WebSphere Business Integration Adapters

Adapter for JMS User Guide

Adapter Version 2.8.x

WebSphere Business Integration Adapters

Adapter for JMS User Guide

Adapter Version 2.8.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 111.

13September2005

This edition of this document applies to IBM WebSphere Business Integration Adapter for JMS (5724-G94), version 2.8.x.

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

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

Contents

About this document

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

This document describes installation, connector property configuration, business object development, and troubleshooting for the IBM WebSphere Business Integration Adapter for JMS.

This document does not describe deployment metrics and capacity planning issues, such as server load balancing, number of adapter processing threads, maximum and minimum throughputs, and tolerance thresholds.

Such issues are unique to every customer deployment and must be measured within or close to the exact environment where the adapter is to be deployed. You should contact your IBM services representative to discuss the configuration of your deployment site, and for details on planning and evaluating these kinds of metrics, given your specific configuration.

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. For links, see "Related documents."

Related documents

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

You can install related documentation from the following sites:

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

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

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. Additional information might also be available in IBM Redbooks at http://www.redbooks.ibm.com/.

Typographic conventions

This document uses the following conventions:

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

New in this release

New in release 2.8.x

Updated in September 2005. The release of this document for adapter version 2.7.x contains the following new or corrected information.

Added support for AIX 5.3

Added support for the management of the JMS adapter by the IBM Tivoli License Manager (ITLM)

Support for Bidirectional text

Enhanced multi-threading for event polling through the optional connector configuration property WorkerThreadCount

Enhanced object message support. The adapter now supports Java object type messages.

Optimized Data Handler calling through the optional connector configurator property DataHandlerPoolSize

Optional formatting for specifying connector properties ConnectionFactoryName and ArchivalConnectionFactoryName

New in release 2.7

Updated in September 2004. The release of this document for adapter version 2.7.x contains the following new or corrected information.

This release adds support for the following platforms:

- Solaris 9: this adapter supports 32-bit JVM on a 64-bit platform
- For already supported AIX 5.1 and 5.2: this adapter supports 32-bit JVM on a 64-bit platform
- Microsoft Windows 2003
- Linux RedHat AS 3.0, ES 3.0 and WS 3.0
- SUSE Linux Standard Server 8.1 and Enterprise Server 8.1 SP3
- IBM JRE/JDK 1.4.2

This release supports use of tracing level 5 to dump the printStackTrace() on exceptions caught by the adapter.

New in release 2.6

Three connector-specific properties have been added: EnableMessageProducerCache, SessionPoolSizeForRequests, and ArchivalConnectionFactoryName. For further information, see "Configuring connector-specific properties" on page 20.

As of version 2.6.x, the adapter is not supported on Solaris 7, so references to that platform version have been deleted from this guide.

New in release 2.5

The adapter can now use WebSphere Integration Message Broker as an integration broker. For further information, see "Broker compatibility" on page 1.

Beginning with the 2.5.x version, the adapter for JMS is not supported on Microsoft Windows NT.

Adapter installation information has been moved from this guide. See Chapter 2 for the new location of that information.

The adapter now supports the publish-and-subscribe (topic-based) messaging style defined by the JMS standard as well as the point-to-point (queue-based) messaging interface. A single instance of the adapter supports one messaging style only; topics and queues cannot both be specified in the same configuration. However, both messaging styles can be supported by running multiple instances of the adapter, with one or more instances implementing topic-based messaging and one or more instances a queue-based style.

New in release 2.4.x

The adapter can now use WebSphere Application Server as an integration broker. For further information, see "Broker compatibility" on page 1.

The connector now runs on the following platforms:

- Microsoft Windows NT 4.0 Service Pack 6A or Windows 2000
- Solaris 7, 8 or AIX 5.1, 5.2 or HP UX 11.i

New in release 2.3.x

Updated in March, 2003. The ″CrossWorlds″ name is no longer used to describe an entire system or to modify the names of components or tools, which are otherwise mostly the same as before. For example ″CrossWorlds System Manager″ is now ″System Manager,″ and ″CrossWorlds InterChange Server″ is now ″WebSphere InterChange Server.″

You can now associate a data handler with an input queue. For further information, see "Mapping data handlers to input destinations" on page 36.

The guaranteed event delivery feature has been enhanced. For further information, see the *Connector Development Guide for Java*.

New in release 2.2.x

The InProgress queue is no longer required and may be disabled. For more information, see "InProgressDestination" on page 26..

The ReplyToQueue can now be dictated via the dynamic child meta-object rather than by the ReplyToQueue connector property. For more information see "JMS headers and dynamic child meta-object attributes" on page 40..

You can use a message selector to identify, filter and otherwise control how the adapter identifies the response message for a given request. This JMS capability applies to synchronous request processing only. For more information, see "Synchronous processing" on page 13.

New in release 2.1.x

The connector has been internationalized. For more information, see "Locale-dependent data" on page 3 and Appendix A, "Standard configuration properties for connectors," on page 51.

This guide provides information about using this adapter with ICS.

Note: To use the guaranteed event delivery feature, you must install release 4.1.1.2 of ICS.

New in release 1.3.x

The IBM WebSphere Business Integration Adapter for JMS includes the connector for JMS. This adapter operates with InterChange Server (ICS) integration broker. An integration broker, which is an application that performs integration of heterogeneous sets of applications, provides services that include data routing. The adapter includes:

- An application component specific to JMS
- Sample business objects
- IBM WebSphere Adapter Framework, which consists of:
  - Connector Framework
  - Development tools (including Business object Designer and IBM CrossWorlds System Manager)
  - APIs (including CDK)

This manual provides information about using this adapter with ICS.

Important: Because the connector has not been internationalized, do not run it against ICS version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed.

The connector is now enabled for AIX 4.3.3 Patch Level 7.

New in release 1.2.x

In previous versions of the connector for JMS, the data handler that was used to convert data between JMS messages and IBM CrossWorlds business objects was determined by the `DataHandlerConfigMO` and `DataHandlerMimeType` connector properties. This had the limiting effect of requiring multiple instances of the connector to process different data formats. In release 1.2.x, the connector now allows you to optionally specify these properties in the connector's static meta-object or in a request business object's dynamic child meta-object. For details, see "Configuring meta-objects" on page 30.

New in release 1.1.x

This release of the document contains information for the following new features and product enhancements:

- The connector recognizes and reads conversion properties from a dynamic meta-object that is added as a child to the business object passed to the connector. The attribute values of this dynamic child meta-object duplicate the conversion properties previously specifiable via the static meta-object used to configure the connector. The connector property specifying the static meta-object is no longer required, but can still be used. You can use a dynamic child meta-object independent of the static meta-object and vice-versa.
- The connector accepts multiple queue names for the connector property *InputQueue*. The connector polls the queues in a round-robin manner and retrieves up to *pollQuantity* number of messages from each queue. Multiple queue names are delimited by semi-colons.

Chapter 1. Adapter for JMS overview

The connector for JMS is a runtime component of the IBM WebSphere Business Integration Adapter for JMS. The connector allows IBM WebSphere integration brokers 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 adapters operate with an integration broker. The connector for JMS operates with:

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

Adapter for JMS environment

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

Broker compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating.

This adapter runs with the WebSphere Business Integration Adapter Framework version 2.6 and requires one of the following:

- WebSphere InterChange Server V 4.2.2 or V 4.3
- WebSphere MQ Integrator V 2.1
- WebSphere MQ Integrator Broker V 2.1
- WebSphere Business Integration Message Broker V 5.0.1
- WebSphere Application Server Enterprise V 5.0.2, in conjunction with WebSphere Studio Application Developer Integration Edition V 5.0.1
- WebSphere Business Integration Server Foundation V 5.1 or V 5.1.1

See the Release Notes for any exceptions.

Hardware and software requirements are also available in the following Techdoc:

http://www.ibm.com/support/docview.wss?uid=swg27006249

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

In addition to a broker, this adapter requires one of the following operating systems:

Note: All operating system environments require the Java compiler (IBM JDK 1.4.2 for Windows 2000) for compiling custom adapters.

- Microsoft Windows 2000 (Professional, Server, or Advanced Server) with Service Pack 4
- Windows XP with Service Pack 1A, for WebSphere Business Integration Adapter Framework (administrative tools only)
- Windows 2003 (Standard Edition or Enterprise Edition)
- Solaris 8 (2.8) with Solaris Patch Cluster dated February 11, 2004 or later.
- Solaris 9 (2.9) with Solaris Patch Cluster dated February 11, 2004. This adapter supports 32-bit JVM on a 64-bit platform.
- AIX 5.2 with Maintenance Level 1 or AIX 5.3. This adapter supports 32-bit JVM on a 64-bit platform.
- HP-UX 11i (11.11) with June 2003 GOLDBASE11i and June 2003 GOLDAPPS11i bundles
- RedHat Enterprise Linux AS 3.0 with Update 1
- RedHat Enterprise Linux ES 3.0 with Update 1
- RedHat Enterprise Linux WS 3.0 with Update 1

- SUSE Linux Enterprise Server x86 8.1 with SP3
- SUSE Linux Standard Server x86 8.1 with SP3 and Enterprise Server 8.1 SP3

  Note: The TMTP (Tivoli Monitoring for Transaction Performance) component of the WebSphere Business Integration Adapter Framework V 2.6 is not supported on Linux Red Hat.
- IBM JRE/JDK 1.4.2

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.

Common Event Infrastructure

This adapter is compatible with IBM's Common Event Infrastructure, a standard for event management that permits interoperability with other IBM WebSphere event-producing applications. If Common Event Infrastructure support is enabled, events produced by the adapter can be received (or used) by another Common Event Infrastructure-compatible application.

For more information, refer to the Common Event Infrastructure appendix in this guide.

Application Response Measurement

This adapter is compatible with the Application Response Measurement application programming interface (API), and API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

For more information, refer to the Application Response Measurement appendix in this guide.

Locale-dependent data

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

This adapter supports the processing of bidirectional (bi-di) script data for the Arabic and Hebrew languages when the adapter is run in a Windows environment. Bidirectional processing is not supported in non-Windows environments. To use the bidirectional capacity, you must configure the bidirectional standard properties. For more information refer to the standard configuration properties for connectors in Appendix A, "Standard configuration properties for connectors," on page 51.

The Java runtime environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encoding for characters in most known character code sets (both single-byte and multibyte). Most

components in the WebSphere business integration system are written in Java. Therefore, when data is transferred between most integration components, there is no need for character conversion.

To log error and informational messages in the appropriate language and for the appropriate country or territory, configure the `Locale` standard configuration property for your environment. For more information on configuration properties, see Appendix A, "Standard configuration properties for connectors," on page 51.

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 WBI 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 (such as WebSphere MQ) and then 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" and "Request message processing" on page 10.

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 6
- "Event retrieval" on page 8

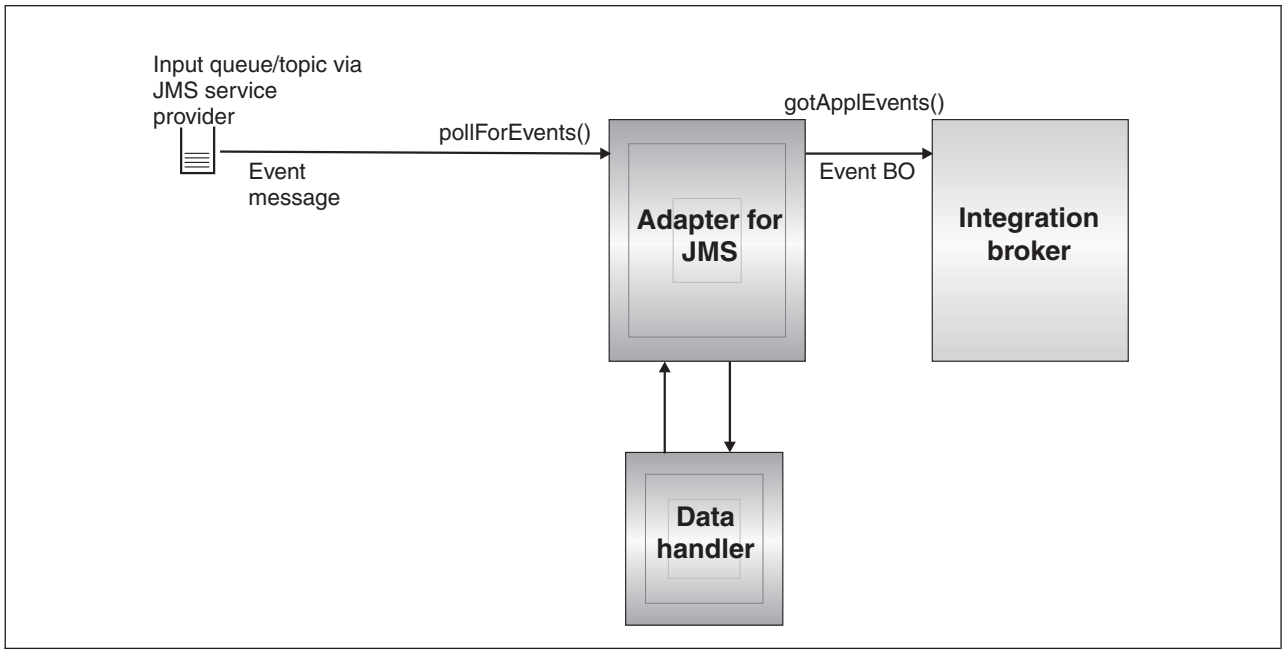


Figure 1. Event message flow

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 19). 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 order 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 7).

