WebSphere Business Integration Express
and Express Plus for Item Synchronization

IBM

Adapter for JMS User Guide

V 2.4.x

Note!

Before using this information and the product it supports, read the information in “Notices” on page 75.

6February2004

This edition of this document applies to IBM WebSphere Business Integration Express for Item Synchronization, version 4.3.1, IBM WebSphere Business Integration Express Plus for Item Synchronization, version 4.3.1, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

Contents

New in this release . . . v

About this document . . . vii

Audience . . . vii
Prerequisites for this document. . . . vii
Related documents . . . vii
Typographic conventions . . . viii

Chapter 1. Overview . . . 1

Connector architecture . . . 1
Application-connector communication method . . . 2
Event handling . . . 4
Guaranteed event delivery. . . . 6
Business object requests . . . 6
Verb processing . . . 6
Processing locale-dependent data . . . 10

Chapter 2. Installing and configuring the connector . . . 13

Compatibility. . . . 13
Prerequisites . . . 13
Overview of installation tasks . . . 13
Installing the connector and related files. . . . 14
Installed file structure . . . 14
Connector configuration . . . 16
Enabling guaranteed event delivery . . . 22
Meta-object attributes configuration . . . 26
Startup . . . 40

Chapter 3. Creating or modifying business objects. . . . 41

Connector business object structure . . . 41
Error handling . . . 44
Tracing . . . 45

Chapter 4. Troubleshooting . . . 47

Start-up problems . . . 47

Appendix A. Standard configuration properties for connectors . . . 49

Configuring standard connector properties . . . 49
Summary of standard properties . . . 50
Standard configuration properties . . . 53

Appendix B. Connector Configurator Express . . . 63

Overview of Connector Configurator Express . . . 63
Starting Connector Configurator Express . . . 64
Running Configurator Express from System Manager . . . 64
Creating a connector-specific property template . . . 64
Creating a new configuration file . . . 67
Using an existing file . . . 68
Completing a configuration file. . . . 69
Setting the configuration file properties . . . 69
Saving your configuration file . . . 73
Completing the configuration . . . 74

Notices 75
Programming interface information 76
Trademarks and service marks 76

New in this release

In this release, version 2.4.x of the Adapter for JMS is supported on the IBM WebSphere Business Integration Express and Express Plus for Item Synchronization release.

Except where noted, all the information in this guide applies to both IBM$^{(R)}$ WebSphere$^{(R)}$ Business Integration Express for Item Synchronization and IBM(R) WebSphere$^{(R)}$ Business Integration Express Plus for Item Synchronization. The term ″WebSphere Business Integration Express for Item Synchronization″ and its variants refer to both products.

The connector runs on the following platforms:
- Microsoft Windows 2000
- OS400 V5R2 (5722-SS1)
- Red Hat Enterprise Linux WS/ES/AS for Intel 2.1, 2.4 Kernel
- SuSE Linux Enterprise Server 7.3, 2.4 Kernel

About this document

The IBM(R) WebSphere(R) Business Integration Express and Express Plus for Item Synchronization product includes InterChange Server Express, the associated Toolset Express are made up of the following components—InterChange Server Express, the associated Toolset Express product, the Item Synchronization collaboration, and a set of software integration adapters. Together, the components provide business process integration and connectivity among leading e-business technologies and enterprise applications as well as integration with the UCCnet GLOBALregistry.

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

Except where noted, all the information in this guide applies to both IBM(R) WebSphere(R) Business Integration Express for Item Synchronization and IBM(R) WebSphere(R) Business Integration Express Plus for Item Synchronization. The term ″WebSphere Business Integration Express for Item Synchronization″ and its variants refer to both products.

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 JMS application.

Related documents

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

This document contains many references to two other documents: the *Installing IBM WebSphere Business Integration Express and Express Plus for Item Synchronization* and the *User Guide for WebSphere Business Integration Express and Express Plus for Item Synchronization*. If you choose to print this document, you may want to print these documents as well.

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

http://www.ibm.com/websphere/wbiitemsync/express/infocenter

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

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *`italic`*	Indicates a variable name or a cross-reference.
blue outline	A blue outline, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click inside the outline to jump to the object of the reference.
`{ }`	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
`[ ]`	In a syntax line, square brackets surround an optional parameter.
`...`	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, `option[,...]` means that you can enter multiple, comma-separated options.
`< >`	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><`*`connector_name`*`>tmp.log`.
ProductDir	Represents the directory where the product is installed.
`/, \`	In this document, backslashes (\) are used as the convention for directory paths. All WebSphere business integration system pathnames are relative to the directory where the WebSphere business integration system is installed on your system.
`%text%`	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable.

Chapter 1. Overview

- "Connector architecture"
- "Application-connector communication method" on page 2
- "Event handling" on page 4
- "Guaranteed event delivery" on page 6
- "Business object requests" on page 6
- "Verb processing" on page 6
- "Processing locale-dependent data" on page 10

The connector for JMS is a runtime component of the IBM WebSphere Business Integration Adapter for JMS. The connector allows IBM WebSphere InterChange Server Express to exchange business objects with applications that send or receive data in the form of JMS messages.

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.

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 *User Guide for WebSphere Business Integration Express and Express Plus for Item Synchronization*.

Note: All WebSphere business integration adapters operate with an integration broker. The connector for JMS operates with the InterChange Server Express integration broker, which is described in the *User Guide for WebSphere Business Integration Express and Express Plus for Item Synchronization*

Connector architecture

The connector is meta-data-driven. Message routing and format conversion are initiated by an event polling technique.

The connector allows collaborations to asynchronously exchange business objects with applications that issue or receive JMS messages when changes to data occur.

The connector retrieves JMS messages from queues, calls data handlers to convert messages to their corresponding business objects, and then delivers them to collaborations. In the opposite direction, the connector receives business objects from collaborations, converts them into JMS messages using the same data handler, and then delivers the messages to a JMS queue.

You can configure the connector to use any data handler when processing messages. For more information, see the *Data Handler Guide*.

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

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

The connector can poll multiple input queues, polling each in a round-robin manner and retrieving a specified number of messages from each queue. For each message retrieved during polling, the connector adds a dynamic child meta-object (if specified in the business object). The child meta-object values can direct the connector to populate attributes with the format of the message as well as with the name of the input queue from which the message was retrieved.

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

Application-connector communication method

The subsections below discuss and illustrate how the connector interacts with InterChange Server Express and an application.

Message request

Figure 1 illustrates a message request communication. When the `doVerbFor()` method receives a business object from a collaboration, the connector passes the business object to the data handler. The data handler converts the business object into JMS-suitable text and the connector issues it as a message to a queue. There, the JMS layer makes the appropriate calls to open a queue session and route the message.

