IBM WebSphere Business Integration Express and
Express Plus
for Item Synchronization

Adapter for iSoft Peer-to-Peer Agent User Guide

Adapter Version 1.2.x

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

19December2003

This edition of this document applies to IBM WebSphere Business Integration Express for Item Synchronization V4.3.1, IBM WebSphere Business Integration Express Plus for Item Synchronization V4.3.1, and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, e-mail doc-comments@us.ibm.com. We look forward to hearing from you. When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

New in this release

In this release, version 1.2 of the adapter for iSoft is supported on the IBM WebSphere Business Integration Express and Express Plus for Item Synchronization release.

The adapter for iSoft is supported on the following operating systems:

- Microsoft Windows 2000

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

About this document

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

This document describes the configuration and business object development for the adapter for iSoft.

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, with WebSphere MQ, and with iSoft Peer-to-Peer Agent.

Related documents

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

You can install the documentation from the following site:

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

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.
{ }	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. All product pathnames are relative to the directory where the product is installed on your system.
%text% and *$text*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable.
ProductDir	Represents the directory where the IBM WebSphere Business Integration Adapters product is installed.

Chapter 1. Overview

The connector for iSoft is a runtime component of the WebSphere Business Integration Adapter for iSoft. The connector allows the WebSphere integration broker to send business data to, and receive business data from, the iSoft Peer-to-Peer (P2P) Agent. This chapter describes the connector component and the relevant business integration system architecture.

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

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

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

For more information about the relationship of the integration broker to the connector, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

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

- the InterChange Server Express integration broker, which is described in the *User Guide for WebSphere Business Integration Express for Item Synchronization*

Connector architecture

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

The connector allows a WebSphere integration broker (InterChange Server Express) to asynchronously exchange business objects with Peer-to-Peer Agent.

The connector supports bidirectional exchanges—it publishes to the integration broker business data events that it receives from Peer-to-Peer Agent, and it processes business data requests sent from the integration broker to Peer-to-Peer Agent. This is done through queues. You must set up queues for receiving messages from Peer-to-Peer Agent, and queues for sending messages to Peer-to-Peer Agent:

- For messages incoming from Peer-to-Peer Agent: The connector monitors a set of queues that are populated with messages received from Peer-to-Peer Agent. The set includes one or more queues for business data messages originating from trading partners, a queue for notifications that are generated by Peer-to-Peer Agent, and a queue for MDNs sent from the trading partners in response to messages that originated from the integration broker. When Peer-to-Peer Agent places a message from a trading partner in the queue, the connector picks up the message, calls a data handler to convert the data contained in the body of the message to a corresponding business object, and then delivers the business object to the integration broker. From that point, the integration broker handles the business object according to the processes of the specific integration broker. (For example, if InterChange Server Express is being used, the business object is published to those collaborations that have subscribed to it.)
- For business data outgoing to Peer-to-Peer Agent: when an integration broker initiates a request, the connector calls a data handler to convert the request business object into a data format for delivery in the body of an WebSphere MQ message, and puts the message in a queue that you have defined for messages that are outgoing to Peer-to-Peer Agent. Peer-to-peer agent picks up the message from the queue, and delivers it to the intended trading partner. If there are multiple trading partners, you can define an outgoing message queue for each.

The connector does not persist the state of events. When the connector delivers a message into the queue monitored by Peer-to-Peer Agent, it returns successfully, without waiting to determine whether the message that it delivered to Peer-to-Peer Agent was sent to the Peer-to-Peer Agent trading partner. And when the connector receives an event from Peer-to-Peer Agent, it publishes it asynchronously to the subscribing collaborations. If the enterprise requires that delivery of the business data messages be tracked and verified, the collaboration (or WebSphere MQ Integrator broker work flow) that defines the business process must be designed to do that by incorporating the monitoring of MDN and notification queues into its process flow.

The connector uses data handlers to convert between the serialized data contained in the WebSphere MQ message and the business object that is handled by the integration broker. Because different data formats require different data handlers, the connector allows you to specify (in the connector's standard configuration properties) the data handler to be used for your implementation of the connector. However, you can adjust this specification at the business object level, specifying that a specific data handler (such as the XML data handler) will be used for a specific business object (such as the Customer_Create business object) and that other data handlers will be used for other business objects. You can do this for both request processing (converting a business object into a message to send to Peer-to-Peer Agent) and for event notification (converting a message from Peer-to-Peer Agent into a business object that is delivered to the integration broker).

For more information about data handlers generally, see the *Data Handler Guide*.

The type of business object and verb used in processing a message received by the connector is based on the queue from which a message is retrieved. The connector uses meta-object entries to specify which business objects can be expected from which queues.

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 allows the user to set and get much of the same information as that defined by the static meta-object, but dynamically at run-time.

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

When a message is retrieved from the input queue, the connector looks up the business object name associated with the queue and 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 the connector cannot determine the business object based on the input queue and format, the message body alone is passed to the data handler. If a business object is successfully populated with message content, the connector checks to see if it is subscribed, and then delivers it to the integration broker using the `gotApplEvents()` method.

Application-connector communication 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 the flow of a message request communication. When the `doVerbFor()` method receives a business object from the integration broker (typically from a collaboration running inInterChange Server Express), 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. Peer-to-Peer Agent monitors the queue, picks up the message, and sends it to the targeted trading partner.

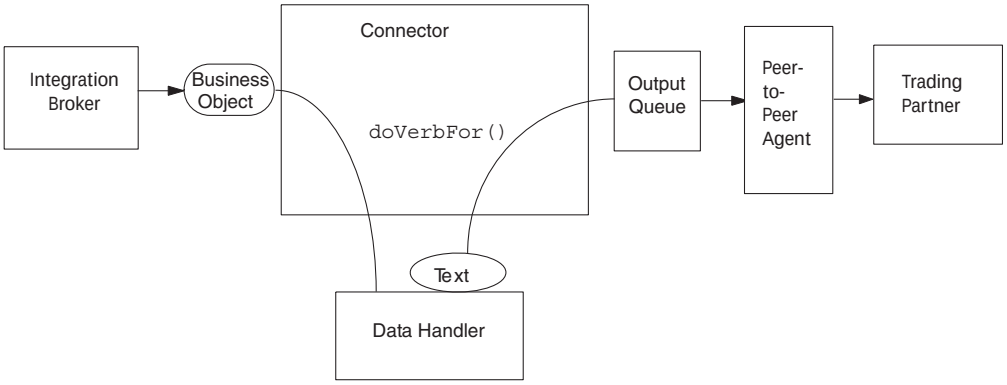

Figure 1. Application-Connector Communication Method: Message Request

Event delivery

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

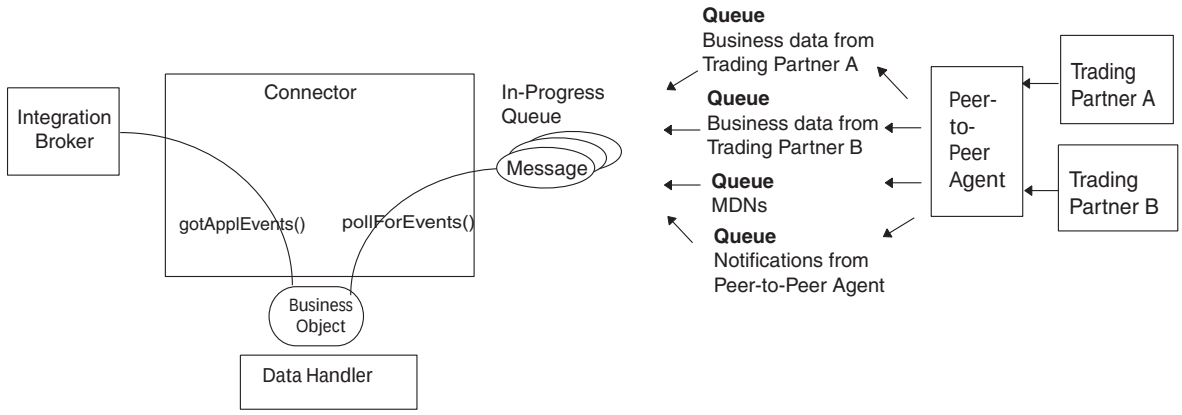

Figure 2. Application-Connector Communication Method: Event Delivery

Event notification and handling

The connector monitors one or more queues that Peer-to-Agent populates with these types of messages:

- Business data events originating from trading partners
- MDNs from the trading partnesr that acknowledge documents that originated from the integration broker and were sent through the connector
- Notification messages originating from Peer-to-Peer Agent, informing the connector of delivery retries and the state of outstanding requests

An event occurs whenever Peer-to-Peer Agent puts a new message into one of these queues. The connector polls the queues periodically, detects the message, retrieves it, and publishes it to subscribing collaborations.

