WebSphere Business Integration Server
Express and Express Plus Adapters

Adapter for JMS User Guide

Version 4.3

WebSphere Business Integration Server
Express and Express Plus Adapters

Adapter for JMS User Guide

Version 4.3

Note!
Before using this information and the product it supports, read the information in "Notices" on page 77.

14May2004

This edition of this document applies to IBM WebSphere Business Integration Server Express, version 4.3, IBM WebSphere Business Integration Server Express Plus, version 4.3, and to all subsequent releases and modifications until otherwise indicated in new editions.

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

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

About this document

The products IBM(R) WebSphere(R) Business Integration Server Express and IBM(R) WebSphere(R) Business Integration Server Express Plus are made up of the following components: InterChange Server Express, the associated Toolset Express, CollaborationFoundation, and a set of software integration adapters. The tools in Toolset Express help you to create, modify, and manage business processes. You can choose from among the prepackaged adapters for your business processes that span applications. The standard processes template—CollaborationFoundation—allows you to quickly create customized processes.

This document describes configuration, business object development, and troubleshooting for the IBM WebSphere Business Integration Server Express Adapter for JMS.

Except where noted, all the information in this guide applies to both IBM(R) WebSphere(R) Business Integration Server Express and IBM(R) WebSphere(R) Business Integration Server Express Plus. The term WebSphere Business Server Express 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 Server Express installations, and includes reference material on specific components.

You can download, install, and view the documentation at the following site: http://www.ibm.com/websphere/wbiserverexpress/infocenter

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

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *`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. For UNIX installations, substitute slashes (/) for backslashes. All WebSphere business integration system pathnames are relative to the directory where the WebSphere business integration system is installed on your system.
UNIX:/Windows:	Paragraphs beginning with either of these indicate notes listing operating system differences.
u	This symbol indicates the end of a **UNIX/Windows** paragraph; it can also indicate the end of a multi paragraph note.
`%text%` and *`$text`*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *`$text`*, indicating the value of the *`text`* UNIX environment variable.

Contents

New in this release

New in release 4.3

This is the first release of this guide.

Chapter 1. Adapter for JMS overview

The adapter for JMS is a runtime component of IBM WebSphere Business Integration Server Express. The primary feature of the adapter is a connector that allows 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 send and receive business data and events.

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.

Note: All WebSphere Business Integration Server Express and Express Plus adapters operate with the WebSphere InterChange Server Express integration broker, which is described in the *System Implementation Guide*.

Adapter for JMS environment

Before installing, configuring, and using the adapter, you must understand its environmental requirements:

Adapter standards

The adapter is written to the JMS 1.0.2 standard. Support for other versions of the standard has not been verified although there are currently no known issues that would preclude this.

The adapter supports both the point-to-point (PTP) messaging and publish-and-subscribe (Pub/Sub) messaging interfaces defined by the JMS standard; these styles are also commonly referred to as queue-based and topic-based messaging, respectively. A single instance of the adapter supports only one messaging style at a time (i.e. topics and queues cannot be mixed in the

configuration); however, both messaging styles can be supported by running multiple instances of the adapter in parallel with instances configured for either PTP or Pub/Sub.

Adapter platforms

The adapter is supported on the following platform:

- Windows 2000

Adapter dependencies

The adapter does not use or depend upon any database. All client libraries required by the JMS provider and the JNDI provider must be included in the adapter classpath. These libraries differ by provider.

Adapter for JMS terminology

- **JMS provider** a messaging system that implements JMS
- **Messages** requests and events containing business data that are consumed by enterprise applications.
- **PTP** point-to-point style or queue-based messaging
- **Pub/Sub** publish-and-subscribe style or topic-based messaging
- **JMS Destination** represents the source of, or target for, a message. In PTP messaging, a destination is a queue. In Pub/Sub, a destination is a topic. This term is used widely in the specification in both descriptions and actual property names when either a queue or a topic could apply in a given situation.
- **ASI** Application-specific information—metadata that appears as semicolon-delimited name=value pairs in business and meta-objects.

Connector for JMS architecture

Messages, in the context of this adapter, are requests and events containing business data that are consumed by enterprise applications. Message Oriented Middleware products (MOM) enable enterprise applications to send messages to and receive messages from one another in an asynchronous fashion. The Java Message Service (JMS) API was established to standardize the way that Java programs communicate with these messaging systems. In the past, a messaging client was often written to work with a single specific MOM system. JMS clients, such as the adapter, can generally take advantage of any messaging system that provides JMS support. The adapter for JMS allows you to integrate with the growing number of enterprise messaging systems that support the JMS standard.

Message processing

The adapter supports two primary operations:

1. The retrieval of messages from a JMS destination
2. The delivery of a message to a JMS destination

The adapter performs both operations by establishing a connection to a JMS provider and then by using the JMS API to:

- Poll and retrieve existing messages from a JMS destination
- Generate and deliver new messages requested by the broker

These two operations are described in detail in “Event message processing” on page 3 and “Request message processing” on page 8.

Event message processing

The connector periodically checks for new messages delivered to one or more JMS destinations. During each poll cycle, the connector:

1. Uses the JMS API to retrieve any waiting messages.
2. Calls a configured data handler to convert the message content to a business object.
3. Delivers or publishes the event business object to the configured integration broker for processing by any subscribing business processes.

These steps are illustrated in Figure 1and described in depth in:

- "Event detection"
- "Event status and recovery" on page 4
- "Event retrieval" on page 6

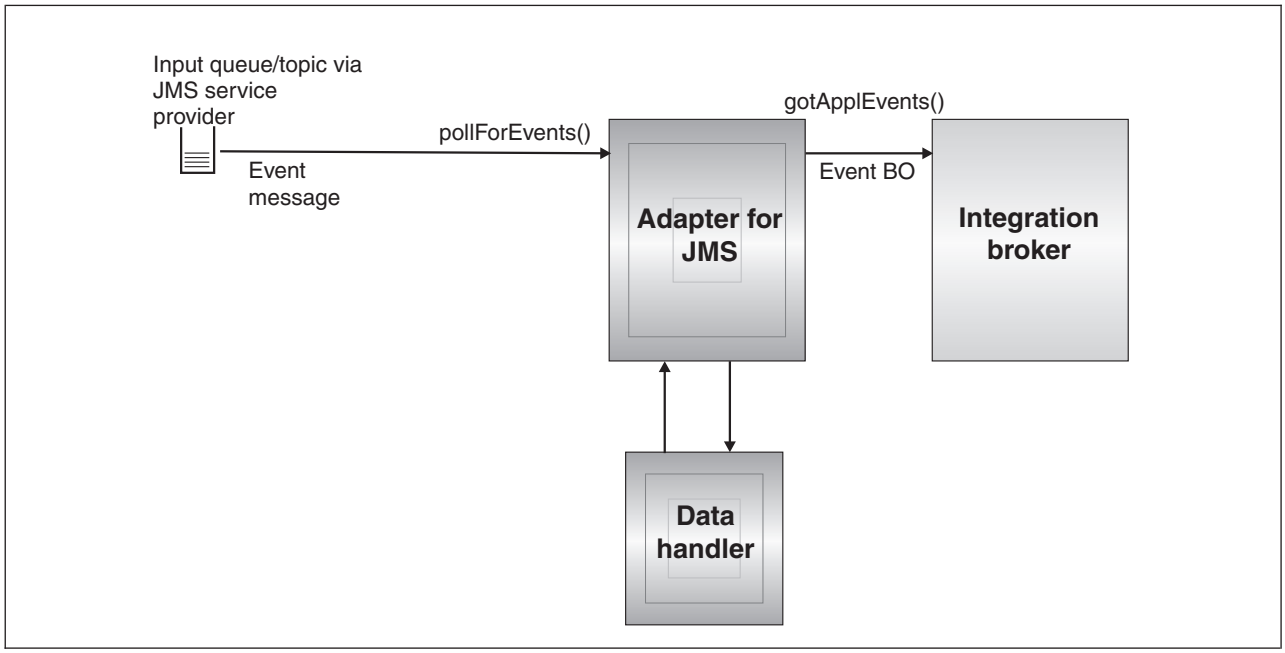

Figure 1. Event message flow

Event detection

During each event polling cycle, the connector performs a non-blocking read of messages at the destination specified by connector property InputDestination (for further information on connector properties, see "Configuring connector properties" on page 16). The connector retrieves messages and then publishes them to the broker.

The connector uses the `pollForEvents()` method to poll at regular intervals for messages. For each poll cycle, message retrieval is limited to the maximum number specified by connector property PollQuantity. If it retrieves all available messages before reaching the specified maximum, the connector does not wait for more messages but instead returns immediately from the poll cycle.

If multiple destinations are specified in connector property InputDestination, the connector polls each destination specified in a round-robin manner. It retrieves and publishes to the broker a maximum of *PollQuanity* number of messages from each

destination. If it empties all destinations before reaching the maximum specified by the PollQuantity, the connector returns immediately from the poll cycle.

For example, in a scenario where

- the connector is configured with a PollQuanity value of 2 and input queues A, B, and C
- queue A contains 2 messages, queue B contains 1 message, and queue C contains 5 messages

the adapter would retrieve messages in the following

in a single poll cycle:

1. Next message from queue A (leaving 1 message remaining)
2. Next message from queue B (now empty)
3. Next message from queue C (leaving 4 messages remaining)
4. Next message from queue A (now empty)
5. Connector checks queue B but it's still empty.
6. Next message from queue C (leaving 3 messages remaining)

The adapter then returns from the polling cycle because it has now polled a maximum (as set by the PollQuanity) of 2 messages from each queue.

Event status and recovery

Event message retrieval is part of a transaction. If the connector terminates unexpectedly before committing the transaction, the transaction is rolled back and the original message restored. Because the connector framework does not currently support distributed transactions, the connector may publish an event to the broker but either terminates unexpectedly or loses communication before an acknowledgement from the broker is received. In such cases, whether or not the event was received by the broker cannot be determined by any information available to the connector. To avoid loss of event messages, the connector does not commit the transaction until after the connector has received a response from the broker confirming receipt of the event. If there is a failure between the time the connector publishes an event and when it receives an acknowledgement, the transaction is rolled back automatically and the original message is restored. Because it's unknown whether or not the message was processed by the broker, such events are referred to as in-doubt events

Upon restart, the connector will start processing messages from the input destination and resubmit the in-doubt event. Although this strategy eliminates any risk that an event could be lost, it cannot prevent the same event from being published twice.

There are two means of reducing or eliminating the risk of duplicate event delivery: use of an in-progress destination (see "Recovery with an in-progress destination") or via guaranteed event delivery (see "Recovery with guaranteed event delivery" on page 5).

Recovery with an in-progress destination: To control how in-doubt events are handled, you can create a separate, temporary destination by specifying the connector property InProgressDestination.

