WebSphere Business Integration Adapters

Adapter for SAP Exchange Infrastructure (SAP XI) User Guide

V1.0.x

Note!
Before using this information and the product it supports, read the information in “Notices” on page 85.“Notices.”

19December003

This edition of this document applies to connector version 2.4.x and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

Contents

New in this release

New in release 1.0.0

Version 1.0.0 of the of the adapter for SAP Exchange Infrastructure.

About this document

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

This document describes the configuration, and business object development for the adapter for SAP Exchange Infrastructure (SAP XI).

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 SAP Exchange Interface specifications.

Related documents

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

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

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
- 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`.
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the product is installed on your system.
`%text%` and *`$text`*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the *`text`* UNIX environment variable.
ProductDir	Represents the directory where the product is installed.

Chapter 1. Overview

The connector for SAP Exchange Infrastructure is a runtime component of the WebSphere Business Integration Adapter for SAP Exchange Infrastructure. The connector allows the WebSphere integration broker to exchange business objects with applications that send or receive SAP Exchange Infrastructure messages. This chapter describes the connector component and the relevant business integration system architecture.

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

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

This document contains information about the application-specific component and connector framework. It refers to both of these components as the connector.

For more information about the relationship of the integration broker to the connector, see the *IBM WebSphere InterChange Server System Administration Guid*e.

Note: All WebSphere business integration adapters operate with an integration broker. The connector for SAP Exchange Infrastructure operates with:

- the InterChange Server integration broker, which is described in the *Technical Introduction to IBM WebSphere InterChange Server*
- the WebSphere Application Server (WAS) integration broker, which is described in *Implementing Adapters with WebSphere Application Server*

The adapter for SAP Exchange Infrastructure 1.0.0 is targeted to support a major segment of the SAP Exchange Infrastructure standards. SAP Exchange Infrastructure (SAP XI) enables you to implement cross-system business processes.

Adapter architecture

The SAP XI adapter allows IBM WebSphere Business Integration Collaborations to asynchronously exchange Business Objects with SAP XI. The adapter uses an MQ implementation of the Java™ Message Service (JMS), an API for accessing enterprise-messaging systems that also makes possible guaranteed event delivery.

The message exchange between SAP XI integration servers and SAP XI adapters is carried out using the WebSphere MQ queues. On the SAP XI side, JMS Inbound and JMS Outbound adapters are configured for WebSphere MQ interaction.

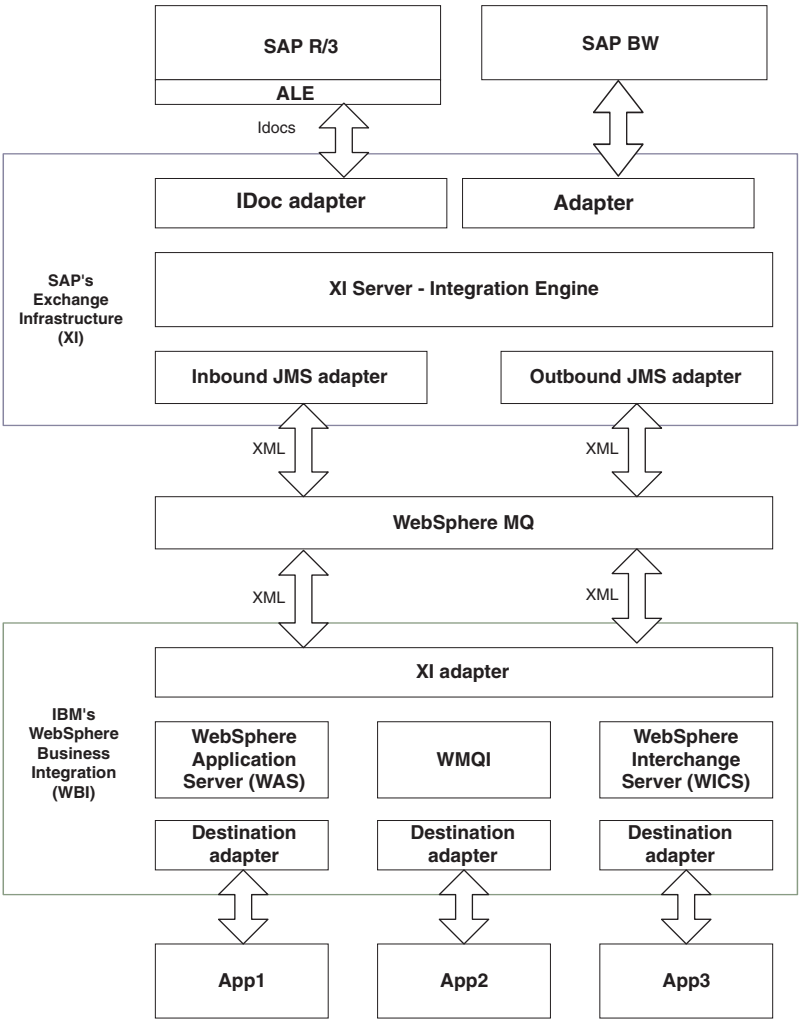

Figure 1. Architecture diagram

The adapter for SAP Exchange Infrastructure is metadata-driven. Message routing and format conversion are initiated by an event polling technique. During event processing, the adapter retrieves SAP XI messages from queues, calls data handlers to convert those messages to their corresponding business objects, and then delivers the business objects to the Integration Broker. In the opposite direction, during request processing, the adapter receives business objects from the Integration Broker, converts them into SAP XI messages using the same data handler, and then delivers the messages to an SAP XI Integration Server.

The SAP XI adapter uses the XML data handler to process SAP XI messages. For information about how to configure this data handler to work with the SAP Exchange Infrastructure connector, see Chapter 3, "XML data handler," in the *Data Handler Guide*.

