IBM WebSphere Business Integration Adapters

Adapter for WebSphere MQ User Guide

Adapter Version 2.7.x

IBM WebSphere Business Integration Adapters

Adapter for WebSphere MQ User Guide

Adapter Version 2.7.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 107.

30September2004

This edition of this document applies to IBM WebSphere Business Integration Adapter for WebSphere MQ (5724-H06), version 2.7.x.

To send us your comments about 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, legacy applications, 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 WebSphere MQ.

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 each 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 WebSphere MQ 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 Business Integration Adapters installations, and includes reference material on specific components.

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

You can install related documentation from the following sites:

- For general adapter information; for using adapters with WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker); and for using adapters with WebSphere Application Server:

  http://www.ibm.com/websphere/integration/wbiadapters/infocenter
- For using adapters with InterChange Server:

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

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

- For more information about message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker):

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

- For more information about WebSphere Application Server:

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

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

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

New in this release

New in release 2.7.x

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

This release supports the processing of bidirectional script data. For more information, see "Locale-dependent data" on page 3.

This release adds support for the following platforms or updates:

- Microsoft Windows 2000 (Professional, Server, or Advanced Server) with Service Pack 4
- Microsoft Windows 2003 (Standard Edition or Enterprise Edition)
- Solaris 8 (2.8) with Solaris Patch Cluster dated Feb. 11, 2004 or later
- Solaris 9 (2.9) with Solaris Patch Cluster dated Feb. 11, 2004 or later. This adapter supports 32-bit JVM on a 64-bit platform
- AIX 5.1 with Maintenance Level 4
- AIX 5.2 with Maintenance Level 1.
  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
- Red Hat Enterprise Linux AS 3.0 with Update 1, ES 3.0 with Update 1, or WS 3.0 with Update 1
- SUSE Linux Enterprise Server x86 8.1 with SP3
- SUSE Linux Standard Server x86 8.1 with SP3
- All operating system environments require the Java compiler (IBM JDK 1.4.2 for Windows 2000) for compiling custom adapters

The connector-specific property ReplyToQueuePollFrequency has been documented.

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

A limitation of 16 characters for the name length of the OutputFormat meta-object property has been documented.

New in release 2.6.x

Two connector-specific properties have been added: EnableMessageProducerCache and SessionPoolSizeForRequests. For further information, see "Connector-specific properties" on page 21.

When processing a response message to a synchronous request, the connector interprets the feedback code MQFB_NONE (the default feedback code if none is set) asVALCHANGE. For further information, see "Synchronous delivery" on page 10.

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.x

The connector now runs on the following platforms:

- Microsoft(R) Windows 2000
- Solaris 7, 8 or AIX 5.1, 5.2 or HP UX 11.i

Beginning with the 2.5.0 version, the adapter for WebSphere MQ is no longer supported on Microsoft Windows NT.

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

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 2.

The connector now runs on the following platforms:

- Microsoft Windows NT(R) 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"(R) 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 "Overview of mapping data handlers to input queues" on page 34.

The guaranteed event delivery feature has been enhanced. For further information, see "Guaranteed event delivery" on page 9.

New in release 2.2.x

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

The connector supports interoperability with applications via WebSphere MQ 5.1, 5.2, and 5.3. For more information, see "Adapter dependencies" on page 3.

The connector now has a UseDefaults property for business object processing. For more information, see "UseDefaults" on page 26..

The connector can now apply a default verb when the data handler does not explicitly assign one to a business object. For more information, see "DefaultVerb" on page 23.

The ReplyToQueue can now be dictated via the dynamic child meta-object rather than by the ReplyToQueue connector property. For more information see “JMS headers and dynamic child meta-object attributes” on page 36..

You can use a message selector to identify, filter and otherwise control how the adapter identifies the response message for a given request. This JMS capability applies to synchronous request processing only. For more information, see “Synchronous delivery” on page 10..

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 InterChange Server.

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

New in release 1.5.x

The IBM WebSphere Business Integration Adapter for WebSphere MQ includes the connector for WebSphere MQ. This adapter operates with the WebSphere 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 WebSphere MQ
- 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 InterChange Server.

Important: Because the connector has not been internationalized, do not run it against InterChange Server 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 9.

New in release 1.4.x

This guide describes the following new or changed features in version 1.4.x of the connector:

- In previous versions of the connector for WebSphere MQ, the data handler that was used to convert data between WebSphere MQ messages and 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 1.4.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 "Meta-objects configuration" on page 29.

- The `DataEncoding` meta-object attribute (static or dynamic) has been added to allow users to specify message type (Text or Binary) and encoding type. For details, see "Meta-objects configuration" on page 29.

New in release 1.3.x

This guide describes the following new or changed features in version 1.3.x of the connector:

- Support for synchronous requests issued to the connector by external applications. See "Synchronous event handling" on page 7 for details.
- The `CollaborationName` property has been added to the meta-objects; see "Meta-objects configuration" on page 29.
- The `DoNotReportBusObj` property has been added to the meta-objects; see "Meta-objects configuration" on page 29.

Chapter 1. Overview

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

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

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

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

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

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

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

Adapter for WebSphere MQ environment

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

Broker compatibility

This adapter runs with the WebSphere Business Integration Adapter Framework V2.6 and requires one of the following brokers:

- WebSphere InterChange Server, version 4.2.2 or 4.3.
- WebSphere Application Server Enterprise, version 5.0.2 with WebSphere Studio Application Developer Integration Edition, version 5.0.1.
- WebSphere Business Intergration Server Foundation, version 5.1.1

See the Release Notes for any exceptions.

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

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

For message brokers (WebSphere MQ Integrator Broker, WebSphere MQ Integrator, and WebSphere Business Integration Message Broker), see *Implementing Adapters with WebSphere Message Brokers*, and the installation documentation for the message broker. Some of this can be found at the following Web site: http://www.ibm.com/software/integration/mqfamily/library/manualsa/. For WebSphere Application Server, see Implementing Adapters with WebSphere Application Server and the documentation at: http://www.ibm.com/software/webservers/appserv/library.html

Adapter platforms

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

- Microsoft Windows 2000 (Professional, Server, or Advanced Server) with Service Pack 4
- Microsoft Windows 2003 (Standard Edition or Enterprise Edition)
- Solaris 8 (2.8) with Solaris Patch Cluster dated Feb. 11, 2004 or later
- Solaris 9 (2.9) with Solaris Patch Cluster dated Feb. 11, 2004 or later. This adapter supports 32-bit JVM on a 64-bit platform
- AIX 5.1 with Maintenance Level 4
- AIX 5.2 with Maintenance Level 1.
  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
- Red Hat Enterprise Linux AS 3.0 with Update 1, ES 3.0 with Update 1, or WS 3.0 with Update 1
- SUSE Linux Enterprise Server x86 8.1 with SP3
- SUSE Linux Standard Server x86 8.1 with SP3
- All operating system environments require the Java compiler (IBM JDK 1.4.2 for Windows 2000) for compiling custom adapters

Note: The Tivoli Monitoring for Transaction Performance (TMTP) component of the WebSphere Business Integration Adapter Framework V2.6 is not supported on Red Hat Linux.

Adapter dependencies

The adapter has the following software prerequisites and other dependencies:

- The connector supports interoperability with applications via WebSphere MQ 5.1, 5.2,[1] and 5.3. Accordingly, you must have one of these software versions installed.
- In addition, you must have the IBM WebSphere MQ Java client libraries.

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

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 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.

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.

Common Event Infrastructure

This adapter is compatible with Common Event Infrastructure from IBM, 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, see the Application Response Management appendix in this guide.