Retrieval

The connector uses the `pollForEvents()` method to poll the WebSphere MQ queue at regular intervals for messages. When the connector finds a message, it retrieves it from the WebSphere MQ queue and examines it to determine its format. If the format and/or input queue 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 and/or queue to the configured data handler; the data handler is expected to populate the business object and specify a verb. If the format and/or queue 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 33 for event failure scenarios.

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

Note: Transactional sessions with a JMS service provider require that every requested action on a queue be performed and committed before events are removed from the queue. Accordingly, when the connector retrieves a message from the queue, it does not commit to the retrieval until three things occur: 1) The message has been converted to a business object; 2) the business object is delivered to InterChange Server Express by the `gotApplEvents()` method, and 3) a return value is received.

Recovery

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

Fail on startup

With the fail on startup option, if the connector finds messages in the in-progress queue during initialization, it logs an error and immediately shuts down. It is the

responsibility of the user or system administrator to examine the message and take appropriate action, either to delete these messages entirely or move them to a different queue.

Reprocess

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

Ignore

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

Log error

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

Archiving

If the connector property `ArchiveQueue` is specified and identifies a valid queue, the connector places copies of all successfully processed messages in the archive queue. If `ArchiveQueue` is undefined, messages are discarded after processing. For more information on archiving unsubscribed or erroneous messages, see "Error handling" on page 33 in Chapter 3, "Creating or modifying business objects," on page 31

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

Business object requests

Business object requests are processed when the integration broker 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 only business object verb supported by the connector is the `Create` verb.

The business objects are delivered asynchronously. A message is created from the business object and then written to the output queue. If the message is delivered to the queue, the connector returns BON_SUCCESS, else BON_FAIL.

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

Chapter 2. Installing and configuring the connector

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

Compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. The 1.2 version of the adapter for iSoft is supported on the following adapter framework and integration brokers:

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

See the Release Notes for any exceptions.

Prerequisites

Prerequisite software

- The connector supports interactions with iSoft Peer-to-Peer Agent version 3.1.3 for WebSphere MQ
- For interoperability with iSoft Peer-to-Peer Agent, the connector requires WebSphere MQ 5.1, 5.2,[1] or 5.3.

  Note: The adapter does not support Secure Socket Layers (SSL) in WebSphere MQ 5.3. Nor does support for WebSphere MQ 5.3 imply that the adapter framework uses WebSphere MQ 5.3 to communicate with the integration broker. For the WebSphere MQ software version appropriate to adapter framework-integration broker communication, see the Installation Guide for your platform.

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.

- You must have the IBM WebSphere MQ Java client libraries, version 5.2.2.
- The adapter for iSoft is supported on the following operating system:
  - Microsoft Windows 2000

Client setup with NT server

See the description in *WebSphere MQ Quick Beginnings for NT*.

Overview of installation tasks

To install the connector for iSoft, you must perform the tasks described in the sections below.

Install the integration broker

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

Install the connector and related files

This task includes installing the files for the adapter from the software package onto your system. See "Installing the connector and related files" on page 10.

Configure the Peer-to-Peer Agent application

Configuration of Peer-to-Peer Agent is done through an XML configuration file. By default, Peer-to-Peer Agent looks for the file `p2pagent.cfg` in the local directory and executes any commands found inside. An iSoft Peer-to-Peer Agent can receive a message from a trading partner and dynamically route it to a queue based various factors, including on the mime type of the message it received. Event notification requires one queue per data format, resulting in fewer defined queues that the connector must poll.

In the following scenario, trading partner B sends an order to trading partner A. The iSoft Peer-to-Peer Agent has the ability to route messages to different queues based on a messages MIME type. The scenario uses messages which are created with the MIME type `txt/xml`. The adroute commands in `p2pagent.cfg_a` route these messages to the `COMPA.XML` queue. iSoft Peer-to-Peer Agent A can also route any messages with a MIME type set to `text/plain` to `COMPA.TXT`. You can configure the connector to poll these queues and configure data handlers to process the MQ messages on a per queue basis. An XML data handler could process all messages in a specific XML queue, `COMPA.XML`, for instance, or a delimited data handler could be configured to process all messages in a text queue, like `COMPA.TXT`. Use the following commands in the `p2pagent.cfg` file.

- To transport a file from P2agentB to P2agentA and have P2agentA place the file in the COMPA.XML queue.
  ```
  send http B A -cX -fNoutboxB\toA\Order1.txt
  ```
- To transport a file from P2agentB to P2agentA and have P2agentA place the file in the COMPA.TXT queue.
  ```
  send http B A -cT -fNoutboxB\toA\Order1.txt
  ```

Following is a sample configuration file, with comments (the comments, indicated by /* in the sample, are for explanation in this document only; do not include comments in your actual configuration file).

The sample makes the following assumptions:

- The WebSphere MQ queue manager that you are using for communication between the connector and Peer-to-Peer Agent is named ″isoft.queue.manager″.
- The integration broker is acting on behalf of trading partner **A**, to exchange XML documents with Peer-to-Peer Agent, which is acting on behalf of Trading Partner **B**.
- Queue ″a_to_b″ is where the integration broker (via the connector) will put any documents intended for delivery through Peer-to-Peer Agent to Trading Partner B.
- Queue ″b_to_a″ is where the Peer-to-Peer Agent will put any documents it receives for Trading Partner A. The connector will poll this queue for documents and publish them as events.
- Queue ″mdns″ is where the Peer-to-Peer Agent will put any MDN's it receives from Trading Partner B, for delivery to Trading Partner A
- Queue ″notifications″ is where the Peer-to-Peer Agent will put any notifications it generates for Trading Partner A

```
<xml>

<command>set -cnBT1</command>

<command>set -bhmq://isoft.queue.manager</command>

/* Configures Peer-to-Peer Agent to put all notifications in queue 'notifications'*/

<command>set -npmq://isoft.queue.manager/notifications -nf -ntW</command>

/* Configures Peer-to-Peer Agent to put all MDNs in queue 'mdns'*/

<command>set -rpmq://isoft.queue.manager/mdns</command>

/*Establishes a relationship so that any messages from trading partner 'B' to
trading partner 'A' are put in queue 'b_to_a' */

<command>addpair A B http://127.0.0.1:4081/ http://127.0.0.1:4080/ b_to_a
mq://isoft.queue.manager/b_to_a</command>

/*Establishes a persistent send command so that Peer-to-Peer Agent periodically
polls queue 'a_to_b' for messages to be sent to trading partner 'B'. */

<command>send http MAILBOX MAILBOX -de -dsMAILBOXID=a_to_b -r -tC5s
-tE230030415000000</command>

<command>send http MAILBOX MAILBOX -de -dsMAILBOXID=a_to_b -r -tC5s
-tE230030415000000</command>

<command>start http://127.0.0.1:4081</command> </xml>
```

Set up queues

Before setting up specific queues, be sure that you have defined the queue manager, server connection channel, and listener that you will use to communicate with Peer-to-Peer Agent.

A typical implementation of the connector requires the creation of an outbound queue for delivering messages to Peer-to-Peer Agent, and at least three inbound

queues that the connector will poll for business data events, MDNs, and notifications. For descriptions of these and more about setting up queues, see: "Setting up queues" on page 16.

Configure event notification and request processing

This task requires that you use connector properties and meta-objects to specify the queues that will be used, the business objects that will be converted, and the data handlers that will be used. See "Setting up request processing" on page 21,"Setting up event notification" on page 21, and "Configuring connector meta-objects" on page 22.

Configure 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 "Connector configuration properties" on page 12.

When you configure connector properties for the adapter, 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 URIs that you are identifying in connector properties are valid and actually exist.

Configure data handlers

Data handlers for converting MDNs and notifications into business objects are provided with the connector; a data handler for converting between business objects and the document format of the business data being exchanged (such as an XML data handler) may be available with the connector, or may need to be customized for your enterprise.

See "Configuring data handlers" on page 18

Modify the startup script

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 30

Installing the connector and related files

To install adapters for Business Integration Express for Item Sync:

1. Insert the product CD.
2. See *Installing WebSphere Business Integration Express and Express Plus for Item Synchronization*.

3. After installing adapters, see the *Quick Start Guide*, which contains configuration information for required adapters.