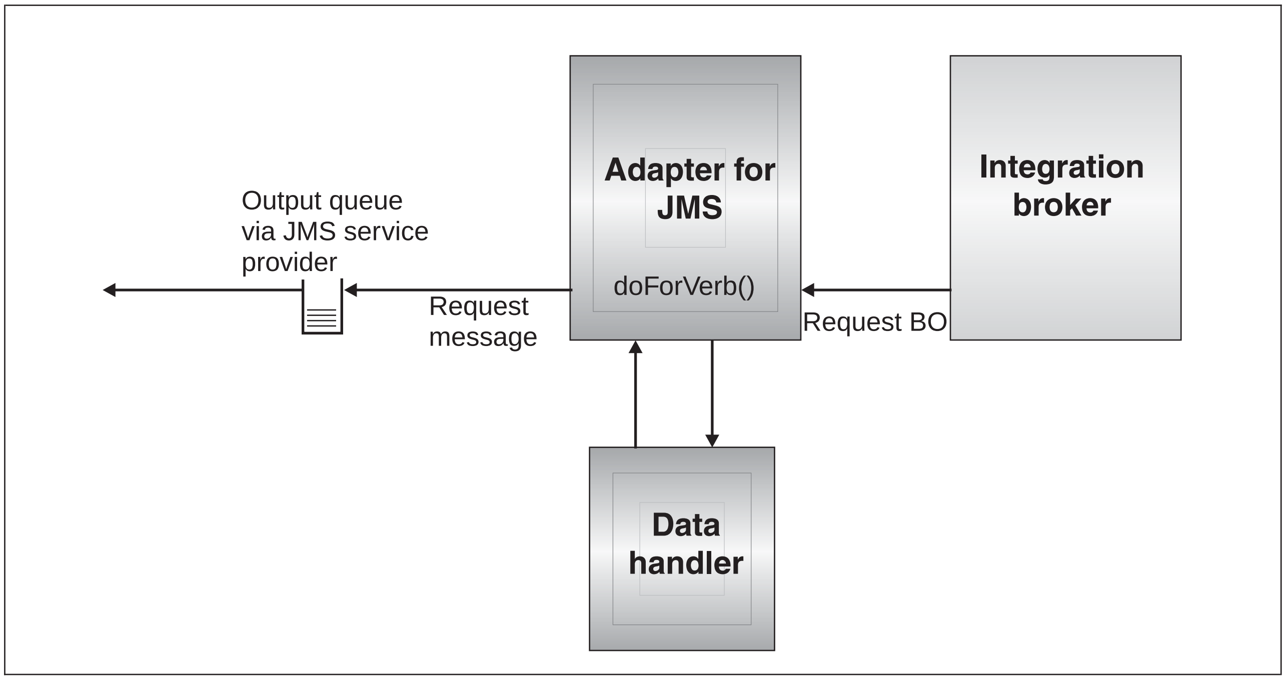


Figure 1. Application-connector communication method: Message request

Message return

Figure 2 shows the message return direction. The `pollForEvents()` method retrieves the next applicable message from the input queue. The message is staged in the in-progress queue where it remains until processing is complete. Using either the static or dynamic meta-objects, the connector first determines whether the message type is supported. If so, the connector passes the message to the configured data handler, which converts the message into a business object. The verb that is set reflects the conversion properties established for the message type. The connector then determines whether the business object is subscribed to by a collaboration. If so, the `gotApplEvents()` method delivers the business object to InterChange Server Express, and the message is removed from the in-progress queue.

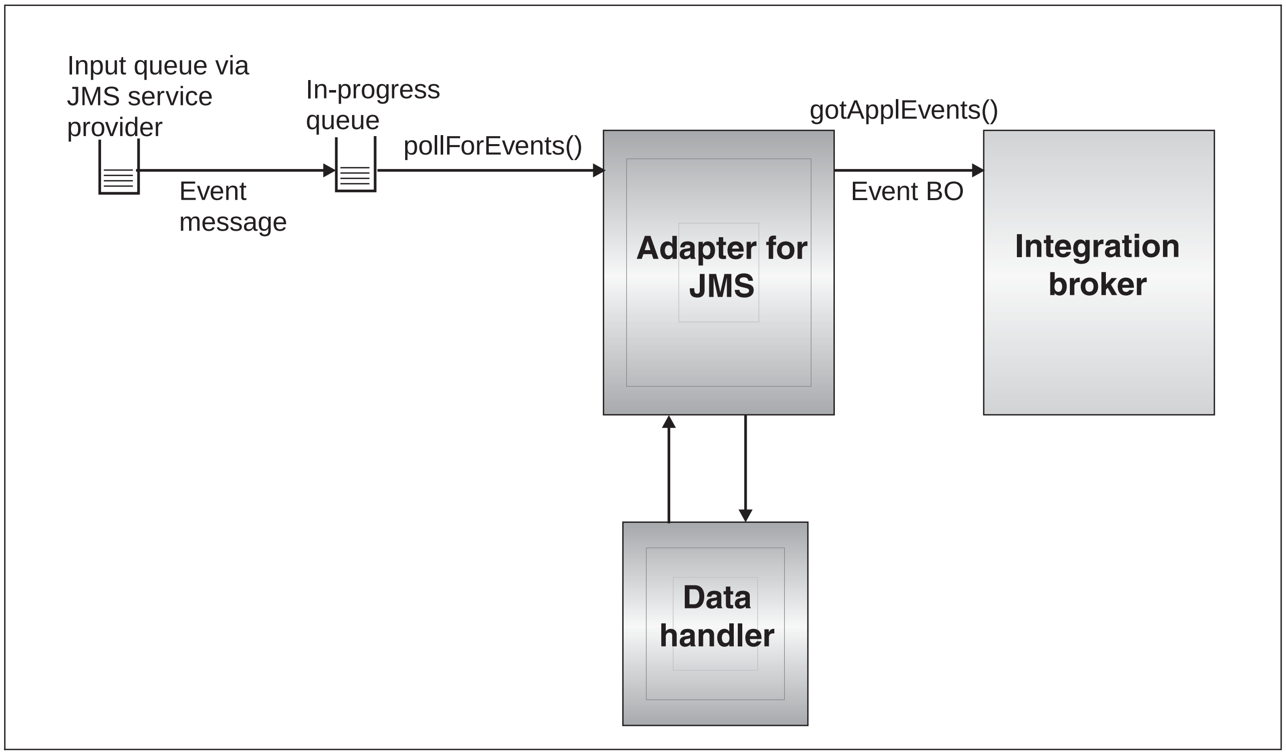


Figure 2. Application-connector communication method: Message return

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 an application or other MQ-capable software generates JMS 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 meta-object, the connector passes only the message body to the data handler; the data handler is expected to determine, create and populate the correct business object for the message. See "Error handling" on page 44 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 Express 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 44.

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

Guaranteed event delivery

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

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

You can configure the guaranteed-event-delivery feature for a JMS-enabled connector with, or without, a JMS event store. To configure the connector for guaranteed event delivery, see "Enabling guaranteed event delivery" on page 22.

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

Business object requests

Business object requests are processed when InterChange ServerExpress sends a business object to the `doVerbFor()` method. Using the configured data handler, the connector converts the business object to a JMS message and issues message. 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 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 JMS, the connector initially issues a message with a header as shown in Table 1.

Table 1. JMS message header

Field	Description	Value
`JMSType`	Message Type Identifier	Output format as defined in the conversion properties if connector property `MessageFormatProperty` is not defined.
`JMSDeliveryMode`	Message delivery Mode	Persistent.
`JMSExpiration`	Message Time-To-Live	No expiration (0)*
`JMSReplyTo`	Destination where a reply to this request should be sent.	Only if a response message is expected will this field be populated.
Optional: property specified by connector property `MessageFormatProperty`	Alternate user-defined message property specified to contain the output format.	Output format as defined in the conversion properties if connector property `MessageFormatProperty` is defined.