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

Application Response Measurement

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

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

Connector architecture

The connector is metadata-driven. Message routing and format conversion are initiated by an event polling technique. The connector uses an IBM WebSphere MQ implementation of the Java™ Message Service (JMS), an API for accessing enterprise-messaging systems that also makes possible guaranteed event delivery.

The connector allows IBM WebSphere Business Integration Collaborations to asynchronously exchange business objects with applications that issue or receive WebSphere MQ messages when changes to data occur.

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

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

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

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

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

When a message is retrieved from an input queue, the connector looks up the business object name associated with that input queue and with the `FORMAT` field contained in the message header. The message body, along with a new instance of

the appropriate business object, is then passed to the data handler. If a business object name that is associated with the input queue and format is not found, the message body alone is passed to the data handler. If a business object is successfully populated with message content, the connector checks to see if it is subscribed, and then delivers it to InterChange Server using the `gotApplEvents()` method.

Application-connector communication method

The connector makes use of IBM's WebSphere MQ implementation of the Java Message Service (JMS). The JMS is an open-standard API for accessing enterprise-messaging systems. It is designed to allow business applications to asynchronously send and receive business data and events.

Message request

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

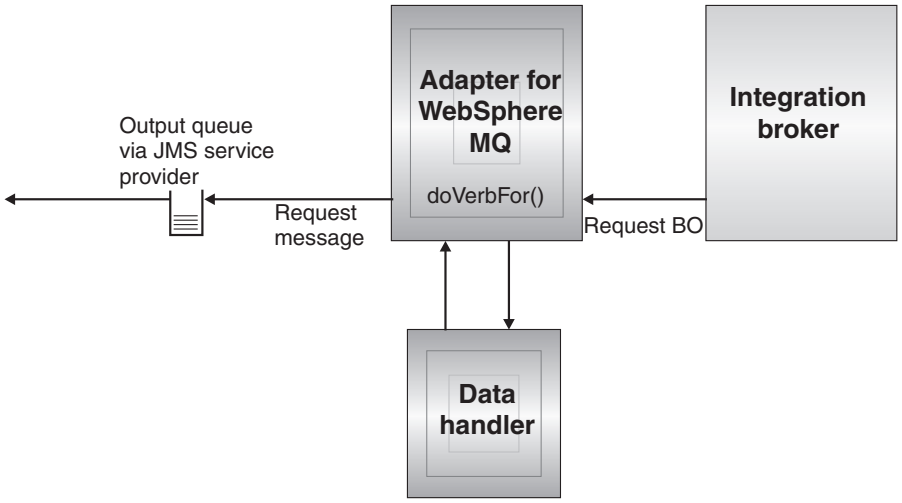

Figure 1. Application-connector communication method: Message request

Event delivery

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

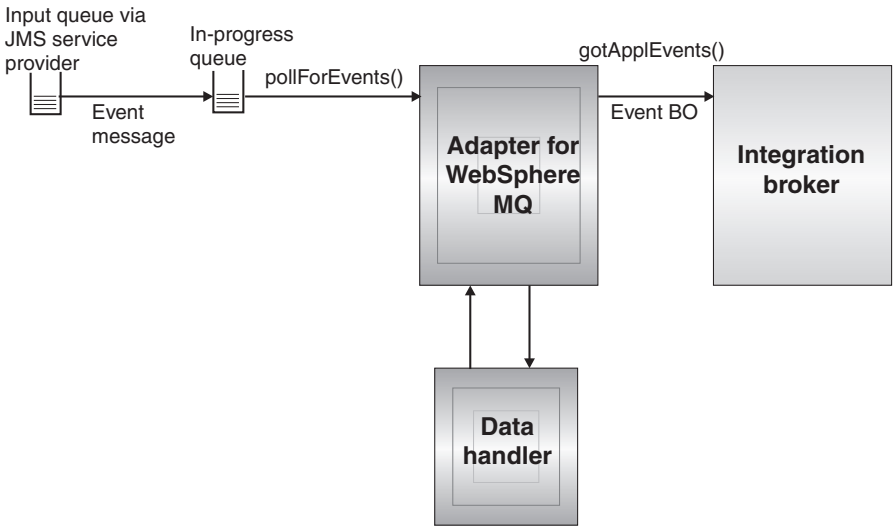

Figure 2. Application-connector communication method: Event delivery

Event handling

For event notification, the connector detects events written to a queue by an application rather than a database trigger. An event occurs when an application or other MQ-capable software generates WebSphere MQ messages and stores them on the MQ message queue.

Retrieval

The connector uses the `pollForEvents()` method to poll the MQ queue at regular intervals for messages. When the connector finds a message, it retrieves it from the MQ queue and examines it to determine its format. If the format has been defined in the connector's static object, the connector passes both the message body and a new instance of the business object associated with the format to the configured data handler; the data handler is expected to populate the business object and specify a verb. If the format is not defined in the static meta-object, the connector passes only the message body to the data handler; the data handler is expected to determine, create and populate the correct business object for the message. See "Error handling" on page 46 for event failure scenarios.

The connector processes messages by first opening a transactional session to the input queue. This transactional approach allows for a small chance that a business object could be delivered to a collaboration twice due to the connector successfully submitting the business object but failing to commit the transaction in the queue. To avoid this problem, the connector moves all messages to an in-progress queue. There, the message is held until processing is complete. If the connector shuts down unexpectedly during processing, the message remains in the in-progress queue instead of being reinstated to the original input queue.

Note: Transactional sessions with a JMS service provider require that every requested action on a queue be performed and committed before events are removed from the queue. Accordingly, when the connector retrieves a message from the queue, it does not commit to the retrieval until three things occur: 1) The message has been converted to a business object; 2) the

business object is delivered to InterChange Server by the gotApplEvents() method, and 3) a return value is received.

Synchronous event handling

Optionally, to support applications that want feedback on the requests they issue through WebSphere MQ, the connector for WebSphere MQ will issue report messages back to the applications detailing the outcome of their requests once they have been processed.

To achieve this, the connector will post the business data for such requests synchronously to InterChange Server. If a collaboration successfully processes the business object, the connector will send a report back to the requestor including the return code from InterChange Server and any business object changes. If the connector or the collaboration fails to process the business object, the connector will send a report containing the appropriate error code and error message.

In either case, an application that sends a request to the connector for WebSphere MQ will be notified of its outcome.

Processing: If the connector for WebSphere MQ receives any messages requesting positive or negative acknowledgement reports (PAN or NAN), it will post the content of the message synchronously to InterChange Server and then incorporate the return code and modified business data in to a report message that will be sent back to the requesting application.

The table below shows the required structure of WebSphere MQ messages sent to the connector to be processed synchronously.

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

Upon receipt of a message as described in the table above, the connector will do the following:

1. Reconstruct the business object in the message body using the configured data handler.
2. Look up the collaboration name specified for the business object and verb in the static metadata object (but not in the dynamic child meta-object, which cannot convey the collaboration name).
3. Post the business object synchronously to the specified collaboration.
4. Generate a report encapsulating the result of the processing and any business object changes or error messages.
5. Send the report to the queue specified in the `replyToQueue` and `replyToQueueManager` fields of the request.

The table below shows the structure of the report that is sent back to the requestor from the connector.

MQMD field	Description	Supported values (multiple values should be OR'd)
`MessageType`	Message Type	`REPORT`
`feedback`	Type of report	One of the following: • `MQRO_PAN` If the collaboration successfully processed the business object. • `MQRO_NAN` If the connector or the collaboration encountered an error while processing the request.
Message Body		If the collaboration successfully processed the business object, the connector will populate the message body with the business object returned by the collaboration. This default behavior can be overridden by setting the `DoNotReportBusObj` property to `true` in the static metadata object. If the request could not be processed, the connector will populate the message body with the error message generated by the connector or the collaboration.