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 26.

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. To use CME, WebSphere MQ must be your JMS provider and the source and desination queues must be on one queue manager.

Note: When configured for Pub/Sub-style messaging, the connector does not support CME. For more on how this method of guaranteed-event-delivery works, see the *Connector Development Guide for Java*. For further information on the ContainerManagedEvents connector property, see "ContainerManagedEvents" on page 62.

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 information on DEE and enabling it, see the *Connector Development Guide for Java*. For further information on the DuplicateEventElimination connector property, see "DuplicateEventElimination" on page 65.

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 9
- "Understanding message header mapping" on page 9
- "Archiving" on page 10
- "Error recovery" on page 10

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 9 and "Understanding message header mapping" on page 9.

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 further information on metadata, see the *Connector Development Guide for Java*. For information on configuring static and dynamic meta-objects, see "Configuring meta-objects" on page 30.

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 8, 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 39.

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 20.

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.

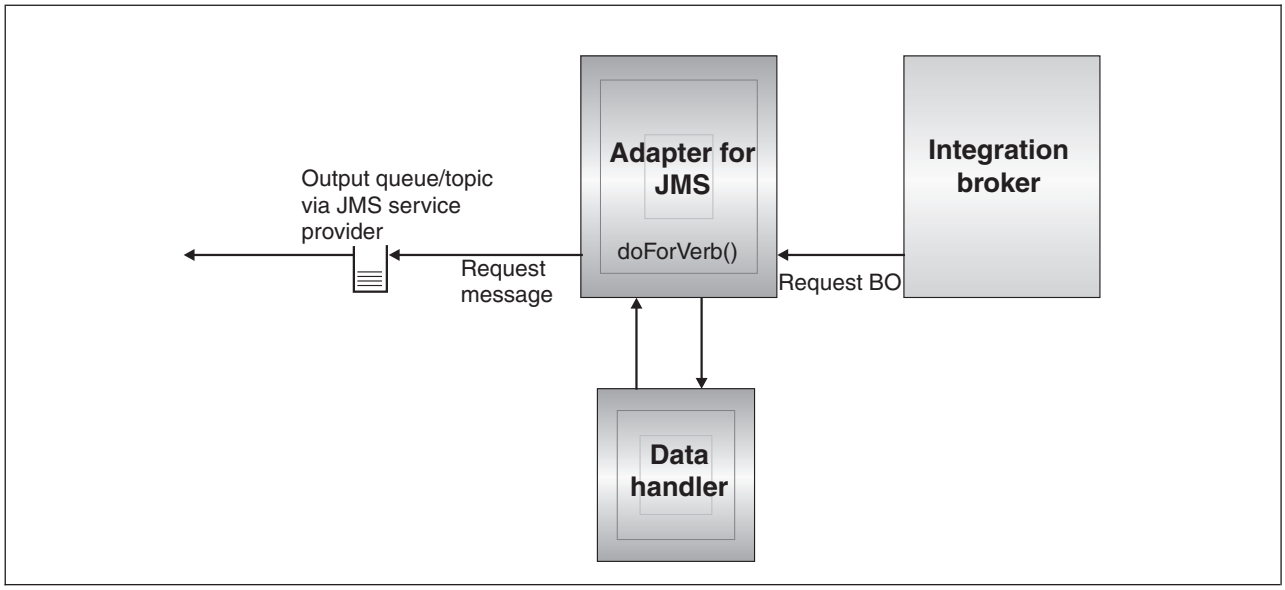


Figure 2. Request flow

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 13

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 30.

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 *undefined value*	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.
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 49.

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 the WebSphere business integration system and starting the integration broker, is described in the installation documentation for your broker and operating system.
- **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 WebSphere Business Integration adapter products, refer to the *Installing WebSphere Business Integration Adapters* guide located in the WebSphere Business Integration Adapters Infocenter at the following site:

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

Installed file structure

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

Windows connector file structure

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

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

Table 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\start_JMS.bat	The startup script for the connector (NT/2000)
connectors\messages\JMSConnector.txt	Message file for the connector
bin\Data\App\JMSConnectorTemplate	Template file for the connector definition
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	Sample business object repository file
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 child business object for dynamic meta-object

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

UNIX connector file structure

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

The utility installs the connector agent into the *ProductDir*/connectors/JMS directory.

Table 8 describes the UNIX file structure used by the connector, and shows the files that are automatically installed when you choose to install the connector through Installer.

Table 8. Installed UNIX file structure for the connector

Subdirectory of *ProductDir*	Description
connectors/JMS/CWJMS.jar	Contains classes used by the JMS connector
connectors/JMS/start_JMS.sh	System startup script for the connector. This script is called from the generic connector manager script. When you click the Connector configuration screen of System Manager, the installer creates a customized wrapper for this connector manager script. Use this customized wrapper to start and stop the connector.

Table 8. Installed UNIX file structure for the connector (continued)

Subdirectory of *ProductDir*	Description
connectors/messages/JMSConnector.txt	Message file for the connector
binData/App/JMSConnectorTemplate	Template file for the connector definition
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	Sample business object repository file
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 child business object for dynamic meta-object

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 28
- "Configuring JNDI" on page 29
- "Configuring meta-objects" on page 30
- "Configuring startup scripts" on page 42

Configuring connector properties

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

- "Configuring standard connector properties" on page 20
- "Configuring connector-specific properties" on page 20

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

You use Connector Configurator to configure connector properties:

- For a description of Connector Configurator and step-by-step procedures, see Appendix B, "Connector Configurator," on page 75.
- For a description of standard connector properties, see "Configuring standard connector properties" on page 20 and then Appendix A, "Standard configuration properties for connectors," on page 51.
- For a description of connector-specific properties, see "Configuring connector-specific properties" on page 20.

Configuring standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 51 for documentation of these properties. Then, for a step-by-step procedure describing how to set these properties see Appendix B, "Connector Configurator," on page 75.

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

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

Table 9. Connector-specific configuration properties

Name	Possible values	Default value	Required
ArchivalConnectionFactoryName	Name of object in JNDI store to retrieve and use for archiving events; supports both publishing styles—PTP (queue-based) and Pub/Sub (topic-based).		No
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_Authoritative	*Constant that holds the name of the environment property for specifying the authoritativeness of the service requested.*		No
CTX_Batchsize	*Constant that holds the name of the environment property for specifying the batch size to use when returning data via the service's protocol.*		No
CTX_DNS_URL	*Constant that holds the name of the environment property for specifying the DNS host and domain names to use for the JNDI URL context (for example, "dns://somehost/wiz.com").*		No
CTX_InitialContextFactory	*Name of factory class to be used to establish an initial JNDI context.*		Yes
CTX_Language	*Constant that holds the name of the environment property for specifying the preferred language to use with the service.*		No
CTX_ObjectFactories	*Constant that holds the name of the environment property for specifying the list of object factories to use.*		No
CTX_ProviderURL	*URL identifying the JNDI context where the connection factory is located.*		Yes

Table 9. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
CTX_Referral	*Constant that holds the name of the environment property for specifying how referrals encountered by the service provider are to be processed.*		No
CTX_SecurityAuthentication	*Constant that holds the name of the environment property for specifying the security level to use.*		No
CTX_SecurityCredentials	*Constant that holds the name of the environment property for specifying the credentials of the principal for authenticating the caller to the service.*		No
CTX_SecurityPrincipal	*Constant that holds the name of the environment property for specifying the identity of the principal for authenticating the caller to the service.*		No
CTX_SecurityProtocol	*Constant that holds the name of the environment property for specifying the security protocol to use.*		No
CTX_URLPackagePrefixes	*Constant that holds the name of the environment property for specifying the list of package prefixes to use when loading in URL context factories.*		No
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
DataHandlerPoolSize	*Maximum number of DataHandler instances to be cached for a particular type of DataHandler*	`30`	No
DefaultVerb	*Specifies the verb to be set within an incoming business object*	`Create`	No
EnableMessageProducerCache	`true` or `false`	`true`	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.
SessionPoolSizeForRequests	*Maximum pool size for caching the sessions used during request processing.*	10	No

Table 9. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
UnsubscribeOnTerminate	*Specify removed topics from InputDestination.*		No

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

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

WorkerThreadCount	*Maximum number of parallel threads for polling.*	1	No

ArchivalConnectionFactoryName

This property enables the connector to support event archival in either point-to-point or topic-based styles. The property names the JMS queue or topic connection factory object, defined in the JNDI store, that the connector should retrieve and use for establishing a connection to the JMS provider. This connection object is then used to create publisher references to the archive destinations. The connector properties that define archival destinations are:

- InProgressDestination
- ErrorDestination
- UnsubscribedDestination
- ArchiveDestination

If this property is undefined, the connector uses the factory specified in the ConnectionFactoryName property to create references to archival destinations.

Format of specifying this property is

```
connection_factory_name:<msg_system_type>
```

where connection_factory_name is mandatory and the second parameter, which takes the value of Topics or Queues (`<msg_system_type>:`), is optional. The second parameter tells the adapter to determine the messaging system type based on the user configuration.

For example, to specify pub/sub messaging, configure the ConnectionFactoryName as follows:

```
<connection_factory_name>:Topics
```

For point-to-point messaging, configure the ConnectionFactoryName as follows:

```
<connection_factory_name>:Queues
```

If you leave the second parameter (either Topics or Queues) blank, the adapter will attempt to identify the messaging system based on the instance of the factory object.

Default = none.

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.

Format of specifying this property is

```
connection_factory_name:<msg_system_type>
```

where connection_factory_name is mandatory and the second parameter, which takes the value of Topics or Queues (`<msg_system_type>:`), is optional. The second parameter tells the adapter to determine the messaging system type based on the user configuration.

For example, to specify pub/sub messaging, configure the ConnectionFactoryName as follows:

```
<connection_factory_name>:Topics
```

For point-to-point messaging, configure the ConnectionFactoryName as follows:

```
<connection_factory_name>:Queues
```

If you leave the second parameter (either Topics or Queues) blank, the adapter will attempt to identify the messaging system based on the instance of the factory object.

Default = none.

CTX_Authoritative
Constant that holds the name of the environment property for specifying the authoritativeness of the service requested. If the value of the property is the string ″true″, it means that the access the most authoritative source (i.e. bypass any cache or replicas). If the value is anything else, the source need not be (but can be) authoritative.

If unspecified, this property defaults to ″false″.

CTX_Batchsize
Constant that holds the name of the environment property for specifying the batch size to use when returning data via the service's protocol. This is a hint to the provider to return the results of operations in batches of the specified size, so the provider can optimize its performance and usage of resources. The value of the property is the string representation of an integer. If unspecified, the batch size is determined by the provider.

CTX_DNS_URL
Constant that holds the name of the environment property for specifying the DNS host and domain names to use for the JNDI URL context (for example, ″dns://somehost/wiz.com″). If the property is not specified in the environment, the system property by the same name is used. If not specified as a system property either and the program attempts to use a JNDI URL containing a DNS name, ConfigurationException is thrown.

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

Default = none.

CTX_Language
Constant that holds the name of the environment property for specifying the preferred language to use with the service. The value of the property is a colon-separated list of language tags defined in RFC 1766. If unspecified, the language preference is determined by the service provider.

CTX_ObjectFactories
Constant that holds the name of the environment property for specifying the list of object factories to use. The value of the property should be a colon-separated list of the fully qualified class names of factory classes that will create an object given information about the object. If the property is not specified in the environment, the system property by the same name is used. If not specified as a system property either, NamingManager.getObjectInstance() will use means to attempt to create the object.

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

Default = none.

CTX_Referral
Constant that holds the name of the environment property for specifying how referrals encountered by the service provider are to be processed. The value of the property is one of the following strings:

- follow
- follow referrals automatically
- ignore
- ignore referrals
- throw
- throw ReferralException when a referral is encountered
- If unspecified, the default is determined by the provider

CTX_SecurityAuthentication
Constant that holds the name of the environment property for specifying the security level to use. Its value is one of the following strings: ″none″, ″simple″, ″strong″. If unspecified, the behaviour is determined by the service provider.

CTX_SecurityCredentials
Constant that holds the name of the environment property for specifying the credentials of the principal for authenticating the caller to the service. The value of the property depends on the authentication scheme. For example, it could be a

hashed password, clear-text password, key, certificate, and so on. If unspecified, the behaviour is determined by the service provider.

