample xml schema" on page 92:

```
<OrderResponse>

    <C0>

       <FT>COR</FT>

       <TT>SO</TT>

       <ORDNO>1COPY004</ORDNO>

       <OT>N</OT>

       <RSYS>SOP</RSYS>

       <TSYS>BND</TSYS>

       <WC>803921</WC>

       <DD>20020730</DD>

    </C0>

       <CUR>
          <STAT>EINP</STAT>
          <MSG>
              <CODE>0173</CODE>
              <TEXT>LSO NOT FOUND IN WC MAPPING REFERENCE TABLE</TEXT>
              <ERRTAGPATH>DELV.SR.RU.DATA.CKT.NEWACL.LSO</ERRTAGPATH>
        </MSG>
       </CUR>
</OrderResponse>
```

Notices

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

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

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

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

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

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

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

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

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

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800

Burlingame, CA 94010
U.S.A

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

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

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

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

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

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

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
Domino
Lotus
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

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.

WebSphere Business Integration Adapter Framework, V2.4.0

Printed in USA