XML and business objects

The type of business object and verb used in processing a message is based on the XML root element name contained in the message body. The connector uses

metaobject entries to determine business object name and verb. For each message, you construct a metaobject to store the business object name and verb to associate with the SAP Exchange Infrastructure message root element.

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

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 metaobject (if specified in the business object). The child metaobject values can direct the connector to populate attributes of the business object 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, using the XML root element name. 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 the Integration Broker using the `gotApplEvents()` method.

Application-connector communication

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 a WebSphere business integration system business object from an Integration Broker, the connector passes the business object to the data handler. The data handler converts the business object into XML 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 routes the message.

For each interface you want to support SAP XI requires definitions of a separate inbound JMS adapter instance bound to separate queues. For example, if you want to support request processing for MATMAS and ORDERS IDocs, you need to configure two separate instances of Inbound JMS adapters in SAP XI. The same instance of the WebSphere adapter for SAP Exchange Infrastructure, can handle multiple request messages and can post to different queues.

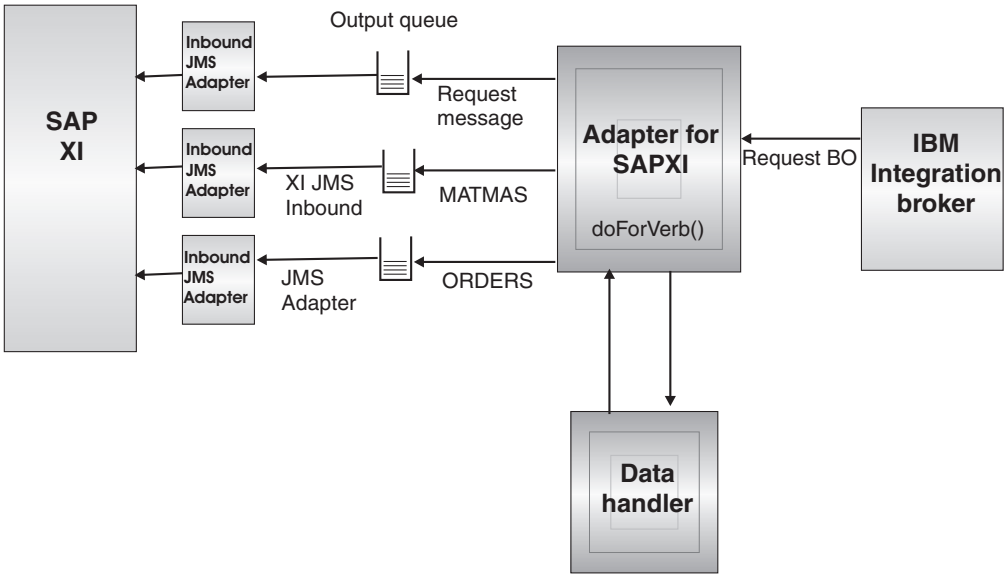

Figure 2. Application-connector communication method: Message request

Event delivery

Figure 3 illustrates the event delivery direction. When SAP XI Outbound JMS Adapter posts a message to the `Input.Queue` of the WebSphere adapter for SAP XI, 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 metaobjects, the connector verifies that the message type (i.e., XML) is supported. The connector then passes the message to the configured data handler, which converts the message into a WebSphere business integration business system business object. The verb that is set reflects the conversion properties established for the message type. The connector then delivers the business object to InterChange Broker, and the message is removed from the in-progress queue.

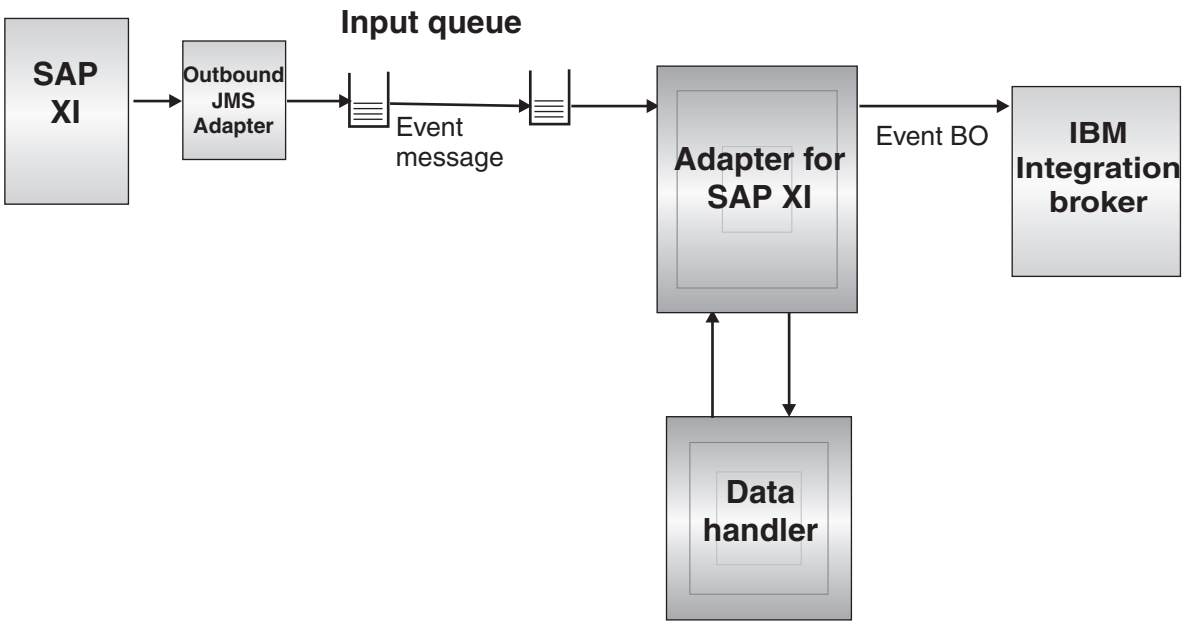

Figure 3. Application-connector communication method: Event delivery

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 instructions in 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.