CTX_SecurityPrincipal

Constant that holds the name of the environment property for specifying the identity of the principal for authenticating the caller to the service. The value of the property depends on the authentication scheme . If unspecified, the behaviour is determined by the service provider.

CTX_SecurityProtocol

Constant that holds the name of the environment property for specifying the security protocol to use. Its value is a string determined by the service provider (e.g. ″ssl″). If unspecified, the behaviour is determined by the service provider.

CTX_URLPackagePrefixes

Constant that holds the name of the environment property for specifying the list of package prefixes to use when loading in URL context factories. The value of the property should be a colon-separated list of package prefixes for the class name of the factory class that will create a URL context factory. If the property is not specified in the environment, the system property by the same name is used. The prefix com.sun.jndi.url is always added to the end of the possibly empty list of package prefixes.

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`.

DataHandlerPoolSize

Set this optional property if you would like the adapter to pool previously created instances of data handlers for future use. The maximum number of DataHandler instances you would like to be cached for a particular type of DataHandler.

Default=`30`

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`

EnableMessageProducerCache

Boolean property to specify that the adapter should enable a message producer cache for sending request messages.

Default= `true`

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.

If the JMS header property specified by the MessageResponseResultProperty does not exist, the connector interprets the return code as VALCHANGE, passes whatever is in the response message to the data handler, and then updates the business object.

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.

SessionPoolSizeForRequests

Maximum pool size for caching the sessions used during request processing.

Default = 10

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

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

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`.

WorkerThreadCount

Maximum number of parallel threads for polling. While concurrently processing the events, the adapter will not be able to submit events to the broker in the order it received. If the sequence needs to be maintained WorkerThreadCount should always be set to 1.

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 23 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.
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.
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" on page 30.

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 99and "Configuring for topic-based messaging" on page 100.

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 8.

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 10 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 10. 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 20.
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 20.
DataHandlerClassName	Yes	Yes	See the description in "Configuring connector-specific properties" on page 20.
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

Table 10. JMS meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
`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.
`DataEncoding`	Yes	Yes	`DataEncoding` refers to whether message data is dealt with by the adapter as text, binary, or object type. If this property is not specified in the static meta-object, the connector tries to read the messages without using any specific encoding. `DataEncoding` defined in a dynamic child meta-object overrides the value defined in the static meta-object. The default value is `Text`. The format for the value of this attribute is `messageType[:enc]`. I.e., `Text:ISO8859_1`, `Text:UnicodeLittle`, `Text`, `Object`, or `Binary`. This property is related internally to the InputFormat property: specify one and only one `DataEncoding` per `InputFormat`. For more information about DataEncoding, see "DataEncoding" on page 33.
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.
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.
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.

Table 10. JMS meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
JMSProperties		Yes	See "JMS properties" on page 40.

DataEncoding

The JMS standard defines different types of messages for storing different types of data. For example, a text message could contain an XML document, while binary data might contain an image. Explicitly specifying the expected data encoding through the meta-object property `DataEncoding` helps the adapter understand what types of messages it should be expecting and how to process them.

By default, the adapter assumes that all messages are text. However, the adapter for JMS supports text, binary, and Java object type messages. During request processing, the adapter uses the configured data handler to convert the request business object to text, which it then delivers as a text message. During event notification, the adapter retrieves the text from a text message and passes it to the data handler to create the event business object. If the adapter receives a binary message, it converts the binary body to text using the default encoding of the JVM before passing the content to the data handler.

In some cases, these scenarios are not appropriate. If you want to process binary messages and your data handler is designed for binary data, you must use the `DataEncoding` property. This property accepts one of three possible values:

- `text`: Default value.
- `binary`: If you choose `binary`, the adapter changes as follows:
  1. During request processing, the adapter passes the business object to the `binary` methods of the data handler and delivers a bytes message.
  2. During event notification, the adapter retrieves the bytes from the binary message and passes them to the data handler as a Java `InputStream` instance (bytes).
  3. If the adapter receives a text message, it converts the text body to binary using the default encoding of the JVM before passing the content to the data handler.

  For binary messages, set `DataEncoding=binary`

  Note: Not all data handlers support binary data, so be sure to check your configured data handler for support restrictions.
- `object`: If you choose `object`, the adapter changes as follows:
  1. During request processing, the adapter passes the business object to the `getStreamFromBO()` method of the data handler to get an `InputStream`.
  2. During event notification, the adapter retrieves the java objects from the object message and passes them to the data handler as a `InputGetStream`.

  For object type messages, set `DataEncoding=object`

  Note: Not all data handlers support Java object data, so be sure to check your configured data handler for support restrictions.

When possible, it is always recommended that you send text data in JMS text messages rather than have the adapter handle the conversion from text to binary (or vice-versa). This is because incorrect encoding can cause subtle data corruption

that is difficult to detect or diagnose. However, it is possible to specify how you would like the adapter to encode the conversion between binary and text (or vice-versa) using an additional modifier provided in the DataEncoding property. If you append a colon and the name of a supported Java encoding, the adapter will use the specified encoding to convert. The format is text:encoding.

For example, DataEncoding=text;ISO8859_1 informs the adapter that if any binary messages are received, it should convert the binary message body to text using encoding ISO8859_1 before passing it to the data handler. If the message type matches the data encoding (for example, if the binary message and data encoding specifies binary or text message and the data encoding specifies text), this value will have no impact.

Data handler design for binary data: If you plan to design and use a custom data handler with the adapter, consider the following:

- You must provide an implementation for the method getStreamFromBO (BusinessObject, Object).
- You must override and provide implementation for the methods getBO (InputStream, Object) and getBO (InputStream, BusinessObject, Object).
- Default inplementations of these methods are provided in the base DataHandler class, but these methods simply convert any binary data to text before invoking the corresponding text methods on your custom data handler subclass (defeating the purpose). You must provide you own implementations for binary data support to work.

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 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 11. 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 12. Sample static meta-object structure

Attribute name	Application-specific information
Customer_Create	OutputFormat=CUST;OutputDestination=QueueA

Table 12. Sample static meta-object structure (continued)

Attribute name	Application-specific information
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_Update` and `Order_Create` as having `OutputDestination=QueueA` in addition to their individual meta-data properties:

Table 13. Sample static meta-object structure

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

See Table 10 on page 31 in “Meta-object properties” on page 30describes 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. For further information, see the *Business Object Development Guide*.
2. Open the sample meta-object `connectors\JMS\Samples\Sample_JMS_MO_Config.xsd`.

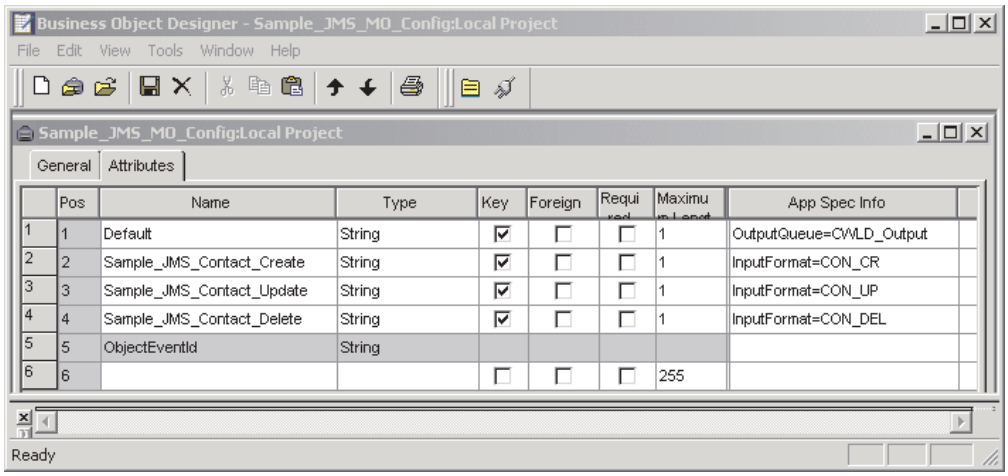


Figure 3. A sample static meta-object

3. Edit the attributes and ASI to reflect your requirements, referring to Table 10 on page 31 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. For further information, see Appendix B, "Connector Configurator," on page 75.
2. Use connector-specific properties (see "InputDestination" on page 26) 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 10 on page 31 in “Meta-object properties” on page 30describes the properties that you can specify as application-specific information in the dynamic meta-object.

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

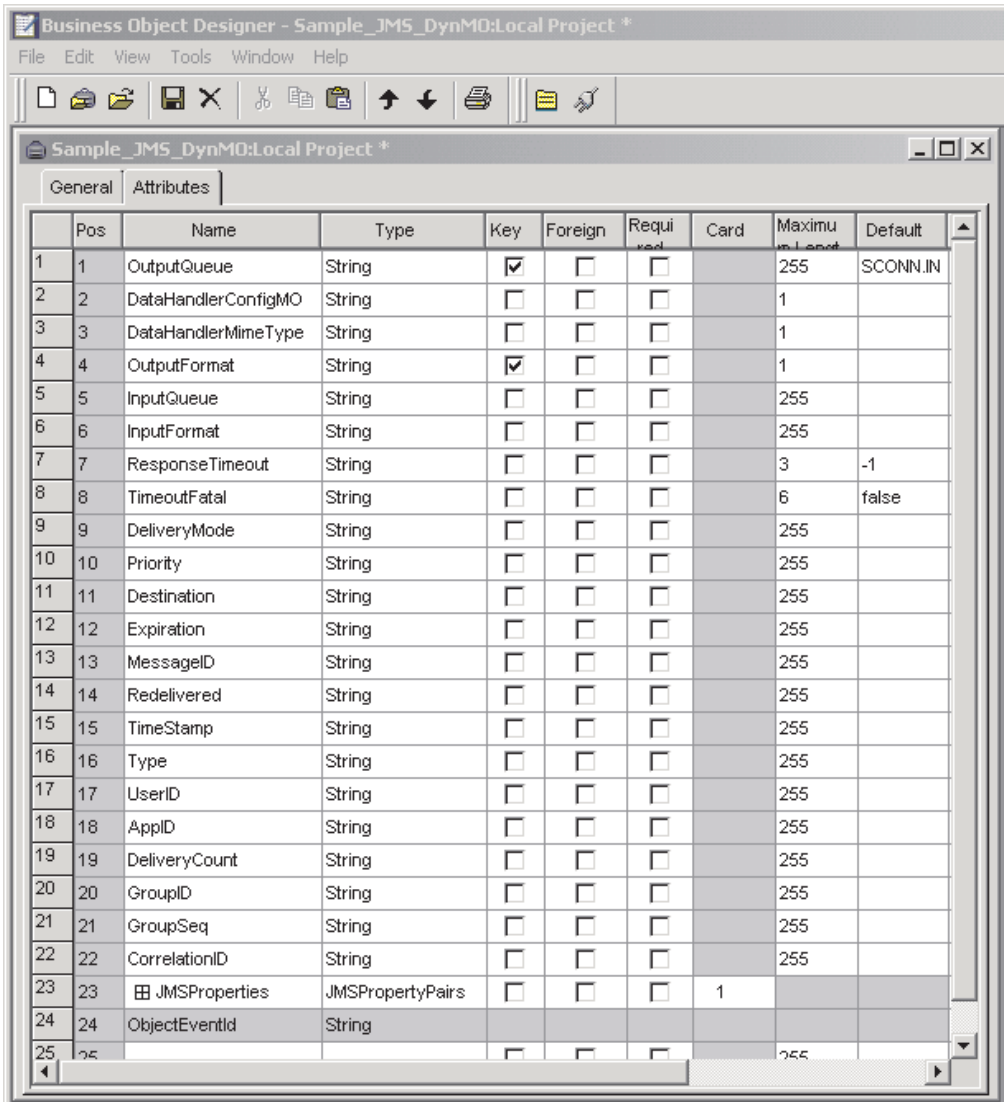
Table 14. Dynamic meta-object header attributes

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

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

To implement a dynamic meta-object:

1. Launch Business Object Designer. For further information, see the *Business Object Development Guide*.
2. Open the sample meta-object `connectors\JMS\Samples\Sample_JMS_DynMO.xsd`.