Installed file structure

Note: WebSphere MQ and JMS typically are installed in separate directories. On Windows, WebSphere MQ is generally installed under `\Program Files\WebSphereMQ` and JMS under `\Program Files\IBM\WebSphereMQ\Java`. Update your classpath accordingly in the iSoft connector startup scrip

Windows connector file structure

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

The utility installs the connector into the *ProductDir*`\connectors\ISOFT` 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\ISoft\CWISoft.jar`	Contains classes used by the iSoft connector
`connectors\ISoft\start_ISoft.bat`	The startup script for the connector (Windows)
`connectors\messages\ISoftConnector.txt`	Message file for the connector
`repository\ISoft\CN_ISoft.txt`	Repository definition for the connector
`connectors\ISoft\samples\WebSphereICS\Company_A_Fruit_Catalog.BO`	A sample instance of a business object
`connectors\ISoft\samples\WebSphereICS\p2pagent\p2pagent.cfg_a`	Sample Peer-to-Peer Agent configuration file.
`connectors\ISoft\samples\WebSphereICS\p2pagent\p2pagent.cfg_b`	Sample Peer-to-Peer Agent configuration file.
`connectors\ISoft\samples\WebSphereICS\p2pagent\outboxB\toA\Order1.txt`	Sample document for orders
`connectors\ISoft\samples\WebSphereICS\Sample_ISoft_BOS.in`	Repository definition for sample iSoft business objects
`connectors\ISoft\samples\WebSphereICS\Sample_ISoft_Collabs.in`	Repository definition for sample collaboration templates

`connectors\ISoft\samples\WebSphereICS\Sample_ISoftConnector_to_Port.cwt`
`connectors\ISoft\samples\WebSphereICS\Sample_ISoftConnector_to_Port_MDN.cwt`
`connectors\ISoft\samples\WebSphereICS\Sample_ISoftConnector_to_Port_Notification.cwt`
`connectors\ISoft\samples\WebSphereICSSample_ISoft_Catalog_SP.cwt`
	Sample collaboration templatess

`connectors\ISoft\samples\WebSphereMQIntegratorBroker\clearQueues.txt`	.
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\Company_A_Fruit_Catalog_with_dynamic_mo.BO`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\createQueues.txt`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\ISoftConnector.cfg`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\PortConnector.cfg`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\p2pagent\p2pagent.cfg_a`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\p2pagent\p2pagent.cfg_b`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\p2pagent\p2pagent.cfg_c`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\p2pagent\outboxB\toA\Order1.txt`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ ImportSample_ISoft_MO_Config.bat`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_Catalog.xsd`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\Sample_ISoft_DataHandler_Default.xsd`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\Sample_ISoft_DataHandler_DefaultDelimitedConfig.xsd`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\Sample_ISoft_DataHandler_DefaultISoftMDNConfig.xsd`	
`connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\Sample_ISoft_DataHandler_DefaultISoftNotificationConfig.xsd`	

Subdirectory of *ProductDir*	Description
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_DataHandler_DefaultXMLConfig.xsd	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_Dyn_MO.xsd	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_Item.xsd	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_JMSProperties.xsd	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_MDN.xsd	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_MO_Config.xsd	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_MO_ConfigParams.txt	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_Notification.xsd	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_Order.xsd	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos\ Sample_ISoft_TLO_Order.xsd	

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

Connector configuration properties

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

You use Connector Configurator Express to configure connector properties:

- For a description of Connector Configurator Express and step-by-step procedures, see Appendix B, "Connector Configurator Express," on page 53.
- For information on required connector configuration, see the *Quick Start Guide*.
- For a description of standard connector properties, see "Standard connector properties" and then Appendix A, "Standard configuration properties for connectors," on page 39.
- For a description of connector-specific properties, see "Connector-specific properties" on page 13.

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

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 39 for documentation of these properties.

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

Connector-specific properties

Connector-specific configuration properties provide information needed by the connector 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 connector. See the sections that follow for explanations of the properties.

You must provide a value for the ApplicationName configuration property before running the connector.

Name	Possible values	Default value	Required
ArchiveQueue	*Queue to which copies of successfully processed messages are sent*	`queue://isoft. queuemanager/ISOFT.ARCHIVE`	No
CCSID	*Character set for queue manager connection*	`null`	No
Channel	*MQ server connector channel*		Yes
ConfigurationMetaObject	*Name of configuration meta-object*		Yes
DataHandlerClassName	*Data handler class name*	`com.isoft.DataHandlers. text.xml`	No
DataHandlerConfigMO	*Data handler meta-object*	`MO_DataHandler_ Default`	Yes
DataHandlerMimeType	*MIME type of file*	`text/xml`	No
DefaultVerb	*Any verb supported by the connector.*	`Create`	
ErrorQueue	*Queue for unprocessed messages*	`queue://isoft. queuemanager/ISOFT.ERROR`	No
HostName	*WebSphere MQ server*		Yes
InDoubtEvents	`FailOnStartup Reprocess IgnoreLogError`	`Reprocess`	No
InputQueue	*Poll queues*	`queue://isoft. queuemanager/ISOFT.IN`	No
InProgressQueue	*In-progress event queue*	`queue://isoft. queuemanager/ISOFT.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
UnsubscribedQueue	*Queue to which unsubscribed messages are sent*	`queue://isoft. queuemanager/ISOFT.UNSUBSCRIBE`	No
UseDefaults	true or false	false	

ArchiveQueue

Queue to which copies of successfully processed messages are sent.

Default = `queue://isoft.queue.manager/ISOFT.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 17.

Default = null.

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

Default = none.

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

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

Default = none.

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

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

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

Default = `MO_DataHandler_Default`

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

Default = `text/xml`

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

Default= `Create`

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

Default = `queue://isoft.queue.manager/ISOFT.ERROR`

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

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

Default = queue://isoft.queue.manager/ISOFT.IN

InProgressQueue

Message queue where messages are held during processing. You can configure the connector to operate without this queue by using CSM 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://isoft.queue.manager/ISOFT.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.

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

Default = `queue://isoft.queue.manager/ISOFT.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`.

Setting up queues

Note: Before setting up specific queues, be sure that you have defined the queue manager, server connection channel, and listener that you will use to communicate with Peer-to-Peer Agent.

A typical implementation of the connector requires the creation of at least one outbound queue for delivering messages to Peer-to-Peer Agent, and at least three inbound queues that the connector will poll for business data events, MDNs, and notifications:

- Inbound from Peer-to-Peer Agent to Integration Broker

  This is the queue in which Peer-to-Peer Agent will place messages that contain data documents received from the trading partner that Peer-to-Peer Agent represents. The connector polls this queue to detect new messages, calls a data handler to convert the documents to business objects, and publishes the business objects as events to subscribing collaborations.

  For implementations in which Peer-to-Peer Agent is sending documents to you from multiple trading partners, you can set up multiple inbound queues, designating a different queue for the documents received from each trading partner. To do so, you must configure Peer-to-Peer Agent to deliver messages to multiple queues, and you must configure the connector to poll multiple queues. You can also use a single queue to receive documents from multiple trading partners. The Peer-to-Peer Agent agent passes information regarding the trading partner in the MQ RHF2 header of the messages it sends to the adapter. To access this information in your business object, ensure that you have defined a dynamic MO in your BO and that it defines the attribute. `JMSProperties`. For more information, see "Configuring connector meta-objects" on page 22.
- Outbound from Integration Broker to Peer-to-Peer Agent

  This is the queue in which the connector places data document messages that are to be sent to Peer-to-Peer Agent and the trading partner that it represents. When the connector receives an appropriate business object from a collaboration or other process flow out of the integration broker, the connector calls a data handler to convert the business object to a data document, and places a message

containing that document in this queue. Peer-to-Peer Agent polls the queue to discover new messages containing documents to be sent to the trading partner it represents.

Or, you can also use a single queue and retrieve the trading partner information passed in the marked header by the Peer-to-Peer Agent.

For implementations in which Peer-to-Peer Agent is sending your documents to multiple trading partners, you can set up multiple queues, designating a different queue for each trading partner. To do so, you must configure the connector to put messages in multiple queues, and you must configure Peer-to-Peer Agent to poll multiple queues. You can also have the Peer-to-Peer Agent poll a single queue for messages to different trading partners. In this case, you must be sure that you pass the trading partner information through the `JMSProperties` attribute of your dynamic MO, as define in your BO. The information specified in the `JMSProperties`attribute will be mapped to the MQ RHF2 header where the Peer-to-Peer Agent will use it when routing the document. For more information, see "Configuring connector meta-objects" on page 22.

- MDNs

  This is the queue in which Peer-to-Peer Agent places any MDNs received from the trading partners that it represents. A single MDN queue can receive MDNs from multiple trading partners. The connector polls this queue to detect new MDNs, calls the MDN data handler to convert the MDNs to business objects, and publishes the business objects as events to subscribing collaborations.