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 XI generates XML messages and stores them on the MQ message queue.

Retrieval

The connector uses the `pollForEvents()` method to poll the MQ queue at regular intervals for messages. When the connector finds a message, it retrieves it from the MQ 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 metaobject, 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 42 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 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.

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 42.

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.

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 WebSphere MQ message and issues it. There are no requirements regarding the type of business objects processed except those of the data handler.

Verb processing

The connector processes business objects passed to it by a collaboration 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. The connector supports the following business object verbs:

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

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

Create, update, and delete

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

Asynchronous delivery

This is the default delivery mode for business objects with Create, Update, and Delete 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 `BON_SUCCESS`, else `BON_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 SAP Exchange Infrastructure, 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, Update, or Delete	SUCCESS VALCHANGE	(Optional) A serialized business object reflecting changes.
	VALDUPES FAIL	(Optional) An error message.

WebSphere MQ feedback code	Equivalent CrossWorlds response*
MQFB_PAN or MQFB_APPL_FIRST	SUCCESS
MQFB_NAN or MQFB_APPL_FIRST + 1	FAIL
MQFB_APPL_FIRST + 2	VALCHANGE
MQFB_APPL_FIRST + 3	VALDUPES
MQFB_APPL_FIRST + 4	MULTIPLE_HITS
MQFB_APPL_FIRST + 5	FAIL_RETRIEVE_BY_CONTENT
MQFB_APPL_FIRST + 6	BO_DOES_NOT_EXIST
MQFB_APPL_FIRST + 7	UNABLE_TO_LOGIN
MQFB_APPL_FIRST + 8	APP_RESPONSE_TIMEOUT (results in immediate termination of connector agent)

*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 WebSphere business integration system 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 WebSphere business integration system 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 metaobject 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 WebSphere MQ feedback codes to override default interpretations by specifying the connector property FeedbackCodeMappingMO. This property allows you to create a metaobject in which all WebSphere business integration system-specific return status values are mapped to the WebSphere MQ feedback codes. The return status assigned (using the metaobject) to a feedback code is passed to InterChange Server.

Retrieve, exists and retrieve by content

Business objects with the Retrieve, Exists, and Retrieve By Content verbs support synchronous delivery only. The connector processes business objects with these verbs as it does for the synchronous delivery defined for create, update and delete. However, when using Retrieve, Exists, and Retrieve By Content verbs, the `responseTimeout` and `replyToQueue` are required. Furthermore, for Retrieve By Content and Retrieve verbs, the message body must be populated with a serialized business object to complete the transaction.

The table below shows the response messages for these verbs.

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

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 preserve 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.

Common configuration tasks

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 for information on installing the adapter.

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 ″Installing and configuring the adapter.″

When you configure connector properties for the adapter for SAP Exchange Infrastructure, make sure that:

- The value specified for connector property HostName matches that of the host of your WebSphere MQ server.
- The value specified for connector property Port matches that of the port for the listener of 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.

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 metaobject to specify the target queue and format. For more on static and dynamic metaobjects, see Chapter 2, ″Installing and configuring the adapter.″
- Set the property ResponseTimeout in the (static or dynamic) metaobject to -1. This forces the connector to issue the business object without checking for a return.

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″ with this exception: you specify a positive ResponseTimeout value to indicate how long the connector waits for a reply.
- See ″Create, update, and delete″ on page 7 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 ″Configuring meta-object attributes.″

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 metaobject. To create and configure this object:

- Follow the steps in ″Static metaobjects″ in Chapter 2.
- Make sure the connector subscribes to the static metaobject by specifying the name of the static metaobject in the connector-specific property ConfigurationMetaObject.

Configuring a dynamic metaobject

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

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

The connector will check for the dynamic metaobject and use its information to determine how to process the business object. For more information, see ″Dynamic metaobject.″

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 metaobject.

Configuring queue URIs

To configure queues for use with the adapter for SAP Exchange Infrastructure:

- 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 the WebSphere MQ programming guide.

Configuring data handlers

There are two ways to configure a data handler:

- Specify the data handler class name in the connector-specific property DataHandlerClassName.
- Or specify both a mime type and the data handler metaobject 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*.

Modifying the startup script

See Chapter 2 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 WebSphere MQ server.

Chapter 2. Installing and configuring the adapter

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

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 1.0.0 version of the adapter for SAP Exchange Infrastructure is supported on the following adapter framework and integration brokers:

- Adapter framework: WebSphere Business Integration Adapter Framework, version 2.4.x.
- Integration brokers:
  - WebSphere InterChange Server, versions 4.1.1 and 4.2.x .
  - WebSphere Application Server Enterprise, version 5.0.1, with WebSphere Studio Application Developer Integration Edition, version 5.0.

See *Release Notes* for any exceptions.

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/apserv/library.html

Required Software

The adapter for SAP XI runs on the following platforms:

- Windows 2000 Server
- Solaris 7, 8 or AIX 5.1, 5.2 or HP UX 11.i

The connector supports interoperability with applications via WebSphere MQ 5.1, 5.2.

Note: The SAP Exchange Infrastructure adapter does not support Secure Socket Layers (SSL) in WebSphere MQ 5.3. For the WebSphere MQ software version appropriate to adapter framework-integration broker communication, see the Installation Guide for your platform (Windows/Unix).

In addition, you must have the IBM WebSphere MQ Java client libraries.