	Pos	Name	Type	Key	Foreign	Requi red	Card	Maximu m Lengt	Default
1	1	OutputQueue	String	☑	☐	☐		255	SCONN.IN
2	2	DataHandlerConfigMO	String	☐	☐	☐		1	
3	3	DataHandlerMimeType	String	☐	☐	☐		1	
4	4	OutputFormat	String	☑	☐	☐		1	
5	5	InputQueue	String	☐	☐	☐		255	
6	6	InputFormat	String	☐	☐	☐		255	
7	7	ResponseTimeout	String	☐	☐	☐		3	-1
8	8	TimeoutFatal	String	☐	☐	☐		6	false
9	9	DeliveryMode	String	☐	☐	☐		255	
10	10	Priority	String	☐	☐	☐		255	
11	11	Destination	String	☐	☐	☐		255	
12	12	Expiration	String	☐	☐	☐		255	
13	13	MessageID	String	☐	☐	☐		255	
14	14	Redelivered	String	☐	☐	☐		255	
15	15	TimeStamp	String	☐	☐	☐		255	
16	16	Type	String	☐	☐	☐		255	
17	17	UserID	String	☐	☐	☐		255	
18	18	AppID	String	☐	☐	☐		255	
19	19	DeliveryCount	String	☐	☐	☐		255	
20	20	GroupID	String	☐	☐	☐		255	
21	21	GroupSeq	String	☐	☐	☐		255	
22	22	CorrelationID	String	☐	☐	☐		255	
23	23	⊞ JMSProperties	JMSPropertyPairs	☐	☐	☐	1		
24	24	ObjectEventId	String						
25				☐	☐	☐		255	

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

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

Attribute name	Sample value
`InputFormat`	`CUST_IN`
`InputQueue`	`MYInputQueue`

Table 15. JMS dynamic child meta-object structure for polling (continued)

Attribute name	Sample value
OutputFormat	CxIgnore
OutputQueue	CxIgnore
ResponseTimeout	CxIgnore
TimeoutFatal	CxIgnore

As shown in Table 15 on page 39, 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 16. 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).

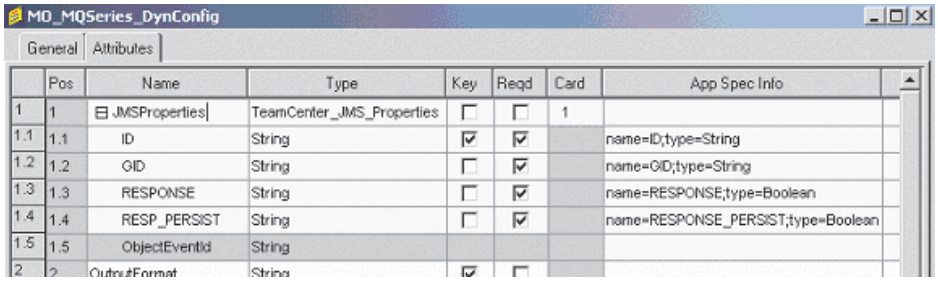


Figure 5. JMS properties attribute in a dynamic meta-object

Configuring startup scripts

The connector comes with startup scripts—`JMS.bat` or `JMS.sh`, depending on your operating system. For information on starting a connector, stopping a connector, and the connector's temporary startup log file, see the see the startup chapter in the *System Installation Guide* for your platform.

Creating multiple connector instances

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

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

Create a new directory

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

```
ProductDir\connectors\connectorInstance
```

where `connectorInstance` uniquely identifies the connector instance.

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

```
ProductDir\repository\connectorInstance
```

Create business object definitions

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

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

   ```
   ProductDir\repository\initialConnectorInstance
   ```

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

Create a connector definition

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

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

Create a start-up script

To create a startup script:

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

   dirname
2. Put this startup script in the connector directory you created in "Create a new directory" on page 42.
3. Create a startup script shortcut (Windows only).
4. Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.

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

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

Starting the connector

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

ProductDir`\connectors\`*connName*

where *connName* identifies the connector.

On UNIX systems the startup script should reside in the *UNIX ProductDir*/bin directory.

The name of the startup script depends on the operating-system platform, as Table 17 shows.

Table 17. Startup scripts for a connector

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

When the startup script runs, it expects by default to find the configuration file in the *Productdir* (see the commands below). This is where you place your configuration file.

Note: You need a local configuration file if the adapter is using JMS transport.

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

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

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

    `start_connName connName brokerName [-cconfigFile ]`
  - On UNIX-based systems:

    `connector_manager -start connName brokerName [-cconfigFile ]`

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

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

  You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Manager (available for all brokers):

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

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

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

Stopping the connector

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

- If you started the connector from the command line, with its connector startup script:

- On Windows systems, invoking the startup script creates a separate "console" window for the connector. In this window, type "Q" and press Enter to stop the connector.
- When using InterChange Server on UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:

  ```
  connector_manager_connName -stop
  ```

  where *connName* is the name of the connector.

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

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

- From System Monitor (WebSphere InterChange Server product only):

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

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

Chapter 3. Creating or modifying business objects

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

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 the integration broker. For more on naming conventions see the naming conventions for your integration broker.

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 the integration broker will send business objects to when the broker 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 19. You can optionally specify special data handler processing rules in static and dynamic meta-objects. For further information, see "Meta-object properties" on page 30.
3. Use Business Object Designer to create the application-specific business objects. For further information see the *Business Object Designer*guide.
4. Add the business objects you create to the Supported Business Objects Using Connector Configurator, 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 to add supported business objects, see "Specifying supported business object definitions" on page 85.

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 APP_RESPONSETIMEOUT 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, 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 in for more on configuring trace messages. For more information on tracing, including how to enable and set it, see the *Connector Development Guide for Java*.

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.

  Use this level to dump the printStackTrace() on exceptions caught by the adapter.

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 WebSphere Business Integration adapters. The information covers connectors running with the following integration brokers:

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

If your adapter supports DB2 Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in Table 18 on page 53.)

The properties you set for the adapter depend on which integration broker you use. You choose the integration broker using Connector Configurator. After you choose the broker, Connector Configurator lists the standard properties you must configure for the adapter.

For information about properties specific to this connector, see the relevant section in this guide.

New properties

This standard property was added in this release:

- BOTrace

Standard connector properties overview

Connectors have two types of configuration properties:

- Standard configuration properties, which are used by the framework
- Application, or connector-specific, configuration properties, which are used by the agent

These properties determine the adapter framework and the agent run-time behavior.

This section describes how to start Connector Configurator and describes characteristics common to all properties. For information on configuration properties specific to a connector, see its adapter user guide.

Starting Connector Configurator

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

Connector Configurator and System Manager run only on the Windows system. If you are running the connector on a UNIX system, you must have a Windows machine with these tools installed.

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

Configuration property values overview

The connector uses the following order to determine a property's value:

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

The default length of a property field is 255 characters. There is no limit on the length of a STRING property type. The length of an INTEGER type is determined by the server on which the adapter is running.

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.

The update characteristics of a property, that is, how and when a change to the connector properties takes effect, depend on the nature of the property.

There are four update methods for standard connector properties:

- **Dynamic**
  The new value takes effect immediately after the change is saved in System Manager. However, if the connector is in stand-alone mode (independently of System Manager), for example, if it is running with one of the WebSphere message brokers, you can change properties only through the configuration file. In this case, a dynamic update is not possible.
- **Agent restart (ICS only)**
  The new value takes effect only after you stop and restart the connector agent.
- **Component restart**
  The new value takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the agent or the server process.
- **System restart**
  The new value takes effect only after you stop and restart the connector agent and the server.

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in Table 18 on page 53.

There are three locations in which a standard property can reside. Some properties can reside in more than one location.

- **ReposController**
  The property resides in the connector controller and is effective only there. If you change the value on the agent side, it does not affect the controller.
- **ReposAgent**
  The property resides in the agent and is effective only there. A local configuration can override this value, depending on the property.

- **LocalConfig**
  The property resides in the configuration file for the connector and can act only through the configuration file. The controller cannot change the value of the property, and is not aware of changes made to the configuration file unless the system is redeployed to update the controller explicitly.

Standard properties quick-reference

Table 18 provides a quick-reference to the standard connector configuration properties. Not all connectors require all of these properties, and property settings may differ from integration broker to integration broker.

See the section following the table for a description of each property.

Note: In the Notes column in Table 18, the phrase "RepositoryDirectory is set to <REMOTE>" indicates that the broker is InterChange Server. When the broker is WMQI or WAS, the repository directory is set to *<ProductDir>*\repository

Table 18. Summary of standard configuration properties

Property name	Possible values	Default value	Update method	Notes
AdapterHelpName	One of the valid subdirectories in *<ProductDir>*\bin\Data\App\Help\ that contains a valid *<RegionalSetting>* directory	Template name, if valid, or blank field	Component restart	Supported regional settings. Include `chs_chn`, `cht_twn`, `deu_deu`, `esn_esp`, `fra_fra`, `ita_ita`, `jpn_jpn`, `kor_kor`, `ptb_bra`, and `enu_usa` (default).
AdminInQueue	Valid JMS queue name	*<CONNECTORNAME>*`/ADMININQUEUE`	Component restart	This property is valid only when the value of DeliveryTransport is `JMS`
AdminOutQueue	Valid JMS queue name	*<CONNECTORNAME>*`/ADMINOUTQUEUE`	Component restart	This property is valid only when the value of DeliveryTransport is `JMS`
AgentConnections	`1` through `4`	`1`	Component restart	This property is valid only when the value of DeliveryTransport is `MQ` or `IDL`, the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
AgentTraceLevel	`0` through `5`	`0`	Dynamic if broker is ICS; otherwise Component restart	
ApplicationName	Application name	The value specified for the connector application name	Component restart	

Table 18. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
BiDi.Application	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transforma tion is true
BiDi.Broker	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true. If the value of BrokerType is ICS, the property is read-only.
BiDi.Metadata	Any valid combination of these bidirectional attributes: 1st letter: I,V 2nd letter: L,R 3rd letter: Y, N 4th letter: S, N 5th letter: H, C, N	ILYNN (five letters)	Component restart	This property is valid only if the value of BiDi.Transformation is true.
BiDi.Transformation	true or false	false	Component restart	This property is valid only if the value of BrokerType is not WAS.
BOTrace	none or keys or full	none	Agent restart	This property is valid only if the value of AgentTraceLevel is lower than 5.
BrokerType	ICS, WMQI, WAS	ICS	Component restart	
CharacterEncoding	Any supported code. The list shows this subset: ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 .	ascii7	Component restart	This property is valid only for C++ connectors.
CommonEventInfrastruc ture	true or false	false	Component restart	
CommonEventInfrastruc tureURL	A URL string, for example, corbaloc:iiop: host:2809.	No default value.	Component restart	This property is valid only if the value of CommonEvent Infrastructure is true.
ConcurrentEventTrig geredFlows	1 through 32,767	1	Component restart	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
ContainerManagedEvents	Blank or JMS	Blank	Component restart	This property is valid only when the value of Delivery Transport is JMS.

Table 18. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
ControllerEventSequencing	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerStoreAndForwardMode	true or false	true	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ControllerTraceLevel	0 through 5	0	Dynamic	This property is valid only if the value of RepositoryDirectory is set to <REMOTE> and the value of BrokerType is ICS.
DeliveryQueue	Any valid JMS queue name	*<CONNECTORNAME>*/DELIVERYQUEUE	Component restart	This property is valid only when the value of Delivery Transport is JMS.
DeliveryTransport	MQ, IDL, or JMS	IDL when the value of RepositoryDirectory is <REMOTE>, otherwise JMS	Component restart	If the value of RepositoryDirectory is not <REMOTE>, the only valid value for this property is JMS.
DuplicateEventElimination	true or false	false	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
EnableOidForFlowMonitoring	true or false	false	Component restart	This property is valid only if the value of BrokerType is ICS.
FaultQueue	Any valid queue name.	*<CONNECTORNAME>*/FAULTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.FactoryClassName	CxCommon.Messaging.jms.IBMMQSeriesFactory, CxCommon.Messaging.jms.SonicMQFactory, or any Java class name	CxCommon.Messaging.jms.IBMMQSeriesFactory	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
jms.ListenerConcurrency	1 through 32767	1	Component restart	This property is valid only if the value of jms.TransportOptimized is true.
jms.MessageBrokerName	If the value of jms.FactoryClassName is IBM, use crossworlds.queue.manager.	crossworlds.queue.manager	Component restart	This property is valid only if the value of DeliveryTransport is JMS .
jms.NumConcurrentRequests	Positive integer	10	Component restart	This property is valid only if the value of DeliveryTransport is JMS .