Recovery

Upon initialization, the connector checks the in-progress queue for messages that have not been completely processed, presumably due to a connector shutdown. The connector configuration property `InDoubtEvents` allows you to specify one of four options for handling recovery of such messages: fail on startup, reprocess, ignore, or log error.

Fail on startup

With the fail on startup option, if the connector finds messages in the in-progress queue during initialization, it logs an error and immediately shuts down. It is the responsibility of the user or system administrator to examine the message and take appropriate action, either to delete these messages entirely or move them to a different queue.

Reprocess

With the reprocessing option, if the connector finds any messages in the in-progress queue during initialization, it processes these messages first during subsequent polls. When all messages in the in-progress queue have been processed, the connector begins processing messages from the input queue.

Ignore

With the ignore option, if the connector finds any messages in the in-progress queue during initialization, the connector ignores them, but does not shut down.

Log error

With the log error option, if the connector finds any messages in the in-progress queue during initialization, it logs an error but does not shut down.

Archiving

If the connector property `ArchiveQueue` is specified and identifies a valid queue, the connector places copies of all successfully processed messages in the archive queue. If `ArchiveQueue` is undefined, messages are discarded after processing. For more information on archiving unsubscribed or erroneous messages, see "Error handling" on page 46.

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

Guaranteed event delivery

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

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

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

If connector framework cannot deliver the business object to the WebSphere InterChange Server integration broker, then the object is placed on a FaultQueue (instead of UnsubscribedQueue and ErrorQueue) and generates a status indicator and a description of the problem. FaultQueue messages are written in MQRFH2 format. For further information, see "Event processing" on page 49.

Business object requests

Business object requests are processed when InterChange Server sends a business object to the `doVerbFor()` method. Using the configured data handler, the connector converts the business object to an WebSphere MQ message and issues it. There are no requirements regarding the type of business objects processed except those of the data handler.

Verb processing

The connector processes business objects passed to it by a collaboration based on the verb for each business object. The connector uses business object handlers and the `doForVerb()` method to process the business objects that the connector supports. The connector supports the following business object verbs:

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

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

Create, update, and delete

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

Asynchronous delivery

This is the default delivery mode for business objects with Create, Update, and Delete verbs. A message is created from the business object using a data handler and then written to the output queue. If the message is delivered, the connector returns SUCCESS, else FAIL.

Note: The connector has no way of verifying whether the message is received or if action has been taken.

Synchronous delivery

If a `ReplyToQueue` has been defined in the connector-specific properties and a `responseTimeout` exists in the conversion properties for the business object, the

connector issues a request in synchronous mode. The connector then waits for a response to verify that appropriate action was taken by the receiving application.

For WebSphere MQ, the connector initially issues a message with a header as shown in the table below.

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

* Indicates constant defined by IBM.

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

The Report field is set to indicate that both positive and negative action reports are expected from the receiving application. The thread that issued the message waits for a response message that indicates whether the receiving application was able to process the request.

When an application receives a synchronous request from the connector, it processes the business object and issues a report message as described in the tables below.

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

*Indicates constant defined by IBM.

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

WebSphere MQ feedback code	Equivalent response*
MQFB_NONE (this is the default if no feedback code is specified)	VALCHANGE
MQFB_PAN orMQFB_APPL_FIRST	SUCCESS
MQFB_NAN orMQFB_APPL_FIRST + 1	FAIL
MQFB_APPL_FIRST + 2	VALCHANGE
MQFB_APPL_FIRST + 3	VALDUPES
MQFB_APPL_FIRST + 4	MULTIPLE_HITS
MQFB_APPL_FIRST + 5	FAIL_RETRIEVE_BY_CONTENT
MQFB_APPL_FIRST + 6	BO_DOES_NOT_EXIST
MQFB_APPL_FIRST + 7	UNABLE_TO_LOGIN
MQFB_APPL_FIRST + 8	APP_RESPONSE_TIMEOUT (results in immediate termination of connector agent)

*See the *Connector Development Guide for Java* for details.

If the business object can be processed, the application creates a report message with the feedback field set to MQFB_PAN (or a specific WebSphere business integration system value). Optionally the application populates the message body with a serialized business object containing any changes. If the business object cannot be processed, the application creates a report message with the feedback field set to MQFB_NAN (or a specific WebSphere business integration system value) and then optionally includes an error message in the message body. In either case, the application sets the correlationID field of the message to the messageID of the connector message and issues it to the queue specified by the replyTo field.

Upon retrieval of a response message, the connector by default matches the correlationID of the response to the messageID of a request message. The connector then notifies the thread that issued the request. Depending on the feedback field of the response, the connector either expects a business object or an error message in the message body. If the feedback code is MQFB_NONE (the default value if no feedback code is specified), the connector by default considers the return code to be VALCHANGE. The connector then passes the response message to the data handler and updates the business object. If a business object was expected but the message body is not populated, the connector simply returns the same business object that was originally issued by InterChange Server for the Request operation. If an error message was expected but the message body is not populated, a generic error message will be returned to InterChange Server along with the response code. However, you can also use a message selector to identify, filter and otherwise control how the adapter identifies the response message for a given request. This message selector capability is a JMS feature. It applies to synchronous request processing only and is described below.

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

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

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

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

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

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

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

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

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

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

to identify the response message.

You can also specify multiple substitutions such as the following:

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

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

An error is reported at run-time when any of the following occurs:

- If you specify a non-integer value between the `'{}'` symbols
- If you specify an index for which no attribute is defined
- If the attribute specified does not exist in the business or meta-object
- If the syntax of the attribute path is incorrect

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

The connector would resolve this message selector as follows:

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

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

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

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

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

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

Bidirectional Support: For the Adapter for WebSphere MQ, support is provided for bidirectional transformation of business object content. Bidirectional support is also provided when the connector exchanges metadata with an external WebSphere MQ application and uses a synchronous delivery filtering mechanism to locate a response message in the ReplyToQueue. Under these circumstances, bidirectional transformation is applied so that the Windows standard format is transformed to the bidirectional format of the WebSphere MQ application. This is done using the attributes set for the BiDi.Application standard configuration property.

When the response_selector is specified with the following syntax,

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

and when bidirectional characters appear in the '*selectorstring*', then the adapter can be configured to transform the '*selectorstring*' to the bidirectional format of the external application, which is the application responsible for sending response messages for the request created by the adapter. This would be necessary when the bidirectional format of the external application is different from the default Windows format of the application-specific information as defined originally in the Connector Designer application. In such cases, the BiDi.Application standard configuration property needs to be configured with the bidirectional format of the external application; then the adapter will transform the message selector string before it is used for filtering response messages.

Creating custom feedback codes: You can extend the WebSphere MQ feedback codes to override default interpretations by specifying the connector property FeedbackCodeMappingMO. This property allows you to create a meta-object in which all WebSphere business integration system-specific return status values are mapped to the WebSphere MQ feedback codes. The return status assigned (using the meta-object) to a feedback code is passed to InterChange Server. For more information, see "FeedbackCodeMappingMO" on page 24.

Retrieve, exists and retrieve by content

Business objects with the Retrieve, Exists, and Retrieve By Content verbs support synchronous delivery only. The connector processes business objects with these

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

The table below shows the response messages for these verbs.

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

Common configuration tasks

After installation, you must configure the connector before starting it. This section provides an overview of some of the configuration and startup tasks that most developers will need to perform.

Installing the adapter

See Chapter 2, "Installing and configuring the adapter," on page 19 for a description of what and where you must install.