Installing the SAP XI adapter

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

Navigating the installed file structure

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

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

Windows file structure

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

The utility installs the connector into the *ProductDir*`\connectors\WebSphereMQConnector` 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\WebSphereMQ\CWWebSphereMQ.jar`	Contains classes used by the WebSphere MQ connector only
`connectors\WebSphereMQConnector\start_WebSphereMQ.bat`	The startup script for the connector (NT/2000)
`connectors\messages\WebSphereMQConnector.txt`	Message file for the connector
`repository\WebSphereMQ\CN_WebSphereMQ.txt`	Repository definition for the connector
`connectors\WebSphereMQ\samples\LegacyContact\WebSphereMQConnector.cfg`	Sample WebSphere MQ configuration file
`connectors\WebSphereMQ\samples\LegacyContact\PortConnector.cfg`	Sample Port connector configuration file
`connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _LegacyContact.xsd`	Sample schema
`connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _MO_Config.xsd`	Sample meta-object
`connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _MO_DataHandler.xsd`	Sample data handler meta-object

Subdirectory of *ProductDir*	Description
connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _MO_DataHandler_DelimitedConfig.xsd	Sample delmited data handler meta-object
connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _DynMO_Config.xsd	Sample dynamic meta-object
connectors\WebSphereMQ\samples\LegacyContact\JMSPropertyPairs.xsd	Sample JMS properties

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

UNIX file structure

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

The utility installs the connector into the *ProductDir*/connectors/WebSphereMQConnector 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/WebSphereMQ/CWWebSphere MQ.jar	Contains classes used by the SAP Exchange Infrastructure connector only
connectors/WebSphereMQ/start_WebSphereMQ.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/WebSphereMQ/Connector.txt	Message file for the connector
repository/WebSphereMQ/CN_WebSphere MQ.txt	Repository definition for the connector
connectors/WebSphereMQ/samples/LegacyContact/WebSphereMQConnector.cfg	Sample WebSphere MQ configuration file
connectors/WebSphereMQ/samples/LegacyContact/PortConnector.cfg	Sample Port connector configuration file
connectors/WebSphereMQ/samples/LegacyContact/Sample_WebSphereMQ _LegacyContact.xsd	Sample schema
connectors/WebSphereMQ/samples/LegacyContact/Sample_WebSphereMQ _MO_Config.xsd	Sample meta-object
connectors/WebSphereMQ/samples/LegacyContact/Sample_WebSphereMQ _MO_DataHandler.xsd	Sample data handler meta-object
connectors/WebSphereMQ/samples/LegacyContact/Sample_WebSphereMQ _MO_DataHandler_DelimitedConfig.xsd	Sample delmited data handler meta-object
connectors/WebSphereMQ/samples/LegacyContact/Sample_WebSphereMQ _DynMO_Config.xsd	Sample dynamic meta-object
connectors/WebSphereMQ/samples/LegacyContact/JMSPropertyPairs.xsd	Sample JMS properties

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

Configuring the connector

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.

Use Connector Configurator to configure connector properties:

- For a description of Connector Configurator and step-by-step procedures, see Appendix A, “Connector Configurator,” on page 67.
- For a description of standard connector properties, see Appendix A, “Connector Configurator,” on page 67.

- 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 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 adapter. See the sections that follow for explanations of the properties.

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

Name	Possible values	Default value	Required
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*		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 WebSphere MQ.

Default = None.

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

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

Default = None.

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

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 "Understanding Queue Uniform Resource Identifiers" on page 26.

Default = null.

Channel
MQ server connector channel through which the connector communicates with WebSphere MQ.

Default = none.

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

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

Default = none.

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

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

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

Default = `MO_DataHandler_Default`

DataHandlerMimeType
Allows you to request a data handler based on a particular MIME type.

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. The connector accepts the following attribute values representing WebSphere MQ-specific feedback codes:

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

The connector accepts the following WebSphere business integration system-specific status codes as attribute values in the meta-object:

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

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 WebSphere MQ.

Default = none.

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

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

Default = `Reprocess`.

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

If the `InputQueue` property is not supplied, the connector will start up properly, print a warning message, and perform request processing only. It will perform no event processing.

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

Since we have a PollQuantity of 2, the connector will retrieve at most 2 messages from each queue per call to pollForEvents. For the first cycle (1 of 2), the connector retrieves the first message from each of MyQueueA, MyQueueB, and MyQueueC. That completes the first round of polling and if we had a PollQuantity of 1, the connector would stop. Since we have a PollQuantity of 2, the connector starts a second round of polling (2 of 2) and retrieves one message each from MyQueueA and MyQueueC--it skips MqQueueB since it is now empty. After polling all queues 2x each, the call to the method pollForEvents is complete. Here's the sequence of message retrieval:

1. 1 message from `MyQueueA`
2. 1 message from `MyQueueB`
3. 1 message from `MyQueueC`
4. 1 message from `MyQueueA`

5. Skip MyQueueB since it's now empty
6. 1 message from MyQueueC

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

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 WebSphere MQ listener.

Default = None.

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, WebSphere MQ message properties, and dynamic child meta-object attributes" on page 37.

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

UnsubscribedQueue
Queue to which messages that are not subscribed are sent.

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

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

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

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

- If the connector uses a JMS event store (implemented as a JMS source queue), the connector framework can manage the JMS event store.

- If the connector uses a non-JMS event store (for example, implemented as a JDBC table, Email mailbox, or flat files), the connector framework can use a JMS monitor queue to ensure that no duplicate events occur.

Guaranteed event delivery for connectors with JMS event stores

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

Enabling the feature for connectors with JMS event stores: To enable the guaranteed-event-delivery feature for a JMS-enabled connector that has a JMS event store, set the connector configuration properties to values shown in Table 1.

Table 1. Guaranteed-event-delivery connector properties for a connector with a JMS event store

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

In addition to configuring the connector, you must also configure the data handler that converts between the event in the JMS store and a business object. This data-handler information consists of the connector configuration properties that Table 2 summarizes.

Table 2. Data-handler properties for guaranteed event delivery

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

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

If you configure a connector that has a JMS event store to use guaranteed event delivery, you must set the connector properties as described in Table 1 and Table 2. To set these connector configuration properties, use the Connector Configurator tool. Connector Configurator displays the connector properties in Table 1 on its Standard Properties tab. It displays the connector properties in Table 2 on its Data Handler tab.