* Indicates current value of constant defined by JMS.

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

The 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 Table 2 on page 8 and Table 3 on page 8.

Table 2. Response message descriptor header (MQMD)

Field	Description	Value
`JMSType`	Message Type Identifier	Input format of business object as specified in the conversion properties (if this field was defined in the original request).
Optional: property specified by connector property `MessageFormatProperty`	Alternate user-defined message property specified to contain the output format.	
Property specified by connector property `MessageResponse ResultProperty`.	Result of original request issued by connector.	One of the following strings: “SUCCESS” “FAIL” “VALCHANGE” “VALDUPES” “MULTIPLE_HITS” “FAIL_RETRIEVE_BY_CONTENT” “BO_DOES_NOT_EXIST” “UNABLE_TO_LOGIN” “APP_RESPONSE_TIMEOUT”. See the *Connector Development Guide for Java* for more information on response codes.

Table 3. Population of response message

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

After processing the business object, the receiving application creates a response message with the connector property `MessageResponseResultProperty` set to SUCCESS, FAIL or one of the other values defined in Table 2. If the business object was processed and changes occurred, the receiving application populates the response message with a serialized business object containing the changes. If the business object could not be processed, the receiving application provides an explanation in the message body that the connector returns to InterChange Server Express. 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 result 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 Express 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 Express along with the response code. However, you can also use a message selector to identify, filter and otherwise control how the adapter identifies the response message for a given request. This message selector capability is a JMS feature. It applies to synchronous request processing only and is described below.

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

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

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

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

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

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

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

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

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

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

to identify the response message.

You can also specify multiple substitutions such as the following:

```
response_selector=PrimaryId LIKE '{1}' AND AddressId LIKE '{2}' :
PrimaryId, Address[4].AddressId
```

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

An error is reported at run-time when any of the following occurs:
- If you specify a non-integer value between the `'{}'` symbols
- If you specify an index for which no attribute is defined
- If the attribute specified does not exist in the business or meta-object
- If the syntax of the attribute path is incorrect

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

The connector would resolve this message selector as follows:

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

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

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

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

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

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

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.

Table 4 shows the response messages for these verbs.

Table 4. Population of response message

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

Processing locale-dependent data

The connector has been internationalized so that it can support double-byte character sets, and deliver message text in the specified language. When the connector transfers data from a location that uses one character code to a location that uses a different code set, it performs character conversion to preserves the meaning of the data.

The Java runtime environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encodings for characters in most known character code sets (both single-byte and multibyte). Most components in the WebSphere business integration system are written in Java. Therefore, when data is transferred between most integration components, there is no need for character conversion.

To log error and informational messages in the appropriate language and for the appropriate country or territory, configure the `Locale` standard configuration property for your environment. For more information on configuration properties, see Appendix A, “Standard configuration properties for connectors,” on page 49.

Chapter 2. Installing and configuring the connector

- "Compatibility"
- "Prerequisites"
- "Overview of installation tasks"
- "Installing the connector and related files" on page 14
- "Installed file structure" on page 14
- "Connector configuration" on page 16
- "Enabling guaranteed event delivery" on page 22
- "Meta-object attributes configuration" on page 26
- "Startup" on page 40

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

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 2.4.x version of the adapter for JMS is supported on the following adapter framework and integration brokers:

- Adapter framework: WebSphere Business Integration Adapter Framework, version 2.3.1.
- Integration broker: InterChange Server Express, version 4.3.1.

See the Release Notes for any exceptions.

Prerequisites

Prerequisite software

- The connector supports JMS 1.02.
- In addition, you must have the following components:
  - Java CDK (see *Installing WebSphere Business Integration Express and Express Plus for Item Synchronization*)
  - JMS and JNDI libraries
- The connector runs on the following platforms:
  - Microsoft Windows 2000
  - OS400 V5R2 (5722-SS1)
  - Red Hat Enterprise Linux WS/ES/AS for Intel 2.1, 2.4 Kernel
  - SuSE Linux Enterprise Server 7.3, 2.4 Kernel

Overview of installation tasks

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

- **Install the integration broker** This task, which includes installing the WebSphere business integration system and starting the integration broker, is described in *Installing IBM WebSphere Business Integration Express and Express Plus for Item Synchronization*.

- **Install the adapter and related files** This task includes installing the files for the adapter from the software package onto your system. See "Installing the connector and related files."

Installing the connector and related files

To install adapters for Business Integration Express for Item Sync:

1. Insert the product CD.
2. See *Installing WebSphere Business Integration Express and Express Plus for Item Synchronization*.
3. After installing adapters, see the *Quick Start Guide*, which contains configuration information for required adapters.

Installed file structure

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

Windows connector file structure

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

The utility installs the connector agent into the `ProductDir\connectors\JMS` directory, and adds a shortcut for the connector agent to the Start menu. Note that *ProductDir* represents the directory where the IBM WebSphere Business Integration Adapters product is installed. The environment variable contains the *ProductDir* directory path, which is IBM\WebSphereAdapters by default.

Table 5 describes the Windows file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through Installer.

Table 5. Installed Windows file structure for the connector

Subdirectory of *ProductDir*	Description
`connectors\JMS\CWJMS.jar`	Contains classes used by the JMS connector
`connectors\JMS\jms.jar`	Third-party library required by connector
`connectors\JMS\start_JMS.bat`	The startup script for the connector (NT/2000)
`connectors\messages\JMSConnector.txt`	Message file for the connector
`repository\JMS\CN_JMS.txt`	Repository definition for the connector
`connectors\JMS\Samples\JMSConnector.cfg`	Sample connector configuration file
`connectors\JMS\Samples\PortConnector.cfg`	Sample port connector configuration file
`connectors\JMS\Samples\Sample_JMS_Contact.xsd`	
`connectors\JMS\Samples\Sample_JMS_MO_Config.xsd`	Sample meta-object
`connectors\JMS\Samples\Sample_JMS_MO_DataHandler.xsd`	Sample data handler meta-object
`connectors\JMS\Samples\Sample_JMS_MO_DataHandler_DelimitedConfig.xsd`	Sample delimited data handler meta-object
`connectors\JMS\Samples\Sample_JMS_DynMO.xsd`	Sample dynamic meta-object
`connectors\JMS\Samples\JMSPropertyPairs.xsd`	Sample JMS properties

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

Linux connector file structure

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

The utility installs the connector agent into the `ProductDir/connectors/JMS` directory, and adds a shortcut for the connector agent to the Start menu. Note that *ProductDir* represents the directory where the IBM WebSphere Business Integration Adapters product is installed. The environment variable contains the *ProductDir* directory path, which is IBM\WebSphereAdapters by default.

Table 5 on page 14 describes the Linux file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through Installer.