- Notifications

  The queue in which Peer-to-Peer Agent places any notifications that it generates. The connector polls this queue to detect messages containing new notifications, calls the notification data handler to convert the notifications to business objects, and publishes the business objects as events to subscribing collaborations.

- WebSphere MQ queue manager

  The queue manager used for communication between the connector and Peer-to-Peer Agent. The default name for this queue is `isoft.queue.manager`.

In addition to creating the queues, you must also configure the connector to use the queues that you create. See "Configure connector properties" on page 10for information about the properties that configure the connector to use the queues.

Queue Uniform Resource Identifiers (URI)

To configure queues for use with the connector:

- 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.

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://isoft.queue.manager/ISOFT.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 13

Note: The connector 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. Not all platforms support all CCSIDs. 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.

For more information about URIs, see the WebSphere MQ programming guide.

Configuring data handlers

To configure a data handler, you must specify that the connector will use that data handler, and you must specify values for configuration attributes of the data handler.

To specify which data handlers a connector will use, set the following connector-specific properties for the connector:

- DataHandlerConfigMO

  Set this to the name of the top-level data handler meta-object. By default, the value is MO_DataHandler_Default.

- DataHandlerMimeType

  Set this value to the MimeType of a data handler that you are using as the default.

  If you are using additional data handlers for specific business objects—for example, if an XML data handler is your default but you are also using an MDN data handler and a notifications data handler—specify each additional data handler by using configuration properties in the AppSpecificInfo property of the appropriate connector meta-object attributes. See "Configuring a static connector meta-object" on page 22and "A Sample connector meta-object" on page 24.

Note: You can also specify a data handler by providing a value for the connector-specific property DataHandlerClassName. However, the DataHandlerClassName property only specifies the data handler; it does not enable you to set configuration values in that data handler's meta-object. To both specify a data handler and configure its properties through the use of data handler meta objects, use the DataHandlerMimeType property and the DataHandlerConfigMO property.

Configuration attributes of data handlers are specified in the top-level data handler meta object (which contains attributes that specify the MimeTypes of each of the data handlers that can be used) and in a child data handler meta-object for each data handler. For instructions on how to create and revise data handler meta objects, see the *Data Handler Guide*

For the iSoft connector, you must configure data handlers for each of the following:

- Data handler for converting business data
- MDN data handler (optional)
- Notification data handler (optional)

Note: For MDNs and notifications, to determine the name of the business object that it creates, the data handler looks first at the value specified by the NameHandlerClass attribute in the data handler's meta-object. If no value has been specified in that attribute, the data handler uses the BOName attribute. By default, the implementation uses just one business object definition for all MDNs and one for all notifications. If you want to have different types of MDN or notification business objects—corresponding, for example, to the business processes of different collaborations—you must create a custom data handler. For information about doing that, see "Building a Custom Name Handler" in the Data Handler Guide.

Data handler for converting business data

You must specify a data handler for converting the business data content that is exchanged between trading partners. Since Peer-to-Peer Agent can send business data in any format, no single data handler can be used for all possible implementations of the connector. However, delivered data handlers are available for some of the most commonly used formats, including an XML data handler. The XML data handler performs conversion both to and from business objects.

For information about designing business objects for use with this type of data handler, see ″Business objects for exchanging business data,″ later in this guide.

MDN data handler

Peer-to-Peer Agent can be configured to send the machine-readable portions of message delivery notifications (MDNs) from its trading partner to the iSoft connector. The connector calls the iSoftMDN data handler, which is provided with the connector, to convert the MDN to a business object. The MDN data handler is unidirectional; it converts MDN data from Peer-to-Peer Agent into a business object that the connector can send to the integration broker for processing. Conversion of business objects into notifications is not necessary, since Peer-to-Peer Agent handles the task of creating and sending MDNs to the trading partner.

Property Name	Description	Default value
BOName	Name of the business object that will store the MDN data. Required.	`Sample_ISoft_MDN`
ClassName	Name of the data handler class to load for use with the specified MIME type. The top-level data handler meta-object has an attirbute whose name matches the specified MIME type and whose type is the MDN child meta-object. Required.	`com.crossworlds. DataHandlers.message. disposition_notification`
DummyKey	Key attribute required by InterChange Server Express. Used to copy a business object into the repository. Required.	`1`

For information about designing business objects for use with the MDN data handler, see “Business objects for MDNs” on page 33 later in this guide.

Notification data handler

The Peer-to-Peer Agent can be configured to generate notification messages to inform the user of retry attempts being exhausted. The Notification data handler is unidirectional; it converts notification messages that Peer-to-Peer Agent sends to the connector into business objects that the connector can send to the integration broker for processing.

Property name	Description	Default value
BOName	Name of the business object that will store the Notification data. Required.	`Sample_ISoft_Notification`
ClassName	Name of the data handler class to load for use with the specified MIME type. The top-level data handler meta-object has an attirbute whose name matches the specified MIME type and whose type is the Notification child meta-object. Required.	`com.crossworlds. DataHandlers.isoftp2p`

Property name	Description	Default value
DummyKey	Key attribute required by InterChange Server Express. Used to copy a business object into the repository. Required.	1

For information about designing business objects for use with the Notification data handler, see "Business objects for notifications" on page 31 later in this guide.

Setting up request processing

The connector processes request business objects received from the integration broker, calls a data handler to convert the business object to an appropriate document type, and sends the document in a message to the output queue that is monitored by Peer-to-Peer Agent. No special configuration of the connector itself is required to make this happen. However, processing of a request business object requires a child meta-object (either static or dynamic) populated with appropriate values for the following:

- OutputQueue that Peer-to-Peer Agent monitors to pick up the message sent from the integration broker
- DataHandlerMimeType, indicating the mime type (such as text/xml) of a configured data-handler that can convert the request business object to an appropriate document type for delivery to Peer-to-Peer Agent

Setting up event notification

Set up and configuration of the event notification mechanism for the iSoft connector requires completion of the following tasks:

- In the connector configuration property InputQueue, add the names of the queues that the connector will poll. For example, assuming that you have set up a queue named ″b_to_a″ for delivering data document messages from Peer-to-Peer Agent to your integration broker, and that you have set up queues named ″mdns″ and ″notifications.″ The connector will poll these queues; you add the names of the queues in the InputQueue property as follows:

  ```
  b_to_a;mdns;notifications
  ```

- Create a static meta-object that defines the business objects that the connector will receive, the queues from which they will be taken, and the data-handlers that will process the business objects from each queue. For each business object defined in the meta-object, specify values for the DataHandlerMimeType, the InputQueue, and the InputFormat. If any one of these values is not specified for a given business object, the connector will use the default data handler (as specified in the standard connector configuration properties) to process that business object.

  In general, all business objects that will be published as events (this includes MDN's, notifications, and any documents to be received from trading partners) should be defined in this static meta-object.

  Note: In this meta-object, you must specify every possible business object that the connector might need to convert from the messages that will appear in the specified queue. If the data handler attempts to convert a message to a business object that is not specified in this meta-object, the connector will fail.

For more information on defining and using a static meta-object, see "Configuring connector meta-objects" "Meta Object Attributes Configuration" later in this guide.

- Create a data-handler configuration meta-object for mapping of data-handler mime-types ("text/xml", "text/mdn", etc.) to actual data-handler classes. See the Data Handler Guide for structure and syntax. In the adapter configuration properties, set the value of DataHandlerConfigMO to the name of this meta-object.
- For all business objects that will be published as events, you can optionally include a dynamic meta-object containing an attribute—InputQueue—that tells the integration broker the name of the queue from which the event was retrieved. Depending on the configuration of Peer-to-Peer Agent, the name of the queue can provide information about the origin or purpose of the message. The value of the InputQueue attribute is populated when the connector publishes the event.

In addition to the above elements, your implementation should include collaborations (for InterChange Server Express) that subscribe to the business objects that will be published by the adapter.

Configuring connector meta-objects

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.

Configuring a static connector 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.

After you have created a static meta-object for the connector, make sure the connector subscribes to the static meta-object by specifying the name of the static meta-object in the connector-specific property DataHandlerConfigMO. For more information, see "Connector-specific properties" on page 13.

The static meta-object for the iSoft connector consists of a list of conversion properties defined for different business objects. To define the conversion properties for a business object, first create a string attribute and name it using the syntax `busObj_verb`. For example, to define the conversion properties for a Customer object with the verb Create, create an attribute named `Customer_Create`. In the application-specific text of the attribute, you specify the actual conversion properties.

Additionally, a reserved attribute named `Default` can be defined in the meta-object. When this attribute is present, its properties act as default values for all business object conversion properties. *Never assign default values to the InputFormat or InputQueue properties; the values of these properties are used to match incoming messsages to business objects only.*

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

The table below describes the meta-object properties.