Table 18. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
jms.Password	Any valid password		Component restart	This property is valid only if the value of DeliveryTransport is JMS .
jms.TransportOptimized	true or false	false	Component restart	This property is valid only if the value of DeliveryTransport is JMS and the value of BrokerType is ICS.
jms.UserName	Any valid name		Component restart	This property is valid only if the value of Delivery Transport is JMS.
JvmMaxHeapSize	Heap size in megabytes	128m	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
ListenerConcurrency	1 through 100	1	Component restart	This property is valid only if the value of DeliveryTransport is MQ.
Locale	This is a subset of the supported locales: en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR	en_US	Component restart	
LogAtInterchangeEnd	true or false	false	Component restart	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MaxEventCapacity	1 through 2147483647	2147483647	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
MessageFileName	Valid file name	InterchangeSystem.txt	Component restart	

Table 18. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
MonitorQueue	Any valid queue name	<*CONNECTORNAME*> /MONITORQUEUE	Component restart	This property is valid only if the value of DuplicateEventElimination is `true` and ContainerManagedEvents has no value.
OADAutoRestartAgent	`true` or `false`	`false`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADMaxNumRetry	A positive integer	`1000`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
OADRetryTimeInterval	A positive integer in minutes	`10`	Dynamic	This property is valid only if the value of Repository Directory is set to <REMOTE> and the value of BrokerType is ICS.
PollEndTime	HH = 0 through 23 MM = 0 through 59	`HH:MM`	Component restart	
PollFrequency	A positive integer (in milliseconds)	`10000`	Dynamic if broker is ICS; otherwise Component restart	
PollQuantity	1 through `500`	`1`	Agent restart	This property is valid only if the value of ContainerManagedEvents is `JMS`.
PollStartTime	HH = 0 through 23 MM = 0 through 59	`HH:MM`	Component restart	
RepositoryDirectory	<REMOTE> if the broker is ICS; otherwise any valid local directory.	For ICS, the value is set to <REMOTE> For WMQI and WAS, the value is <*ProductDir* `\repository`	Agent restart	
RequestQueue	Valid JMS queue name	<*CONNECTORNAME*> /REQUESTQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is `JMS`
ResponseQueue	Valid JMS queue name	<*CONNECTORNAME*> /RESPONSEQUEUE	Component restart	This property is valid only if the value of DeliveryTransport is `JMS`.
RestartRetryCount	`0` through `99`	`7`	Dynamic if ICS; otherwise Component restart	

Table 18. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
RestartRetryInterval	A value in minutes from 1 through 2147483647	1	Dynamic if ICS; otherwise Component restart	
ResultsSetEnabled	true or false	false	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of DeliveryTransport is JMS, and the value of BrokerType is WMQI.
ResultsSetSize	Positive integer	0 (means the results set size is unlimited)	Component restart	Used only by connectors that support DB2II. This property is valid only if the value of ResultsSetEnabled is true.
RHF2MessageDomain	mrm or xml	mrm	Component restart	This property is valid only if the value of DeliveryTransport is JMS and the value of WireFormat is CwXML.
SourceQueue	Any valid WebSphere MQ queue name	*<CONNECTORNAME>*/SOURCEQUEUE	Agent restart	This property is valid only if the value of ContainerManagedEvents is JMS.
SynchronousRequest Queue	Any valid queue name.	*<CONNECTORNAME>*/SYNCHRONOUSREQUEST QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
SynchronousResponse Queue	Any valid queue name	*<CONNECTORNAME>*/SYNCHRONOUSRESPONSE QUEUE	Component restart	This property is valid only if the value of DeliveryTransport is JMS.
TivoliMonitorTransaction Performance	true or false	false	Component restart	
WireFormat	CwXML or CwBO	CwXML	Agent restart	The value of this property must be CwXML if the value of RepositoryDirectory is not set to <REMOTE>. The value must be CwBO if the value of RepositoryDirectory is set to <REMOTE>.
WsifSynchronousRequest Timeout	0 to any number (milliseconds)	0	Component restart	This property is valid only if the value of BrokerType is WAS.

Table 18. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
XMLNameSpaceFormat	short or long or no	short	Agent restart	This property is valid only if the value of BrokerType is WMQI or WAS

Standard properties

This section describes the standard connector configuration properties.

AdapterHelpName

The AdapterHelpName property is the name of a directory in which connector-specific extended help files are located. The directory must be located in <*ProductDir*>\bin\Data\App\Help and must contain at least the language directory enu_usa. It may contain other directories according to locale.

The default value is the template name if it is valid, or it is blank.

AdminInQueue

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

The default value is <*CONNECTORNAME*>/ADMININQUEUE

AdminOutQueue

The AdminOutQueue property specifies 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 (Object Request Broker) connections opened when the ORB initializes.

It is valid only if the value of the RepositoryDirectory is set to <REMOTE> and the value of the DeliveryTransport property is MQ or IDL.

The default value of this property is 1.

AgentTraceLevel

The AgentTraceLevel property sets the level of trace messages for the application-specific component. The connector delivers all trace messages applicable at the tracing level set and lower.

The default value is 0.

ApplicationName

The ApplicationName property uniquely identifies the name of the connector application. This name is used by the system administrator to monitor the integration environment. This property must have a value before you can run the connector.

The default is the name of the connector.

BiDi.Application

The BiDi.Application property specifies the bidirectional format for data coming from an external application into the adapter in the form of any business object supported by this adapter. The property defines the bidirectional attributes of the application data. These attributes are:

- Type of text: implicit or visual (`I` or `V`)
- Text direction: left-to-right or right-to-left (`L` or `R`)
- Symmetric swapping: on or off (`Y` or `N`)
- Shaping (Arabic): on or off (`S` or `N`)
- Numerical shaping (Arabic): Hindi, contextual, or nominal (`H`, `C`, or `N`)

This property is valid only if the BiDi.Transformation property value is set to true.

The default value is `ILYNN` (implicit, left-to-right, on, off, nominal).

BiDi.Broker

The BiDi.Broker property specifies the bidirectional script format for data sent from the adapter to the integration broker in the form of any supported business object. It defines the bidirectional attributes of the data, which are as listed under `BiDi.Application` above.

This property is valid only if the BiDi.Transformation property value is set to true. If the BrokerType property is ICS, the property value is read-only.

The default value is `ILYNN` (implicit, left-to-right, on, off, nominal).

BiDi.Metadata

The BiDi.Metadata property defines the bidirectional format or attributes for the metadata, which is used by the connector to establish and maintain a link to the external application. The attribute settings are specific to each adapter using the bidirectional capabilities. If your adapter supports bidirectional processing, refer to the section on adapter-specific properties for more information.

This property is valid only if the BiDi.Transformation property value is set to true.

The default value is `ILYNN` (implicit, left-to-right, on, off, nominal).

BiDi.Transformation

The BiDi.Transformation property defines whether or not the system performs a bidirectional transformation at run time.

If the property value is set to `true`, the BiDi.Application, BiDi.Broker, and BiDi.Metadata properties are available. If the property value is set to `false`, they are hidden.

The default value is `false`.

BOTrace

The BOTrace property specifies whether or not business object trace messages are enabled at run time.

Note: It applies only when the AgentTraceLevel property is set to less than 5.

When the trace level is set to less than 5, you can use these command line parameters to reset the value of BOTrace.

- Enter `-xBOTrace=Full` to dump all the business object's attributes.
- Enter `-xBOTrace=Keys` to dump only the business object's keys.
- Enter `-xBOTrace=None` to disable business object attribute dumping.

The default value is `false`.

BrokerType

The BrokerType property identifies the integration broker type that you are using. The possible values are `ICS`, `WMQI (for WMQI, WMQIB or WBIMB)`, or `WAS`.

CharacterEncoding

The CharacterEncoding property 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. C++ connectors use the value `ascii7` for this property.

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

CommonEventInfrastructure

The Common Event Infrastructure (CEI) is a simple event management function handling generated events. The CommonEventInfrastructure property specifies whether the CEI should be invoked at run time.

The default value is `false`.

CommonEventInfrastructureContextURL

The CommonEventInfrastructureContextURL is used to gain access to the WAS server that executes the Common Event Infrastructure (CEI) server application. This property specifies the URL to be used.

This property is valid only if the value of CommonEventInfrastructure is set to `true.`

The default value is a blank field.

ConcurrentEventTriggeredFlows

The ConcurrentEventTriggeredFlows property determines how many business objects can be concurrently processed by the connector for event delivery. You set the value of this attribute to the number of business objects that are mapped and delivered concurrently. For example, if you set the value of this property to 5, five business objects are processed concurrently.

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), the following properties must configured:

- The collaboration must be configured to use multiple threads by setting its `Maximum number of concurrent events` property high enough to use multiple threads.
- The destination application's application-specific component must be configured to process requests concurrently. That is, it must be multithreaded, or it must be able to use connector agent parallelism and be configured for multiple processes. The Parallel Process Degree configuration property must be set to a value larger than 1.

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

This property is valid only if the value of the RepositoryDirectory property is set to <REMOTE>.

The default value is 1.

ContainerManagedEvents

The ContainerManagedEvents 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 one JMS transaction.

When this property is set to `JMS`, the following properties must also be set to enable guaranteed event delivery:

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

You must also configure a data handler with the MimeType and DHClass (data handler class) properties. You can also add DataHandlerConfigMOName (the meta-object name, which is optional). To set those values, use the **Data Handler** tab in Connector Configurator.

Although these properties are adapter-specific, here are some example values:

- MimeType = `text\xml`
- DHClass = `com.crossworlds.DataHandlers.text.xml`
- DataHandlerConfigMOName = `MO_DataHandler_Default`

The fields for these values in the **Data Handler** tab are displayed only if you have set the ContainerManagedEvents property to the value `JMS`.

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

The ContainerManagedEvents property is valid only if the value of the DeliveryTransport property is set to `JMS`.

There is no default value.

ControllerEventSequencing

The ControllerEventSequencing property enables event sequencing in the connector controller.

This property is valid only if the value of the RepositoryDirectory property is set to set to <REMOTE> (`BrokerType` is `ICS`).

The default value is `true`.

ControllerStoreAndForwardMode

The ControllerStoreAndForwardMode property 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.

This property is valid only if the value of the RepositoryDirectory property is set to <REMOTE> (the value of the BrokerType property is ICS).

The default value is `true`.

ControllerTraceLevel

The ControllerTraceLevel property sets the level of trace messages for the connector controller.

This property is valid only if the value of the RepositoryDirectory property is set to set to <REMOTE>.

The default value is `0`.

DeliveryQueue

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

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

The default value is <*CONNECTORNAME*>/DELIVERYQUEUE.

DeliveryTransport

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

- If the value of the RepositoryDirectory property is set to <REMOTE>, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.
- If the value of the RepositoryDirectory property is a local directory, the value can be only JMS.

The connector sends service-call requests and administrative messages over CORBA IIOP if the value of the RepositoryDirectory property is MQ or IDL.

If the value of the DeliveryTransport property is MQ, you can set the command-line parameter WhenServerAbsent in the adapter start script to indicate whether the adapter should pause or shut down when the InterChange Server is shut down.

- Enter WhenServerAbsent=pause to pause the adapter when ICS is not available.
- Enter WhenServerAbsent=shutdown to shut down the adapter when ICS is not available.

WebSphere MQ and IDL

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

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

JMS

The JMS transport mechanism 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

are listed in Connector Configurator. The properties `jms.MessageBrokerName` and `jms.FactoryClassName` are required for this transport.

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

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

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

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

  This script is located in the \bin directory below the product directory (<*ProductDir*>). Using a text editor, add the following line as the first line in the CWSharedEnv.sh script:

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

  This line restricts heap memory usage to a maximum of 768 MB (3 segments * 256 MB). If the process memory grows larger than this limit, page swapping can occur, which can adversely affect the performance of your system.
- Set the value of the IPCCBaseAddress property to 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

DuplicateEventElimination

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

Note: When the value of this property is `true`, the MonitorQueue property must be enabled to provide guaranteed event delivery.

The default value is `false`.

EnableOidForFlowMonitoring

When the value of this property is `true`, the adapter runtime will mark the incoming ObjectEventID as a foreign key for flow monitoring.

This property is only valid if the BrokerType property is set to ICS.

The default value is `false`.

FaultQueue

If the connector experiences an error while processing a message, it moves the message (and a status indicator and description of the problem) to the queue specified in the FaultQueue property.

The default value is *<CONNECTORNAME>*`/FAULTQUEUE`.

jms.FactoryClassName

The jms.FactoryClassName property specifies the class name to instantiate for a JMS provider. This property must be set if the value of the DeliveryTransport property is `JMS`.

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

jms.ListenerConcurrency

The jms.ListenerConcurrency property specifies the number of concurrent listeners for the JMS controller. It specifies the number of threads that fetch and process messages concurrently within a controller.

This property is valid only if the value of the jms.OptimizedTransport property is `true`.

The default value is `1`.

jms.MessageBrokerName

The jms.MessageBrokerName specifies the broker name to use for the JMS provider. You must set this connector property if you specify `JMS` as the delivery transport mechanism (in the DeliveryTransport property).

When you connect to a remote message broker, this property requires the following values:
`QueueMgrName``:`*`Channel`*`:`*`HostName`*`:`*`PortNumber`*
where:
`QueueMgrName` is the name of the queue manager.
`Channel` is the channel used by the client.
`HostName` is the name of the machine where the queue manager is to reside.
*`PortNumber`*is the port number used by the queue manager for listening

For example:
```
jms.MessageBrokerName = WBIMB.Queue.Manager:CHANNEL1:RemoteMachine:1456
```

The default value is `crossworlds.queue.manager`. Use the default when connecting to a local message broker.

jms.NumConcurrentRequests

The jms.NumConcurrentRequests property 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 are blocked and must wait for another request to complete before proceeding.

The default value is `10`.

jms.Password

The jms.Password property specifies the password for the JMS provider. A value for this property is optional.