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

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

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

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

   The connector framework calls the data handler that has been configured with the properties in Table 2.
4. When WebSphere MQ Integrator Broker is the integration broker, convert the business object to a message based on the configured wire format (XML).
5. Send the resulting message to the JMS destination queue.
   If you are using the WebSphere ICS integration broker, the message sent to the JMS destination queue is the business object. If you are using WebSphere MQ Integrator broker, the message sent to the JMS destination queue is an XML message (which the data handler generated).
6. Commit the JMS transaction.

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

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

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

Enabling the feature for connectors with non-JMS event stores: To enable the guaranteed-event-delivery feature for a JMS-enabled connector that has a non-JMS event store, you must set the connector configuration properties to values shown in Table 3.

Table 3. Guaranteed-event-delivery connector properties for a connector with a non-JMS event store

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

If you configure a connector to use guaranteed event delivery, you must set the connector properties as described in Table 3. To set these connector configuration properties, use the Connector Configurator tool. It displays these connector properties on its Standard Properties tab. For information on Connector Configurator, see Appendix A.

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

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

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

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

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

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

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

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

Understanding Queue Uniform Resource Identifiers

The Uniform Resource Identifiers (URIs) for a queue begins with the sequence `queue://` followed by:

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

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

```
queue://crossworlds.queue.manager/MQCONN.IN?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.

Property name	Description	Values
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.
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 adapter has no control of the character set (CCSID) or encoding attributes of data in MQMessages. Because data conversion is applied as the data is retrieved from or delivered to the message buffer, the 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 WebSphere business integration system Technical Support to discuss the possibility of using an alternate Java Virtual Machine to run the connector.

Configuring meta-object attributes

The connector for SAP Exchange Infrastructure 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-objects

The SAP Exchange Infrastructure 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 40.
`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 "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.

Property name	Description
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.
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.
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.
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.

Property name	Description
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.

Additionally, a reserved property named Default can be defined in the meta-object. When this property is present, its application-specific information specifies default values for all business object conversion properties.

Consider the following sample meta-object.

Property name	Application-specific text
Default	DataEncoding=Text:UnicodeLittle; OutputFormat=CUST_OUT; OutputQueue=QueueA;ResponseTimeout=10000; TimeoutFatal=False

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 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.

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=CustomerQueue1;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 `CustomerQueue1` 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 `Customer_Create` indicates to the connector that any messages of format `NEW` should be converted to a Customer 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`).

Customer object with verbs update and delete: Input format `MODIFY` is overloaded—defined for both business object Customer with verb Update and business object Customer with verb Delete. In order to successfully process retrieved messages of this format, the business object name and possibly the verb should be contained in the message content for the data handler to identify. For Request processing operations, the connector will send messages for either verb using the input format `MODIFY` since an output format is not defined.

Customer object with verb retrieve: Attribute `Customer_Retrieve` specifies that business objects of type Customer with verb Retrieve should be sent as messages with format Retrieve. Note that the default response time has been overridden so that the connector will wait up 10000 milliseconds before timing out (it will still terminate if a response is not received).

```
[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=CustomerQueue1;ResponseTimeout=5000;TimeoutFatal=true
IsRequiredServerBound = false
[End]
[Attribute]
Name = Customer_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputFormat=NEW
IsRequiredServerBound = false
[End]
[Attribute]
Name = Customer_Update
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputFormat=MODIFY
IsRequiredServerBound = false
[End]
[Attribute]
Name = Customer_Delete
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputFormat=MODIFY
IsRequiredServerBound = false
[End]
[Attribute]
Name = Customer_Retrieve
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = OutputFormat=RETRIEVE;ResponseTimeout=10000
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
```

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

[Verb]
Name = Delete
[End]

[Verb]
Name = Retrieve
[End]

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

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 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 business object `Customer_Create`. Note that the application-specific information consists of semi-colon delimited name-value pairs.

Property name	Value
DataEncoding	Text:UnicodeLittle
DataHandlerMimeType*	text/delimited
OutputFormat	CUST_OUT
OutputQueue	QueueA
ResponseTimeout	10000
TimeoutFatal	False

*Assumes that DataHandlerConfigMO has been specified in either the connector configuration properties or 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 collaborations with more information regarding messages retrieved during polling, the connector populates specific attributes of the dynamic meta-object, if already defined for the business object created.

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

Property 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 collaborations.

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]

[Verb]
Name = Delete
[End]

[Verb]
Name = Retrieve
[End]

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

[Attribute]
Name = FirstName
```

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

[Verb]
Name = Create
[End]

[Verb]
Name = Delete
[End]

[Verb]
Name = Retrieve
[End]

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

JMS headers, WebSphere MQ 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 WebSphere MQ header properties, are recognized in the dynamic meta-object.

Table 4. 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 5. 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`).

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 WebSphere MQ 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 WebSphere MQ 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 ReplyToQueuevalue specified in the connector configuration properties. If you additionally specify a negative ResponseTimeout (meaning that the connector should not wait for a response), theReplyToQueue 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.

Configuring the startup file

Before you start the connector for SAP Exchange Infrastructure, you must configure the startup file.

Windows

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

1. Open the start_WebSphereMQ.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_WebSphereMQ.sh file:

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

Starting the adapter

For information on starting a connector, stopping a connector, and the connector's temporary startup log file, see the see the startup chapter in the *System Installation Guide* for your platform.

Error handling

All error messages generated by the connector are stored in a message file named WebSphere MQConnector.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.

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 adapter," on page 15 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.

Chapter 3. Creating or modifying business objects

- "Creating business objects"
- "Generating busines objects"

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.