Table 6. Installed Linux file structure for the connector

Subdirectory of *ProductDir*	**Description**
`connectors/JMS/CWJMS.jar`	Contains classes used by the JMS connector
`connectors/JMS/jms.jar`	Third-party library required by connector
`connectors/JMS/start_JMS.sh`	The startup script for the connector
`connectors/messages/JMSConnector.txt`	Message file for the connector
`repository/JMS/CN_JMS.txt`	Repository definition for the connector
`connectors/JMS/Samples/JMSConnector.cfg`	Sample connector configuration file
`connectors/JMS/Samples/PortConnector.cfg`	Sample port connector configuration file
`connectors/JMS/Samples/Sample_JMS_Contact.xsd`	
`connectors/JMS/Samples/Sample_JMS_MO_Config.xsd`	Sample meta-object
`connectors/JMS/Samples/Sample_JMS_MO_DataHandler.xsd`	Sample data handler meta-object
`connectors/JMS/Samples/Sample_JMS_MO_DataHandler_DelimitedConfig.xsd`	Sample delimited data handler meta-object
`connectors/JMS/Samples/Sample_JMS_DynMO.xsd`	Sample dynamic meta-object
`connectors/JMS/Samples/JMSPropertyPairs.xsd`	Sample JMS properties

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

OS/400 connector file structure

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

The utility installs the connector agent into the `ProductDir/connectors/JMS` directory, and adds a shortcut for the connector agent to the Start menu. Note that *ProductDir* represents the directory where the IBM WebSphere Business Integration Adapters product is installed. The environment variable contains the *ProductDir* directory path, which is IBM\WebSphereAdapters by default.

Table 5 on page 14 describes the OS/400 file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through Installer.

Table 7. Installed OS/400 file structure for the connector

Subdirectory of *ProductDir*	Description
`connectors/JMS/CWJMS.jar`	Contains classes used by the JMS connector
`connectors/JMS/jms.jar`	Third-party library required by connector
`connectors/JMS/start_JMS.sh`	The startup script for the connector
`connectors/messages/JMSConnector.txt`	Message file for the connector
`repository/JMS/CN_JMS.txt`	Repository definition for the connector
`connectors/JMS/Samples/JMSConnector.cfg`	Sample connector configuration file
`connectors/JMS/Samples/PortConnector.cfg`	Sample port connector configuration file
`connectors/JMS/Samples/Sample_JMS_Contact.xsd`	
`connectors/JMS/Samples/Sample_JMS_MO_Config.xsd`	Sample meta-object
`connectors/JMS/Samples/Sample_JMS_MO_DataHandler.xsd`	Sample data handler meta-object
`connectors/JMS/Samples/Sample_JMS_MO_DataHandler_DelimitedConfig.xsd`	Sample delimited data handler meta-object
`connectors/JMS/Samples/Sample_JMS_DynMO.xsd`	Sample dynamic meta-object
`connectors/JMS/Samples/JMSPropertyPairs.xsd`	Sample JMS properties

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

Connector configuration

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

You use Connector Configurator Express to configure connector properties:

- For a description of Connector ConfiguratorExpress and step-by-step procedures, see Appendix B, "Connector Configurator Express," on page 63.
- For information on required connector configuration, see the *Quick Start Guide*.
- For a description of standard connector properties, see "Standard connector properties" and then Appendix A, "Standard configuration properties for connectors," on page 49.
- For a description of connector-specific properties, see "Connector-specific properties" on page 17.

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 Express.

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 49 for documentation of these properties.

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

Connector-specific properties

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

Table 8 lists the connector-specific configuration properties for the connector. See the sections that follow for explanations of the properties.

Table 8. Connector-specific configuration properties

Name	Possible values	Default value	Required
ArchiveQueue	*Queue to which copies of successfully processed messages are sent*	`CWLD_ARCHIVE`	No
CCSID	*Character set for queue manager connection*	`null`	Yes
ConfigurationMetaObject	*Configuration meta-object*		Yes
DataHandlerClassName	*Name of data handler class*		No
DataHandlerConfigMO	*Name of data handler configuration meta-object*	`MO_DataHandler_ Default`	No
DataHandlerMimeType	*Mime type of file*		No
ErrorQueue	*Queue for unprocessed messages*	`CWLD_ERROR`	No
InDoubtEvents	`FailOnStartup` `Reprocess` `Ignore` `LogError`	`Reprocess`	No
InProgressQueue	*In-progress event queue*	`CWLD_InProgress`	No
InputQueue	*Name of poll queues*	`CWLD_Input`	Yes
LookupQueuesUsingJNDI	`true` or `false`	`false`	No
MessageFormatProperty	*Property name specifying message format*	`JMSType`	No
MessageResponseResultProperty	*Property in response message that indicates the result of the request operation*	`CWLD_Result`	Yes
PollQuantity	*Number of messages to retrieve from each queue specified in the* `InputQueue` *property*	`1`	No
QueueConnectionFactoryName	*Name of connection factory to look up using JNDI context*		Yes
ReplyToQueue	*Queue to which response messages are delivered when the connector issues requests*	`CWLD_REPLYTO`	No
UnsubscribedQueue	*Queue to which unsubscribed messages are sent*	`CWLD_UNSUBSCRIBED`	No
UseDefaults	true or false	false	

ArchiveQueue

Queue to which copies of successfully processed messages are sent.

The default value is `CWLD_ARCHIVE`.

CCSID

The character set for the queue manager connection.

Default = `null`.

ConfigurationMetaObject

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

There is no default value.

DataHandlerClassName

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

DataHandlerConfigMO

Name of data handler meta-object. Provides configuration information for the data handler.

The default value is `MO_DataHandler_Default`.

DataHandlerMimeType

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

ErrorQueue

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

The default value is `CWLD_ERROR`.

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

The default value is `Reprocess`.

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.

The default value is `CWLD_InProgress`.

InputQueue

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

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

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

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

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

The default value is CWLD_`Input.`

LookupQueuesUsingJNDI

If this property is `true`, the connector assumes that queue names represent stored objects in the JNDI context, and resolves the queue name by looking it up in the configured JNDI store. A queue object must exist in the store and be associated with the name provided. When this property is `false`, the connector passes the queue name to the JMS provider and allows it to resolve it in a vendor-specific manner.

The default value is `false`.

MessageFormatProperty

Property name specifying message format. This value determines the conversion properties used during subscription delivery and is then set for output messages. If value is `JMSType` or unspecified, the connector uses the JMS message type as the format.

The default value is `JMSType`.

MessageResponseResultProperty

Property in response message that indicates the result of the request operation. The default value is `CWLD_Result`.

PollQuantity

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

The default value is 1.

QueueConnectionFactoryName

Name of connection factory to look up using JNDI context.

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 and dynamic child meta-object Attributes" on page 35

The default value is `CWLD_REPLYTO`.

UnsubscribedQueue

Queue to which unsubscribed messages are sent.

The default value is `CWLD_UNSUBSCRIBED`.

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`.

JNDI provider properties

This section describes connector properties for establishing the context of a Java Naming and Directory Interface (JNDI) provider. This information is taken from J2EE (Java 2 Enterprise Edition). For more information on JNDI environment variables and configuration, see `www.javasoft.com`. For information on configuring JNDI with the MA88 Patch, see "Configuring JNDI with WebSphere MQ Java client libraries" on page 21.