Configuring connector properties

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

When you configure connector properties for the adapter for WebSphere MQ, make sure that:

- The value specified for connector property HostName matches that of the host of your WebSphere MQ server.
- The value specified for connector property Port matches that of the port for the listener of your queue manager
- The value specified for connector property Channel matches the server connection channel for your queue manager.
- The queue URI's for connector properties InputQueue, InProgressQueue, ArchiveQueue, ErrorQueue, and UnsubscribeQueue are valid and actually exist.

Configuring the connector to send requests without notification

To configure the connector to send requests without notification (the default asynchronous mode, also known as "fire and forget"):

- Create a business object that represents the request you want to send and is also compatible with the data handler that you have configured for the connector.

- Use either a static or a dynamic meta-object to specify the target queue and format. For more on static and dynamic meta-objects, see “Overview of creating static meta-objects” on page 33 and “Overview of creating dynamic child meta-objects” on page 35.
- Set the property ResponseTimeout in the (static or dynamic) meta-object to -1. This forces the connector to issue the business object without checking for a return.
- For more information, see “Create, update, and delete” on page 10, “Meta-objects configuration” on page 29, and Chapter 3, “Creating or modifying business objects,” on page 43

Configuring the connector to send requests and get notifications

To configure the connector to send requests and get notifications (synchronous event handling):

- Follow the steps described in “Configuring the connector to send requests without notification” on page 15 with this exception: you specify a positive ResponseTimeout value to indicate how long the connector waits for a reply.
- See “Create, update, and delete” on page 10 for details of exactly what the connector expects in a response message. If the requirements listed are not met by the response message, the connector may report errors or fail to recognize the response message. See also sections on “Meta-objects configuration” on page 29, and Chapter 3, “Creating or modifying business objects,” on page 43.

Configuring a static meta-object

A static meta-object contains application-specific information that you specify about business objects and how the connector processes them. A static meta-object provides the connector with all the information it needs to process a business object when the connector is started.

If you know at implementation time which queues that different business objects must be sent to, use a static meta-object. To create and configure this object:

- Follow the steps in “Overview of creating static meta-objects” on page 33.
- Make sure the connector subscribes to the static meta-object by specifying the name of the static meta-object in the connector-specific property ConfigurationMetaObject. For more information, see “Connector-specific properties” on page 21.

Configuring a dynamic meta-object

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

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

The connector will check for the dynamic meta-object and use its information to determine how to process the business object. For more information, see "Overview of creating dynamic child meta-objects" on page 35.

Configuring MQMD formats

MQMDs are message descriptors. MQMDs contain the control information accompanying application data when a message travels from one application to another. You must specify a value for the MQMD attribute OutputFormat in either your static or dynamic meta-object. For more information, see "Create, update, and delete" on page 10.

Configuring queue URIs

To configure queues for use with the Adapter for WebSphere MQ:

- Specify all queues as Uniform Resource Identifiers (URIs). The syntax is:
  `queue://<queue manager name>/<actual queue>`
- Specify the host for the queue manager in connector-specific configuration properties.
- If your target application expects an MQMD header only and cannot process the extended MQRFH2 headers used by JMS clients, append `?targetClient=1` to the queue URI. For more information, see "Queue Uniform Resource Identifiers (URI)" on page 28 and the WebSphere MQ programming guide.

Configuring data handlers

There are two ways to configure a data handler:

- Specify the data handler class name in the connector-specific property DataHandlerClassName. For more information, see "Connector-specific properties" on page 21.
- Or specify both a mime type and the data handler meta-object that defines the configuration for that mime type in the connector-specific properties DataHandlerMimeType and DataHandlerConfigMO, respectively. For more information, see the *Data Handler Guide*.

Modifying the startup script

See Chapter 2, "Installing and configuring the adapter," on page 19 for a description of how to start the connectors. You must configure connector properties before startup. You must also modify the startup file:

- Make sure you modify the `start_connector` script to point to the location of the client libraries. Do not install multiple versions of the client libraries or versions that are not up-to-date with your WebSphere MQ server. For more information, see "Startup file configuration" on page 39.

Chapter 2. Installing and configuring the adapter

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

Overview of installation tasks

To install the connector for WebSphere MQ, 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.

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

Windows file structure

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

The utility installs the connector into the *ProductDir*\connectors\WebSphereMQ directory, and adds a shortcut for the connector to the Start menu.

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

Subdirectory of *ProductDir*	Description
connectors\WebSphereMQ\CWWebSphereMQ.jar	Contains classes used by the WebSphere MQ connector only
connectors\WebSphereMQ\start_WebSphereMQ.bat	The startup script for the connector (NT/2000)
connectors\messages\WebSphereMQConnector.txt	Message file for the connector
bin\Data\App\WebSphereMQConnectorTemplate	Template file for adapter definition
connectors\WebSphereMQ\samples\LegacyContact\WebSphereMQConnector.cfg	Sample WebSphere MQ configuration file
connectors\WebSphereMQ\samples\LegacyContact\PortConnector.cfg	Sample Port connector configuration file
connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _LegacyContact.xsd	Sample schema
connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _MO_Config.xsd	Sample meta-object
connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _MO_DataHandler.xsd	Sample data handler meta-object
connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _MO_DataHandler_DelimitedConfig.xsd	Sample delmited data handler meta-object
connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ _DynMO_Config.xsd	Sample dynamic meta-object
connectors\WebSphereMQ\samples\LegacyContact\JMSPropertyPairs.xsd	Sample JMS properties 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 file structure

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

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

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

Subdirectory of *ProductDir*	Description
connectors/WebSphereMQ/CWWebSphere MQ.jar	Contains classes used by the WebSphere MQ connector only
connectors/WebSphereMQ/start_WebSphereMQ.sh	System startup script for the connector. This script is called from the generic connector manager script. When you click the Connector Configuration screen of System Manager, the installer creates a customized wrapper for this connector manager script. Use this customized wrapper to start and stop the connector.
connectors/messages/WebSphereMQ/Connector.txt	Message file for the connector
bin/Data/App/WebSphereMQConnectorTemplate	Template file for the adapter definition
connectors/WebSphereMQ/samples/LegacyContact/WebSphereMQConnector.cfg	Sample WebSphere MQ configuration file
connectors/WebSphereMQ/samples/LegacyContact/PortConnector.cfg	Sample Port connector configuration file
connectors/WebSphereMQ/samples/LegacyContact/Sample_WebSphereMQ _LegacyContact.xsd	Sample business object repository file
connectors/WebSphereMQ/samples/LegacyContact/Sample_WebSphereMQ _MO_Config.xsd	Sample meta-object

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

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

You use Connector Configurator to configure connector properties:

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

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

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 51 for documentation of these properties.

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

Connector-specific properties

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

The table below lists the connector-specific configuration properties for the adapter. See the sections that follow for explanations of the properties.