Note: Recovery with an in-progress destination is not supported with Pub/Sub style messaging.

Before publishing an event to the broker, the connector moves the event message

from the input destination to the in-progress destination. Once it receives acknowledgement from the broker, the connector will remove the message from the in-progress destination. This isolates in-doubt messages that have not been processed. Upon startup, if the connector finds messages in the in-progress destination, the connector can safely assume that these were left from a previous instance of the connector that terminated unexpectedly. You can specify different actions for the connector to take on such messages (if duplicate event notification is unacceptable). You do this by specifying one of four options for the connector configuration property InDoubtEvents as follows:

- **Fail on startup** If it finds messages in the in-progress destination during initialization, the connector logs an error and immediately shuts down. You or a system administrator then examine the message and take appropriate action: either deleting these messages entirely or moving them to a different location.
- **Reprocess** If it finds any messages in the in-progress destination during initialization, the connector processes these messages first during subsequent polls. When all messages in the in-progress destination have been processed, the connector begins processing messages from the input destination.
- **Ignore** If it finds any messages in the in-progress destination 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 destination during initialization, it logs an error but does not shut down.

For further information, see "InDoubtEvents" on page 20.

Recovery with guaranteed event delivery: The guaranteed-event-delivery feature enables the connector framework to ensure that events are never lost and never sent twice. The connector framework supports guaranteed event delivery through two mechanisms: container managed events (CME) and duplicate event elimination (DEE).

Container managed events (CME): You can use CME when the connector is configured for PTP-style messaging; when configured for Pub/Sub-style messaging, the connector does not support CME. For further information on the ContainerManagedEvents connector property, see "ContainerManagedEvents" on page 46.

Duplicate event elimination (DEE): DEE is the recommended approach to implement guaranteed event delivery for the JMS adapter. DEE also is the only approach supported for Pub/Sub-style messaging.

With DEE, a connector includes a unique ID with each event it publishes to the broker. The framework checks that the connector does not submit the same event ID consecutively. If this does occur, the framework assumes that the connector is publishing the same event twice and discards the second submission. For PTP-style messaging, DEE reduces the substantial overhead involved in copying messages to and from an in-progress destination

This connector includes the message ID of all events when publishing business objects to the broker. If the connector fails to successfully post an event to the broker due to communication failure or unexpected termination, the original message is rolled back to the input queue as described previously. Upon restart, the connector begins resubmitting events from the queue including any in-doubt messages. If DEE is enabled, any in-doubt message that successfully reached the broker in the past will be discarded. This assures that each message is posted once and only once to the broker.

When using DEE, you should avoid manipulating the order of messages in destinations while the connector is off-line. DEE records only the last message ID retrieved by the adapter. DEE will fail in situations where, for example, new messages with higher priorities have pushed the last in-doubt message down in the queue before the adapter could restart.

For further information on the DuplicateEventElimination connector property, see "DuplicateEventElimination" on page 48.

Event retrieval

Event retrieval encompasses the typical processing of events by the connector. It begins when an incoming event is detected and ends when it has been converted into a format suitable for the target application and successfully delivered to the designated integration broker. The connector delivers all events asynchronously ("fire and forget") to the broker.

The following sections discuss event retrieval:

- "Metadata and meta-objects"
- "Business object mapping" on page 7
- "Understanding message header mapping" on page 7
- "Archiving" on page 8
- "Error recovery" on page 8

Metadata and meta-objects: In order for the connector to successfully convert messages to business objects and vice-versa, it needs additional information known as metadata. Metadata describes how data in an object or message or application is represented or should be processed. Meta-data includes such details as which business object to create if the connector retrieves a message from destination XYZ, or which data-handler should be used to serialize a request business object of type `Customer` with verb `Create`.

Attributes, properties, verbs, and application-specific information constitute the metadata for a business object definition. In addition, you can specify one or more meta-objects that contain metadata about destinations, data formats, data handlers, and more.

There are two types of meta-objects: static and dynamic. You create a static meta-object during implementation. It contains attributes that provide meta-data for each business object type the connector must support. The static meta-object is specified in connector-specific properties and is read by the connector during initialization. For an overview of meta-object properties and how they affect message transformation, see "Business object mapping" on page 7 and "Understanding message header mapping" on page 7.

The second type of meta-object is dynamic. This meta-object allows you to change the metadata used by the adapter to process a business object on a per-request basis during request processing. During event processing, the dynamic meta-object acts as a container to hold transport-specific information about the event (for example, message ID, priority, etc.) so that downstream business processes can use the information in their business logic. The dynamic meta-object is represented as a specially marked child object defined in the event (or request) top-level object.

You may opt to use one or both types of meta-objects in the same implementation. Values provided in a dynamic meta-object generally take precedence over any

values provided in the static meta-object. For information on configuring static and dynamic meta-objects, see “Configuring meta-objects” on page 23.

Business object mapping: Upon retrieval of a message, the connector attempts to identify which business object the message should be mapped to.

By default, the connector allows the data handler configured in the connector properties to determine the business object type. It will pass the message body to the data handler and publish the business object returned by the data handler to the broker. If the data handler cannot determine the appropriate business object, the connector will fail the event.

If a static meta-object is specified for connector configuration property ConfigurationMetaObject, the connector searches this object to find a rule that matches the message in terms of input format or input destination. If the rule specified in the meta-object specifies both an input format and input destination, the connector observes this rule only if the message matches both those properties. If one of these properties is missing, the connector uses the specified property only.

For example, a message with input format `Cust_In` from input destination `MyInputDest` would match the following static meta-object rules:

1. `InputFormat=Cust_In;InputDestination=MyInputDest`
2. `InputFormat=Cust_In`
3. `InputDestination=MyInputDest`

If it can match the event message to a single rule, the connector will dictate the business object by creating a new instance of this business object and passing it along with the message body to the data handler specified in the rule. If no data handler is specified in the rule, the connector will use the default data handler specified in the connector configuration properties.

If the adapter can match the event message to multiple rules or to none at all, the connector allows the data handler to determine the business object type by passing only the message body to the data-handler specified in the connector configuration properties.

Understanding message header mapping: To transform an event message into a business object, the connector compares metadata about the business object to metadata about the message, mapping one to the other. As described in “Metadata and meta-objects” on page 6, metadata about business objects resides in business object definitions (the application-specific information as well as child dynamic meta-objects), connector properties, and in static meta-objects. Message meta-data is contained in message headers.

To gain access to transport-specific message header information, and to get more information about, and more control over, the message transport, you can add attributes to a dynamic meta-object that is a child of a business object definition. Adding such attributes allows you to read from and optionally write to message headers, thereby modifying message metadata. Such modifications may include changing JMS properties, controlling the destination on a per-request basis (rather than using the default destination specified in the adapter properties), re-targeting a message CorrelationID, and more. When you specify such properties in a dynamic meta-object that is a child of a business object definition, the connector will check for their counterparts in message headers and then populate a dynamic meta-object based on the contents of the message header. You can define one or all

of the supported dynamic meta-object attributes; the connector will populate the meta-object accordingly. For further information, including a list of the message header properties that you can read or write to, see "Population of the dynamic child meta-object during polling" on page 30.

Archiving: If you specify the connector-specific property ArchiveDestination, the connector places a copy of all successfully processed messages in this destination. If ArchiveDestination is undefined, successfully processed messages are discarded. For further information see "Configuring connector-specific properties" on page 17.

Error recovery: If it encounters errors reading from the input destination(s), the connector will immediately return a constant value APPRESPONSETIMEOUT to the broker resulting in the termination and possible restart of the connector. Such unrecoverable errors are generally caused by either loss of connection to the JMS provider or internal errors reported by the JMS provider that the connector either does not recognize, or recognizes but deems unrecoverable (for example, transaction failure).

If it encounters errors converting the inbound message to an event business object (for example, the data handler reports invalid message format), the connector will fail the event and log an appropriate error message explaining the reason. If connector property ErrorDestination is defined and valid, the connector will place a copy of the failed message in this error destination; otherwise, the message is discarded.

If the broker reports an error after the connector publishes the event business object, the connector will fail the event and log the error message reported by the broker. If connector property ErrorDestination is defined and valid, the connector will put copy of the failed message in this destination; otherwise, the message is discarded.

If it is unable to determine a business object for a message, or if it publishes a message to a broker and the broker reports that the message is not supported, the connector will consider it unsubscribed. If connector property UnsubscribedDestination is defined and valid, the connector will put a copy of the unsubscribed message in this destination; otherwise, the message is discarded.

Request message processing

When a business object request is sent to the connector, it creates a new message in the target destination. The message header is populated with a combination of user-defined values specified in the request meta-object(s) and default parameters specified by connector properties. The body of the message is populated with the resulting content generated by passing the request business object through the configured data handler.

Figure 2 illustrates a message request communication. When the `doVerbFor()` method receives a business object from a broker, the connector passes the business object to the data handler. The data handler converts the business object into a suitable message, and the connector issues it as a message to a destination.

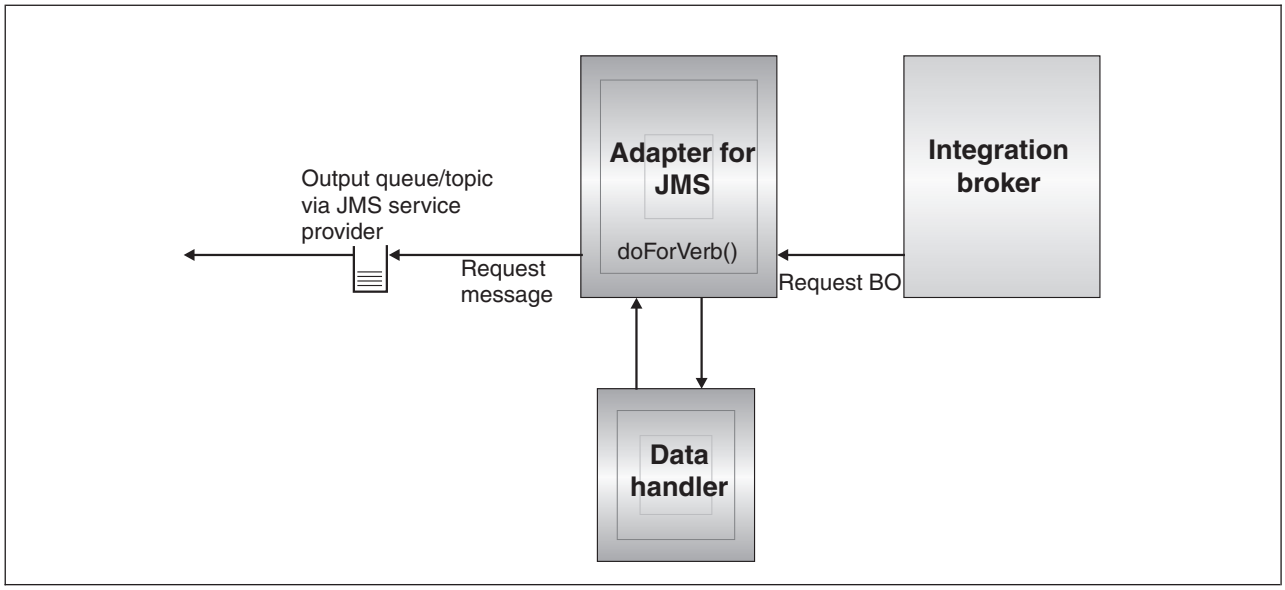

Figure 2. Request flow

There are two types of actions the connector can take during request processing. In the first, described below as asynchronous processing, the connector will put a message in the target destination and return successfully. This is commonly referred to as 'fire-and-forget'. In the second, described below as synchronous processing, the connector will again put a message in the target destination but will also wait for a response to be returned by the target application.