There is no default value.

jms.TransportOptimized

The jms.TransportOptimized property determines if the WIP (work in progress) is optimized. You must have a WebSphere MQ provider to optimize the WIP. For optimized WIP to operate, the messaging provider must be able to:

1. Read a message without taking it off the queue
2. Delete a message with a specific ID without transferring the entire message to the receiver's memory space
3. Read a message by using a specific ID (needed for recovery purposes)
4. Track the point at which events that have not been read appear.

The JMS APIs cannot be used for optimized WIP because they do not meet conditions 2 and 4 above, but the MQ Java APIs meet all four conditions, and hence are required for optimized WIP.

This property is valid only if the value of DeliveryTransport is `JMS` and the value of BrokerType is `ICS`.

The default value is `false`.

jms.UserName

the jms.UserName property specifies the user name for the JMS provider. A value for this property is optional.

There is no default value.

JvmMaxHeapSize

The JvmMaxHeapSize property specifies the maximum heap size for the agent (in megabytes).

This property is valid only if the value for the RepositoryDirectory property is set to <REMOTE>.

The default value is 128m.

JvmMaxNativeStackSize

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

This property is valid only if the value for the RepositoryDirectory property is set to <REMOTE>.

The default value is 128k.

JvmMinHeapSize

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

This property is valid only if the value for the RepositoryDirectory property is set to <REMOTE>.

The default value is 1m.

ListenerConcurrency

The ListenerConcurrency property supports multithreading in WebSphere MQ Listener when ICS is the integration broker. It enables batch writing of multiple events to the database, thereby improving system performance.

This property valid only with connectors that use MQ transport. The value of the DeliveryTransport property must be `MQ`.

The default value is 1.

Locale

The Locale property specifies the language code, country or territory, and, optionally, the associated character code set. The value of this property determines cultural conventions such 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 is a two-character language code (in lowercase letters)
TT is a two-letter country or territory code (in uppercase letters)
codeset is the name of the associated character code set (may be optional).

By default, only a subset of supported locales are listed. To add other supported values to the list, you modify the \Data\Std\stdConnProps.xml file in the <*ProductDir*>\bin directory. For more information, refer to the Connector Configurator appendix in this guide.

If the connector has not been internationalized, the only valid value for this property is `en_US`. To determine whether a specific connector has been globalized, refer to the user guide for that adapter.

The default value is `en_US`.

LogAtInterchangeEnd

The LogAtInterchangeEnd property specifies whether to log errors to the log destination of the integration broker.

Logging to the log destination also turns on e-mail notification, which generates e-mail messages for the recipient specified as the value of MESSAGE_RECIPIENT in the InterchangeSystem.cfg file when errors or fatal errors occur. For example, when a connector loses its connection to the application, if the value of LogAtInterChangeEnd is `true`, an e-mail message is sent to the specified message recipient.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is `false`.

MaxEventCapacity

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

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

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

The default value is 2147483647.

MessageFileName

The MessageFileName property specifies the name of the connector message file. The standard location for the message file is \connectors\messages in the product directory. Specify the message file name 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 connector has its own message file, see the individual adapter user guide.

The default value is `InterchangeSystem.txt`.

MonitorQueue

The MonitorQueue property specifies the logical queue that the connector uses to monitor duplicate events.

It is valid only if the value of the DeliveryTransport property is `JMS` and the value of the DuplicateEventElimination is `true`.

The default value is *<CONNECTORNAME>*`/MONITORQUEUE`

OADAutoRestartAgent

the OADAutoRestartAgent property specifies whether the connector uses the automatic and remote restart feature. This feature uses the WebSphere 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 WebSphere MQ-triggered OAD feature. see the *Installation Guide for Windows* or *for UNIX*.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is `false`.

OADMaxNumRetry

The OADMaxNumRetry property specifies the maximum number of times that the WebSphere MQ-triggered Object Activation Daemon (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.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is 1000.

OADRetryTimeInterval

The OADRetryTimeInterval property specifies the number of minutes in the retry-time interval for the WebSphere MQ-triggered Object Activation Daemon (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.

This property is valid only if the value of the RespositoryDirectory property is set to <REMOTE> (the value of BrokerType is ICS).

The default value is 10.

PollEndTime

The PollEndTime property specifies the time to stop polling the event queue. The format is *HH:MM*, where *HH* is 0 through 23 hours, and *MM* represents 0 through 59 minutes.

You must provide a valid value for this property. The default value is HH:MM without a value, and it must be changed.

If the adapter runtime detects:

- PollStartTime set and PollEndTime not set, or
- PollEndTime set and PollStartTime not set

it will poll using the value configured for the PollFrequency property.

PollFrequency

The PollFrequency property specifies the amount of time (in milliseconds) between the end of one polling action and the start of the next polling action. This is not the interval between polling actions. Rather, the logic is as follows:

- Poll to obtain the number of objects specified by the value of the PollQuantity property.
- Process these objects. For some connectors, this may be partly done on separate threads, which execute asynchronously to the next polling action.
- Delay for the interval specified by the PollFrequency property.
- Repeat the cycle.

The following values are valid for this property:

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

The default is 10000.

Important: Some connectors have restrictions on the use of this property. Where they exist, these restrictions are documented in the chapter on installing and configuring the adapter.

PollQuantity

The PollQuantity property designates the number of items from the application that the connector polls for. If the adapter has a connector-specific property for setting the poll quantity, the value set in the connector-specific property overrides the standard property value.

This property is valid only if the value of the `DeliveryTransport` property is `JMS`, and the `ContainerManagedEvents` property has a value.

An e-mail message is also considered an event. The connector actions are as follows when it is polled for e-mail.

- When it is polled once, the connector detects the body of the message, which it reads as an attachment. Since no data handler was specified for this mime type, it will then ignore the message.
- The connector processes the first BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- When it is polled for the second time, the connector processes the second BO attachment. The data handler is available for this MIME type, so it sends the business object to Visual Test Connector.
- Once it is accepted, the third BO attachment should be transmitted.

PollStartTime

The PollStartTime property specifies the time to start polling the event queue. The format is *HH:MM*, where *HH* is 0 through 23 hours, and *MM* represents 0 through 59 minutes.

You must provide a valid value for this property. The default value is `HH:MM` without a value, and it must be changed.

If the adapter runtime detects:

- PollStartTime set and PollEndTime not set, or
- PollEndTime set and PollStartTime not set

it will poll using the value configured for the PollFrequency property.

RepositoryDirectory

The RepositoryDirectory property is the location of the repository from which the connector reads the XML schema documents that store the metadata for business object definitions.

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

When the integration broker is a WebSphere message broker or WAS, this value is set to <*ProductDir*>\repository by default. However, it may be set to any valid directory name.

RequestQueue

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

This property is valid only if the value of the DeliveryTransport property is `JMS`.

The default value is <*CONNECTORNAME*>/REQUESTQUEUE.

ResponseQueue

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

This property is valid only if the value of the DeliveryTransport property is JMS.

The default value is <*CONNECTORNAME*>/RESPONSEQUEUE.

RestartRetryCount

The RestartRetryCount property specifies the number of times the connector attempts to restart itself. When this property is used for a connector that is connected in parallel, it specifies the number of times the master connector application-specific component attempts to restart the client connector application-specific component.

The default value is 7.

RestartRetryInterval

The RestartRetryInterval property specifies the interval in minutes at which the connector attempts to restart itself. When this property is used for a connector that is linked in parallel, it specifies the interval at which the master connector application-specific component attempts to restart the client connector application-specific component.

Possible values for the property range from 1 through 2147483647.

The default value is 1.

ResultsSetEnabled

The ResultsSetEnabled property enables or disables results set support when Information Integrator is active. This property can be used only if the adapter supports DB2 Information Integrator.

This property is valid only if the value of the DeliveryTransport property is JMS, and the value of BrokerType is WMQI.

The default value is false.

ResultsSetSize

The ResultsSetSize property defines the maximum number of business objects that can be returned to Information Integrator. This property can be used only if the adapter supports DB2 Information Integrator.

This property is valid only if the value of the ResultsSetEnabled property is true.

The default value is 0. This means that the size of the results set is unlimited.

RHF2MessageDomain

The RHF2MessageDomain property allows you to configure the value of the field domain name in the JMS header. When data is sent to a WebSphere message broker over JMS transport, the adapter framework writes JMS header information, with a domain name and a fixed value of `mrm`. A configurable domain name lets you track how the WebSphere message broker processes the message data.

This is an example header:

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

This property is valid only if the value of BrokerType is WMQI or WAS. Also, it is valid only if the value of the DeliveryTransport property is `JMS`, and the value of the WireFormat property is `CwXML`.

Possible values are `mrm` and `xml`. The default value is `mrm`.

SourceQueue

The SourceQueue property 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 62.

This property is valid only if the value of DeliveryTransport is `JMS`, and a value for ContainerManagedEvents is specified.

The default value is *<CONNECTORNAME>*`/SOURCEQUEUE`.

SynchronousRequestQueue

The SynchronousRequestQueue property 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 synchronous request queue and waits for a response from the broker on the synchronous response queue. The response message sent to the connector has a correlation ID that matches the ID of the original message.

This property is valid only if the value of DeliveryTransport is `JMS`.

The default value is *<CONNECTORNAME>*`/SYNCHRONOUSREQUESTQUEUE`

SynchronousRequestTimeout

The SynchronousRequestTimeout property specifies the time in milliseconds that the connector waits for a response to a synchronous request. If the response is not received within the specified time, the connector moves the original synchronous request message (and error message) to the fault queue.

This property is valid only if the value of DeliveryTransport is `JMS`.

The default value is `0`.

SynchronousResponseQueue

The SynchronousResponseQueue property delivers response messages in reply to a synchronous request from the broker to the connector framework. This queue is necessary only if the connector uses synchronous execution.

This property is valid only if the value of DeliveryTransport is `JMS`.

The default is `<CONNECTORNAME>/SYNCHRONOUSRESPONSEQUEUE`

TivoliMonitorTransactionPerformance

The TivoliMonitorTransactionPerformance property specifies whether IBM Tivoli Monitoring for Transaction Performance (ITMTP) is invoked at run time.

The default value is `false`.

WireFormat

The WireFormat property specifies the message format on the transport:

- If the value of the RepositoryDirectory property is a local directory, the value is `CwXML`.
- If the value of the RepositoryDirectory property is a remote directory, the value is `CwBO`.

WsifSynchronousRequestTimeout

The WsifSynchronousRequestTimeout property specifies the time in milliseconds that the connector waits for a response to a synchronous request. If the response is not received within the specified time, the connector moves the original synchronous request message (and an error message) to the fault queue.

This property is valid only if the value of BrokerType is `WAS`.

The default value is 0.

XMLNameSpaceFormat

The XMLNameSpaceFormat property specifies short or long namespaces in the XML format of business object definitions.

This property is valid only if the value of BrokerType is set to WMQI or WAS.

The default value is `short`.

Appendix B. Connector Configurator

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

You use Connector Configurator to:

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

The topics covered in this appendix are:

Overview of Connector Configurator

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

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

If your adapter supports DB2 Information Integrator, use the WMQI options and the DB2 II standard properties (see the Notes column in the Standard Properties appendix.)

You use Connector Configurator to:

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

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see "Running Configurator in stand-alone mode" on page 76).

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

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

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

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 77 to set up a new one.

Running connectors on UNIX

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

Some properties in the Connector Configurator use directory paths, which default to the Windows convention for directory paths. If you use the configuration file in a UNIX environment, revise the directory paths to match the UNIX convention for these paths. Select the target operating system in the toolbar drop-list so that the correct operating system rules are used for extended validation.

Starting Connector Configurator

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

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

Running Configurator in stand-alone mode

You can run Connector Configurator without running System Manager and work with connector configuration files, irrespective of your broker.

To do so:

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

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 82.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

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

To edit an existing configuration file:

- In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it. Connector Configurator opens and displays the configuration file with the integration broker type and file name at the top.
- From Connector Configurator, select **File>Open**. Select the name of the connector configuration file from a project or from the directory in which it is stored.
- 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 connector definition as the template.

- To create a new template, see "Creating a new template" on page 77.
- To use an existing file, simply modify an existing template and save it under the new name. You can find existing templates in your `\WebSphereAdapters\bin\Data\App` directory.

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 in Connector Configurator:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears.
   - Enter a name for the new template in the **Name** field below **Input a New Template Name.** You will see this name again when you open the dialog box for creating a new configuration file from a template.

- 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 appears in the **Template Preview** display.

3. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template. If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.
   - If you are planning to modify an existing template, select the name of the template from the list in the **Template Name** table below **Select the Existing Template to Modify: Find Template.**
   - This table displays the names of all currently available templates. You can also search for a template.

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
  Property Subtype
  Updated Method
  Description
- **Flags**
  Standard flags
- **Custom Flag**
  Flag

The **Property Subtype** can be selected when **Property Type** is a String. It is an optional value which provides syntax checking when you save the configuration file. The default is a blank space, and means that the property has not been subtyped.