Name	Possible values	Default value	Required
ApplicationPassword	*Login password*		No
ApplicationUserName	*Login user ID*		No
ArchiveQueue	*Queue to which copies of successfully processed messages are sent*	`queue://crossworlds.queue.manager/MQCONN.ARCHIVE`	No
CCSID	*Character set for queue manager connection*		No
Channel	*MQ server connector channel*		Yes
ConfigurationMetaObject	*Name of configuration meta-object*		Yes
DataHandlerClassName	*Data handler class name*	`com.crossworlds.DataHandlers.text.xml`	No
DataHandlerConfigMO	*Data handler meta-object*	`MO_DataHandler_ Default`	Yes
DataHandlerMimeType	*MIME type of file*	`text/xml`	No
DefaultVerb	*Any verb supported by the connector.*		No
EnableMessageProducerCache	true or false	`true`	No
ErrorQueue	*Queue for unprocessed messages*	`queue://crossworlds.queue.manager/MQCONN.ERROR`	No
FeedbackCodeMappingMO	Feedback code meta-object		No
HostName	*WebSphere MQ server*		Yes
InDoubtEvents	`FailOnStartup Reprocess Ignore LogError`	`Reprocess`	No
InputQueue	*Poll queues*	`queue://crossworlds.queue.manager/MQCONN.IN`	No
InProgressQueue	*In-progress event queue*	`queue://crossworlds.queue.manager/MQCONN.IN_PROGRESS`	No
PollQuantity	*Number of messages to retrieve from each queue specified in the `InputQueue` property*	1	No
Port	*Port established for the WebSphere MQ listener*		Yes
ReplyToQueue	*Queue to which response messages are delivered when the connector issues requests*	`queue://crossworlds.queue.manager/MQCONN.REPLY`	No
ReplyToQueuePollFrequency	*Polling interval in number of milliseconds for receiver during synchronous request processing*		No
SessionPoolSizeForRequests	*Maximum pool size for caching the sessions used during request processing*	10	No
UnsubscribedQueue	*Queue to which unsubscribed messages are sent*	`queue://crossworlds.queue.manager/MQCONN.UNSUBSCRIBED`	No
UseDefaults	true or false	`false`	No

ApplicationPassword

Password used with `UserID` to log in to WebSphere MQ.

Default = None.

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

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

Default = None.

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

ArchiveQueue
Queue to which copies of successfully processed messages are sent.

Default = `queue://crossworlds.queue.manager/MQCONN.ARCHIVE`

CCSID
The character set for the queue manager connection. The value of this property should match that of the CCSID property in the queue URI; see "Queue Uniform Resource Identifiers (URI)" on page 28

Default = none.

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

Default = none.

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

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

Default = none.

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

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

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

Default = `MO_DataHandler_Default`

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

Default = `text/xml`

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

Default= none

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

Default= `true`

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

Default = `queue://crossworlds.queue.manager/MQCONN.ERROR`

FeedbackCodeMappingMO
Allows you to override and reassign the default feedback codes used to synchronously acknowledge receipt of messages to InterChange Server. This property enables you to specify a meta-object in which each attribute name is understood to represent a feedback code. The corresponding value of the feedback code is the return status that is passed to InterChange Server. For a listing of the default feedback codes, see "Synchronous delivery" on page 10.. The connector accepts the following attribute values representing WebSphere MQ-specific feedback codes:

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

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

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

The table below shows a sample meta-object.

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

Default = none.

HostName
The name of the server hosting WebSphere MQ.

Default = none.

InDoubtEvents

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

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

Default = `Reprocess`.

InputQueue

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

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

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

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

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

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

InProgressQueue

Message queue where messages are held during processing. You can configure the connector to operate without this queue by using System Manager to remove the default InProgressQueue name from the connector-specific properties. Doing so prompts a warning at startup that event delivery may be compromised if the connector is shut down while are events pending.

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

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

Default =1

Port
Port established for the WebSphere MQ listener.

Default = None.

ReplyToQueue
Queue to which response messages are delivered when the connector issues requests. You can also use attributes in the child dynamic meta-object to ignore a response. For more information on the these attributes, see "JMS headers and dynamic child meta-object attributes" on page 36.

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

ReplyToQueuePollFrequency
Specifies the polling interval for the receiver during synchronous request processing. The value is the number of milliseconds.

Default = None.

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

Default = 10

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

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

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

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

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."
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*.

Queue Uniform Resource Identifiers (URI)

The URI for a queue begins with the sequence `queue://` followed by:

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

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

```
queue://crossworlds.queue.manager/MQCONN.IN?targetClient=1&priority=5
```

The table below shows property names for queue URIs.

Property name	Description	Values
`expiry`	Lifetime of the message in milliseconds.	0 = unlimited. positive integers = timeout (in ms).
`priority`	Priority of the message.	0-9, where 1 is the highest priority. A value of -1 means that the property should be determined by the configuration of the queue. A value of -2 specifies that the connector can use its own default value.
persistence	Whether the message should be 'hardened' to disk.	1 = non-persistent 2 = persistent A value of -1 means that the property should be determined by the configuration of the queue. A value of -2 specifies that the connector can use its own default value.
CCSID	Character set encoding of the outbound message.	Integers - valid values listed in base WebSphere MQ documentation. This value should match that of the CCSID connector-specific configuration property; see "CCSID" on page 23
targetClient	Whether the receiving application is JMS compliant or not.	0 = JMS (MQRFH2 header) 1 = MQ (MQMD header only)
`encoding`	How to represent numeric fields.	An integer value as described in the base WebSphere MQ documentation.

Note: The adapter has no control of the character set (CCSID) or encoding attributes of data in MQMessages. Because data conversion is applied as the data is retrieved from or delivered to the message buffer, the connector relies upon the IBM WebSphere MQ implementation of JMS to convert data (see the IBM WebSphere MQ Java client library documentation). Accordingly, these conversions should be bi-directionally equivalent to those performed by the native WebSphere MQ API using option `MQGMO_CONVERT`. The connector has no control over differences or failures in the conversion

process. The connector can retrieve message data of any CCSID or encoding supported by WebSphere MQ without additional modifications. To deliver a message of a specific `CCSID` or encoding, the output queue must be a fully-qualified URI and specify values for `CCSID` and `encoding`. The connector passes this information to WebSphere MQ, which (via the JMS API) uses the information when encoding data for MQMessage delivery. Often, lack of support for CCSID and encoding can be resolved by downloading the most recent version of the IBM WebSphere MQ Java client library from IBM's web site. If problems specific to CCSID and encoding persist, contact WebSphere business integration system Technical Support to discuss the possibility of using an alternate Java Virtual Machine to run the connector.

Meta-objects configuration

The connector uses meta-object entries to determine which business object to associate with a message. The type of business object and verb used in processing an event message is based on the `FORMAT` field contained in the WebSphere MQ message header. You construct a meta-object attribute to store the business object name and verb to associate with the WebSphere MQ message header `FORMAT` field text. Meta-object attributes also contain message processing guidelines.

When a message is retrieved from the input queue, the connector looks up the business object name associated with the `FORMAT` text field. The message, along with the business object name, is then passed to the data handler. If a business object is successfully populated with message content, the connector checks to see if it is subscribed, and then delivers it to the integration broker using the `gotApplEvents()` method.

The connector 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.

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**
  - Gives 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 1 provides a complete list of properties supported in meta-objects. Refer to these properties when implementing meta-objects. Your meta object should have one or more of the properties shown in Table 1.

Not all properties are available in both static and dynamic meta-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, "Overview," on page 1, to determine how a specific property is interpreted and used by the connector.

Table 1. WebSphere MQ adapter meta-object properties

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
`CollaborationName`	Yes	No	The `CollaborationName` must be specified in the application specific text of the attribute for the business object/verb combination. For example, if a user expects to handle synchronous event delivery for the business object Customer with the Create verb, the static metadata object must contain an attribute named `Customer_Create`. The `Customer_Create` attribute must contain application specific text that includes a name-value pair. For example, `CollaborationName=MyCustomerProcessingCollab`. See the "Overview of creating static meta-objects" on page 33 section for syntax details. Failure to do this will result in run-time errors when the connector attempts to synchronously process a request involving the Customer business object. **Note:** This property is only available for synchronous requests.
`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 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 Appendix B, "Connector Configurator," on page 75.
`DataHandlerMimeType`	Yes	Yes	Allows you to request a data handler based on a particular MIME type. If specified in the meta-object, this will override the value specified in the `DataHandlerMimeType` connector property. Use this 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 Appendix B, "Connector Configurator," on page 75.