Creating business objects

In WebSphere business integration system business objects, metadata 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 adapter 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 SAP XI, your modifications must conform to the rules the connector is designed to follow, so that the connector can correctly process new or modified business objects.

After installing the adapter, create business objects. There are no requirements regarding the structure of the business objects other than those imposed by the configured data handler. The business objects that the connector processes can have any name allowed by Integration Broker. For more on naming conventions, see *Naming CrossWorlds Components*.

Generating busines objects

There are two methods for generating business objects. The preferred method is to generate business objects from imported IDOC structures. The alternate method is to generate business objects from XML schema definitions in the SAP Interface Repository (IFR).

Generating Business Objects from XML Schema (XSD) definitions

XML Object Discovery Agent (ODA) generates business objects from any schema or document type definition (DTD) that conforms to W3C guidelines. For more information about this functionality, see the *XML ODA User Guide*. To generate business objects for the SAP XI adapter:

1. Download the XML Schema definition from a URL location or from the SAP XI Repository.
2. Launch XML ODA.

Note: When using Business Objects generated from XML Schema definitions, you may need to define interface mapping in SAP XI.

Generating business objects from IDoc structures

The preferred method for generating business objects is to import them from existing IDoc structures. To generate business objects for the SAP XI adapter:

1. Start the SAP XI repository.
2. Double-click on the software component with which you want to work. (For information on maintaining software components in the System Landscape directory (SLD), refer to the SAP XI documentation.)
3. Maintain connection data for the SAP system.
4. Under your software interface, right-click on Imported Objects.
5. Choose Import RFC/IDoc.
6. Save the imported IDoc structure to a local XSD schema file.
7. Use the IDoc schema file with XML ODA to generate business objects.

Generating business objects from IFR IDoc XML schemas

XML Object Discovery Agent (ODA) generates business objects from any schema or document type definition (DTD) that conforms to W3C guidelines. For more information about this functionality, see the *XML ODA User Guide*. To generate business objects for the SAP XI adapter:

1. Connect to http://ifr.sap.com.
2. Choose Enter the Interface Repository from the navigation bar.
3. Download the object you need, and save it as an .xsd file.
4. Launch XML ODA and use Business Object Designer to connect to the ODA and load the .xsd file.

Note: When using Business Objects generated from IFR IDoc schemas, you need to define interface mapping in SAP XI. The interface mapping should be defined for the IFR schema to the imported IDOC struture.

Sample business object properties

Below is a sample business object properties for an SAP XI connector with the XML data handler.

```
 [ReposCopy]Version = 3.0.0

[End]

[BusinessObjectDefinition]

Name = MATMAS01

Version = 3.0.0

AppSpecificInfo = elem_fd=unqualified;attr_fd=unqualified

    [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 = MATMAS01_MATMAS01
ContainedObjectVersion = 3.0.0
Relationship = Containment
Cardinality = 1
    MaxLength = 255
IsKey = false
    IsForeignKey = false
    IsRequired = false
    AppSpecificInfo = elem_name=MATMAS01
IsRequiredServerBound = false
    [End]
    [Attribute]
    Name = ObjectEventId
    Type = String
Cardinality = 1
    MaxLength = 255
    IsKey = false
    IsForeignKey = false
IsRequired = false
    IsRequiredServerBound = false
```

```
    [End]
    [Verb]
    Name = Create
    [End]
    [Verb]
Name = Delete
    [End]
    [Verb]
Name = Retrieve
    [End]
    [Verb]
    Name = Update
    [End]
```

Chapter 4. Standard configuration properties for connectors

This appendix describes the standard configuration properties for WebSphere Business Integration adapter connectors. The information covers connectors running on the following brokers:

- WebSphere InterChange Server (ICS)
- WebSphere MQ Integrator Broker (WMQI)
- WebSphere Application Server (WAS)

Not every connector makes use of all these standard properties. When you select a 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 either added or deleted in the 2.3 release of the adapters.

New properties

- RFH2Message Domain
- ListenerConcurrency
- RestartCount
- WsifSynchronousRequestTimeout

Deleted properties

None

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 the WebSphere MQ Integrator Broker, 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 6 on page 51 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.

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 6. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdminInQueue	*Valid JMS queue name*	*CONNECTORNAME* /ADMININQUEUE		Delivery Transport is JMS
AdminOutQueue	*Valid JMS queue name*	*CONNECTORNAME*/ADMINOUTQUEUE		Delivery Transport is JMS
AgentConnections	1-4	1	Component restart	ICS: Delivery Transport is MQ or IDL
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	*application name*	*The value that is specified for the connector application name*	Component restart	Value required
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	No value	Component restart	
ContainerManagedEvents	No value or JMS	JMS		Guaranteed event delivery
ControllerStoreAndForwardMode	true or false	True	Dynamic	ICS only
ControllerTraceLevel	0-5	0	Dynamic	ICS only
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	MQ, IDL, or JMS	JMS	Component restart	For WAS or WMQI: JMS only
DuplicateEventElimination	True/False	False	Component restart	JMS transport only: Container Managed Events must be <NONE>
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	
jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory or CxCommon.Messaging.jms.SonicMQFactory or any Java class name	CxCommon.Messaging.jms.IBMMQSeriesFactory	Server restart	JMS transport only
jms.MessageBrokerName	If FactoryClassName is IBM, use crossworlds.queue.manager. If FactoryClassName is Sonic, use localhost:2506.	crossworlds.queue.manager	Server restart	JMS transport only
jms.NumConcurrentRequests	Positive integer	10	Component restart	JMS transport only
jms.Password	Any valid password		Server restart	JMS transport only