The mode of processing is determined by the numeric property ResponseTimeout, which is specified in either the dynamic or static meta-object for the business object request. If this property is not defined or is equal to -1, the connector delivers the request asynchronously. If this property is 0 or greater, the adapter processes the request synchronously, waiting at least that many milliseconds for a response message to be returned by the target application. The request processing illustrated in Figure 2 is described in detail in:

- "Verb support"
- "Asynchronous processing"
- "Synchronous processing" on page 11

Verb support

The connector places no semantic value on the verb specified in the request business object. It performs the same action, namely putting a message in a JMS destination, regardless of the verb specified.

Asynchronous processing

In asynchronous processing, the connector converts the request business object to a message, places that message in the target destination, and then returns immediately to the broker. The success or failure of the request is based entirely on the ability of the connector to put the message to the JMS destination. Note that the success of this delivery does not imply that the target application has or even will receive the message. Because of the asynchronous nature of messaging systems, a message may remain with the JMS provider indefinitely until the target application is able to process it or expires (if so configured).

The connector first serializes the request business object to text using the configured data handler. The connector uses the data handler that is specified, in order of preference, by:

1. the dynamic meta-object
2. the static meta-object
3. the connector configuration properties

The connector creates a new message containing the serialized business object data as the body of the message. It populates the message headers as described in the following table. In all cases where a property can be specified in either the dynamic or static meta-object, the value specified in the dynamic meta-object takes precedence over any value specified in the static meta-object. For descriptions and a list of which properties you can specify in meta-objects, see "Configuring meta-objects" on page 23.

Table 1. JMS message header population during asynchronous request processing

Meta-object property	Default action if property is undefined	Action taken if property is defined
`OutputFormat`	Connector does not specify a message format	Connector specifies this value for message format.
`CorrelationID`	Connector leaves this value blank in message header.	Connector specifies this value for correlation ID in request message header.
`ReplyToDestination`	Connector leaves this value blank in message header.	Connector specifies this value for reply destination in request message header.
`Priority`	Connector allows JMS provider to use default priority.	Connector sets numeric message priority using this value.
`JMSProperties`	None	Connector maps JMS properties specified to JMS properties in message header.

The following attributes in the meta-object determine how the message is delivered:

Table 2. Asynchronous delivery to destination

Meta-object property	Default action if property is undefined	Action taken if property is defined
`OutputDestination`	A value is required.	Target destination for message.
`DeliveryMode`	Connector allows JMS provider to dictate message persistence.	Connector writes message persistently/non-persistently as indicated by user.

Depending on the connector's ability to successfully deliver the request message to the output (target) destination, one of the following codes is returned to the broker:

Table 3. Asynchronous return codes

Connector action	Return code to broker
Successfully delivers message to target destination.	SUCCEED
Fails to deliver due to recoverable errors such as improper or incomplete meta-data, failure of data handler, or general processing problems.	FAIL
Fails to deliver due to unrecoverable errors reported by the JMS provider such as connection failure	APPRESPONSETIMEOUT

Synchronous processing

In synchronous processing, the connector delivers the request to the target destination and then waits on a second destination for a response message. Creation of the request message is identical to that described in asynchronous processing. However, the connector also checks the following additional attributes in the meta-object:

Table 4. Synchronous meta-object properties

Meta-object property	Default action if property is undefined	Action taken if property is defined
`ResponseTimeout`	A value is required.	Minimum amount of time (in milliseconds) that the adapter should wait for a response message to be returned.
`TimeoutFatal`	If it does not receive response by time specified by `ResponseTimeout`, connector returns APPRESPONSETIMEOUT to the broker, which typically results in connector termination.	If it does not receive response, connector fails request (return FAIL to broker) but does not terminate.

The delivery of the message to the target destination is the same as that described for asynchronous processing except for the following:

Table 5. Synchronous delivery to destination

Meta-object property	Default action if property is undefined	Action taken if property is defined
`ReplyToDestination`	Same as that for asynchronous.	Connector populates this field in request message with value of connector-specific property ReplyToDestination.

The connector waits for a response message from the target application specified by ReplyToDestination for at least as much time as specified by the meta-object attribute ResponseTimeout. If a response is not returned in that time, the connector will timeout and report an error.

Response criteria: The connector does not assume that the first message in the reply destination is the correct response message. Instead, it follows JMS request-response conventions and looks for the first message that has a correlation ID that matches the message ID of the request. In other words, the application that receives the request message must create a response message whose correlation ID equals the request message ID and it must put that message in the reply destination specified by the request message.

Not all applications follow the convention of using the correlation ID to map request and response messages. In such cases, the connector accepts custom criteria for identifying a response message.

Upon receipt of a business object for synchronous request processing, the connector checks for the presence of the name-value pair `response_selector=` in the application-specific information of the verb. If no such name-value pair exists, the connector identifies the response message using the message correlation ID as described above.

If a response selector name-value pair is defined, the connector considers the value to represent a JMS message selector string that can identify the response message. The following are a few examples of usage; for further information on JMS message selector syntax, see the JMS API specification. Note that the JMS message selector syntax is not parsed by the connector. Rather, the syntax is understood by the JMS provider. The connector makes available the selector to the JMS provider as a means of filtering messages (akin to a database query).

For example, verb application-specific information containing the name-value pair

```
response_selector=JMSType = 'xmlResponse'
```

informs the connector that the response message must match selector string `JMSType = 'xmlResponse'`. The connector provides this selector to the JMS provider, which then returns the first message delivered to the reply destination where the JMS type field of the message equals `xmlResponse`.

In all cases, the message selector string must be capable of uniquely identifying one and only one response. If multiple messages were to be delivered to the reply destination that met the criteria of the response selector, the adapter would retrieve only the first. Any other potential response message matching the criteria would be ignored.

To allow for unique message selectors at run-time, the connector supports dynamic substitutions of attribute values into the message selector itself. To do so, you must specify a placeholder in the form of an integer surrounded by curly braces (″{1}″) in the response selector. This must be followed by 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
```

informs the connector to replace the token {1} with the value of attribute `CorrelationID` in child-object `MyDynamicMO`. If attribute `CorrelationID` had a value of `123ABC`, the connector would generate and use message selector

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

You can also specify multiple substitutions as shown below:

```
response_selector=Name LIKE '{1}'AND Zip LIKE '{2}':PrimaryID,Address[4].AddressID
```

In this example, the connector would substitute `'{1}'` with the value of attribute `PrimaryID` from the top-level business object and `'{2}'` with the value of `AddressID` from the fifth position (base 0) 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.

To specify the literal value ″{″ in the message selector, use ″{{″ instead. For example, the following selector

```
response_selector=PrimaryID LIKE {{1}
```

would be recognized by the adapter as the literal value

```
PrimaryID LIKE {1}
```

The connector would not perform any substitution on the value ’{1}’ in this case.

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. For example, the following selector

```
Response_selector=PrimaryID = '{1}':Foo
```

when attribute `Foo` has a value of `{A:B};{C:D}` would be converted into a literal message selector like