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**, enter your values.

To create a new property value:

1. Right-click on the square to the left of the Value column heading.
2. From the pop-up menu, select **Add** to display the Property Value dialog box. Depending on the property type, the dialog box allows you to enter either a value, or both a value and a range.
3. Enter the new property value and click OK. The value appears in the **Value** panel on the right.

The **Value** panel 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 stores the information you have entered as an XML document, under `\data\app` in the `\bin` directory where you have installed Connector Configurator.

Setting pathnames

Some general rules for setting pathnames are:

- The maximum length of a filename in Windows and UNIX is 255 characters.
- In Windows, the absolute pathname must follow the format [Drive:][Directory]\filename: for example, C:\WebSphereAdapters\bin\Data\Std\StdConnProps.xml
  In UNIX the first character should be /.

- Queue names may not have leading or embedded spaces.

Creating a new configuration file

When you create a new configuration file, you must name it and select an integration broker.

You also select an operating system for extended validation on the file. The toolbar has a droplist called **Target System** that allows you to select the target operating system for extended validation of the properties. The available options are: Windows, UNIX, Other (if not Windows or UNIX), and None-no extended validation (switches off extended validation). The default on startup is Windows.

To start Connector Configurator:

- In the System Manager window, select **Connector Configurator** from the **Tools** menu. Connector Configurator opens.
- In stand-alone mode, launch Connector Configurator.

To set the operating system for extended validation of the configuration file:

- Pull down the **Target System:** droplist on the menu bar.
- Select the operating system you are running on.

Then select **File>New>Connector Configuration**. In the New Connector window, enter the name of the new connector.

You also need to select an integration broker. The broker you select determines the properties that will appear in the configuration file. To select a broker:

- In the **Integration Broker** field, select ICS, WebSphere Message Brokers or WAS connectivity.
- Complete the remaining fields in the **New Connector** window, as described later in this chapter.

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. Set the operating system for extended validation of the configuration file using the **Target System:** droplist on the menu bar (see "Creating a new configuration file" above).
2. Click **File>New>Connector Configuration**.
3. The **New Connector** dialog box appears, with the following fields:
   - **Name**

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

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

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

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

Select the template you want to use and click **OK**.

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

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

Using an existing file

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

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

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

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

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

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

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

- All files (*.*)

  Choose this option if a `*.txt` file was delivered in the adapter package for the connector, or if a definition file is available under another extension.

3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

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

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

Completing a configuration file

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

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

1. Under the **Standard Properties** tab, select the value field for the `BrokerType` property. In the drop-down menu, select the value `ICS`, `WMQI`, or `WAS`.
2. The Standard Properties tab will display the connector properties associated with the selected broker. The table shows **Property name**, **Value**, **Type**, **Subtype** (if the Type is a string), **Description**, and **Update Method**.
3. You can save the file now or complete the remaining configuration fields, as described in "Specifying supported business object definitions" on page 85..
4. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

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

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

   Before you created the configuration file, you used the **Target System** droplist that allows you to select the target operating system for extended validation of the properties.

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

   If you have elected to use the extended validation feature by selecting a value of Windows, UNIX or Other from the **Target System** droplist, the system will validate the property subtype s well as the type, and it displays a warning message if the validation fails.

Setting the configuration file properties

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

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

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

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

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

- Associated Maps
- Resources
- Messaging (where applicable)
- Security

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

Standard properties differ from connector-specific properties as follows:

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

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

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.
- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.

- The **Update Method** field is displayed for each property. It indicates whether a component or agent restart is necessary to activate changed values. You cannot configure this setting.

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.

   Note: If the property has a Type of String, it may have a subtype value in the Subtype column. This subtype is used for extended validation of the property.
3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
   - To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

To get more information on a particular standard property, left-click the entry in the Description column for that property in the Standard Properties tabbed sheet. If you have Extended Help installed, an arrow button will appear on the right. When you click on the button, a Help window will open and display details of the standard property.

Note: If the hot button does not appear, no Extended Help was found for that property.

If installed, the Extended Help files are located in <*ProductDir*>\bin\Data\Std\Help\<*RegionalSetting*>\.

Setting connector-specific configuration properties

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

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. 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.

   Note: If the property has a Type of String, you can select a subtype from the Subtype droplist. This subtype is used for extended validation of the property.
3. To encrypt a property, select the **Encrypt** box.

4. To get more information on a particular property, left-click the entry in the Description column for that property. If you have Extended Help installed, a hot button will appear. When you click on the hot button, a Help window will open and display details of the standard property.

   Note: If the hot button does not appear, no Extended Help was found for that property.
5. Choose to save or discard changes, as described for "Setting standard connector properties" on page 84.

If the Extended Help files are installed and the AdapterHelpName property is blank, Connector Configurator will point to the adapter-specific Extended Help files located in <*ProductDir*>\bin\Data\App\Help\<*RegionalSetting*>\. Otherwise, Connector Configurator will point to the adapter-specific Extended Help files located in <*ProductDir*>\bin\Data\App\Help\<*AdapterHelpName*>\<*RegionalSetting*>\. See the AdapterHelpName property described in the Standard Properties appendix.

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 Connector-specific Properties 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 Properties appendix, under "Configuration property values overview" on page 52.

Specifying supported business object definitions

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

Note: Some connectors require that certain business objects be specified as supported in order to perform event notification or additional configuration

(using meta-objects) with their applications. For more information, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

If ICS is your broker

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

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

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

To delete a business object from the supported list:

1. To select a business object field, click the number to the left of the business object.
2. From the **Edit** menu of the Connector Configurator window, click **Delete Row**. The business object is removed from the list display.
3. From the **File** menu, click **Save to Project**.

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

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

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

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

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

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

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

If a WebSphere Message Broker is your broker

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

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

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

If WAS is your broker

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

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

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

Associated maps (ICS)

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

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

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

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

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

- **Business Object Name**

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

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

- **Explicit Binding**

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

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

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

  To explicitly bind a map:

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

Resources (ICS)

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

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

Messaging (ICS)

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

Validating messaging queues

Before you can validate a messaging queue, you must:

- Make sure that WebSphere MQ Series is installed.
- Create a messaging queue with channel and port on the host machine.
- Set up a connection to the host machine.

To validate the queue, use the Validate button to the right of the Messaging Type and Host Name fields on the Messaging tab.

Security (ICS)

You can use the **Security** tab in Connector Configurator to set various privacy levels for a message. You can only use this feature when the DeliveryTransport property is set to JMS.

By default, Privacy is turned off. Check the **Privacy** box to enable it.

The **Keystore Target System Absolute Pathname** is:

- For Windows:
  `<ProductDir>\connectors\security\<connectorname>.jks`
- For UNIX:
  `opt/IBM/WebSphereAdapters/connectors/security/<connectorname>.jks`

This path and file should be on the system where you plan to start the connector, that is, the target system.

You can use the Browse button at the right only if the target system is the one currently running. It is greyed out unless **Privacy** is enabled and the **Target System** in the menu bar is set to Windows.

The **Message Privacy Level** may be set as follows for the three messages categories (All Messages, All Administrative Messages, and All Business Object Messages):

- "" is the default; used when no privacy levels for a message category have been set.
- none
  Not the same as the default: use this to deliberately set a privacy level of none for a message category.
- integrity
- privacy
- integrity_plus_privacy

The **Key Maintenance** feature lets you generate, import and export public keys for the server and adapter.

- When you select **Generate Keys**, the Generate Keys dialog box appears with the defaults for the keytool that will generate the keys.
- The keystore value defaults to the value you entered in **Keystore Target System Absolute Pathname** on the Security tab.
- When you select OK, the entries are validated, the key certificate is generated and the output is sent to the Connector Configurator log window.

Before you can import a certificate into the adapter keystore, you must export it from the server keystore. When you select **Export Adapter Public Key**, the Export Adapter Public Key dialog box appears.

- The export certificate defaults to the same value as the keystore, except that the file extension is <filename>.cer.

When you select **Import Server Public Key**, the Import Server Public Key dialog box appears.

- The import certificate defaults to <*ProductDir*>\bin\ics.cer (if the file exists on the system).
- The import Certificate Association should be the server name. If a server is registered, you can select it from the droplist.

The **Adapter Access Control** feature is enabled only when the value of DeliveryTransport is IDL. By default, the adapter logs in with the guest identity. If the **Use guest identity** box is not checked, the **Adapter Identity** and **Adapter Password** fields are enabled.

Setting trace/log file values

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

To change the logging and tracing values:

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

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

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

Data handlers

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

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

Saving your configuration file

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

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

- From System Manager, as a file with a `*.con` extension in an Integration Component Library, or
- In a directory that you specify.
- In stand-alone mode, as a file with a `*.cfg` extension in a directory folder. By default, the file is saved to `\WebSphereAdapters\bin\Data\App`.
- You can also save it to a WebSphere Application Server project if you have set one up.

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

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

Changing a configuration file

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

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

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

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

Completing the configuration

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

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

Using Connector Configurator in a globalized environment

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

Connector Configurator supports non-English characters in:

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

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

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

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

Appendix C. 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 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 WBI 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 WBI 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, open *sample_folder*\`JMSConnector.cfg`. For further information on using Connector Configurator, see Appendix B, "Connector Configurator," on page 75; for more on connector-specific properties, see "Configuring connector-specific properties" on page 20.
   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:
   -
   - `DuplicateEventElimination` Set this property to `true`.
   - `MonitorQueue` Set this property to `JMSConnector/MONITORQUEUE`
   - `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`.
   - `InputQueue` Set this property to `CWLD_Input`.

- UnsubscribedQueue Set this property to CWLD_Unsubscribed.

5. **Configure the Port Connector** Using Connector Configurator, 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, click the **Supported Business Objects** tab for the JMS adapter, add the business objects shown in Table 19 and set the **Message Set ID** to a unique value for each supported business object.

Table 19. 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, 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 20.

Table 20. 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

   UNIX:
   a. Open the file: *Product_Dir*/bin/connector_manager_JMS.
   b. Set the value of the AGENTCONFIG_FILE property to -c followed by the *<full path and filename for the JMSConnector.cfg file>* For example:
      AGENTCONFIG_FILE=-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 34.

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 37. 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:

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 WebSphere Business Integration 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
   ```

Appendix E. Common Event Infrastructure

WebSphere Business Integration Server Foundation includes the Common Event Infrastructure Server Application, which is required for Common Event Infrastructure to operate. The WebSphere Application Server Foundation can be installed on any system (it does not have to be the same machine on which the adapter is installed.)

The WebSphere Application Server Application Client includes the libraries required for interaction between the adapter and the Common Event Infrastructure Server Application. You must install WebSphere Application Server Application Client on the same system on which you install the adapter. The adapter connects to the WebSphere Application Server (within the WebSphere Business Integration Server Foundation) by means of a configurable URL.

Common Event Infrastructure support is available using any integration broker supported with this release.

Required software

In addition to the software prerequisites required for the adapter, you must have the following installed for Common Event Infrastructure to operate:

- WebSphere Business Integration Server Foundation 5.1.1
- WebSphere Application Server Application Client 5.0.2, 5.1, or 5.1.1.

  (WebSphere Application Server Application Client 5.1.1 is provided with WebSphere Business Integration Server Foundation 5.1.1.)

Note: Common Event Infrastructure is not supported on any HP-UX or Linux platform.

Enabling Common Event Infrastructure

Common Event Infrastructure functionality is enabled with the standard properties `CommonEventInfrastructure` and `CommonEventInfrastructureContextURL`, configured with Connector Configurator. By default, Common Event Infrastructure is not enabled. The `CommonEventInfrastructureContextURL` property enables you to configure the URL of the Common Event Infrastructure server.(Refer to the "Standard Properties" appendix of this document for more information.)

Obtaining Common Event Infrastructure adapter events

If Common Event Infrastructure is enabled, the adapter generates Common Event Infrastructure events that map to the following adapter events:

- Starting the adapter
- Stopping the adapter
- An application response to a timeout from the adapter agent
- Any `doVerbFor` call issued from the adapter agent
- A `gotApplEvent` call from the adapter agent

For another application (the "consumer application") to receive the Common Event Infrastructure events generated by the adapter, the application must use the

Common Event Infrastructure event catalog to determine the definitions of appropriate events and their properties. The events must be defined in the event catalog for the consumer application to be able to consume the sending application's events.

The "Common Event Infrastructure event catalog definitions" appendix of this document contains XML format metadata showing, for WebSphere Business Information adapters, the event descriptors and properties the consumer application should search for.

For more information

For more information about Common Event Infrastructure, refer to the Common Event Infrastructure information in the WebSphere Business Integration Server Foundation documentation, available at the following URL:

http://publib.boulder.ibm.com/infocenter/ws51help

For sample XML metadata showing the adapter-generated event descriptors and properties a consumer application should search for, refer to"Common Event Infrastructure event catalog definitions."

Common Event Infrastructure event catalog definitions

The Common Event Infrastructure event catalog contains event definitions that can be queried by other applications. The following are event definition samples, using XML metadata, for typical adapter events. If you are writing another application, your application can use event catalog interfaces to query against the event definition. For more information about event definitions and how to query them, refer to the Common Event Infrastructure documentation that is available from the online IBM WebSphere Server Foundation Information Center.

For WebSphere Business Integration adapters, the extended data elements that need to be defined in the event catalog are the keys of the business object. Each business object key requires an event definition. So for any given adapter, various events such as start adapter, stop adapter, timeout adapter, and any `doVerbFor` event (create, update, or delete, for example) must have a corresponding event definition in the event catalog.

The following sections contain examples of the XML metadata for start adapter, stop adapter, and event request or delivery.

XML format for "start adapter" metadata