Table 6. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
jms.UserName	Any valid name		Server restart	JMS transport only
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	ICS only
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	ICS only
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	ICS only
ListenerConcurrency	1- 100	1	Component restart	ICS only: Delivery Transport must be `MQ`
Locale	`en_US`, `ja_JP`, `ko_KR`, `zh_C`, `zh_T`, `fr_F`, `de_D`, `it_I`, `es_E`, `pt_BR` **Note:** This is a subset of the supported locales.	`en_US`	Component restart	
LogAtInterchangeEnd	`True` or `False`	`False`	Component restart	ICS only
MaxEventCapacity	1-2147483647	2147483647	Dynamic	ICS: Repository Directory must be <REMOTE>
MessageFileName	*path/filename*	*`Connectorname`*`.txt` or `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	ICS only: Repository Directory must be <REMOTE>
OADMaxNumRetry	*A positive number*	`1000`	Dynamic	ICS only: Repository Directory must be <REMOTE>
OADRetryTimeInterval	*A positive number in minutes*	`10`	Dynamic	ICS only: 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	

Table 6. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
PollQuantity	1-500	1	Component restart	JMS transport only: DuplicateEvent Elimination must be True
PollStartTime	HH:MM*(HH is 0-23, MM is 0-59)*	HH:MM	Component restart	
RepositoryDirectory	Location of meta-data repository		Component restart	For ICS: set to <REMOTE> For WMQI and WAS: set to *<local directory>*
RequestQueue	*Valid JMS queue name*	*CONNECTORNAME*/REQUESTQUEUE	Component restart	
ResponseQueue	*Valid JMS queue name*	*CONNECTORNAME*/RESPONSEQUEUE	Component restart	
RestartCount	0-100		Dynamic	Connector must be in polling mode
RestartRetryCount	0-99	3	Dynamic	
RestartRetryInterval	*A sensible positive value in minutes*	1	Dynamic	
RHF2MessageDomain	mrm, xml	mrm	Component restart	Only if Delivery Transport is JMS and WireFormat is CwXML
SourceQueue	*Valid WebSphere MQ name*	*CONNECTORNAME*/SOURCEQUEUE	Component restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue		*CONNECTORNAME*/ SYNCHRONOUSREQUESTQUEUE	Component restart	
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	
SynchronousResponseQueue		*CONNECTORNAME*/ SYNCHRONOUSRESPONSEQUEUE	Component restart	
WireFormat	CwXML, CwBO	CwXML	Component restart	CwXML for WMQI and WAS; CwBO if Repository Directory is <REMOTE> (ICS)
WisfSynchronousRequest Timeout	0 - any number (millisecs)	0	Component restart	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

WebSphere ICS only.

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, WMQI 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 ASCII for this property. If you previously configured the value of this property to `ascii7` or `ascii8`, you must reconfigure the connector to use either `ASCII` or one of the other supported values.

Important: By default only a subset of supported character encodings display in the drop list. To add other supported values to the drop list, you must

manually modify the \Data\Std\stdConnProps.xml file in the product directory. For more information, see the appendix on Connector Configurator.

The default value is ascii.

ConcurrentEventTriggeredFlows

WebSphere ICS only.

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 JMS. It can also be set to no value.

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

- PollQuantity = 1 to 500
- SourceQueue = SOURCEQUEUE

You must also configure a data handler with the MimeType, DHClass, 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

WebSphere ICS only.

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

WebSphere ICS only.

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

DeliveryQueue

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

The default value is DELIVERYQUEUE.

DeliveryTransport

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

- If ICS is the broker type, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.
- If WMQI is the broker type, the value may only be JMS.
- If WAS is the broker type, 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

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 integration broker is ICS and the `Repository Directory` is <REMOTE>.

Specifies whether the Object Activation Daemon (OAD) automatically attempts to restart the application-specific component after an abnormal shutdown. This property is required for automatic restart.

The default value is `false`.

OADMaxNumRetry

Valid only when the integration broker is ICS and the `Repository Directory` is <REMOTE>.

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

The default value is `1000`.

OADRetryTimeInterval

Valid only when the integration broker is ICS and the `Repository Directory` is <REMOTE>.

Specifies the number of minutes for the interval during which the OAD automatically attempts to restart the application-specific component after an

abnormal shutdown. If the application-specific component does not start within the specified interval, the OAD repeats the attempt as many times as specified in "OADMaxNumRetry" on page 60.

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 `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 WMQI or WAS, this value must be set to *<local directory>*.

ResponseQueue

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

RestartCount

Causes the connector to shut down and restart automatically after it has processed a set number of events. You set the number of events in `RestartCount.` The connector must be in polling mode (set `PollFrequency` to "p") for this property to take effect.

Once the set number of events has passed through request processing, the connector is shut down and restarted the next time it polls.

RestartRetryCount

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

The default is `3`.

RestartRetryInterval

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

The default is `1`.

RHF2MessageDomain

WebSphere MQ Integrator Broker only.

This property allows you to configure the value of the field domain name in the JMS header. When data is sent to WebSphere MQ Integrator Broker over JMS transport, the connector framework writes JMS header information, with a domain name and a fixed value of `mrm`. A configurable domain name enables users to track how WebSphere MQ Integrator Broker processes the message data.

A sample header would look like this:

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

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

SourceQueue

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 55.

The default value is `SOURCEQUEUE`.

SynchronousRequestQueue

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

SynchronousResponseQueue

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

SynchronousRequestTimeout

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

The default value is 0.

WireFormat

Message format on the transport.

- If the integration broker is WMQI or WAS, the setting is `CwXML.`
- if the integration broker is ICS and the value of `RepositoryDirectory` is <REMOTE>, the setting is`CwBO`.

WisfSynchronousRequest 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.

Chapter 5. Troubleshooting

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

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 `WebSphere MQ_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 `WebSphere MQ_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 `WebSphere MQ_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

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 adapter," on page 15 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. 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 Broker (WMQI)
- WebSphere Application Server (WAS)

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 68).