Table 9. Java Naming and Directory Interface (JNDI) provider properties

Property Name	Description
`CTX_InitialContextFactory`	Fully qualified class name of the factory class that will create an initial context. (e.g "com.sun.jndi.fscontext")
`CTX_ProviderURL`	Configuration information for the service provider to use. The value of the property should contain a URL (e.g. "ldap://somehost:389")

`CTX_ObjectFactories`
`CTX_StateFactories`
`CTX_URLPackagePrefixes`
`CTX_DNS_URL`
`CTX_Authoritative`
`CTX_Batchsize`
`CTX_Referral`
`CTX_SecurityProtocol`
`CTX_SecutiryAuthentication`
`CTX_SecurityPrincipal`
`CTX_SecurityCredentials`
`CTX_Language`
	Properties specifying additional information about security and object lookup in the JNDI context. See J2EE documentation for more information.

Configuring JNDI with WebSphere MQ Java client libraries

This section describes how to configure JNDI with WebSphere MQ Java client libraries. If you are using WebSphere MQ V5.2 or earlier, you must download these libraries from www.ibm.com/support/us/ and install them. Note that the download may be referred to on the site as ″SupportPac MA88.″

If you are using WebSphere MQ V5.3, the client libraries are packaged with the product and thus you are not required to perform the following steps.

1. Download and install the MQ Java client libraries.
2. Modify the classpath and variables as follows **on Windows**:

   ```
   SET PATH=%PATH%;"%MQ_JAVA_INSTALL_PATH%"\lib
   SET CP="%MQ_JAVA_INSTALL_PATH%"\lib\fscontext.jar;
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\jms.jar
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\jndi.jar
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\jta.jar
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\ldap.jar
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\providerutil.jar
   SET CP=%CP%;%CROSSWORLDS%\rt.jar;
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\com.ibm.mq.jar;
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\com.ibm.mq.iiop.jar;
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\com.ibm.mqbind.jar;
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib;
   SET CP=%CP%;"%MQ_JAVA_INSTALL_PATH%"\lib\com.ibm.mqjms.jar
   SET CLASSPATH=%CP%;%CLASSPATH%
   ```

   Modify the classpath and variables as follows **on OS/400**:

   ```
   MQ_JAVA_INSTALL_PATH=/QIBM/ProdData/mqm/java EXPORT
    PATH=${PATH}:${MQ_JAVA_INSTALL_PATH}/lib EXPORT
    CP=${MQ_JAVA_INSTALL_PATH}/lib/fscontext.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/jms.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/jndi.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/jta.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/ldap.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/providerutil.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/rt.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/com.ibm.mq.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/com.ibm.mq.iiop.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/com.ibm.mqbind.jar EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib EXPORT
    CP=${CP}:${MQ_JAVA_INSTALL_PATH}/lib/com.ibm.mqjms.jar EXPORT
    CLASSPATH=${CP}:${CLASSPATH}
   ```

3. On Windows, add %CP% to the -classpath in

   ```
   %MQ_JAVA_INSTALL_PATH%\bin\JMSAdmin.bat and start_JMS.bat:
   ```

   On OS/400, add %CP% to the -classpath in

   ```
   %MQ_JAVA_INSTALL_PATH%/bin/JMSAdmin and start_JMS.sh:
   ```

4. Modify

   ```
   %MQ_JAVA_INSTALL_PATH%\bin\JMSAdmin.config
   ```

   as follows:

   ```
   INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory
   PROVIDER_URL=file:/C:/JNDI-Directory
   SECURITY_AUTHENTICATION=none
   ```

   where PROVIDER_URL points to an existing directory where the `.bindings` file will be created.

5. Create queues for the default queue manager.

6. Create the `.jndi` file. Here is a sample:

```
*
*  JNDI Configured Objects for Information Pipeline
*  Used by WebSphere MQ JMS*
*
*
DEFINE QCF(ISSMWS14QCF) +
TRAN(client) HOST(MZIBERW2K) PORT(1414) +
CHANNEL(CHANNEL1) CLIENTID(' ')
*
* Queues
*
DEFINE Q(INPUTQ) +
QUEUE(INPUTQ) +
TARGCLIENT(JMS)
*
DEFINE Q(REPLYQ) +
QUEUE(REPLYQ) +
TARGCLIENT(JMS)
```

where MZIBERW2K is the name of the host machine,
CHANNEL1 is the name of the channel, and
INPUTQ and REPLYQ are the Q names.

7. Change directories to %MQ_JAVA_INSTALL_PATH%\bin and run JMSAdmin:

   ```
   >JMSAdmin.bat<sample.jndi
   ```

   where sample.jndi is the file created in step 3.

The result should be a .binding file in the directory specified in PROVIDER_URL.

Enabling guaranteed event delivery

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

- If the connector uses a JMS event store (implemented as a JMS source queue), the connector framework can manage the JMS event store. For more information, see “Guaranteed event delivery for connectors with JMS event stores.”
- If the connector uses a non-JMS event store (for example, implemented as a JDBC table, Email mailbox, or flat files), the connector framework can use a JMS monitor queue to ensure that no duplicate events occur. For more information, see “Guaranteed event delivery for connectors with non-JMS event stores” on page 24.

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” on page 23
- “Effect on event polling” on page 24

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 10.

Table 10. 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 11 summarizes.

Table 11. 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 10 *and* Table 11. To set these connector configuration properties, use the Connector Configurator Express tool. Connector Configurator Express displays the connector properties in Table 10 on its Standard Properties tab. It displays the connector properties in Table 11 on its Data Handler tab.

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

For information on Connector Configurator Express, see Appendix B, "Connector Configurator Express," on page 63.

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 11 on page 23.
4. Send the resulting message to the JMS destination queue.
   With the WebSphere ICS Express integration broker, the message sent to the JMS destination queue is the business object.
5. 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.
6. Repeat step 1 through 5 in a loop. The `PollQuantity` connector property determines the number of repetitions in this loop.

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

Guaranteed event delivery for connectors with non-JMS event stores

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

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

Enabling the feature for connectors with non-JMS event stores: To enable the guaranteed-event-delivery feature for a JMS-enabled connector that has a non-JMS event store, you must set the connector configuration properties to values shown in Table 12.

Table 12. 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 12. To set these connector configuration properties, use the Connector ConfiguratorExpress tool. It displays these connector properties on its Standard Properties tab. For information on Connector Configurator Express, see Appendix B, "Connector Configurator Express," on page 63.

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.

Meta-object attributes configuration

The connector for JMS can recognize and read two kinds of meta-objects:

- a static connector meta-object
- a dynamic child meta-object

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

Static meta-object

The JMS configuration static meta-object contains a list of conversion properties defined for different business objects. To define the conversion properties for a business object, 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 information of the attribute, specify the actual conversion properties.

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

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.

Table 13 describes the static meta-object properties.

Table 13. JMS static meta-object properties

Property name	Description
`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 agent.
`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.
`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. 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. Do not set this property using default conversion properties; its value is used to match incoming messages to business objects.
`OutputQueue`	The `OutputQueue` is the output queue to which messages derived from the given business object are delivered. An OutputQueue defined in a dynamic child meta-object overrides the value defined in the static meta-object.
`ResponseTimeout`	Indicates the length of time in milliseconds to wait before timing out when waiting for a response. The connector returns SUCCESS immediately without waiting for a response if this is left undefined or with a value less than zero. A `ResponseTimeout` defined in a dynamic child meta-object overrides the value defined in the static meta-object.