```
PrimaryID = '{A:B};{C:D}'
```

Response processing: To determine what action to take upon receipt of the response message, the connector checks the JMS result property specified by connector property MessageResponseResultProperty. Depending on the value of this JMS property, the connector expects the response message to contain either a business object or an error message in the message body (see table below). In all cases, the connector returns the corresponding return code to the broker; for example, if the JMS result property equals `VALCHANGE` in the message, the connector takes the action described below for `VALCHANGE` and returns the numeric value corresponding to broker constant `VALCHANGE` to the broker.

Table 6. Response message processing

Value of JMS result property	Connector action
SUCCESS	Makes no changes to request business object and simply returns successfully to broker.
VALCHANGE MULTIPLE_HITS	Repopulates request business object with content of response message body. If response message body is empty, request business object is left unchanged. Repopulates the dynamic meta-object of the request business object with the JMS header fields of the response message.
FAIL FAIL_RETRIEVE_BY_CONTENT BO_DOES_NOT_EXIST UNABLE_TO_LOGIN VALDUPES	If response is populated, connector assumes it is an error message and returns it to the broker. If response message body is empty, connector returns generic error message to broker.
APPRESPONSETIMEOUT	Same as above except that return of APPRESPONSETIMEOUT to broker normally results in the termination of the adapter agent.
undefined or unrecognized value	Connector fails the request.

Error handling: If it encounters errors when reading or writing the request message to the target destination or checking for the response message (as applicable), the connector immediately returns APPRESPONSETIMEOUT to the broker. This results in the termination or possible restart of the adapter. Such unrecoverable errors are generally caused by either loss of connection to the JMS provider or internal errors reported by the JMS provider that the connector does not recognize or recognizes but deems unrecoverable (for example, transaction failure).

If it encounters errors converting a business object to a message or vice-versa (for example, the data handler reports an invalid message format), the connector fails the request and logs an appropriate error message explaining the reason.

For further information including event failure scenarios, see "Error handling" on page 39.

Chapter 2. Installing and configuring the adapter

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

Installation tasks

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

- **Install the integration broker** This task, which includes installing and starting WebSphere Business Integration Server Express, is described in the *Installation Guide for WebSphere Business Integration Server Express*.
- **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 adapter and related files."

Installing the adapter and related files

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

http://www.ibm.com/websphere/wbiserverexpress/infocenter

Installed file structure

The section below describes 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 adapter product is installed. The environment variable contains the *ProductDir* directory path, which is IBM\WebSphereServer by default.

Table 7 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 7. 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.

Connector configuration

After installing the adapter, you must configure the connector. To do so, you must perform the tasks described in the following sections:

- "Configuring connector properties"
- "Configuring message style" on page 22
- "Configuring JNDI" on page 23
- "Configuring meta-objects" on page 23
- "Configuring startup scripts" on page 33

Configuring connector properties

Connectors have two types of configuration properties that are described in the following sections:

- "Configuring standard connector properties" on page 17
- "Configuring connector-specific properties" on page 17

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 Configurator Express and step-by-step procedures, see Appendix B, "Connector Configurator Express," on page 55.

- For a description of standard connector properties, see "Configuring standard connector properties" and then Appendix A, "Standard configuration properties for connectors," on page 41.
- For a description of connector-specific properties, see "Configuring connector-specific properties."

Configuring standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 41 for documentation of these properties. Then, for a step-by-step procedure describing how to set these properties see "Overview of Connector Configurator Express" on page 55.

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.

Configuring 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 re-code 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
ArchiveDestination	*Destination to which copies of successfully processed messages are sent*		No
ConfigurationMetaObject	*Configuration meta-object*		See property description
ConnectionFactoryName	*JMS queue or topic connection factory defined in JNDI store.*		Yes
CTX_InitialContextFactory	*Name of factory class to be used to establish an initial JNDI context.*		Yes
CTX_ProviderURL	*URL identifying the JNDI context where the connection factory is located.*		Yes
DataHandlerClassName	*Name of data handler class to instantiate.*		See property description
DataHandlerConfigMO	*Name of data handler meta-object containing configuration information for DataHandlerMimeType*	`MO_DataHandler_Default`	See property description
DataHandlerMimeType	*Mime type to use when selecting default data handler*	`text/delimited`	See property description
DefaultVerb	*Specifies the verb to be set within an incoming business object*	`Create`	No
ErrorDestination	*Destination for unprocessed messages*		No

InDoubtEvents	`FailOnStartup`
	`Reprocess`
	`Ignore`
	`LogError`
		`Reprocess`	No

InProgressDestination	*Temporary storage destination*		No
InputDestination	*Name of poll destination(s)*		No
LookupDestinationsUsingJNDI	`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*	`WBI_Result`	Yes, for synchronous processing.
PollQuantity	*Number of messages to retrieve from each destination specified in the `InputDestination` property*	`1`	No
ReplyToDestination	*Destination to which response messages are delivered when the connector issues requests*		Yes, for synchronous processing.
UnsubscribedDestination	*Destination to which copies of inbound messages are put if the message is unrecognized or if the business object to which it maps is not supported.*		No
UnsubscribeOnTerminate	*Specify removed topics from InputDestination.*		No

UseDefaults	`true`
	or
	`false`
		`false`	No

UseDurableSubscriptions	`true`
	or
	`false`
		`false`	No

ArchiveDestination
Destination to which copies of successfully processed messages are sent.

The default value is `CWLD_ARCHIVE`.

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

There is no default value.

ConnectionFactoryName
Name of JMS queue or topic connection factory object defined in JNDI store that the connector should retrieve and use for establishing a connection to the JMS provider. When looking up this name, the connector uses the initial JNDI context established by the CTX_InitialContextFactory and CTX_ProviderURL properties.

Default = none.

CTX_InitialContextFactory
The name of the factory class that is used to establish an initial JNDI context.

Default = none.

CTX_ProviderURL
Fully-qualified URL identifying JNDI context where the connection factor is located. This value is passed to the context factor.

Default = none.

DataHandlerClassName
Data handler class to use when converting messages to and from business objects. Specify either both DataHandlerConfigMO and DataHandlerMimeType or DataHandlerClassName only. Do not specify all three properties.

Note: A DataHandlerClassName value in a static or dynamic meta-object takes precedence over a one specified in this connector configuration property. If you do not provide a DataHandlerClassName value in a meta-object, then the connector obtains the value from this connector-configuration property.

Default = none.

DataHandlerConfigMO
Name of meta-object that contains configuration information for the mimetype specified in the DataHandlerMimeType property. Provides configuration information for the data handler. Specify either DataHandlerConfigMO and DataHandlerMimeType or DataHandlerClassName only. Do not specify all three properties.

Note: A DataHandlerConfigMO value in a static or dynamic meta-object takes precedence over a one specified in this connector configuration property. If you do not provide a DataHandlerConfigMO value in a meta-object, then the connector obtains the value from this connector-configuration property.

The default value is `MO_DataHandler_Default`.

DataHandlerMimeType
Allows you to request a data handler based on a particular MIME type. Specify either DataHandlerConfigMO and DataHandlerMimeType or DataHandlerClassName only. Do not specify all three properties.

Note: A DataHandlerMimeType value in a static or dynamic meta-object takes precedence over a one specified in this connector configuration property. If you do not provide a DataHandlerMimeType value in a meta-object, then the connector obtains the value from this connector-configuration property.

Default = `text/delimited`.

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

Default= `Create`

ErrorDestination
Destination to which copies of inbound messages are sent when the connector encounters errors while processing.

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`.

Note: You must specify a value for this property if you configure the InProgressDestination property.

InProgressDestination
Temporary destination where messages are held during processing.

Default = none.

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

If the `InputDestination` property is not supplied, the connector will not poll.

Default = none.

LookupDestinationsUsingJNDI

If this property is `true`, the connector will look up all JMS destination names in the JNDI store. This requires that any specified destination is defined in the JNDI store.

By default, the connector skips this step and allows the JMS provider to resolve the name to the appropriate destination at run-time.

Default = `false`.

MessageFormatProperty

The field in a JMS message that contains the input or output format for the message. By default, the connector checks the JMSType field of inbound messages for the message format and writes the message format to the JMSType field of outbound messages.

Default =`JMSType`.

MessageResponseResultProperty

Required for synchronous request processing, this property specifies the field in a response JMS message that the connector should check to determine the result of the request. This property is not used for asynchronous processing.

The default value is `WBI_Result`.

PollQuantity

Maximum umber of messages to retrieve from each destination specified in the `InputDestination` property during a `pollForEvents` cycle.

The default value is `1`.

ReplyToDestination

Destination to which response messages are delivered when the connector issues requests. This is, by default, the destination that the connector uses to coordinate the exchange of request messages with the target application. Specify this property for synchronous processing only.

Default = none.

UnsubscribedDestination

Destination to which copies of inbound messages are put if the message is unrecognized or if the business object to which it maps is not supported. If this property is defined and valid, the connector will put a copy of unsubscribed messages in this destination; otherwise, the message is discarded.

Default = none.

UnsubscribeOnTerminate

Applicable only when UserDurableSubscriptions is set to `true`. The use of durable subscriptions creates a problem if you remove topics from the connector configuration. The JMS provider will continue to store messages for the durable subscriptions even though the connector will never again check those subscriptions.

Whenever you remove topics from the list specified in InputDestination, specify those removed topics (delimited by semicolons) for this property value. To destroy the existing durable subscriptions, follow these steps:

1. Move those topic names to which you no longer want to subscribe from InputDestination to UnsubscribeOnTerminate.
2. Start and stop the connector (This destroys durable subscription).
3. Clear any topics specified for UnsubscribeOnTerminate.

This action does not change any of the InputDestination values.

Failing to perform the above steps will not impact the connector but will cause the JMS provider to store unnecessary messages.

Default = none.

UseDefaults

If UseDefaults is set to `true`, the connector checks whether a valid value or a default value is provided for each business object attribute that is marked isRequired.

Default = `false`.

UseDurableSubscriptions

Use this with Pub/Sub topic-style messaging only. If this property is set to `true`, the connector will act as a durable subscriber for applicable destinations. At the cost of higher overhead, the connector will instruct the JMS provider to store all messages for those topics to which it subscribes even while the connector is off-line. When brought back on-line, the connector will reprocess any published messages it missed.

Default = `false`.

Configuring message style

The adapter supports both the point-to-point (PTP) messaging and publish-and-subscribe (Pub/Sub) messaging interfaces defined by the JMS standard. The messaging style used by the adapter is determined by the type of administered object specified by the user in connector-specific property ConnectionFactoryName. See "ConnectionFactoryName" on page 19 before proceeding with these procedures:

- "Configuring PTP message style"
- "Configuring Pub/Sub style"

Configuring PTP message style

To configure an instance of the adapter in PTP message style:

1. Open Connector Configurator Express.
2. Click the Connector-Specific Properties tab.
3. Specify a name for ConnectionFactoryName that maps to an instance of a JMS *Queue*`ConnectionFactory` in your JNDI store. The adapter will work in PTP style and assume that all connector and meta-object properties identifying destinations (for example, the OutputDestination property) represent queues.

Configuring Pub/Sub style

To configure an instance of the adapter in Pub/Sub message style:

1. Open Connector Configurator Express.

2. Click the Connector-Specific Properties tab.
3. Specify a name for ConnectionFactoryName that maps to an instance of a JMS *Topic*`ConnectionFactory` in your JNDI store. The adapter will work in publish/subscribe style and assume that all connector and meta-object properties identifying destinations (for example, the OutputDestination property) represent topics.

Configuring JNDI

To establish a connection to the JMS provider, the connector needs access to a JMS connection factory. JMS defines the interface for the factory. But each individual JMS provider must supply its own implementation. Once the connector has a reference to this factory implementation, the connector can establish a connection to, and communicate with, the JMS provider without any knowledge of proprietary protocols or even the identify of the provider.

To be portable, the connector requires that the connection factory be located in a JNDI store. During implementation, the user or system administrator must create and configure a connection factory and place it in the JNDI store under a user-defined name. At run-time, the connector will establish a connection to the JNDI store, lookup the connection factory and use it to establish a connection to the JMS provider.

Some JMS providers provide their own JNDI implementations containing any connection factories or other administered JMS objects that you create; this approach makes it fairly straight-forward for you to configure the JMS adapter. For other JMS providers, users may need to install and configure an external JNDI provider, create the connection factory and make it available to the adapter. See your JNDI provider documentation for further information.

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."

Configuring JNDI with WebSphere MQ Java client libraries

For a tutorial that shows how to configure JNDI with WebSphere MQ Java client libraries, see "Configuring for queue-based messaging" on page 75and "Configuring for topic-based messaging" on page 76.

Configuring meta-objects

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. For an overview of metadata and static versus dynamic meta-objects, see "Metadata and meta-objects" on page 6.

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

- **Static meta-object**
  - Useful if all meta-data for different messages is fixed and can be specified at configuration time.

- Limits you to specifying values by business-object type. For example, all Customer-type objects must be sent to the same destination.

- **Dynamic meta-object**
  - Provides business processes access to information in message headers
  - Allows business processes to change processing of messages at run-time, regardless of business type. For example, a dynamic meta-object would allow you to specify a different destination for every Customer-type object sent to the adapter.
  - Requires changes to the structure of supported business objects—such changes may require changes to maps and business processes.
  - Requires changes to custom data handlers.

Meta-object properties

Table 9 provides a complete list of properties supported in meta-objects. Refer to these properties when implementing meta-objects.

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

Table 9. JMS meta-object properties

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
DataHandlerConfigMO	Yes	Yes	Meta-object passed to data handler to provide configuration information. If specified in the static meta-object, this will override the value specified in the DataHandlerConfigMO connector property. Use this static meta-object property when different data handlers are required for processing different business object types. Use the dynamic child meta-object for request processing when the data format may be dependent on the actual business data. The specified business object must be supported by the connector agent. See the description in "Configuring connector-specific properties" on page 17.
DataHandlerMimeType	Yes	Yes	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. Use the dynamic child meta-object for request processing when the data format might be dependent on the actual business data. The business object specified in DataHandlerConfigMO should have an attribute that corresponds to the value of this property. See the description in "Configuring connector-specific properties" on page 17.
DataHandlerClassName	Yes	Yes	See the description in "Configuring connector-specific properties" on page 17.

Table 9. JMS meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
`InputFormat`	Yes	Yes	Format or type of inbound (event) messages. This value assists in identifying the content of the message and would be specified by the application that generated the message. The field that the connector considers as defining the format in the message can be user-defined via the connector-specific property MessageFormatProperty.
`OutputFormat`	Yes	Yes	Format to be populated in outbound messages. If the `OutputFormat` is not specified, the input format is used, if available.
`InputDestination`	Yes	Yes	This property is used to match incoming messages to business objects only. By contrast, the InputDestination connector-specific property defines which destinations the adapter polls and is the only property that the adapter uses to determine which destinations to poll. In the MO, the InputDestination property and the InputFormat property can serve as criteria for the adapter to map a given message to a specific business object. To implement this feature, you would use connector-specific properties to configure multiple input destinations and optionally map different data handlers to each one based on the input formats of incoming messages. Do not set this property using default conversion properties; its value is used
`OutputDestination`	Yes	Yes	Destination to which the outbound message is written.
`ResponseTimeout`	Yes	Yes	Indicates the length of time in milliseconds to wait before timing out when waiting for a response in synchronous request processing. The connector returns SUCCESS immediately without waiting for a response if this is left undefined or with a value less than zero.
`TimeoutFatal`	Yes	Yes	Used in synchronous request processing to trigger the connector to return an error message if a response is not received. If this property is `True`, the connector returns APPRESPONSETIMEOUT to the broker when a response is not received within the time specified by `ResponseTimeout`. If this property is undefined or set to `False`, then on a response timeout the connector fails the request but does not terminate. Default = `False`.
Below are fields mapping specifically to the JMS message header. For specific explanations, interpretation of values, and more, see the JMS API specification. JMS providers may interpret some fields differently so also check your JMS provider documentation for any deviations.			
ReplyToDestination		Yes	Destination to which a response message for a request is to be sent.
Type		Yes	Type of message. Generally user-definable, depending on JMS provider.
MessageID		Yes	Unique ID for message (JMS provider specific).
CorrelationID	Yes	Yes	Used in response messages to indicate the ID of the request message that initiated this response.

Table 9. JMS meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
Delivery Mode	Yes	Yes	Specifies whether the message is persisted or not in the MOM system. Acceptable values: 1=non-persistent 2=persistent Other values, depending on the JMS provider, may be available.
Priority		Yes	Numeric priority of message. Acceptable values: 0 through 9 inclusive (low to high priority).
Destination		Yes	Current or last (if removed) location of message in MOM system.
Expiration		Yes	Time-to-live of message. If specified as zero, expiration is set to zero. Zero indicates to the JMS provider that the message does not expire.
Redelivered		Yes	Indicates that the JMS provider most likely attempted to deliver the message to the client earlier but receipt was not acknowledged.
Timestamp		Yes	Time message was handed off to JMS provider.
UserID		Yes	Identity of the user sending the message.
AppID		Yes	Identity of the application sending the message.
DeliveryCount		Yes	Number of delivery attempts.
GroupID		Yes	Identity of the message group.
GroupSeq		Yes	Sequence of this message in the message group specified in GroupID.
JMSProperties		Yes	See "JMS properties" on page 31.

Configuring a static meta-object

The JMS configuration static meta-object contains a list of conversion properties defined for different business objects. To view a sample static meta-object, launch Business Object Designer Express and open the following sample that is shipped with the adapter: `connectors\JMS\Samples\Sample_JMS_MO_Config.xsd`.

The connector supports at most one static meta-object at any given time. You implement a static meta-object by specifying its name for connector property ConfigurationMetaObject.

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

Table 10. Static meta-object structure

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

For example, consider the following meta-object:

Table 11. Sample static meta-object structure

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

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

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

Table 12. Sample static meta-object structure

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

See Table 9 on page 24 in "Meta-object properties" on page 24 describes the properties that you can specify as application-specific information in the static meta-object.

To implement a static meta-object:

1. Launch Business Object Designer Express. For further information, see the *Business Object Development Guide*.
2. Open the sample meta-object connectors\JMS\Samples\Sample_JMS_MO_Config.xsd. Figure 3 shows a sample static meta-object in Business Object Designer Express.

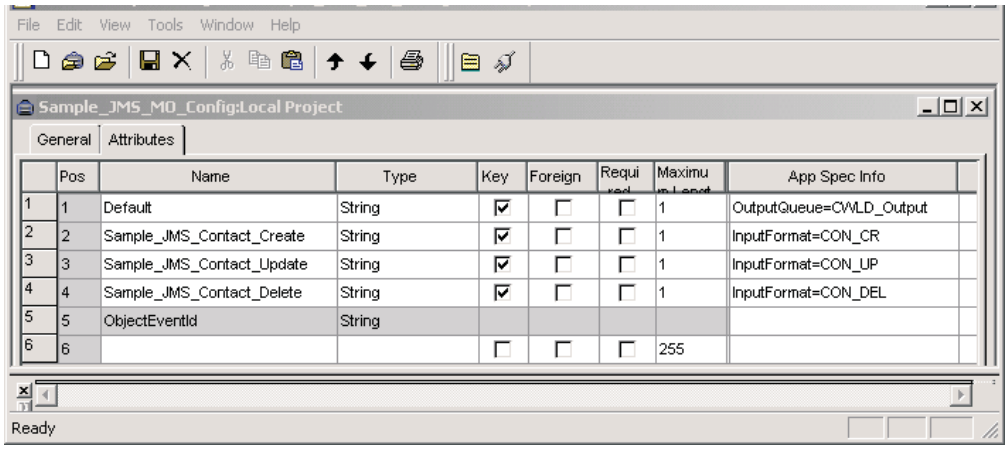

Figure 3. A sample static meta-object

3. Edit the attributes and ASI to reflect your requirements, referring to Table 9 on page 24 and then save the meta-object file.
4. Specify the name of this meta-object file as the value of the connector property ConfigurationMetaObject.

Mapping data handlers to input destinations

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

To map a data handler to an input destination:

1. Launch Connector Configurator Express. For further information, see Appendix B, "Connector Configurator Express," on page 55.
2. Use connector-specific properties (see "InputDestination" on page 20) to configure one or more input destination. Multiple destination names must be delimited by a semicolon.
3. For each input destination, specify the destination (queue manager if you are implementing PTP messaging style) and input destination 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 InputDestination named CompReceipts:

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

Configuring a 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 meta-data delivered at run-time for each business object instance.

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

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

To implement a dynamic meta-object, you add it as a child to your top-level object and include the name-value pair `cw_mo_conn=<MO attribute>` in your top-level object ASI where *<MO attribute>* is the name of the attribute in your top-level object representing the dynamic meta-object. For example:

```
Customer (ASI = cw_mo_conn=MetaData)
  |-- Id
  |-- FirstName
  |-- LastName
  |-- ContactInfo
  |-- MetaData
        |-- OutputFormat = CUST
        |-- OutputDestination = QueueA
```

Upon receipt of a request populated as shown above, the connector would convert the `Customer` object to a message with format `CUST` and then put the message in queue `QueueA`.

Business objects can use the same or different dynamic meta-object or none at all.

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

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.

See Table 9 on page 24 in “Meta-object properties” on page 24describes the properties that you can specify as application-specific information in the dynamic meta-object.

To implement a dynamic meta-object:

1. Launch Business Object Designer Express. For further information, see the *Business Object Development Guide*.

2. Open the sample meta-object `connectors\JMS\Samples\Sample_JMS_DynMO.xsd`. Figure 4 shows a sample dynamic meta-object in Business Object Designer Express.

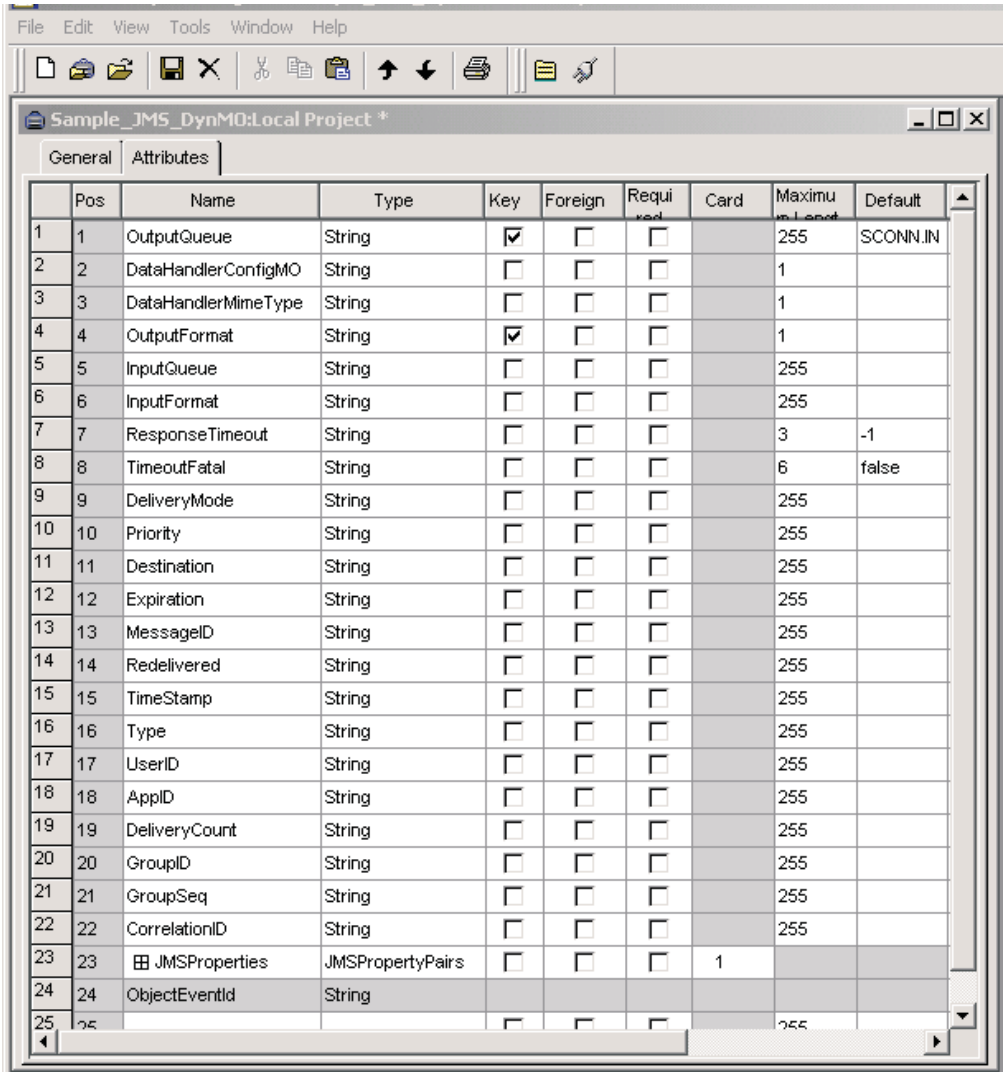

Figure 4. A sample dynamic meta-object

3. Edit the attributes and properties to reflect your requirements for this business object and save it.
4. Add the dynamic meta-object as a child to your top-level object and include the name-value pair `cw_mo_conn=<MO attribute>` in your top-level object ASI where `<MO attribute>` is the name of the attribute in your top-level object representing the dynamic 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 13 on page 31 shows how a dynamic child meta-object might be structured for polling.

Table 13. 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 13, you can define additional attributes, `Input_Format` and `Inputdestination`, in a dynamic child meta-object. The `Input_Format` is populated with the format of the message retrieved, while the `InputDestination` attribute contains the name of the destination 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 `InputDestination` and `InputFormat` attributes accordingly, then publishes the business object to available collaborations.

JMS headers and dynamic child meta-object attributes

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

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

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

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

Table 14. Application-specific information for JMS property attributes

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

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

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

To illustrate another example, Figure 5 shows attribute JMSProperties in the dynamic meta-object and definitions for four properties in the JMS message header: ID, GID, RESPONSE and RESPONSE_PERSIST. The application-specific information of the attributes defines the name and type of each. For example, attribute ID maps to JMS property ID of type String).

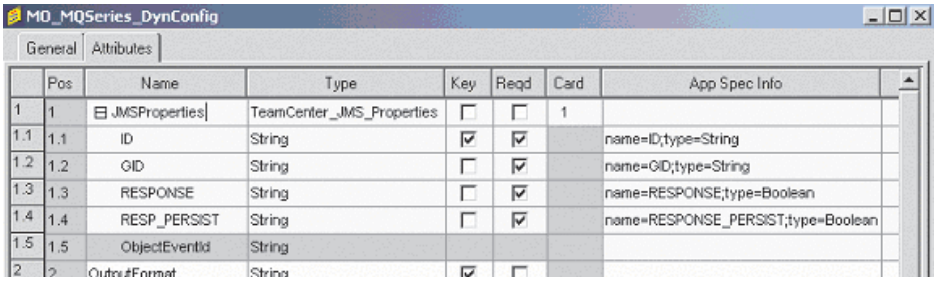

Figure 5. JMS properties attribute in a dynamic meta-object

Configuring startup scripts

The connector comes with the `JMS.bat startup script`. For information on starting a connector, stopping a connector, and the connector's temporary startup log file, see the startup chapter in the *Installation Guide for WebSphere Business Integration Server Express*.

Creating multiple connector instances

Creating multiple instances of a connector is in many ways the same as creating a custom connector.

Note: Each additional instance of any adapter supplied with WebSphere Business Integration Server Express will be treated as a separate adapter by the function that limits the total number of adapters that can be deployed.

You can set your system up to create and run multiple instances of a connector by following the steps below. You must:

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

Create a new directory

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