Table 1. WebSphere MQ adapter meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
DataHandlerClassName	Yes	Yes	See the description in Appendix B, "Connector Configurator," on page 75.
InputFormat	Yes	Yes	Format or type of inbound (event) message to associate with the given business object. This value helps identify the message content and is specified by the application that generated the message. When a message is retrieved and is in this format, it is converted to the given business object, if possible. If this format is not specified for a business object, the connector does not handle subscription deliveries for the given business object. Do not set this property using default meta-object conversion properties; its value is used to match incoming messages to business objects. 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. The OutputFormat name length must not exceed 16 characters, otherwise WebSphere MQ might generate an error. If the OutputFormat is not specified, the input format is used, if available.
InputQueue	Yes	Yes	The input queue that the connector polls to detect new messages. This property is used to match incoming messages to business objects only. Do not set this property using default conversion properties; its value is used to match incoming messages to business objects. **Note:** The InputQueue property in the connector-specific properties defines which queues the adapter polls. This is the only property that the adapter uses to determine which queues to poll. In the static MO, the InputQueue 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. For information, see "Overview of mapping data handlers to input queues" on page 34
OutputQueue	Yes	Yes	Queue to which messages derived from the given business object are delivered.
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.

Table 1. WebSphere MQ adapter meta-object properties (continued)

Property name	Definable in static meta-object	Definable in dynamic meta-object	Description
`TimeoutFatal`	Yes	Yes	Used in synchronous request processing to trigger the connector to return an error message if a response is not received. If this property is `True`, the connector returns APPRESPONSETIMEOUT to the broker when a response is not received within the time specified by `ResponseTimeout`. If this property is undefined or set to `False`, then on a response timeout the connector fails the request but does not terminate. Default = `False`.
`DataEncoding`	Yes	Yes	`DataEncoding` is the encoding to be used to read and write messages. If this property is not specified in the static meta-object, the connector tries to read the messages without using any specific encoding. `DataEncoding` defined in a dynamic child meta-object overrides the value defined in the static meta-object. The default value is `Text`. The format for the value of this attribute is `messageType[:enc]`. I.e., `Text:ISO8859_1`, `Text:UnicodeLittle`, `Text`, or `Binary`.This property is related internally to the InputFormat property: specify one and only one `DataEncoding per InputFormat`.
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.			
ReplyToQueue		Yes	Queue 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.
JMSProperties		Yes	See "JMS properties" on page 37.

Overview of creating static meta-objects

The WebSphere MQ adapter configuration meta-object consists of 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\WebSpereMQ\samples\LegacyContact\WebSpereMQ_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 2. 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 3. Sample static meta-object structure

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

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

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

Table 4. Sample static meta-object structure

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

Table 4. Sample static meta-object structure (continued)

Attribute name	Application-specific information
Order_Create	OutputFormat=ORDER

Table 1 on page 30 describes the properties that you can specify as application-specific information in the static meta-object.

Note: If a static meta object is not specified, the connector is unable to map a given message format to a specific business object type during polling. When this is the case, the connector passes the message text to the configured data handler without specifying a business object. If the data handler cannot create a business object based on the text alone, the connector reports an error indicating that this message format is unrecognized.

Steps for creating static meta-objects

To implement a static meta-object, do the following:

1. Launch Business Object Designer. For further information, see the *Business Object Development Guide*.
2. Open the sample meta-object `connectors\WebSphereMQ\samples\LegacyContact\Sample_WebSphereMQ_MO_Config.xsd`.
3. Edit the attributes and ASI to reflect your requirements, referring to Table 1 on page 30 and then save the meta-object file.
4. Specify the name of this meta-object file as the value of the connector property ConfigurationMetaObject.

Overview of mapping data handlers to input queues

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

Steps for mapping data handlers to input queues

To map a data handler to an InputQueue, do the following:

1. Use connector-specific properties (see "InputQueue" on page 25) to configure one or more input queues.
2. Open the static meta-object in **Business Object Designer**.
3. For each input queue in the static meta-object, specify the queue manager and input queue name as well as data handler class name and mime type in the application-specific information.

For example, the following attribute in a static meta-object associates a data handler with an InputQueue named CompReceipts:

```
[Attribute]
Name = Cust_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
```

```
AppSpecificInfo = InputQueue=queue://queue.manager/CompReceipts;
DataHandlerClassName=com.crossworlds.DataHandlers.WBIMB.
disposition_notification;DataHandlerMimeType=message/
disposition_notification
IsRequiredServerBound = false
[End]
```

Overview of creating dynamic child meta-objects

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

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

Table 1 on page 30 describes the properties that you can specify as application-specific information in the dynamic meta-object.

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

Table 5. Dynamic meta-object header attributes

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

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

The 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`

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.

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 Table 6 shows how a dynamic child meta-object might be structured for polling.

Table 6. Dynamic child meta-object structure for polling

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

As shown in Table 6, you can define additional attributes, `Input_Format` and `InputQueue`, in a dynamic child meta-object. The `Input_Format` is populated with the format of the message retrieved, while the `InputQueue` attribute contains the name of the queue from which a given message has been retrieved. If these properties are not defined in the child meta-object, they will not be populated.

Example scenario:

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

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 7. 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 3 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`).

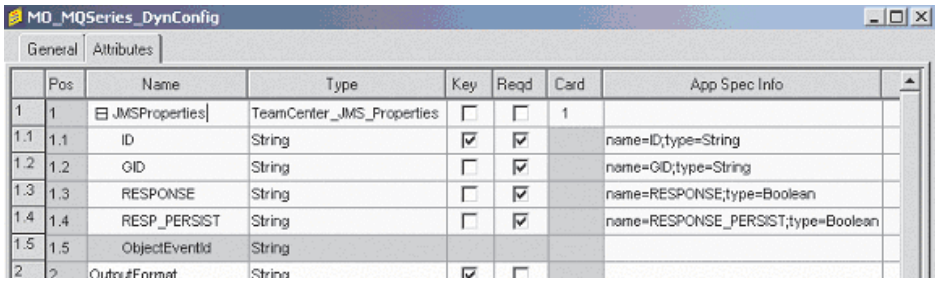

Figure 3. JMS properties attribute in a dynamic meta-object

Steps for creating a dynamic meta-object

To implement a dynamic meta-object, do the following:

1. Launch **Business Object Designer**. For further information, see the *Business Object Development Guide*.
2. Open the top-level business object whose processing you want to the dynamic meta-object to influence.
3. 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. 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`.
4. Save the top-level business object.

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

Startup file configuration

Before you start the connector for WebSphere MQ, you must configure the startup file.

Windows

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

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

UNIX

To complete the configuration of the connector for UNIX platforms, you must modify the `start_WebSphereMQ.sh` file:

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

Startup

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 *ProductDir*/bin directory.

The name of the startup script depends on the operating-system platform, as Table 8 shows.

Table 8. 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 (available only when the broker is WebSphere Application Server or InterChange Server), which is launched when you start System Manager

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

  `connector_manager_connName -stop`

  where *connName* is the name of the connector.

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

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

- From System Monitor (WebSphere InterChange Server product only)

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

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

Chapter 3. Creating or modifying business objects

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

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

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

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

Adapter business object structure

After installing the adapter, you must create business objects. There are no requirements regarding the structure of the business objects other than those imposed by the configured data handler. The business objects that the connector processes can have any name allowed by InterChange Server.

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

Sample business object properties

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

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

```
Name = Sample_WebSphereMQ_LegacyContact
Version = 1.0.0

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

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

[Verb]
Name = Create
[End]

[Verb]
Name = Delete
[End]

[Verb]
Name = Retrieve
[End]
```

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

Error handling

All error messages generated by the connector are stored in a message file named WebSphere MQConnector.txt. (The name of the file is determined by the LogFileName standard connector configuration property.) Each error has an error number followed by the error message:

```
Message number
Message text
```

The connector handles specific errors as described in the following sections.