Table 13. JMS static meta-object properties (continued)

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.

Table 14. JMS static meta-object structure for Customer_Create

Attribute name	Application-specific text
`Customer_Create`	`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 (see "InputQueue" on page 18) 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 = Customer_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo =
InputQueue=//queue.manager/CompReceipts;DataHandlerClassName=com.crossworlds.
DataHandlers.MQ.disposition_notification;DataHandlerMimeType=message/
disposition_notification
IsRequiredServerBound = false
[End]
```

Overloading input formats

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

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

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

A sample meta-object

The static meta-object shown below configures the connector to convert Customer business objects using verbs create, update, delete, and retrieve. Note that attribute `Default` is defined in the meta-object. The connector uses the conversion properties of this attribute:

```
OutputQueue=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 (see "Overloading input formats"). 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).

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

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

Table 14 on page 28 and Table 15 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.

Table 15. JMS dynamic child meta-object structure for Customer_Create

Attribute name	Value
`DataHandlerMimeType`[1]	text/delimited
`OutputFormat`	`CUST_OUT`
`OutputQueue`	`QueueA`
`ResponseTimeout`	`10000`
`TimeoutFatal`	`False`

1. 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.

Table 16 shows how a dynamic child meta-object might be structured for polling.

Table 16. JMS dynamic child meta-object structure for polling

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

As shown in Table 16, you can define additional attributes, `Input_Format` and `InputQueue`, in a dynamic child meta-object. The `Input_Format` is populated with the format of the message retrieved, while the `InputQueue` attribute contains the name of the queue from which a given message has been retrieved. If these properties are not defined in the child meta-object, they will not be populated.

Example scenario:

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

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 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 17. 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:

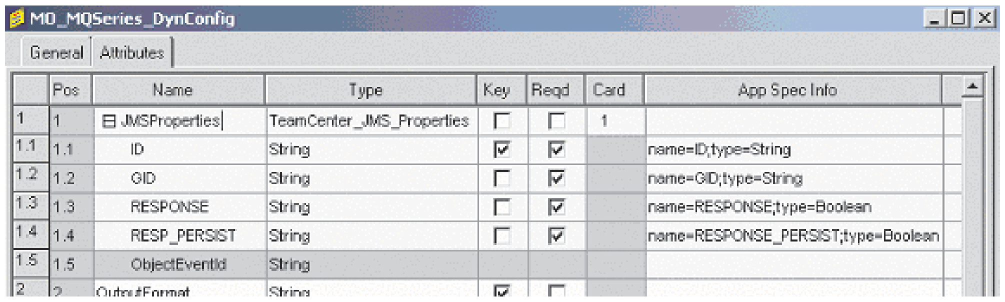
1. The name of the attribute has no semantic value.
2. The type of the attribute should always be `String` regardless of the JMS property type.
3. The application-specific information of the attribute must contain two name-value pairs defining the name and format of the JMS message property to which the attribute maps.

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

Table 18. 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`).



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

Figure 3. JMS properties attribute in a dynamic meta-object

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

1. Populates the `CorrelationId` attribute of the meta-object with the value specified in the `JMSCorrelationID` header field of the message.
2. Populates the `ReplyToQueue` attribute of the meta-object with the queue specified in the `JMSReplyTo` header field of the message. Since this header field is represented by a Java object in the message, the attribute is populated with the name of the queue (often a URI).
3. Populates the `DeliveryMode` attribute of the meta-object with the value specified in the `JMSDeliveryMode` header field of the message.
4. Populates the `Priority` attribute of the meta-object with the `JMSPriority` header field of the message.
5. Populates the `Destination` attribute of the meta-object with the name of the `JMSDestination` header field of the message. Since the `Destination` is represented by an object, the attribute is populated with the name of the `Destination` object.
6. Populates the `Expiration` attribute of the meta-object with the value of the `JMSExpiration` header field of the message.
7. Populates the `MessageID` attribute of the meta-object with the value of the `JMSMessageID` header field of the message.
8. Populates the `Redelivered` attribute of the meta-object with the value of the `JMSRedelivered` header field of the message.
9. Populates the `TimeStamp` attribute of the meta-object with the value of the `JMSTimeStamp` header field of the message.
10. Populates the `Type` attribute of the meta-object with the value of the `JMSType` header field of the message.
11. Populates the `UserID` attribute of the meta-object with the value of the `JMSXUserID` property field of the message.
12. Populates the `AppID` attribute of the meta-object with the value of the `JMSXAppID` property field of the message.
13. Populates the `DeliveryCount` attribute of the meta-object with the value of the `JMSXDeliveryCount` property field of the message.
14. Populates the `GroupID` attribute of the meta-object with the value of the `JMSXGroupID` property field of the message.
15. Populates the `GroupSeq` attribute of the meta-object with the value of the `JMSXGroupSeq` property field of the message.
16. Examines the object defined for the `JMSProperties` attribute of the meta-object. The adapter populates each attribute of this object with the value of the corresponding property in the message. If a specific property is undefined in the message, the adapter sets the value of the attribute to CxBlank.

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

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

1. Leave the value unchanged. The `CorrelationID` of the response message will be the same as the `CorrelationID` of the request message. This is equivalent to the 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 `ReplyToQueue`value specified in the connector configuration properties. If you additionally specify a negative `ResponseTimeout` (meaning that the connector should not wait for a response), the`ReplyToQueue` is set in the response message, even though the connector does not actually wait for a response.
3. If attribute `DeliveryMode` is set to `2`, the message is sent persistently. If `DeliveryMode` is set to `1`, the message is not sent persistently. Any other value may fail the connector. If `DeliveryMode` is not specified in the MO, then the JMS provider establishes the persistence setting.
4. If attribute `Priority` is specified, the connector sets the value in the outgoing request. The `Priority` attribute can take values 0 through 9; any other value may cause the connector to terminate.
5. If attribute `JMSProperties` is specified in the dynamic meta-object, the corresponding JMS properties specified in the child dynamic meta-object are set in the outbound message sent by the connector.

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

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

1. If attribute `CorrelationID` is present in the dynamic meta-object, the adapter updates this attribute with the `JMSCorrelationID` specified in the response message.
2. If attribute `ReplyToQueue` is defined in the dynamic meta-object, the adapter updates this attribute with the name of the `JMSReplyTo` specified in the response message.
3. If attribute `DeliveryMode` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSDeliveryMode` header field of the message.
4. If attribute `Priority` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSPriority` header field of the message.
5. If attribute `Destination` is defined in the dynamic meta-object, the adapter updates this attribute with the name of the `JMSDestination` specified in the response message.
6. If attribute `Expiration` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSExpiration` header field of the message.
7. If attribute `MessageID` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSMessageID` header field of the message.
8. If attribute `Redelivered` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSRedelivered` header field of the message.
9. If attribute `TimeStamp` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSTimeStamp` header field of the message.
10. If attribute `Type` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSType` header field of the message.
11. If attribute `UserID` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSXUserID` header field of the message.
12. If attribute `AppID` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSXAppID` property field of the message.
13. If attribute `DeliveryCount` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSXDeliveryCount` header field of the message.
14. If attribute `GroupID` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSXGroupID` header field of the message.
15. If attribute `GroupSeq` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSXGroupSeq` header field of the message.
16. If attribute `JMSProperties` is defined in the dynamic meta-object, the adapter updates any properties defined in the child object with the values found in the response message. If a property defined in the child object does not exist in the message, the value is set to CxBlank.