```
ProductDir\connectors\connectorInstance
```

where `connectorInstance` uniquely identifies the connector instance.

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

```
ProductDir\repository\connectorInstance
```

Create business object definitions

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

1. If you need to modify business object definitions that are associated with the initial connector, copy the appropriate files and use Business Object Designer

Express to import them. You can copy any of the files for the initial connector. Just rename them if you make changes to them.

2. Files for the initial connector should reside in the following directory:

   `ProductDir\repository\initialConnectorInstance`

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

Create a connector definition

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

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

Create a start-up script

To create a startup script:

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

   dirname

2. Put this startup script in the connector directory you created in "Create a new directory" on page 33.
3. Create a startup script shortcut.
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

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

Starting the connector

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

ProductDir`\connectors\`*connName*

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 15 shows.

Table 15. Startup scripts for a connector

Operating system	Startup script
Windows	`start_`*connName*`.bat`

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

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

  Select **Programs>IBM WebSphere Business Integration Express>Adapters>Connectors>***your_connector_name*.

  By default, the program name is "IBM WebSphere Business Integration Express". However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.

- From the command line
  - On Windows systems:

    `start_connName connName ICS [-cconfigFile ]`

  where *connName* is the name of the connector.
- From System Monitor

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

For more information on how to start a connector, including the command-line startup options, refer to the *System Administration Guide*.

Stopping the connector

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

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

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

Chapter 3. Creating or modifying business objects

- "Connector business object structure"

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

This chapter describes connector requirements for business objects. 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 WebSphere InterChange Server Express.

The connector retrieves messages from a destination 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.

Creating business objects

1. Identify and configure the application that InterChange Server Express will send business objects to when InterChange Server Express is configured with an instance of the JMS adapter.
2. Configure the connector with a data handler that can transform messages from JMS destinations into business objects suitable for processing by the target application. You do this by specifying the DataHandlerConfigMO and DataHandlerMimeType connector properties, or by specifying the DataHandlerClassName property. For further information, see "Configuring connector properties" on page 16. You can optionally specify special data handler processing rules in static and dynamic meta-objects. For further information, see "Meta-object properties" on page 24.
3. Use Business Object Designer Express to create the application-specific business objects. For further information see the *Business Object Development Guide*.
4. Add the business objects you create to the Supported Business Objects. Using Connector Configurator Express, click the **Supported Business Objects** tab for the JMS adapter, add the business objects you have created, and set the **Message Set ID** to a unique value for each supported business object. For further information on using Connector Configurator Express to add supported business objects, see "Specifying supported business object definitions" on page 63.

Chapter 4. Troubleshooting

The chapter describes how the connector handles errors and tracing, as well as problems that you may encounter when starting up or running the connector.

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 `UnsubscribedDestination` property.

Note: If the `UnsubscribedDestination` 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 Destination, if specified.

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 `ErrorDestination`. If the `ErrorDestination` is not defined, messages that cannot be processed due to errors are discarded.

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

Tracing

Trace messages are hard-coded in the adapter. 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 for more on configuring trace messages.

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 a broker, 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.

Fixing 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 Server Express, running on WebSphere InterChange Server Express.

Not every connector makes use of all these standard properties. When you select an integration broker 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 the integration broker.

- **Server restart**
  The change takes effect only after you stop and restart the application-specific component and the integration broker.
- **Agent restart**
  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 16 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

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

Table 16. 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 IDL
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS	ICS		
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	truetrue	Dynamic	Repository directory is <REMOTE>

Table 16. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE>
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	IDL or JMS	IDL	Component restart	
DuplicateEventElimination	true or false	false	Component restart	JMS transport only: Container Managed Events must be <NONE>
EnableOidForFlowMonitoring	true or false	false	Component restart	
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	JMS transport only
jms.FactoryClassName	CxCommon.Messaging.jms .IBMMQSeriesFactory or any Java class name	CxCommon.Messaging. jms.IBMMQSeriesFactory	Component restart	JMS transport only
jms.MessageBrokerName	crossworlds.queue. manager	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>
Locale	en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR **Note:** This is a subset of the supported locales.	en_US	Component restart	
LogAtInterchangeEnd	true or false	false	Component restart	
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory is <REMOTE>
MessageFileName	Path or filename	InterchangeSystem.txt	Component restart	

Table 16. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
MonitorQueue	Any valid queue name	*CONNECTORNAME*/MONITORQUEUE	Component restart	JMS transport only: DuplicateEvent Elimination must be true
OADAutoRestartAgent	`true` or `false`	`false`	Dynamic	Repository Directory 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 is 0-23, MM is 0-59)	`HH:MM`	Component restart	
PollFrequency	A positive integer in milliseconds `no` (to disable polling) `key` (to poll only when the letter `p` is entered in the connector's Command Prompt window)	`10000`	Dynamic	
PollQuantity	`1-500`	`1`	Agent restart	JMS transport only: Container Managed Events is specified
PollStartTime	`HH:MM`(HH is 0-23, MM is 0-59)	`HH:MM`	Component restart	
RepositoryDirectory	Location of metadata repository		Agent restart	Set to <REMOTE>
RequestQueue	Valid JMS queue name	*CONNECTORNAME*/REQUESTQUEUE	Component restart	Delivery Transport is JMS
ResponseQueue	Valid JMS queue name	*CONNECTORNAME*/RESPONSEQUEUE	Component restart	Delivery Transport is JMS:
RestartRetryCount	`0-99`	`3`	Dynamic	
RestartRetryInterval	A sensible positive value in minutes: 1 - 2147483547	`1`	Dynamic	
SourceQueue	Valid JMS queue name	*CONNECTORNAME*/SOURCEQUEUE	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue	Valid JMS queue name	*CONNECTORNAME*/ SYNCHRONOUSREQUESTQUEUE	Component restart	Delivery Transport is JMS

Table 16. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery Transport is JMS
SynchronousResponseQueue	Valid JMS queue name	*CONNECTORNAME*/SYNCHRONOUSRESPONSEQUEUE	Component restart	Delivery Transport is JMS
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 the integration broker to send administrative messages to the connector.

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

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

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

AgentTraceLevel

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

ApplicationName

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

BrokerType

Identifies the integration broker that you are using, which is ICS.

CharacterEncoding

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

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

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

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 the integration broker, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

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

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

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

ContainerManagedEvents

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

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

The default value is No value.

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

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

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

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 ICS, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forwards the request to it.

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

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

The default is true.

ControllerTraceLevel

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

DeliveryQueue

Applicable only if DeliveryTransport is JMS.

The queue that is used by the connector to send business objects to the WebSphere InterChange Server Express.

The default value is CONNECTORNAME/DELIVERYQUEUE.

DeliveryTransport

Specifies the transport mechanism for the delivery of events. Possible values are 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 IDL.

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 WebSphere 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.

EnableOidForFlowMonitoring

When you set this property to `true`, the adapter framework will mark the incoming **ObjectEventId** as a foreign key for the purpose of flow monitoring.

The default is `false`.

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).

The default value is 128m.

JvmMaxNativeStackSize

The maximum native stack size for the agent (in kilobytes).

The default value is 128k.

JvmMinHeapSize

The minimum heap size for the agent (in megabytes).

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 (see DeliveryTransport).

The default is crossworlds.queue.manager.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

Locale

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

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 using Connector Configurator Express in this guide.

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

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

LogAtInterchangeEnd

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

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

MaxEventCapacity

The maximum number of events in the controller buffer. This property is used by flow control.

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

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

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

The default value is false.

OADMaxNumRetry

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

The default value is 1000.

OADRetryTimeInterval

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

The default is 10.

PollEndTime

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

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

PollFrequency

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

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

The default is 10000.

Important: Some connectors have restrictions on the use of this property. To determine whether a specific connector does, see the installing and configuring chapter of its adapter guide.

PollQuantity

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

PollStartTime

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

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

RequestQueue

The queue that is used by WebSphere InterChange Server Express to send business objects to the connector.

The default value is CONNECTOR/REQUESTQUEUE.

RepositoryDirectory

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

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

ResponseQueue

Applicable only if `DeliveryTransport` is JMS.

Designates the JMS response queue, which delivers a response message from the connector framework to the integration broker. WebSphere InterChange Server Express sends the request and waits for a response message in the JMS response queue.

RestartRetryCount

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

The default is 3.

RestartRetryInterval

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

The default is 1.

SourceQueue

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

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 46.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is `CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE`

SynchronousResponseQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is `CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE`

SynchronousRequestTimeout

Applicable only if `DeliveryTransport` is JMS.

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

The default value is 0.

WireFormat

This is the 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.

The topics covered in this appendix are:

Overview of Connector Configurator Express

Connector Configurator Express allows you to configure the connector component of your adapter for use with WebSphere 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 maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

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

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

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

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 57 to set up a new one.

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

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 and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere Business Integration Server Express> Toolset Express>Development>Connector Configurator Express**.
- Select **File>New>Configuration File**.

You may choose to run Connector Configurator Express independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 61.)

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.

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.
2. Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 57.
- To use an existing file, simply modify an existing template and save it under the new name.

Creating a new template

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

To create a template:

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

     Enter a unique name that identifies the connector, or type of connector, for which this template will be used. You will see this name again when you open the dialog box for creating a new configuration file from a template.
   - **Old Template**, and **Select the Existing Template to Modify**

     The names of all currently available templates are displayed in the Template Name display.
   - To see the connector-specific property definitions in any template, select that template's name in the **Template Name** display. A list of the property definitions contained in that template will appear in the **Template Preview** display. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template.
3. Select a template from the **Template Name** display, enter that template name in the **Find Name** field (or highlight your selection in **Template Name**), and click **Next**.

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

Specifying general characteristics

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

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

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

Specifying values

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

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.

2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, make any changes. The changes will not be accepted unless you also open the **Property Value** dialog box for the property, described in the next step.
4. Right-click the box in the top left-hand corner of the value table and click **Add**. A **Property Value** dialog box appears. Depending on the property type, the dialog box allows you to enter either a value, or both a value and range. Enter the appropriate value or range, and click **OK**.
5. The **Value** panel refreshes to display any changes you made in **Max Length** and **Max Multiple Values**. It displays a table with three columns:

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

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

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

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

Setting dependencies

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

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

   <=(less than or equal to)
4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator 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 configuration file:

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

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

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

     The default broker is ICS. You cannot change this value.
   - **Select Connector-Specific Property Template**

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

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

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`).

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 used in a previous InterChange Server Express implementation of the connector may be available to you in a repository file that was used in the configuration of that connector. Such a file typically has the extension `.in` or `.out`.
- A previous configuration file for the connector.
  Such a file typically has the extension `*.cfg.`