Each time you install a new adapter, you need to set up a **configuration file** for the connector. This file:

- Sets the standard and application-specific properties for the connector
- Designates which business objects and meta-objects it supports
- Sets the logging and tracing values that the connector will use at run time
- Sets the property values used by messaging and data handlers in the adapter
- Allows you to modify connector properties for an existing connector

You use Connector Configurator to create this configuration file and to modify its settings.

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 69 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 (all brokers).
- From System Manager (ICS and WAS only).

Running Configurator in stand-alone mode

You can run Connector Configurator independently and work with connector configuration files, irrespective of your broker. However, if WMQI is your integration broker, you can only use Connector Configurator in stand-alone mode.

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 Connectivity, WMQI Connectivity, or WAS Connectivity, depending on your broker.

If you are creating a configuration file for use with ICS or WAS as the broker, you may prefer 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 73.)

Running Configurator from System Manager

If ICS or WAS is your integration broker, 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 i**con 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 Connectivity or WAS Connectivity, 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 69.
- 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:
   - **New 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. Connector Configurator provides a template named **None**, containing no property definitions, as a default choice.

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:

- **Edit properties**

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

  A child property is an attribute of another property, the parent property. The parent property can obtain simple values, or child properties, or both. These property relationships are hierarchical. When you create a configuration file from these properties, Connector Configurator will identify hierarchical property sets with a plus sign in a box at the left of any parent property.

- **Property type**

  Choose one of these property types: Boolean, String, Integer, or Time.

- Flags

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

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. 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 left-hand corner of the adapter display panel. 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 **Dependencies** 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, WMQI or WAS connectivity.

Note: To get the choice of WMQI, Connector Configurator must be launched from the Start menu, not from System Manager.

- Complete the remaining fields in the **New Connector** window, as described later in this chapter.

To create a new file for ICS or WAS, 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, must end with the word "connector", and must be consistent with the file name for a connector that is installed on the system. For example, enter `PeopleSoftConnector` if the connector file name is `PeopleSoft.jar`.

     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 WMQI or WAS connectivity.
   - **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>Save to the project**. To save to a project, you must be using ICS or WAS as the broker, and 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 save the file as a configuration file (`*.cfg` file).

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

1. In Connector Configurator, 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 76..
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 handlers (applicable for connectors that use JMS messaging with guaranteed event delivery)

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

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 or **Add Child** to add a child property to a property.
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" on page 75.

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

Connector properties are almost all static and the **Update Method** is `COmponent restart`. For changes to take effect, you must restart the connector after saving the revised connector configuration file.

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.

For you to specify a supported business object, the business objects and their maps must exist in the system.

- Business object definitions and map definitions should be saved into System Manager projects if ICS or WAS is your integration broker.
- Business object definitions and MQ message set files should exist if WMQI is your integration broker.

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, because most application APIs do not support the `Stringent` level.

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

If WMQI is your broker

The MQ message set files (`*.set` files) contain message set IDs that Connector Configurator requires for designating the connector's supported business objects. See *Implementing Adapters with WebSphere MQ Integrator Broker* for information about creating the MQ message set files.

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

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

To specify supported business objects:

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

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.

Configuring 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, you save the connector configuration file. Connector Configurator will save it 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:

For ICS:

- From System Manager, as a file with a `*.con` extension in a an ICS User Project, or
- In a directory that you specify.
- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder, if you are using the file as a local configuration file.

For WMQI:

- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder.

For WAS:

- From System Manager, as a file with a `*.con` extension in a WAS User Project, or
- In a directory that you specify.
- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder.

After you have created the configuration file and set its properties, you need to deploy it to the correct location for your connector.

- If you are using ICS as your integration broker, save the configuration in a System Manager project, and use System Manager to load the file into ICS.
- If you are using WMQI as your integration broker, copy the configuration file to the correct location, which must match exactly the configuration file location specified in the startup file for your connector.
- If you are using WAS as your integration broker, save the file in a WAS user project. Use **File>Export** to create `.wsdl` files that you can then import into WSAD-IE.
  You can also export the configuration file as a `.jar` file to a specified directory.

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 WMQI: *Implementing Adapters with WebSphere MQ Integrator Broker*
- 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 B. Quick Steps

This appendix lists quick steps for configuring the Adapter for SAP XI for request processing and event processing.

Request processing

To configure the adapter for SAP XI for request processing, use the following procedure:

1. Save an XSD definition of an existing XI Repository Interface Object to a file, or download an XSD IDoc definition from http://ifr.sap.com.
2. Generate business objects using XML ODA delivered with the WebSphere Business Integration Adapter for SAP XI.
3. Configure the adapter to support the generated objects.
4. Configure SAP XI, if it is not already configured.
   a. Define a software component for the WebSphere Business Integration Adapter for SAP XI in the System Landscape Directory (SLD).
   b. Configure the Inbound JMS adapters. Each interface requires a separate Inbound JMS adapter.
   c. Maintain the directory.
5. Test request objects from the end application or by using Test Connector if you are sending test requests.

Event processing

To configure the adapter for SAP XI for event processing, use the following procedure:

1. Save an XSD definition of an existing XI Repository Interface Object to a file, or download an XSD IDoc definition from http://ifr.sap.com.
2. Generate business objects using XML ODA delivered with the WebSphere Business Integration Adapter for SAP XI.
3. Configure the WebSphere Business Integration Adapter for SAP XI:
   a. Add support for the generated objects.
   b. Define business processes (collaborations) and mapping if necessary.
   c. For testing, configure the test connector to imitate end applications.
4. Configure SAP XI:
   a. Define a software component for the WebSphere Business Integration Adapter for SAP XI in the System Landscape Directory (SLD).
   b. Configure the Outbound JMS adapters. Each interface requires a separate Outbound JMS adapter.
   c. Maintain the directory.
5. Configure the SAP application to trigger IDoc or other types of messages.
6. Test end-to-end connectivity by triggering events in the SAP application.

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