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

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

Startup

For information on starting a connector, stopping a connector, and the connector's temporary startup log file, see the *User Guide for IBM WebSphere Business Integration Express and Express Plus for Item Synchronization*.

Chapter 3. Creating or modifying business objects

- "Connector business object structure"
- "Error handling" on page 44
- "Tracing" on page 45

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

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

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

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

Connector business object structure

After installing the connector, you must create business objects. 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 InterChange Server Express. For more on naming conventions see *Naming IBM WebSphere InterChange Server Express Components*.

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

Sample business object properties

This section describes sample business object properties for a connector with a Name-Value data handler.

```
[ReposCopy]
Version = 3.0.0
[End]
```

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

[Attribute]
Name = ContactId
Type = String
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = true
DefaultValue = 1001
IsRequiredServerBound = false
[End]
[Attribute]
Name = FirstName
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = Jim
IsRequiredServerBound = false
[End]
[Attribute]
Name = LastName
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = Smith
IsRequiredServerBound = false
[End]
[Attribute]
Name = OfficePhoneArea
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = 650
IsRequiredServerBound = false
[End]
[Attribute]
Name = OfficePhone
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = 555-1234
IsRequiredServerBound = false
[End]
[Attribute]
Name = OfficePhoneExt
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = x100
IsRequiredServerBound = false
[End]
[Attribute]
Name = FaxArea
Type = String
```