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

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

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

1. In Connector Configurator 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.
   - All files (*.*)

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

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

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

Completing a configuration file

When you open a configuration file or a connector from a project, the Connector Configurator 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 application-specific configuration properties" on page 62
- "Specifying supported business object definitions" on page 63
- "Associated maps" on page 64
- "Setting trace/log file values" on page 65

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 66.

Setting the configuration file properties

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

Standard properties differ from connector-specific properties as follows:

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

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

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

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.

3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator 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 application-specific configuration properties

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

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. To add a child property, right-click on the parent row number and click **Add child**.
2. Enter a value for the property or child property.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for "Setting standard connector properties" on page 61.

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under "Setting and updating property values" on page 41.

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

Specifying supported business object definitions

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

For you to specify a supported business object, the business objects and their maps must exist in the system. Business object definitions, including those for data handler meta-objects, and map definitions should be saved into Integration Component Library (ICL) projects. For more information on ICL projects, see the *User Guide for WebSphere Business Integration Server Express*.

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 agent.

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

To indicate that the business object is supported by the connector agent, check the **Agent Support** box. The Connector Configurator 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.

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 InterChange Server Express. This list appears when you select the **Associated Maps** tab.

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

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

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

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

- **Business Object Name**

  These are the business objects supported by this connector, as designated in the **Supported Business Objects** tab. If you designate additional business objects under the Supported Business Objects tab, they will be reflected in this list after you save the changes by choosing **Save to Project** from the **File** menu of the Connector Configurator 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.
- **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.
3. In the **File** menu of the Connector Configurator Express window, click **Save to Project**.
4. Deploy the project to InterChange Server Express.
5. Reboot the server for the changes to take effect.

Resources

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

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

Setting trace/log file values

When you open a connector configuration file or a connector definition 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.

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

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

Data handlers

The data handlers section is available for configuration only if you have designated a value of JMS for DeliveryTransport and a value of JMS for ContainerManagedEvents. 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.

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

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

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

Completing the configuration

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

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

Using Connector Configurator Express in a globalized environment

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

Connector Configurator Express supports non-English characters in:

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

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

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

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

Appendix C. Tutorial

This appendix shows you how to use the adapter to send and receive business objects to and from an application communicating via JMS message queues. The scenarios in the tutorial are designed to show the basic points of the adapter's functionality.

See the Preface of this document for a guide to notational conventions.

Tutorial overview

The tutorial consists of two scenarios, one using a static meta-object and the other using a dynamic meta-object. Both scenarios involve ApplicationX, which can exchange corporate contact information as it is created, updated, or deleted. Business object `Sample_JMS_Contact`, which you create, matches the fields defined in messages from ApplicationX. ApplicationX sends and receives messages in a format that is compatible with the Delimited data handler that is available in IBM WebSphere Business Integration Server Express and Express Plus development kits.

The tutorial also makes use of the Port connector repository, which is included with the installation of WebSphere Adapters. The Port connector consists of a connector definition with no underlying code, and as such is well-suited for simulation scenarios.

Once started, the JMS adapter retrieves contact messages posted by ApplicationX to its input queue. Using the Delimited data handler, the adapter converts these messages to `Sample_JMS_Contact` business objects and delivers them to the integration broker. Using the Test connector (also included in the WebSphere Business Integration Server Express installation), you simulate the Port connector, retrieve the business object posted by the JMS adapter, and examine the attributes. After changing the data, you re-deliver the message to the integration broker where it is sent to the JMS adapter, converted to a message, and delivered to the output queue of the adapter (input queue of ApplicationX). In the tutorial, the adapter is configured for the WebSphere MQ Integrator Broker, but you need not actually install this broker to run the tutorial.

Before proceeding with this tutorial, make sure that:

- You installed and are experienced with the IBM WebSphere product.
- You installed a JMS service provider.
- You installed the adapter for JMS.
- You have installed and are experienced with WebSphere MQ 5.3.

Setting up your environment

This section describes how to prepare your environment to work with the tutorial. In what follows, *sample_folder* refers to the folder in which the samples reside.

1. **Create the queues** The tutorial requires that six queues be defined with your JMS service provider. Check your JMS provider documentation before defining these queues. Define the following queues (or make them available via JNDI lookup):
   - `CWLD_Input`
   - `CWLD_InProgress`
   - `CWLD_Error`
   - `CWLD_Archive`
   - `CWLD_Unsubscribed`
   - `CWLD_Output`
2. **Create and start a WebSphere MQ queue manager** with a running channel initiator and listener.
3. **Define the queues** required by the WebSphere MQ adapter and Port Connector for the WMQI broker configuration as follows:
   - `DEFINE QL('JMSConnector/ADMININQUEUE')`
   - `DEFINE QL('JMSConnector/ADMINOUTQUEUE')`
   - `DEFINE QL('JMSConnector/DELIVERYQUEUE')`
   - `DEFINE QL('JMSConnector/FAULTQUEUE')`
   - `DEFINE QL('JMSConnector/REQUESTQUEUE')`
   - `DEFINE QL('JMSConnector/RESPONSEQUEUE')`
   - `DEFINE QL('JMSConnector/SYNCHRONOUSREQUESTQUEUE')`
   - `DEFINE QL('JMSConnector/SYNCHRONOUSRESPONSEQUEUE')`
   - `DEFINE QL('PortConnector/ADMININQUEUE')`
   - `DEFINE QL('PortConnector/ADMINOUTQUEUE')`
   - `DEFINE QL('PortConnector/DELIVERYQUEUE')`
   - `DEFINE QL('PortConnector/FAULTQUEUE')`
   - `DEFINE QL('PortConnector/REQUESTQUEUE')`
   - `DEFINE QL('PortConnector/RESPONSEQUEUE')`
   - `DEFINE QL('PortConnector/SYNCHRONOUSREQUESTQUEUE')`
   - `DEFINE QL('PortConnector/SYNCHRONOUSRESPONSEQUEUE')`
4. **Configure the adapter** Using Connector Configurator Express, open *sample_folder*\`JMSConnector.cfg`. For further information on using Connector Configurator Express, see "Overview of Connector Configurator Express" on page 55; for more on connector-specific properties, see "Configuring connector-specific properties" on page 17.
   Set the following standard properties:
   - `Broker Type` Set this property to `WMQI`.
   - `Repository Directory` Set this property to the *sample_folder* directory.

   Set the following connector-specific properties:
   - `ConfigurationMetaObject` Set this property to `Sample_JMS_MO_Config`.
   - `DataHandlerConfigMO` Set this property to `Sample_JMS_MO_DataHandler`.
   - `DataHandlerMimeType` Set this property to `text/delimited`.
   - `ErrorQueue` Set this property to `CWLD_Error`.
   - `InProgressQueue` Set this property to `CWLD_InProgress`.
   - `InputQueue` Set this property to `CWLD_Input`.
   - `UnsubscribedQueue` Set this property to `CWLD_Unsubscribed`.

5. **Configure the Port Connector** Using Connector Configurator Express, set the following standard properties:
   - `Broker Type` Set this property to `WMQI`.
   - `Repository Directory` Set this property to the *sample_folder* directory.
   - `RequestQueue` Set this property to `JMSConnector/DELIVERYQUEUE` (the `DeliveryQueue` property value for the JMS adapter).
   - `DeliveryQueue` Set this property to `JMSConnector/REQUESTQUEUE` (the `RequestQueue` property value for the JMS adapter).
6. **Support business objects** In order to use business objects, adapters must first support them. Using Connector Configurator Express, click the **Supported Business Objects** tab for the JMS adapter, add the business objects shown in Table 17 and set the **Message Set ID** to a unique value for each supported business object.

Table 17. Supported sample business objects for JMS adapter

Business Object Name	Message Set ID
`Sample_JMS_MO_Config`	1
`Sample_JMS_MO_DataHandler`	2
`Sample_JMS_Contact`	3

Using Connector Configurator Express, open the Port connector definition `PortConnector.cfg` provided in the *sample_folder*, and add the supported business object and Message Set ID shown in Table 18.

Table 18. Supported sample business objects for Port connector

Business Object Name	Message Set ID
`Sample_JMS_Contact`	1

7. **Create or update connector start scripts**
   Windows:
   a. Open the properties of the shortcut for the adapter for JMS.
   b. As the last argument in the target, add `-c` followed by the *<full path and filename for the JMSConnector.cfg file>* For example:

      `-c`*`Product_Dir`*`\connectors\JMS\samples\JMSConnector.cfg`

Running the scenarios

Before you run the scenarios:

1. **Start the adapter for JMS** if it is not already running.
2. **Start the Visual Test connector** if it is not already running.

Running the static meta-object scenario

This part of the tutorial describes a scenario using a static meta-object. For further information on static meta-objects, see "Configuring a static meta-object" on page 26.

1. **Simulate the Port connector** Using the Visual Test connector, define a profile for PortConnector:
   a. Select **File->Create/Select Profile** from the Visual Test connector menu, then select **File-> New Profile** from the Connector Profile menu.

b. Select the Port Connector Configuration File PortConnector.cfg in the Samples directory, then configure the Connector Name and Broker Type and click **OK**.

c. Select the profile you created and click **OK**.

d. From the Visual Test connector menu, select **File->Connect** to begin simulating.

2. **Test request processing**

   a. Using the Test Connector, create a new instance of business object Sample_JMS_Contact by selecting the business object in the **BoType** drop-down box and then selecting **Create** for the BOInstance.

   b. Change the default values if desired, set the verb to **Create** and send the message by clicking **Send BO**.

3. **Check message delivery** Open queue CWLD_Output to see if a new contact message with format CON_CR has arrived from the JMS adapter.

4. **Test event processing** Send a message to the JMS adapter's input queue. Note: this step requires that you have a utility capable of sending messages to a queue. Otherwise, to implement an easier approach, you can set the JMS adapter's InputQueue property to CWLD_Output so that the adapter will poll its own messages. Once you have a message in the input queue, the JMS adapter will poll it and attempt to convert it into a Sample_JMS_Contact business object. The key to having the adapter poll the message is to ensure that the message format equals the value associated with the Sample_JMS_Contact business object in meta-object Sample_JMS_MO_Config. In this scenario, that format is CON_CR. If the adapter identifies the incoming message format as CON_CR, it will use the data handler to convert the message to business object Sample_JMS_Contact with the verb create. The newly created business object is subsequently delivered to the to the Test Connector.

5. **Confirm message delivery** If you've performed all the above steps successfully, you should have a working scenario that enables the JMS adapter to retrieve messages and convert them to Sample_JMS_Contact business objects, and to convert Sample_JMS_Contact business objects to contact messages.

Running the dynamic meta-object scenario

This scenario demonstrates how to use a dynamic meta-object to re-route a business object to various queues defined in your JMS service provider. For further information on dynamic meta-objects, see "Configuring a dynamic child meta-object" on page 29. The steps below take you through creating an attribute for a child meta-object for Sample_JMS_Contact. Specifically, you will be modifying the output queue values in this child meta object to redirect the Sample_JMS_Contact business object to various queues.

The child meta-object repository, Sample_JMS_DynMO.xsd, resides in the *sample_folder*.

1. **Identify the dynamic meta-object attribute** First you must add application-specific information to identify the attribute containing the dynamic meta-object: in Sample_JMS_Contact, add cw_mo_conn=DynMO to the application-specific information. This identifies the attribute.

2. **Add the attribute** Using Business Object Designer Express:

   a. Open Sample_JMS_DynMO.xsd and Sample_JMS_Contact.xsd from the *sample_folder*.

   b. In the Sample_JMS_Contact Object window, add an attribute named DynMO of type Sample_JMS_DynMO.

   c. Double-click the Sample_JMS_Contact Object.

d. Select the attributes folder and add an attribute named DynMO of type Sample_JMS_DynMO.

3. **Define a new target queue** Define a temporary queue REROUTE.IN with the JMS service provider. This is where the dynamic meta-object will re-route the Sample_JMS_Contact business object.
4. **Start the adapter for JMS** if it is not already running.
5. **Start the Visual Test connector** if it is not already running.
6. **Simulate the Port connector** Using the Visual Test connector, define a profile for PortConnector:
   a. Select **File->Create/Select Profile** from the Visual Test connector menu, then select **File-> New Profile** from the Connector Profile menu.
   b. Select the Port Connector Configuration File PortConnector.cfg in the Samples directory, then configure the Connector Name and Broker Type and click **OK**.
   c. Select the profile you created and click **OK**.
   d. From the Visual Test connector menu, select **File->Connect** to begin simulating.
7. **Create instances of parent business object and child meta object** Using the Visual Test Connector:
   a. Create a new instance of business object Sample_JMS_Contact, changing the default values if desired.
   b. Right-click on the DynMO attribute and create an instance of it, Sample_JMS_DynMO.
8. **Set the new target queue**
   a. Expand the DynMO attribute by clicking on the + sign beside it.
   b. In the attribute named outputQueue, enter the name of the target queue: REROUTE.IN
9. **Send the business object** Click **Send BO**.
10. **Confirm message delivery** Open queue REROUTE.IN to see if a new contact message has arrived from the JMS adapter. If a new message has arrived from the JMS adapter to the queue named REROUTE.IN, then the re-routing has worked.

Appendix D. Configuring for topic- and queue-based messaging

This appendix shows you how to configure the adapter for JMS with WebSphere MQ as a common JMS provider. For further information, see the *WebSphere MQ Using Java* guide.

Note: If you are using WebSphere MQ as your JMS provider, it is strongly suggested that you use the adapter for WebSphere MQ for integration. The steps below are provided for reference only to show how to configure the JMS adapter using a common JMS provider.

See the Preface of this document for a guide to notational conventions.

Configuring for queue-based messaging

1. Install WebSphere MQ and WebSphere MQ client libraries (including JMS support).
2. Ensure that all MQ client libraries, including `fscontext.jar` and `providerutil.jar`, are in your system's classpath. Alternatively, you can modify the `jmsAdmin.bat` file and add `-Djava.ext.dirs="<your MQ home directory>/Java/lib` to the java command-line script to ensure that all client library files are available to the tool. Note that any ClassDefNotFoundExceptions reported by the tool are the result of missing libraries—recheck your classpaths.
3. Open `<your MQ home directory>Java/bin/jmsAdmin.config` and set the following properties:
   - `INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory`
   - `PROVIDER_URL=file://c:/temp`
   - `SECURITY_AUTHENTICATION=none`
4. Create a file named `MyJNDI.txt` containing the following:
```
DEFINE QCF(MyQCF)
HOST(<your host name>)  +PORT(<your MQ listener port name e.g. 1414>)  +
CHANNEL(<your MQ server connection channel name, for example, CHANNEL1>)
+
QMGR(<your MQ queue manager name>) +
TRAN(client)
END
```
5. Bind objects to JNDI names by running `<your MQ home directory>/java/bin/jmsAdmin.bat < MyJNDI.txt`
6. Configure the following JMS connector-specific properties as shown:

```
CTX_InitialContextFactory = com.sun.jndi.fscontext.RefFSContextFactory
CTX_ProviderURL = file://c:/temp
ConnectionFactoryName = MyQCF
```

Configuring for topic-based messaging

1. Install WebSphere MQ and WebSphere MQ client libraries (including JMS support).
2. Ensure that all MQ client libraries, including `fscontext.jar` and `providerutil.jar`, are in your system's classpath. Alternatively, you can modify the `jmsAdmin.bat` file and add `-Djava.ext.dirs="<your MQ home directory>/Java/lib` to the java command-line script to ensure that all client library files are available to the tool. Note that any ClassDefNotFoundExceptions reported by the tool are the result of missing libraries—recheck your classpaths.
3. Download and install the appropriate WebSphere MQ MA0C SupportPac from IBM to enable topic-based (publish/subscribe) messaging support in MQ. For example, a search for `ma0c_ntmq52` will locate the topic-based messaging patch for MQ 5.2 on Windows.
4. Change directories to `<your MQ home directory>/Java/bin` and execute `runmqsc < MQJMS_PSQ.mqsc`.
5. Execute `IVTSetup.bat` The process should display `Done!` without reporting any errors.
6. Open `<your MQ home directory>Java/bin/jmsAdmin.config` and set the following properties:
   - `INITIAL_CONTEXT_FACTORY=com.sun.jndi.fscontext.RefFSContextFactory`
   - `PROVIDER_URL=file://c:/temp`
   - `SECURITY_AUTHENTICATION=none`
7. Create a file named `MyJNDI.txt` containing the following:
   ```
   DEFINE QCF(MyQCF) HOST(<your host name>)  +PORT(<your MQ listener port
   name e.g. 1414>)  +
   CHANNEL(<your MQ server connection channel name, for example, CHANNEL1>)
   +
   QMGR(<your MQ queue manager name>) +
   TRAN(client)
   END
   ```
8. Bind objects to JNDI names by running `<your MQ home directory>/java/bin/jmsAdmin.bat < MyJNDI.txt`
9. Configure the following JMS connector-specific properties as shown:

   ```
   CTX_InitialContextFactory = com.sun.jndi.fscontext.RefFSContextFactory
   CTX_ProviderURL = file://c:/temp
   ConnectionFactoryName = MyQCF
   ```

Notices

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

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

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

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

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

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

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

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

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

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

Burlingame, CA 94010
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

IBM
the IBM logo
AIX
CrossWorlds
DB2
DB2 Universal Database
Domino
Lotus
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

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

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

System Manager includes software developed by the Eclipse Project (http://www.eclipse.org/)

WebSphere Business Integration Server Express V4.3 and WebSphere Business Integration Server Express Plus V4.3

Printed in USA