Application timeout

The error message ABON_APPRESPONSETIMEOUT is returned when:

- The connector cannot establish a connection to the JMS service provider during message retrieval.
- The connector successfully converts a business object to a message but cannot deliver it the outgoing queue due to connection loss.
- The connector issues a message but times out waiting for a response for a business object with conversion property TimeoutFatal equal to True.
- The connector receives a response message with a return code equal to APP_RESPONSE_TIMEOUT or UNABLE_TO_LOGIN.

Unsubscribed business object

If the connector retrieves a message that is associated with an unsubscribed business object, the connector delivers a message to the queue specified by the UnsubscribedQueue property.

Note: If the UnsubscribedQueue is not defined, unsubscribed messages will be discarded.

When a NO_SUBSCRIPTION_FOUND code is returned by the gotApplEvent() method, the connector sends the message to the queue specified by the UnsubscribedQueue property and continues processing other events.

Connector not active

When the gotApplEvent() method returns a CONNECTOR_NOT_ACTIVE code, the pollForEvents() method returns an APP_RESPONSE_TIMEOUT code and the event remains in the InProgress queue.

Data handler conversion

If the data handler fails to convert a message to a business object, or if a processing error occurs that is specific to the business object (as opposed to the JMS provider), the message is delivered to the queue specified by ErrorQueue. If the ErrorQueue is not defined, messages that cannot be processed due to errors will be discarded.

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

Tracing

Tracing is an optional debugging feature you can turn on to closely follow connector behavior. Trace messages, by default, are written to STDOUT. See the connector configuration properties in Chapter 2, "Installing and configuring the adapter," on page 19 for more on configuring trace messages. For more information on tracing, including how to enable and set it, see the *Connector Development Guide*.

What follows is recommended content for connector trace messages.

Level 0 — This level is used for trace messages that identify the connector version.

Level 1 — Use this level for trace messages that provide key information on each business object processed or record each time a polling thread detects a new message in an input queue.

Level 2 — Use this level for trace messages that log each time a business object is posted to InterChange Server, either from `gotApplEvent()` or `executeCollaboration()`.

Level 3 — Use this level for trace messages that provide information regarding message-to-business-object and business-object-to-message conversions or provide information about the delivery of the message to the output queue.

Level 4 — Use this level for trace messages that identify when the connector enters or exits a function.

Level 5 — Use this level for trace messages that indicate connector initialization, represent statements executed in the application, indicate whenever a message is taken off of or put onto a queue, or record business object dumps.

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

Chapter 4. Troubleshooting

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

Start-up problems

Problem	Potential solution / explanation
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/jms/JMSException...`	Connector cannot find file `jms.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `WebSphere MQ_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: com/ibm/mq/jms/MQConnectionFactory...`	Connector cannot find file `com.ibm.mqjms.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `WebSphere MQ_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/naming/Referenceable...`	Connector cannot find file `jndi.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `WebSphere MQ_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following exception is reported: `java.lang.UnsatisfiedLinkError: no mqjbnd01 in shared library path`	Connector cannot find a required run-time library (`mqjbnd01.dll` [NT] or `libmqjbnd01.so` [Solaris]) from the IBM WebSphere MQ Java client libraries. Ensure that your path includes the IBM WebSphere MQ Java client library folder.
The connector reports `MQJMS2005: failed to create MQQueueManager for ':'`	Explicitly set values for the following properties: `HostName`, `Channel`, and `Port`.

Event processing

Problem	Potential solution / explanation
The connector delivers all messages with an MQRFH2 header.	To deliver messages with only the MQMD WebSphere MQ header, append `?targetClient=1` to the name of output queue URI. For example, if you output messages to queue `queue://my.queue.manager/OUT`, change the URI to `queue://my.queue.manager/OUT?targetClient=1`. See Chapter 2, "Installing and configuring the adapter," on page 19 for more information.
The connector truncates all message formats to 8-characters upon delivery regardless of how the format has been defined in the connector meta-object.	This is a limitation of the WebSphere MQ MQMD message header and not the connector.

Appendix A. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of WebSphere Business Integration adapters. The information covers connectors running 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 9 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

These standard properties have been added in this release:

- AdapterHelpName
- BiDi.Application
- BiDi.Broker
- BiDi.Metadata
- BiDi.Transformation
- CommonEventInfrastructure
- CommonEventInfrastructureContextURL
- ControllerEventSequencing
- jms.ListenerConcurrency
- jms.TransportOptimized
- ResultsSetEnabled
- ResultsSetSize
- TivoliTransactionMonitorPerformance

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 9 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 9 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 9, 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 9. 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

Table 9. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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	
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 .
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.

Table 9. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
CommonEventInfrastructure	true or false	false	Component restart	
CommonEventInfrastructureURL	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.
ConcurrentEventTriggeredFlows	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.
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.

Table 9. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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.NumConcurrent Requests	Positive integer	10	Component restart	This property is valid only if the value of DeliveryTransport is JMS .
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	

Table 9. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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	
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	

Table 9. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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	3	Dynamic if ICS; otherwise Component restart	
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	

Table 9. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
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.
XMLNameSpaceFormat	short or long	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 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 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 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`.

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.

The default value is JMS.

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 3.

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 WebSphere MQ. 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.

Note: For information on a separate, downloadable GUI utility that you can use to store and send MQ messages, search for "IH03: WBI Message Broker V5 - Message display, test and performance utilities" at http://www.ibm.com/support.

About the tutorial

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_WebSphereMQ_LegacyContact`, 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 (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 adapter for WebSphere MQ retrieves contact messages posted by ApplicationX to its input queue. Using the Delimited data handler, the adapter converts these messages to `Sample_WebSphereMQ_LegacyContact` 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 adapter for WebSphere MQ, and examine the attributes. After changing the data, you re-deliver the message to the integration broker where it is sent to the adapter for WebSphere MQ, converted to a message, and delivered to the output queue of the adapter (the input queue of ApplicationX). In the tutorial, the adapter is configured for the WebSphere MQ Integrator Broker, but you need not actually install and configure this broker to run the tutorial.

Before you begin

Before proceeding with this tutorial, make sure that:

- You installed and are experienced with the IBM WebSphere product.
- You installed WebSphere MQ 5.1 or later.
- You installed the WebSphere MQ client libraries for Java.
- You installed the adapter for WebSphere MQ (configuration instructions are provided in this tutorial).
- Your WebSphere MQ adapter queue manager is named `crossworlds.queue.manager` (the default value during installation). Otherwise, substitute your queue manager name whenever this document refers to `crossworlds.queue.manager`.

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. The business object repository is provided in the *sample_folder* as `.xsd` files.

1. **Define the queues** The tutorial requires that six queues be defined in your queue manager. To create the necessary queues, type `RUNMQSC crossworlds.queue.manager` from the command line and issue the following commands:
   - `DEFINE QL('MQCONN.IN')`
   - `DEFINE QL('MQCONN.IN_PROGRESS')`
   - `DEFINE QL('MQCONN.ERROR')`
   - `DEFINE QL('MQCONN.ARCHIVE')`
   - `DEFINE QL('MQCONN.REPLY')`
   - `DEFINE QL('LEGACYAPP.IN')`

   Now you can define the queues required by the WebSphere MQ adapter and Port Connector for the WMQI broker configuration as follows:
   - `DEFINE QL('WebSphereMQConnector/ADMININQUEUE')`
   - `DEFINE QL('WebSphereMQConnector/ADMINOUTQUEUE')`
   - `DEFINE QL('WebSphereMQConnector/DELIVERYQUEUE')`
   - `DEFINE QL('WebSphereMQConnector/FAULTQUEUE')`
   - `DEFINE QL('WebSphereMQConnector/REQUESTQUEUE')`
   - `DEFINE QL('WebSphereMQConnector/RESPONSEQUEUE')`
   - `DEFINE QL('WebSphereMQConnector/SYNCHRONOUSREQUESTQUEUE')`
   - `DEFINE QL('WebSphereMQConnectorSYNCHRONOUSRESPONSEQUEUE')`
   - `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')`
2. **Configure the adapter** Using Connector Configurator, open *sample_folder*`\WebSphereMQConnector.cfg`. For further information on using

Connector Configurator, see Appendix B, "Connector Configurator," on page 75; for more on connector-specific properties, see "Connector-specific properties" on page 21.
If you have not done so already, configure the adapter as described in the installation guide for your system. Additionally, check or change adapter configuration properties to match the values listed below:

- `Broker Type` Set this property to `WMQI`.
- `Repository Directory` Set this property to the *sample_folder* directory.