Property name	Description
DataEncoding	`DataEncoding` is the encoding to be used to read and write messages. If this property is not specified in the static meta-object, the connector tries to read the messages without using any specific encoding. `DataEncoding` defined in a dynamic child meta-object overrides the value defined in the static meta-object. The default value is `Text`. The format for the value of this attribute is `messageType[:enc]`. I.e., `Text:ISO8859_1`, `Text:UnicodeLittle`, `Text`, or `Binary`.This property is related internally to the InputFormat property: specify one and only one `DataEncoding per InputFormat`.
DataHandlerConfigMO	Meta-object passed to data handler to provide configuration information. If specified in the static meta-object, this will override the value specified in the `DataHandlerConfigMO` connector property. Use this static meta-object property when different data handlers are required for processing different business object types. If defined in a dynamic child meta-object, this property will override the connector property and the static meta-object property. Use the dynamic child meta-object for request processing when the data format may be dependent on the actual business data. The specified business object must be supported by the connector.
DataHandlerMimeType	Allows you to request a data handler based on a particular MIME type. If specified in the static meta-object, this will override the value specified in the `DataHandlerMimeType` connector property. Use this static meta-object property when different data handlers are required for processing different business object types. For example, If defined in a dynamic child meta-object, this property will override the connector property and the static meta-object property. Use the dynamic child meta-object for request processing when the data format might be dependent on the actual business data. The business object specified in `DataHandlerConfigMO` should have an attribute that corresponds to the value of this property.
`DataHandlerClassName`	Name of the data handler class to use when converting messages to and from business objects.
`InputFormat`	The `InputFormat` is the message format to associate with the given business object. When a message is retrieved and is in this format, it is converted to the given business object if possible. Do not set this property using default conversion properties; its value is used to match incoming messages to business objects that will store message content.
`InputQueue`	The `InputQueue` is the input queue that the connector will poll to discover new messages received from Peer-to-Peer Agent. Do not set this property using default conversion properties; its value is used to match incoming messages to business objects.

Property name	Description
OutputFormat	The OutputFormat is set on messages created from the given business object. If the OutputFormat is not specified, the input format is used, if available. An OutputFormat defined in a dynamic child meta-object overrides the value defined in the static meta-object.
OutputQueue	The OutputQueue is the output queue to which messages derived from the given business object are delivered. An OutputQueue defined in a dynamic child meta-object overrides the value defined in the static meta-object.

Application-specific information

The application-specific information is structured in name-value pair format, separated by semicolons. For example:

```
InputFormat=CUST_IN;OutputFormat=CUST_OUT
```

Mapping data handlers to InputQueues

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

1. Use connector-specific properties (see "InputQueue" on page 15) to configure one or more input queues.
2. For each input queue, specify the queue manager and input queue name as well as data handler class name and mime type in the application-specific information.

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

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

A Sample connector meta-object

The static meta-object shown below configures the connector to handle the following:

- Catalog_Create business objects, which the connector places in the outbound queue to be picked up by Peer-to-Peer Agent
- Order_Create business objects, which the connector picks up when it polls an input queue
- MDN_Create and Notification_Create business objects, which the connector picks up when it polls the MDN and Notification input queues

Notice that in this sample meta-object, no data handler has been specified in the AppSpecificInfo of the attribute for the Sample_ISoft_Order_Create business object or for the Sample_ISoft_Catalog_Create business object. Because this sample meta-object does not contain the configuration values that would specify data handlers for those business objects, the data handler specified in the connector-specific properties will be used for them instead. However, in the attribute for the Sample_ISoft_MDN_Create business object and the Sample_ISoft_Notification_Create business object, this meta-object does provide configuration values in AppSpecificInfo to specify data handlers, and those values override the data handler values specified in the connector-specific properties.

```
[ReposCopy]
Version = 3.1.0
Repositories = 1cHyILNuPTc=
[End]
[BusinessObjectDefinition]
Name = Sample_ISoft_MO_Config
Version = 1.0.0

[Attribute]
Name = Default
Type = String
Cardinality = 1
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
AppSpecificInfo = OutputQueue=queue://isoft.queue.manager/
  COMPA.OUT;DataEncoding=Text
IsRequiredServerBound = false
[End]

[Attribute]
Name = Sample_ISoft_Order_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputQueue=queue://isoft.queue.manager/
  COMPA.IN
IsRequiredServerBound = false
[End]

[Attribute]
Name = Sample_ISoft_Catalog_Create
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = OutputFormat=application/xml;
IsRequiredServerBound = false
[End]

[Attribute]
Name = Sample_ISoft_MDN_Create
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputQueue=queue://isoft.queue.manager/
  COMPA.RECEIPTS;
DataHandlerClassName=com.crossworlds.DataHandlers.message
```

```
  .disposition_notification;
DataHandlerMimeType=message/disposition-notification
IsRequiredServerBound = false
[End]

[Attribute]
Name = Sample_ISoft_Notification_Create
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = DataHandlerMimeType=isoftp2p/command;
InputQueue=queue://isoft.queue.manager/COMPA.NOTICES;
DataHandlerClassName=com.crossworlds.DataHandlers.isoftp2p.command
IsRequiredServerBound = false
[End]

[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Verb]
Name = Create
[End]

[Verb]
Name = Delete
[End]

[Verb]
Name = Retrieve
[End]

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

Dynamic child connector meta-object

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

The table below shows a sample dynamic child meta-object for business object `Customer_Create`. Note that the application-specific information consists of semi-colon delimited name-value pairs.

Attribute name	Value
DataEncoding	Text:UnicodeLittle
DataHandlerMimeType*	text/delimited
OutputFormat	CUST_OUT
OutputQueue	QueueA

Attribute name	Value
JMSProperties	The iSoft Peer-to-Peer Agent sets the AS/2 “to” and “from” names associated with a given message in the MQ RFH2 header of messages it sends to the adapter. In order to have these populated in the business object, the user should define two child attributes in the “JMSProperties” object. The attribute names are insignificant; however the app-specific information of the attributes must contain name-value pairs `type=String;name=to` and `type=String;name=from` corresponding to the fields in the MQ RFH2 header that the iSoft Peer-to-Peer Agent populates with the AS/2 “to” and “from” names, respectively.

*Assumes that DataHandlerConfigMO has been specified in either the connector configuration properties or the static meta-object.

The connector checks the application-specific information of top-level business object received to determine whether tag `cw_mo_conn` specifies a child meta-object. If so, the dynamic child meta-object values override those specified in the static meta-object.

Population of the dynamic child meta-object during polling

In order to provide collaborations with more information regarding messages retrieved during polling, the connector populates specific attributes of the dynamic meta-object, if already defined for the business object created.

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

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

As shown in the table above, you can define an additional attribute, `InputQueue`, in a dynamic child meta-object. This attribute contains the name of the queue from which a given message has been retrieved. If this property is not defined in the child meta-object, it will not be populated.

Example scenario:

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

JMS headers, WebSphere MQ message properties, and dynamic child meta-object attributes

You can add attributes to a dynamic meta-object to gain more information about, and more control over, the message transport. Adding such attributes allows you

to modify JMS properties and to re-target a message CorrelationID. This section describes these attributes and how they affect event notification and request processing.

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

Table 1. Dynamic Meta-Object Header Attributes

Header attribute name	Mode
`CorrelationID`	Read/Write
`DeliveryMode`	Read/Write
`Priority`	Read/Write
`Destination`	Read
`Expiration`	Read
`MessageID`	Read
`Redelivered`	Read
`TimeStamp`	Read
`Type`	Read
`UserID`	Read
`AppID`	Read
`DeliveryCount`	Read
`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.

For example, if attribute `DeliveryMode` is present in the dynamic meta-object, the adapter updates this attribute with the value of the `JMSDeliveryMode` header field of the message. If `DeliveryMode` is not specified in the MO, then the JMS provider establishes the persistence setting.

Note: If attribute `DeliveryMode` is set to `2` in an asynchronous request, the message is sent persistently. If `DeliveryMode` is set to `1`, the message is not sent persistently. Any other value may fail the connector.

The interpretation and use of these attributes are described in the sections below.

Note: None of the above attributes are required. You may add any attributes to the dynamic meta-object that relate to your business process.

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

1. The name of the attribute has no semantic value.
2. The type of the attribute should always be `String` regardless of the JMS property type.
3. The application-specific information of the attribute must contain two name-value pairs defining the name and format of the JMS message property to which the attribute maps.
4. Values specified in the JMSProperty attribute are mapped to the MQ RHF2 message header, which can be used to exchange trading partner information and the iSoft Peer-to-Peer Agent.

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

Table 2. Application-Specific Information for JMS Property Attributes

Name	Possible values	Comments
Name	Any valid JMS property name	This is the name of the JMS property. Some vendors reserve certain properties to provide extended functionality. In general, users should not define custom properties that begin with `JMS` unless they are seeking access to these vendor-specific features.
Type	`String`, `Int`, `Boolean`, `Float`, `Double`, `Long`, `Short`	This is the type of the JMS property. The JMS API provides a number of methods for setting values in the JMS Message: `setIntProperty`, `setLongProperty`, `setStringProperty`, etc. The type of the JMS property specified here dictates which of these methods is used for setting the property value in the message.

The figure below shows the attribute `JMSProperties` in the dynamic meta-object and definitions for two properties in the MQ RFH2 header. These properties map to the MQ RFH2 fields ″to″ and ″from″ which the iSoft Peer-to-Peer Agent recognizes as containing trading partner information. Currently, the iSoft Peer-to-Peer Agent only recognizes these two fields in the MQ RFH2 header. In the future, if the iSoft Peer-to-Peer Agent is enhanced to recognized additional fields, they can be defined here in the JMSProperties object.

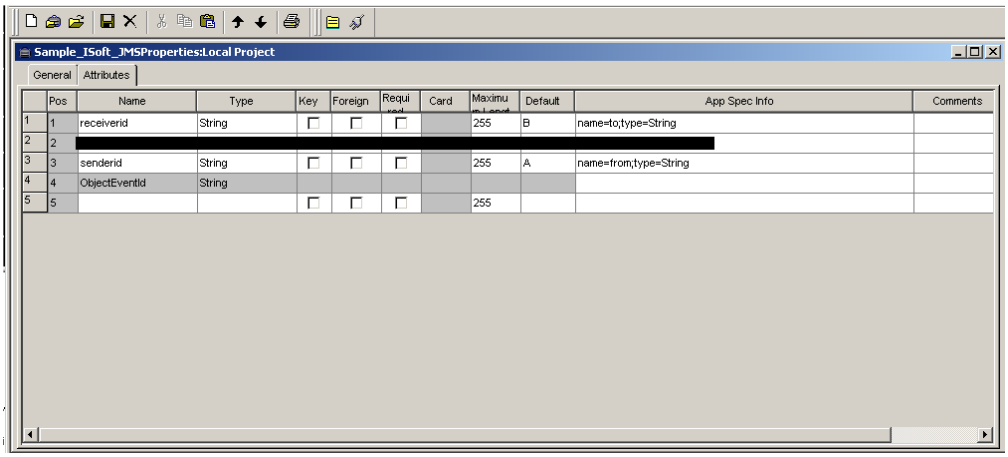

Figure 3. JMSProperties attribute in a dynamic meta-object

Startup file configuration

Before you start the connector, you must configure the startup file.

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

1. Open the `start_iSoft.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.

Startup

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

Chapter 3. Creating or modifying business objects

After installing the connector, you must create business objects.

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

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

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

Business object metadata includes the structure of a business object, the settings of its attribute properties, and the content of its application-specific 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 the iSoft connector, your modifications must conform to the rules the connector is designed to follow, or the connector cannot process new or modified business objects correctly.

The business objects that the connector processes can have any name allowed by the InterChange server Express. For more on naming conventions, see *Naming WebSphere InterChange Components*.

The connector retrieves messages from a queue and attempts to populate a business object (defined by the meta-object) with the message contents. The required business object structure is dependent on the meta-object definitinos as well as the connector's data handler requirements.

Connector business object structure

The structure of business objects must conform to the requirements of the data handler that converts them. The iSoft connector uses business objects that conform to the requirements of the iSoftNotification data handler, the iSoftMDN data handler, and the data handler that you configure for exchanging business data.

Business objects for notifications

The Peer-to-Peer Agent can be configured to generate notification messages to inform the user of retry attempts being exhausted. The Peer-to-Peer Agent notice

record format consists of values delimited primarily by US-ASCII unit-separator character 31 (represented by ″US″ in the syntax shown below). The syntax of the notice record is as follows:

```
HTTP/1.0 200 OK
Content-Type: isoftp2p/command
Content-Length: 337
CRLF
notice
CRLF
notice-id /*format of "YYYYMMDDHHMMSSRRRRRRRR"
  where "RRRRRRRR" is a random HEX string*/ US
op-code /*one of "SEND" or "RECV"*/ US
sender-AS2-name US
receiver-AS2-name US
sender-notify-name US
message-id US
message-subject US
message-digest US
transaction-begin-date-time US
transaction-end-date-time US
agent-role /*one of "A", "R" or "T"*/ US
batch-number US
bytes-in-count US
bytes-out-count US
result-code US
file-byte-count US
source-IP-address US
destination-IP-address US
source-IP-port US
destination-IP-port US
attempt-count US
total-attempts US
original-filename US
agent-name US
send-parameters US
```

Since the notice record format is not self-describing, the data-handler simply tokenizes the values and populates the given business object sequentially. This also means that the names used for the attributes in the business object will have no semantic value to the data-handler. As the data-handler reads the values in the notification message, it simply populates the next available attribute in the business object.

The following shows a sample business object structure, with attribute names taken from the notice record syntax shown above. All attributes in the object should be of type String.

```
Object:
P2PAgentNotificationMessage

Attributes:
NoticeId
OpCode
SenderAS2Name
. . .
SendParameters
```

Note that it is mandatory that each of the delimited values of the notice record be included as an attribute in the business object, and in the order shown in the notice record.

Business objects for MDNs

The attributes of the business object for MDN correspond to data fields in the machine-readable message body of an MDN sent from Peer-to-Peer Agent to the connector.

To map fields in the MDN to business object attributes, the data handler looks at name-value pairs in the application-specific info of the attributes of the business object. The name value pairs use this syntax: name=<MDN field name>.

For example, for the following fields from a sample MDN message body:

```
Reporting-UA:
Final-Recipient:
Original-Message-Id:
Disposition:
```

The business object would have attributes with the following application-specific info:

You can control which data gets mapped from the MDN to the business object by adding or deleting attributes from the business object. You can use any name you like for the business object attribute, but the Application-Specific Info that you provide for the attribute must specify exactly the name of a field in the MDN message body.

Business objects for exchanging business data

The business data in messages received from Peer-to-Peer Agent can be in virtually any format, and multiple data handlers may be required. The business objects that you define for exchanging business data must conform to the requirements of whichever data handlers you configure.

One common format for business data exchange is the XML document format. For information about creating business objects that conform to data handlers for this and other formats, see the *Data Handler Guide*.

Error handling

All error messages generated by the connector are stored in a message file named `iSoftConnector.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 successfully converts a business object to a message but cannot deliver it the outgoing queue due to connection loss.

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 connector,” on page 7 for more on configuring trace messages. For more information on tracing, including how to enable and set it, see the Connector Configurator Express appendix.

What follows is recommended content for connector trace messages.

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

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

Level 2 — Use this level for trace messages that log each time a business

	object is posted to InterChange Server Express, either from `gotApplEvent()` or `executeCollaboration()`.
Level 3	Use this level for trace messages that provide information regarding message-to-business-object and business-object-to-message conversions or provide information about the delivery of the message to the output queue.
Level 4	Use this level for trace messages that identify when the connector enters or exits a function.
Level 5	Use this level for trace messages that indicate connector initialization, represent statements executed in the application, indicate whenever a message is taken off of or put onto a queue, or record business object dumps.

Chapter 4. Troubleshooting

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

Start-up problems

Problem	Potential solution / explanation
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/jms/JMSException...`	Connector cannot find file `jms.jar` from the IBMWebSphere MQ Java client libraries. Ensure that variable `MQSERIES_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 `MQSERIES_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 `MQSERIES_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 connector," on page 7 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 the adapters in WebSphere Business Integration Express for Item Synchronization, running on WebSphere InterChange Server Express.

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

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

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

This section describes the standard configuration properties. For information on configuration properties specific to a connector, see its adapter user guide.

Using Connector Configurator Express

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

Setting and updating property values

The default length of a property field is 255 characters.

The connector uses the following order to determine a property's value (where the highest number overrides other values):

1. Default
2. Repository
3. Local configuration file
4. Command line

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a run-time session, the property's **Update Method** determines how the change takes effect. There are four different update methods for standard connector properties:

- **Dynamic**
  The change takes effect immediately after it is saved in System Manager.
- **Component restart**
  The change takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the application-specific component or InterChange Server Express.

- **Agent restart**
  The change takes effect only after you stop and restart the application-specific component.

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator Express window, or see the Update Method column in the Property Summary table below.

Summary of standard properties

Table 3 provides a quick reference to the standard connector configuration properties.

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

Table 3. Summary of standard configuration properties

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

Table 3. Summary of standard configuration properties (continued)

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

Table 3. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
OADAutoRestartAgent	`True` or `False`	`False`	Dynamic	Repository Directory is <REMOTE>
OADMaxNumRetry	*A positive number*	`1000`	Dynamic	Repository Directory is <REMOTE>
OADRetryTimeInterval	*A positive number in minutes*	`10`	Dynamic	Repository Directory is <REMOTE>
PollEndTime	`HH:MM`	`HH:MM`	Component restart	
PollFrequency	*a positive integer in milliseconds* `no` (to disable polling) `key` (to poll only when the letter `p` is entered in the connector's Command Prompt window)	`10000`	Dynamic	
PollQuantity	1-500	1	Component restart	JMS transport only: DuplicateEvent Elimination must be True
PollStartTime	`HH:MM`*(HH is 0-23, MM is 0-59)*	`HH:MM`	Component restart	
RepositoryDirectory	Location of metadata repository	<remote>	Agent restart	
RequestQueue	*Valid JMS queue name*	*CONNECTORNAME*`/REQUESTQUEUE`	Component restart	
ResponseQueue	*Valid JMS queue name*	*CONNECTORNAME*`/RESPONSEQUEUE`	Component restart	Delivery transport is JMS
RestartRetryCount	`0-99`	`3`	Dynamic	
RestartRetryInterval	A sensible positive value in minutes 1 - 2147483547:	1	Dynamic	
SourceQueue	Valid WebSphere MQ name	*CONNECTORNAME*`/SOURCEQUEUE`	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue		*CONNECTORNAME*`/` `SYNCHRONOUSREQUESTQUEUE`	Component restart	Delivery transport is JMS
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery transport is JMS
SynchronousResponseQueue		*CONNECTORNAME*`/` `SYNCHRONOUSRESPONSEQUEUE`	Component restart	Delivery transport is JMS

Table 3. Summary of standard configuration properties (continued)

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

Standard configuration properties

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

AdminInQueue

The queue that is used by InterChange Server Express to send administrative messages to the connector.

The default value is `CONNECTORNAME/ADMININQUEUE`.

AdminOutQueue

The queue that is used by the connector to send administrative messages to InterChange Server Express.

The default value is `CONNECTORNAME/ADMINOUTQUEUE`.

AgentConnections

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

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

AgentTraceLevel

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

ApplicationName

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

CharacterEncoding

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

Note: Java-based connectors do not use this property. A C++ connector currently uses the value ASCII for this property. If you previously configured the value of this property to `ascii7` or `ascii8`, you must reconfigure the connector to use either `ASCII` or one of the other supported values.

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

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

The default value is ascii.

ConcurrentEventTriggeredFlows

Determines how many business objects can be concurrently processed by the connector for event delivery. Set the value of this attribute to the number of business objects you want concurrently mapped and delivered. For example, set the value of this property to 5 to cause five business objects to be concurrently processed. The default value is 1.

Setting this property to a value greater than 1 allows a connector for a source application to map multiple event business objects at the same time and deliver them to multiple collaboration instances simultaneously. This speeds delivery of business objects to Interchange Server Express, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

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

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

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

ContainerManagedEvents

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

The default value is JMS. It can also be set to no value.

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

- PollQuantity = 1 to 500
- SourceQueue = SOURCEQUEUE

You must also configure a data handler with the MimeType, DHClass, and DataHandlerConfigMOName (optional) properties. To set those values, use the **Data Handler** tab in Connector Configurator Express. The fields for the values under the Data Handler tab will be displayed only if you have set ContainerManagedEvents to JMS.

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

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

ControllerStoreAndForwardMode

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

If this property is set to true and the destination application-specific component is unavailable when an event reaches Interchange Server Express, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forwards the request to it.

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

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

The default is true.

ControllerTraceLevel

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

DeliveryQueue

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

The default value is DELIVERYQUEUE.

DeliveryTransport

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

The connector sends service call requests and administrative messages over CORBA IIOP if the value configured for the DeliveryTransport property is MQ or IDL.

WebSphere MQ and IDL

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

- Asynchronous communication:
  WebSphere MQ allows the application-specific component to poll and persistently store events even when the server is not available.
- Server side performance:
  WebSphere MQ provides faster performance on the server side. In optimized mode, WebSphere MQ stores only the pointer to an event in the repository database, while the actual event remains in the WebSphere MQ queue. This saves having to write potentially large events to the repository database.

- Agent side performance:
  WebSphere MQ provides faster performance on the application-specific component side. Using WebSphere MQ, the connector's polling thread picks up an event, places it in the connector's queue, then picks up the next event. This is faster than IDL, which requires the connector's polling thread to pick up an event, go over the network into the server process, store the event persistently in the repository database, then pick up the next event.

JMS

Enables communication between the connector and client connector framework using Java Messaging Service (JMS).

If you select JMS as the delivery transport, additional JMS properties such as `jms.MessageBrokerName`, `jms.FactoryClassName`, `jms.Password`, and `jms.UserName`, appear in Connector Configurator Express. The first two of these properties are required for this transport.

Important: There may be a memory limitation if you use the JMS transport mechanism for a connector running on InterChange Server Express.

In this environment, you may experience difficulty starting both the connector controller (on the server side) and the connector (on the client side) due to memory use within the WebSphere MQ client.

DuplicateEventElimination

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

This property can also be set to `false`.

Note: When `DuplicateEventElimination` is set to `true`, you must also configure the `MonitorQueue` property to enable guaranteed event delivery.

FaultQueue

If the connector experiences an error while processing a message then the connector moves the message to the queue specified in this property, along with a status indicator and a description of the problem.

The default value is `CONNECTORNAME/FAULTQUEUE`.

JvmMaxHeapSize

The maximum heap size for the agent (in megabytes). This property is applicable only if the `RepositoryDirectory` value is <REMOTE>.

The default value is 128m.

JvmMaxNativeStackSize

The maximum native stack size for the agent (in kilobytes). This property is applicable only if the `RepositoryDirectory` value is <REMOTE>.

The default value is 128k.

JvmMinHeapSize

The minimum heap size for the agent (in megabytes). This property is applicable only if the `RepositoryDirectory` value is <REMOTE>.

The default value is 1m.

jms.FactoryClassName

Specifies the class name to instantiate for a JMS provider. You *must* set this connector property when you choose `JMS` as your delivery transport mechanism (DeliveryTransport).

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

jms.MessageBrokerName

Specifies the broker name to use for the JMS provider. You *must* set this connector property when you choose `JMS` as your delivery transport mechanism (DeliveryTransport).

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

ListenerConcurrency

This property supports multi-threading in MQ Listener for InterChange Server Express. It enables batch writing of multiple events to the database, thus improving system performance. The default value is 1.

This property applies only to connectors using MQ transport. The `DeliveryTransport` property must be set to `MQ`.

Locale

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

A locale name has the following format:

ll_TT.codeset

where:

ll	a two-character language code (usually in lower case)
TT	a two-letter country or territory code (usually in upper case)
codeset	the name of the associated character code set; this portion of the name is often optional.

By default, only a subset of supported locales appears in the drop list. To add other supported values to the drop list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, see the appendix on Connector Configurator Express.

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

LogAtInterchangeEnd

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

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

MaxEventCapacity

The maximum number of events in the controller buffer. This property is used by flow control and is applicable only if the value of the `RepositoryDirectory` property is <REMOTE>.

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

MessageFileName

The name of the connector message file. The standard location for the message file is `\connectors\messages`. Specify the message filename in an absolute path if the message file is not located in the standard location.

If a connector message file does not exist, the connector uses `InterchangeSystem.txt` as the message file. This file is located in the product directory.

Note: To determine whether a specific connector has its own message file, see the individual adapter user guide.

MonitorQueue

The logical queue that the connector uses to monitor duplicate events. It is used only if the `DeliveryTransport` property value is JMS and `DuplicateEventElimination` is set to `TRUE`.

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

The Repository Directory must be set to <REMOTE>.

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

The default value is false.

OADMaxNumRetry

The Repository Directory must be set to <REMOTE>.

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

The default value is 1000.

OADRetryTimeInterval

The Repository Directory must be set to <REMOTE>.

Specifies the number of minutes for the interval during which the OAD automatically attempts to restart the application-specific component after an abnormal shutdown. If the application-specific component does not start within the specified interval, the OAD repeats the attempt as many times as specified in "OADMaxNumRetry."

The default is 10.

PollEndTime

Time to stop polling the event queue. The format is HH:MM, where *HH* represents 0-23 hours, and *MM* represents 0-60 seconds.

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

PollFrequency

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

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

The default is 10000.

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

PollQuantity

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

PollStartTime

The time to start polling the event queue. The format is *HH:MM*, where *HH* represents 0-23 hours, and *MM* represents 0-60 seconds.