```
<eventDefinition name="startADAPTER"
      parent="event">
    <property name ="creationTime" //Comment: example value would be
 "2004-05-13T17:00:16.319Z"
         required="true" />
    <property name="globalInstanceId" //Comment: Automatically generated
 by Common Event Infrastructure
         required="true"/>
    <property name="sequenceNumber"    //Comment: Source defined number
for messages to be sent/sorted logically
         required="false"/>
    <property name="version"    //Comment: Version of the event
         required="false"
         defaultValue="1.0.1"/>
```

```
    <property name="sourceComponentId"
         path="sourceComponentId"
         required="true"/>
    <property name="application"   //Comment: The name#version of the
source application generating the event. Example is "SampleConnector#3.0.0"
         path="sourceComponentId/application"         required="false"/>
    <property name="component"   //Comment: This will be the name#version
 of the source component.
         path="sourceComponentId/component"
         required="true"
         defaultValue="ConnectorFrameWorkVersion#4.2.2"/>
    <property name="componentIdType"    //Comment: specifies the format
and meaning of the component
          path="sourceComponentId/componentIdType"
          required="true"
          defaultValue="Application"/>
    <property name="executionEnvironment"
 //Comment: Identifies the environment the application is running
 in...example is "Windows 2000#5.0"
          path="sourceComponentId/executionEnvironment"
          required="false" />
     <property name="location"    //Comment: The value of this is the
 server name...example is "WQMI"
          path="sourceComponentId/location"
          required="true"/>
    <property name="locationType"   //Comment specifies the format and
     meaning of the location
          path="sourceComponentId/locationType"
          required="true"
          defaultValue="Hostname"/>
     <property name="subComponent"     //Comment:further distinction
of the logical component
          path="sourceComponentId/subComponent"
          required="true"
          defaultValue="AppSide_Connector.AgentBusinessObjectManager"/>
     <property name="componentType"      //Comment: well-defined name
used to characterize all instances of this component
          path="sourceComponentId/componentType"
          required="true"
          defaultValue="ADAPTER"/>
     <property name="situation"   //Comment: Defines the type of
 situation that caused the event to be reported
          path="situation"
          required="true"/>
     <property name="categoryName="    //Comment: Specifies the type
of situation for the event
         path="situation/categoryName"
         required="true"
         defaultValue="StartSituation"/>
    <property name="situationType"    //Comment: Specifies the type
of situation and disposition of the event
         path="situation/situationType"
         required="true"
    <property name="reasoningScope" //Comment: Specifies the scope
 of the impact of the event
         path="situation/situationType/reasoningScope"
         required="true"
         permittedValue="INTERNAL"
         permittedValue="EXTERNAL"/>
    <property name="successDisposition" //Comment: Specifies the
 success of event
         path="situation/situationType/successDisposition"
         required="true"
         permittedValue="SUCCESSFUL"
         permittedValue="UNSUCCESSFUL" />
    <property name="situationQualifier"  //Comment: Specifies the
 situation qualifiers for this event
```

```
        path="situation/situationType/situationQualifier"
        required="true"
        permittedValue="START_INITIATED"
        permittedValue="RESTART_INITIATED"
        permittedValue="START_COMPLETED" />
</eventDefinition>
```

XML format for ″stop adapter″ metadata

The metadata for “stop adapter” is the same as that for “start adapter” with the following exceptions:

- The default value for the `categoryName` property is `StopSituation`:

```
<property name="categoryName="
 //Comment: Specifies the type
 of situation for the event
             path="situation/categoryName"
             required="true"
             defaultValue="StopSituation"/>
```

- The permitted values for the `situationQualifier` property differ and are as follows for “stop adapter”:

```
<property name="situationQualifier"
 //Comment: Specifies the situation qualifiers for this event
          path="situation/situationType/situationQualifier"
          required="true"
          permittedValue="STOP_INITIATED"
          permittedValue="ABORT_INITIATED"
          permittedValue="PAUSE_INITIATED"
          permittedValue="STOP_COMPLETED"
 />
```

XML format for “timeout adapter” metadata

The metadata for “timeout adapter” is the same as that for “start adapter” and “stop adapter” with the following exceptions:

- The default value for the `categoryName` property is `ConnectSituation`:

```
<property name="categoryName="
 //Comment: Specifies the type
 of situation for the event
             path="situation/categoryName"
             required="true"
             defaultValue="ConnectSituation"/>
```

- The permitted values for the `situationQualifier` property differ and are as follows for “timeout adapter”:

```
<property name="situationQualifier"  //Comment: Specifies
 the situation qualifiers for this event
          path="situation/situationType/situationQualifier"
          required="true"
          permittedValue="IN_USE"
          permittedValue="FREED"
          permittedValue="CLOSED"
          permittedValue="AVAILABLE"
 />
```

XML format for ″request″ or ″delivery″ metadata

At the end of this XML format are the extended data elements. The extended data elements for adapter request and delivery events represent data from the business object being processed. This data includes the name of the business object, the key (foreign or local) for the business object, and business objects that are children of parent business objects. The children business objects are then broken down into the same data as the parent (name, key, and any children business objects). This data is represented in an extended data element of the event definition. This data will change depending on which business object, which keys, and which child business objects are being processed. The extended data in this event definition is just an example and represents a business object named `Employee` with a key `EmployeeId` and a child business object `EmployeeAddress` with a key `EmployeeId`. This pattern could continue for as much data as exists for the particular business object.

```
<eventDefinition name="createEmployee"     //Comment: This
 extension name is always the business object verb followed by the business
 object name
          parent="event">
    <property name ="creationTime"  //Comment: example value would be
"2004-05-13T17:00:16.319Z"
         required="true" />
    <property name="globalInstanceId" //Comment: Automatically generated
 by Common Event Infrastructure
         required="true"/>
    <property name="localInstanceId"    //Comment: Value is business
 object verb+business object name+#+app name+ business object identifier
         required="false"/>
    <property name="sequenceNumber"    //Comment: Source defined number
for messages to be sent/sorted logically
         required="false"/>
    <property name="version"  //Comment: Version of the event...value is
 set to 1.0.1
         required="false"
         defaultValue="1.0.1"/>
    <property name="sourceComponentId"
         path="sourceComponentId"
         required="true"/>
    <property name="application"    //Comment: The name#version of the
 source application generating the event...example is
 "SampleConnector#3.0.0"
         path="sourceComponentId/application"
         required="false"/>
    <property name="component"   //Comment: This will be the name#version
of the source component.
         path="sourceComponentId/component"
         required="true"
         defaultValue="ConnectorFrameWorkVersion#4.2.2"/>
    <property name="componentIdType"     //Comment: specifies the format
 and meaning of the component
         path="sourceComponentId/componentIdType"
         required="true"
         defaultValue="Application"/>
    <property name="executionEnvironment" //Comment: Identifies the
 environment#version the app is running in...example is "Windows 2000#5.0"
         path="sourceComponentId/executionEnvironment"
         required="false" />
    <property name="instanceId" //Comment: Value is business object
  verb+business object name+#+app name+ business object identifier
          path="sourceComponentId/instanceId"
          required="false"
    <property name="location"   //Comment: The value of this is the
server name...example is "WQMI"
          path="sourceComponentId/location"
```

```
          required="true"/>
     <property name="locationType" //Comment specifies the format and
 meaning of the location
          path="sourceComponentId/locationType"
          required="true"
          defaultValue="Hostname"/>
     <property name="subComponent"  //Comment:further distinction of the
 logical component-in this case the value is the name of the business
 object
          path="sourceComponentId/subComponent"
          required="true"/>
     <property name="componentType"      //Comment: well-defined name used
 to characterize all instances of this component
          path="sourceComponentId/componentType"
          required="true"
          defaultValue="ADAPTER"/>
     <property name="situation" //Comment: Defines the type of
situation that caused the event to be reported
          path="situation"
          required="true"/>
    <property name="categoryName"    //Comment: Specifies the type
 of situation for the event
         path="situation/categoryName"
         required="true"
         permittedValue="CreateSituation"
         permittedValue="DestroySituation"
         permittedValue="OtherSituation" />
    <property name="situationType"    //Comment: Specifies the type
of situation and disposition of the event
         path="situation/situationType"
         required="true"
    <property name="reasoningScope" //Comment: Specifies the scope
of the impact of the event
         path="situation/situationType/reasoningScope"
         required="true"
         permittedValue="INTERNAL"
         permittedValue="EXTERNAL"/>
    <property name="successDisposition" //Comment: Specifies the
 success of event
         path="situation/situationType/successDisposition"
         required="true"
         permittedValue="SUCCESSFUL"
         permittedValue="UNSUCCESSFUL" />
    <extendedDataElements name="Employee" //Comment: name of business
 object itself
            type="noValue"
            <children name="EmployeeId"
                 type="string"/>  //Comment: type is one of the
 permitted values within Common Event Infrastructure documentation
            <children name="EmployeeAddress"
                 type="noValue"/>
                    <children name="EmployeeId"
                        type="string"/>
                     -
                     -
                     -
    </extendedDataElements
</eventDefinition>
```

Appendix F. Application Response Measurement

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

Application Response Measurement instrumentation support

This adapter is compatible with the Application Response Measurement application programming interface (API), an API that allows applications to be managed for availability, service level agreements, and capacity planning. An ARM-instrumented application can participate in IBM Tivoli Monitoring for Transaction Performance, allowing collection and review of data concerning transaction metrics.

Required software

In addition to the software prerequisites required for the adapter, you must have the following installed for ARM to operate:

- WebSphere Application Server 5.0.1 (contains the IBM Tivoli Monitoring for Transaction Performance server). This does not have to be installed on the same system as the adapter.
- IBM Tivoli Monitoring for Transaction Performance v. 5.2 Fixpack 1. This must be installed on the same system on which the adapter is installed and configured to point to the system on which the IBM Tivoli Monitoring for Transaction Performance server resides.

Application Response Measurement support is available using any integration broker supported with this release.

Note: Application Response Measurement instrumentation is supported on all operating systems supported with this IBM WebSphere Business Integration Adapters release *except* HP-UX (any version) and Red Hat Linux 3.0.

Enabling Application Response Measurement

ARM instrumentation is enabled via by setting the standard property `TivoliMonitorTransactionPerformance` in Connector Configurator to "True." By default ARM support is not enabled. (Refer to the ″Standard Properties″ appendix of this document for more information.)

Transaction monitoring

When ARM is enabled, the transactions that are monitored are service events and event deliveries. The transaction is measured from the start of a service request or event delivery to the end of the service request or event delivery. The name of the transaction displayed on the Tivoli Monitoring for Transaction Performance console will start with either `SERVICE REQUEST` or `EVENT DELIVERY`. The next part of the name will be the business object verb (such as `CREATE`, `RETRIEVE`, `UPDATE` or `DELETE`). The final part of the name will be the business object name such as "`EMPLOYEE`."

For example, the name of a transaction for an event delivery for creation of an employee might be `EVENT DELIVERY CREATE EMPLOYEE`. Another might be `SERVICE REQUEST UPDATE ORDER`.

The following metrics are collected by default for each type of service request or event delivery:

- Minimum transaction time
- Maximum transaction time
- Average transaction time
- Total transaction runs

You (or the system administrator of the WebSphere Application Server) can select which of these metrics to display, for which adapter events, by configuring Discovery Policies and Listener Policies for particular transactions from within the Tivoli Monitoring for Transaction Performance console. (Refer to "For more information.")

For more information

Refer to the IBM Tivoli Monitoring for Transaction Performance documentation for more information. In particular, refer to the *IBM Tivoli Monitoring for Transaction Performance User's Guide* for information about monitoring and managing the metrics generated by the adapter.

Index

P

Q

R

S

T

U

V

W

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other 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 grant 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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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 the 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 Corporation
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 measurements 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 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.

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

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

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:

i5/OS
IBM
the IBM logo
AIX
AIX 5L
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
HelpNow
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
Notes
OS/400
Passport Advantage
pSeries
Redbooks
SupportPac
WebSphere
z/OS

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

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

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

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

WebSphere Business Integration Adapters, Version 6.0

WebSphere Business Integration Adapter Framework V2.6.0

Printed in USA