```
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = 650
IsRequiredServerBound = false
[End]
[Attribute]
Name = FaxPhone
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = 555-1235
IsRequiredServerBound = false
[End]
[Attribute]
Name = Department
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = Engineering
IsRequiredServerBound = false
[End]
[Attribute]
Name = Title
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = Software Engineer
IsRequiredServerBound = false
[End]
[Attribute]
Name = EmailAddr
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = jim.smith@ibm.com
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 0
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]
```

Error handling

All error messages generated by the connector are stored in a message file named `JMSConnector.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, or if a NO_SUBSCRIPTION_FOUND code is returned by the `gotApplEvent()` method, 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.

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* for more on configuring trace messages. For more information on tracing, including how to enable and set it, see the Connector Configurator Express appendix to this 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 Express, 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 4. Troubleshooting

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

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`.
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`.

Appendix A. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of the adapters in WebSphere Business Integration Express for Item Synchronization, running on WebSphere InterChange Server Express.

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

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

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 Express

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

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
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.
- **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 InterChange Server Express.

- **Agent restart**
  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 Express window, or see the Update Method column in the Property Summary table below.

Summary of standard properties

Table 19 provides a quick reference to the standard connector configuration properties.

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

Table 19. Summary of standard configuration properties

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

Table 19. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
DuplicateEventElimination	True/False	False	Component restart	JMS transport only: Container Managed Events must be <NONE>
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	JMS transport only
jms.FactoryClassName	CxCommon.Messaging.jms .IBMMQSeriesFactory or CxCommon.Messaging .jms.SonicMQFactory or any Java class name	CxCommon.Messaging. jms.IBMMQSeriesFactory	Component restart	JMS transport only
jms.MessageBrokerName	If FactoryClassName is IBM, use crossworlds.queue. manager. If FactoryClassName is Sonic, use localhost:2506.	crossworlds.queue.manager	Component restart	JMS transport only
jms.NumConcurrentRequests	Positive integer	10	Component restart	JMS transport only
jms.Password	Any valid password		Component restart	JMS transport only
jms.UserName	Any valid name		Component restart	JMS transport only
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	Repository Directory is <REMOTE>
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	Repository Directory is <REMOTE>
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	Repository Directory is <REMOTE>
ListenerConcurrency	1- 100	1	Component restart	Delivery Transport must be MQ
Locale	en_US, ja_JP, ko_KR, zh_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	Repository Directory is <REMOTE>
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory is <REMOTE>
MessageFileName	*path/filename*	InterchangeSystem.txt	Component restart	
MonitorQueue	Any valid queue name	*CONNECTORNAME*/MONITORQUEUE	Component restart	JMS transport only: DuplicateEvent Elimination must be True

Table 19. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
OADAutoRestartAgent	`True` or `False`	`False`	Dynamic	Repository Directory is <REMOTE>
OADMaxNumRetry	*A positive number*	`1000`	Dynamic	Repository Directory is <REMOTE>
OADRetryTimeInterval	*A positive number in minutes*	`10`	Dynamic	Repository Directory is <REMOTE>
PollEndTime	`HH:MM`	`HH:MM`	Component restart	
PollFrequency	*a positive integer in milliseconds* `no` (to disable polling) `key` (to poll only when the letter `p` is entered in the connector's Command Prompt window)	`10000`	Dynamic	
PollQuantity	1-500	1	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 metadata repository	<remote>	Agent restart	
RequestQueue	*Valid JMS queue name*	*`CONNECTORNAME`*`/REQUESTQUEUE`	Component restart	
ResponseQueue	*Valid JMS queue name*	*`CONNECTORNAME`*`/RESPONSEQUEUE`	Component restart	Delivery transport is JMS
RestartRetryCount	`0-99`	`3`	Dynamic	
RestartRetryInterval	A sensible positive value in minutes 1 - 2147483547:	1	Dynamic	
SourceQueue	Valid WebSphere MQ name	*`CONNECTORNAME`*`/SOURCEQUEUE`	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue		*`CONNECTORNAME/`* `SYNCHRONOUSREQUESTQUEUE`	Component restart	Delivery transport is JMS
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery transport is JMS
SynchronousResponseQueue		*`CONNECTORNAME/`* `SYNCHRONOUSRESPONSEQUEUE`	Component restart	Delivery transport is JMS

Table 19. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
WireFormat	CwBO	CwBO	Agent restart	

Standard configuration properties

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

AdminInQueue

The queue that is used by InterChange Server Express 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 InterChange Server Express.

The default value is `CONNECTORNAME/ADMINOUTQUEUE`.

AgentConnections

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.

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 Express.

The default value is `ascii`.

ConcurrentEventTriggeredFlows

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 Interchange Server Express, 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 Express. 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

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

If this property is set to `true` and the destination application-specific component is unavailable when an event reaches Interchange Server Express, 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

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 Interchange Server Express.

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. The default is IDL.

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 Express. 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 running on InterChange Server Express.

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.

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 for InterChange Server Express. 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 Express.

The default value is `en_US`. If the connector has not been globalized, the only valid value for this property is `en_US`.

LogAtInterchangeEnd

Specifies whether to log errors to InterChange Server Express's log destination. Logging to the server'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

The `Repository Directory` must be set to <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

The `Repository Directory` must be set to <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

The `Repository Directory` must be set to <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."

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 0-60 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 0-60 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 InterChange Server Express 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.

This value must be set to <REMOTE> because the connector obtains this information from the InterChange Server Express repository.

ResponseQueue

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

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 54.

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. The setting is `CwBO`.

Appendix B. Connector Configurator Express

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

If you are configuring any of the following adapters, you may also want to refer to the *Quick Start Guide*:

- JTextRWLConnector
- iSoftConnector
- JTextISoftConnector
- ERP-source connector
- Emailconnector
- PortConnector

A more recent version of the *Quick Start Guide* may be available at the following link: http://www.ibm.com/websphere/wbiitemsync/express/infocenter

You use Connector Configurator Express to:

- Create a connector-specific property template for configuring your connector
- Create a connector configuration file
- Set properties, specify business objects and associated maps, and establish tracing and logging values in a configuration file

The topics covered in this appendix are:

- "Overview of Connector Configurator Express" on page 63
- "Starting Connector Configurator Express" on page 64
- "Creating a connector-specific property template" on page 64
- "Creating a new configuration file" on page 67
- "Setting the configuration file properties" on page 69

Overview of Connector Configurator Express

Connector Configurator Express allows you to configure the connector component of your adapter for use with InterChange Server Express.

You use Connector Configurator Express to:

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

You use Connector Configurator Express 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).

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

Starting Connector Configurator Express

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

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

Running Configurator Express in stand-alone mode

You can run Connector Configurator Express independently to work with connector configuration files. To do so:

- From **Start>Programs**, click **IBM WebSphere Business Integration Express for Item Sync v4.3>Toolset Express > Development > Connector Configurator Express**.
- Select **File > New > Configuration File**.

If you are creating a configuration file, you may prefer to run Connector Configurator Express independently to generate the file, and then connect to System Manager to save it in an InterChange Server Express project (see "Completing a configuration file" on page 69.)

Running Configurator Express from System Manager

You can run Connector Configurator Express from System Manager.

To run Connector Configurator Express:

1. Open the System Manager.
2. In the System Manager window, expand the **Integration Component Libraries** icon and highlight **Connectors**.
3. From the System Manager menu bar, click **Tools>Connector Configurator Express**. The Connector Configurator Express window opens and displays a **New Connector** dialog box.

Creating a connector-specific property template

To create a configuration file for your connector, you first 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 65.
- To use an existing file, simply modify an existing template and save it under the new name.

Note: Connector-specific templates are provided for the iSoft, JText, and e-Mail connectors only. If you are configuring one of these connectors, see the *Quick Start Guide*, or skip this section and go to "Creating a new configuration file" on page 67.

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. You then 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 Express Express 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 Express 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 Express stores the information you have entered as an XML document, under `\data\app` in the`\bin` directory where you have installed Connector Configurator Express.

Creating a new configuration file

You create a connector configuration file from a connector-specific template or by modifying an existing configuration file.

Creating a configuration file from a connector-specific template

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

1. In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator Express opens and displays the **New Connector** dialog box, with the following fields:
   - **Name**

     Enter the name of the connector followed by the word `connector`. Names are case-sensitive. The name you enter must be unique and consistent with the file name for a connector that is installed on the system. For example, enter `iSoftconnector` if the connector file name is `iSoft`.

     Important: Connector Configurator Express does not check the spelling of the name that you enter. You must ensure that the name is correct.
   - **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**.
2. 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.
3. To save the file, click **File>Save>Save to the project**. To save to a project, System Manager must be running.
   If you save as a file, the **Save File Connector** dialog box appears. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator Express 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.

4. To complete the connector definition, enter values in the fields for each of the tabs of the Connector Configurator Express window, as described later in this appendix.

Using an existing file

To use an existing file to configure a connector, you must open the file in Connector Configurator Express, revise the configuration, and then save the file as a configuration file (`*.cfg` 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 InterChange Server Express repository file.
  Definitions already created for the connector may be available to you in a repository file. 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 appendix.

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

1. In Connector Configurator Express, 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`)
   - InterChange Server Express Repository (`*.in`, `*.out`)

     Choose this option if a repository file was used to configure the connector. A repository file may include multiple connector definitions, all of which will appear when you open the file.
3. In the directory display, navigate to the correct connector definition file, select it, and click **Open**.

Opening an existing file from System Manager

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

1. Start System Manager.
2. Start Connector Configurator Express.
3. Click **File > Open > From Project**.

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 Express opens and displays the configuration file with the file name at the top.
2. Click the **Properties** tab to see which properties are included in this configuration file.

Completing a configuration file

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

Connector Configurator Express requires values for properties described in the following sections:
- "Setting standard connector properties"
- "Setting connector-specific configuration properties" on page 70
- "Specifying supported business object definitions" on page 70
- "Associated maps" on page 72
- "Setting trace/log file values" on page 73
- "Configuring messaging" on page 73

Note: For connectors that use JMS messaging, an additional category may display, for special configuration of data handlers that convert the data to business objects. For further information, see "Data handlers" on page 73.

Setting the configuration file properties

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.
   - To set values for standard property values for your connector, see the Standard Properties appendix of this guide.
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 Express, click **File > Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator Express, 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 connector-specific configuration properties

For connector-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.
   - To set values for connector-specific property values for your connector, **see the connector-specific properties section of this guide**.
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 69.

The Update Method displayed for each property indicates whether a component or agent restart is necessary to activate changed values. For further information, see *User Guide for WebSphere Business Integration Express for Item Synchronization*

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

Connector-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. For further information, see *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Specifying supported business object definitions

Use the **Supported Business Objects** tab in Connector Configurator Express to specify the business objects that the connector will use. You must specify both generic business object definitions and application-specific business object definitions, 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, including those for data handler meta-objects, and map definitions should be saved into ICL projects. For further information on ICL projects, see *User Guide for WebSphere Business Integration Express for Item Synchronization*

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 chapter on business objects in this guide as well as the *Business Object Development Guide*.

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 Express 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 Express 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. 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 Express 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.

Associated maps

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 connector 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 defined for specific source and destination business objects, the maps will already be associated with their business objects when you open the display, and you will not need 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 application-specific and generic 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 Express 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. To display the maps, you must first designate the supported business objects, and then save the connector configuration to project. To see the maps, you must first designate the supported business objects and save the connector configuration to project.
- **Explicit**

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

  Explicit binding is required only when more than one map exists for a particular supported business object. When InterChange Server Express 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.

Configuring messaging

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

Setting trace/log file values

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

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.
   - 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. Adapters that make use of the guaranteed event delivery enable this tab.

See the descriptions under `ContainerManagedEvents` in the Standard Properties appendix for values to use for these properties.

Saving your configuration file

After you have created the configuration file and set its properties, you need to deploy it to the correct location for your connector. Save the configuration in an ICL project, and use System Manager to load the file into InterChange Server Express.

For details about using projects in System Manager, and for further information about deployment, see the *User Guide for IBM WebSphere Business Integration Express for Item Synchronization.*

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. for more information on the startup file, see the appropriate section of your adapter user guide as well as the *User Guide for IBM WebSphere Business Integration Express for Item Synchronization*.

Notices

Programming interface information

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

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

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

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

Trademarks and service marks

IBM
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 Express for Item Synchronization V4.3.1.
WebSphere Business Integration Express Plus for Item Synchronization V4.3.1