Set the following connector-specific properties:

- `ConfigurationMetaObject` Set this property to `Sample_WebSphereMQ_MO_Config`.
- `DataHandlerConfigMO` Set this property to `Sample_WebSphereMQ_MO_DataHandler`.
- `DataHandlerMimeType` Set this property to `text/delimited`.
- `DataHandlerClassName` Set this property to `com.crossworlds.DataHandlers.text.delimited`
- `ErrorQueue` Set this property to `queue://crossworlds.queue.manager/MQCONN.ERROR`.
- `InProgressQueue` Set this property to `queue://crossworlds.queue.manager/MQCONN.IN_PROGRESS`.
- `InputQueue` Set this property to `queue://crossworlds.queue.manager/MQCONN.IN`.
- `hostname` Set this property to the name of your machine.
- `port` Set this property to `1414`.
- `channel` Set this property to `CHANNEL1`
- `UnsubscribedQueue` Set this property to `queue://crossworlds.queue.manager/MQCONN.UNSUBSCRIBED`.

3. **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 `WebSphereMQConnector/DELIVERYQUEUE` (the `DeliveryQueue` property value for the WebSphere MQ adapter).
   - `DeliveryQueue` Set this property to `WebSphereMQConnector/REQUESTQUEUE` (the `RequestQueue` property value for the WebSphere MQ adapter).
4. **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 WebSphere MQ adapter, add the business objects shown in Table 10 and set the **Message Set ID** to a unique value for each supported business object.

Table 10. Supported sample business objects for JMS adapter

Business object name	Message Set ID
`Sample_WebSphereMQ_MO_Config`	1
`Sample_WebSphereMQ_MO_DataHandler`	2
`Sample_WebSphereMQ_LegacyContact`	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 11.

Table 11. Supported sample business objects for Port connector

Business object name	Message Set ID
Sample_WebSphereMQ_LegacyContact	1

5. **Configure the meta-object** If your WebSphere MQ Queue Manager is not named crossworlds.queue.manager, update the URI in the default attribute's AppSpecificInfo field in the Sample_WebSphereMQ_MO_Config business object.
6. **Create or update connector start scripts**
   Windows:
   a. Open the properties of the shortcut for the adapter for WebSphere MQ.
   b. As the last argument in the target, add -c followed by the *<full path and filename for the WebSphereMQConnector.cfg file>* For example:

   ```
   -cProduct_Dir\connectors\WebSphereMQ\samples\
   LegacyContact\WebSphereMQConnector.cfg
   ```

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

   ```
   AGENTCONFIG_FILE=-cProduct_Dir/connectors/WebSphereMQ/samples/
   LegacyContact/WebSphereMQConnector.cfg
   ```

Running the scenarios

Before you run the scenarios:

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

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 "Overview of creating static meta-objects" on page 33.

1. **Simulate the Port connector** Using the Visual Test connector, define a profile for the Port connector:
   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 *sample_folder*, 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_WebSphereMQ_LegacyContactby 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** Using WebSphere MQ Explorer or a similar application, open queue queue://crossworlds.queue.manager/LEGACYAPP.IN to see if a new contact message with format LC_CR has arrived from the adapter.

4. **Test event processing** Send a message to the WebSphere MQ 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 WebSphere adapter's `InputQueue` property to `queue://crossworlds.queue.manager/LEGACYAPP.IN` so that the adapter will poll its own messages. Once you have a message in the input queue, the adapter will poll it and attempt to convert it into a `Sample_WebSphereMQ_LegacyContact` 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_WebSphereMQ_LegacyContact` business object in meta-object `Sample_WebSphereMQ_MO_Config`. In this scenario, that format is `LC_CR`. If the adapter identifies the incoming message format as `LC_CR`, it will use the data handler to convert the message to business object `Sample_WebSphereMQ_LegacyContact` 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 WebSphere MQ adapter to retrieve messages and convert them to `Sample_WebSphereMQ_LegacyContact` business objects, and to convert `Sample_WebSphereMQ_LegacyContact` business objects to contact messages.

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 the static meta-object scenario. For further information on dynamic meta-objects, see "Overview of creating dynamic child meta-objects" on page 35. For prerequisites for this scenario, see "Before you begin" on page 94. In addition, you must install and configure the Port connector as described in "Static meta-object scenario" on page 96. The steps below take you through creating an attribute for a child meta-object for `Sample_WebSphereMQ_LegacyContact`. Specifically, you will be modifying the output queue values in this child meta object to redirect the `Sample_WebSphereMQ_LegacyContact` business object to a new queue.

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_WebSphereMQ_LegacyContact`, 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_WebSphereMQ_DynMO_Config.xsd` and `Sample_WebSphereMQ_LegacyContact.xsd` from the *sample_folder*.
   b. In the `Sample_WebSphereMQ_LegacyContact` window, add an attribute named `DynMO` of type `Sample_WebSphereMQ_DynMO_Config` .
3. **Define a new target queue** Define a temporary queue `REROUTE.IN` in WebSphere MQ. This is where the dynamic meta-object will re-route the `Sample_WebSphereMQ_LegacyContact` business object. To create the necessary queue, type `RUNMQSC crossworlds.queue.manager` from the command line and issue the following command: `DEFINE QL('REROUTE.IN')`
4. **Start the adapter for WebSphere MQ** if it is not already running.
5. **Start the Visual Test connector** if it is not already running.
6. **Simulate the Port connector** (If you have already performed this step for the static meta-object scenario, skip this task and go to the next step.) Using the Visual Test connector, define a profile forthe Port connector:

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_WebSphereMQ_LegacyContact`, changing the default values if desired.
   b. Right-click on the `DynMO` attribute and create an instance of it, `Sample_WebSphereMQ_DynMO_Config`.
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. For this scenario, the target queue is `REROUTE.IN`. Make sure you enter the complete URI, for example, `queue://<queue manager>/REROUTE.IN?targetClient=1`.
9. **Send the business object** Click **Send BO**.
10. **Confirm message delivery** Using WebSphere MQ Explorer or a similar application, open queue `queue://<queue manager>/REROUTE.IN` to see if a new contact message has arrived from the adapter. If a new message has arrived from the WebSphere MQ adapter to the queue named `REROUTE.IN`, then the re-routing has worked. You can create different queues in WebSphere MQ and send the business objects with different queue names in their respective dynamic meta-objects.

Appendix D. 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 E. 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.

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
CICS
CrossWorlds
DB2
DB2 Universal Database
Domino
IMS
Informix
iSeries
Lotus
Lotus Notes
MQIntegrator
MQSeries
MVS
OS/400
Passport Advantage
SupportPac
WebSphere
z/OS
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both. MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both. Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. 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.

WebSphere Business Integration Adapter Framework V2.6.0

Index

S

T

U

V

W

Printed in USA