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

RequestQueue

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

The default value is `REQUESTQUEUE`.

RepositoryDirectory

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

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

ResponseQueue

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

RestartCount

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

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

RestartRetryCount

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

The default is 3.

RestartRetryInterval

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

The default is 1.

SourceQueue

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 44.

The default value is `SOURCEQUEUE`.

SynchronousRequestQueue

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

SynchronousResponseQueue

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

SynchronousRequestTimeout

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

The default value is 0.

WireFormat

Message format on the transport. The setting is `CwBO`.

Appendix B. Connector Configurator Express

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

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

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

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

You use Connector Configurator Express to:

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

The topics covered in this appendix are:

Overview of Connector Configurator Express

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

You use Connector Configurator Express to:

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

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

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

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

Starting Connector Configurator Express

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

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

Running Configurator Express in stand-alone mode

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

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

If you are creating a configuration file, you may prefer to run Connector Configurator Express independently to generate the file, and then connect to System Manager to save it in an InterChange Server Express project (see "Completing a configuration file" on page 59.)

Running Configurator Express from System Manager

You can run Connector Configurator Express from System Manager.

To run Connector Configurator Express:

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

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 55.
- To use an existing file, simply modify an existing template and save it under the new name.

Note: Connector-specific templates are provided for the iSoft, JText, and e-Mail connectors only. If you are configuring one of these connectors, see the *Quick Start Guide*, or skip this section and go to "Creating a new configuration file" on page 57.

Creating a new template

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

To create a template:

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

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

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

If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one. Connector Configurator Express Express provides a template named **None**, containing no property definitions, as a default choice.

Specifying general characteristics

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

- **Edit properties**

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

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

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

- Flags

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

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

Specifying values

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

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

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

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

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

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

Setting dependencies

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

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

< (less than)

>= (greater than or equal to)

<=(less than or equal to)

4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator Express stores the information you have entered as an XML document, under `\data\app` in the`\bin` directory where you have installed Connector Configurator Express.

Creating a new configuration file

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

Creating a configuration file from a connector-specific template

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

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

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

     Important: Connector Configurator Express does not check the spelling of the name that you enter. You must ensure that the name is correct.
   - **Select Connector-Specific Property Template**

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

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

Important: The directory path and name that you establish here must match the connector configuration file path and name that you supply in the startup file for the connector.

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

Using an existing file

To use an existing file to configure a connector, you must open the file in Connector Configurator Express, revise the configuration, and then save the file as a configuration file (`*.cfg` file).

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

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

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

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

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

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

Opening an existing file from System Manager

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

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

To edit an existing configuration file:

1. In the System Manager window, select any of the configuration files listed in the **Connector** folder and right-click on it. Connector Configurator Express opens and displays the configuration file with the file name at the top.
2. Click the **Properties** tab to see which properties are included in this configuration file.

Completing a configuration file

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

Connector Configurator Express requires values for properties described in the following sections:

- "Setting standard connector properties"
- "Setting connector-specific configuration properties" on page 60
- "Specifying supported business object definitions" on page 60
- "Associated maps" on page 62
- "Setting trace/log file values" on page 63
- "Configuring messaging" on page 63

Note: For connectors that use JMS messaging, an additional category may display, for special configuration of data handlers that convert the data to business objects. For further information, see "Data handlers" on page 63.

Setting the configuration file properties

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

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.
- A field with a white background indicates an application-specific property. These properties vary according to the specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.
- The **Update Method** field is informational and not configurable. This field specifies the action required to activate a property whose value has changed.

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.
   - To set values for standard property values for your connector, see the Standard Properties appendix of this guide.
3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator Express, click **File > Exit** (or close the window), and click **No** when prompted to save changes.
   - To enter values for other categories in Connector Configurator Express, select the tab for the category. The values you enter for **Standard Properties** (or

any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.

- To save the revised values, click **File > Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save > To File** from either the File menu or the toolbar.

Setting connector-specific configuration properties

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

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property or **Add Child** to add a child property to a property.
2. Enter a value for the property or child property.
   - To set values for connector-specific property values for your connector, **see the connector-specific properties section of this guide**.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for "Setting standard connector properties" on page 59.

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

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

Encryption for connector properties

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

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

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

Update method

Connector properties are almost all static and the **Update Method** is `Component restart`. For changes to take effect, you must restart the connector after saving the revised connector configuration file. For further information, see *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Specifying supported business object definitions

Use the **Supported Business Objects** tab in Connector Configurator Express to specify the business objects that the connector will use. You must specify both

generic business object definitions and application-specific business object definitions, and you must specify associations for the maps between the business objects.

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

Note: Some connectors require that certain business objects be specified as supported in order to perform event notification or additional configuration (using meta-objects) with their applications. For more information, see the chapter on business objects in this guide as well as the *Business Object Development Guide*.

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

Business object name

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

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

To delete a business object from the supported list:

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

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

Agent support: If a business object has Agent Support, the system will attempt to use that business object for delivering data to an application via the connector. Typically, application-specific business objects for a connector are supported by that connector's agent, but generic business objects are not.

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

Maximum transaction level

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

For most connectors, `Best Effort` is the only possible choice, because most application APIs do not support the `Stringent` level.

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

Associated maps

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

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

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

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

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

- **Business Object Name**

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

- **Associated Maps**

  The display shows all the maps that have been installed to the system for use with the supported business objects of the connector. The source business object for each map is shown to the left of the map name, in the **Business Object Name** display. To display the maps, you must first designate the supported business objects, and then save the connector configuration to project. To see the maps, you must first designate the supported business objects and save the connector configuration to project.

- **Explicit**

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

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

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

To explicitly bind a map:

1. In the **Explicit** column, place a check in the check box for the map you want to bind.
2. Select the map that you intend to associate with the business object.

Configuring messaging

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

Setting trace/log file values

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

To change the logging and tracing values:

1. Click the **Trace/Log Files** tab.
2. For either logging or tracing, you can choose to write messages to one or both of the following:
   - To console (STDOUT):
     Writes logging or tracing messages to the STDOUT display.
   - To File:
     Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

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

Data handlers

The data handlers section is available for configuration only if you have designated a value of JMS for DeliveryTransport and a value of JMS for ContainerManagedEvents. Adapters that make use of the guaranteed event delivery enable this tab.

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

Saving your configuration file

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

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

Completing the configuration

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

To do so, open the startup file used for the connector, and verify that the location and file name used for the connector configuration file match exactly the name you have given the file and the directory or path where you have placed it. for more information on the startup file, see the appropriate section of your adapter user guide as well as the *User Guide for IBM WebSphere Business Integration Express for Item Synchronization*.

Appendix C. Connector feature list

This appendix details the features supported by the connector.

Business object request handling features

The table below details the business object request handling features supported by the connector.

Category	Feature	Support	Notes
Create	Create verb	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
Delete	Delete verb	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	Logical delete	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
Exist	Exist verb	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
Misc	Attribute names	Full	
	Business object names	Full	
Retrieve	Ignore missing child object	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
RetrieveByContent	Ignore missing child object	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	Multiple results	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	RetrieveByContent verb	N/A	The connector supports RetrieveByContent verb in full when using synchronous request/response.
Update	After-image support	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	Delta support	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	KeepRelations	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
Verbs	Retrieve verb	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	Subverb support	Partial	Depends on data handler chosen for the connector.
	Verb stability	Full	

Event notification features

The table below details the event notification features supported by the connector.

Category	Feature	Support	Notes
Connector Properties	Event distribution	No	
	PollQuantity	Full	
Event Table	Event status values	N/A	
	Object key	N/A	
	Object name	N/A	
	Priority	Full	Connector retrieves messages based on the priority specified in their message header (range 0-9).
Misc.	Archiving	Full	Connector can deliver copies of messages to different queues depending on whether the message was unsubscribed, was successfully processed, or resulted in errors.
	CDK method gotApplEvent	Full	
	Delta event notification	No	
	Event sequence	Full	
	Future event processing	No	
	In-Progress event recovery	Full	
	Physical delete event	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	RetrieveAll	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	Smart filtering	No	
	Verb stability	N/A	

General features

The table below details the general features supported by the connector.

Category	Feature	Support	Notes
Business Object Attributes	Foreign key	No	
	Foreign Key attribute property	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.
	Key	No	
	Max Length	Partial	May be used by the data handler chosen for the connector.
	metadata-driven design	Full	
	Required	No	
Connection Lost	Connection lost on poll	No	Connector may report an error or may continue polling and not retrieve events. The adapter listens for errors reported by MQSeries—whether such errors are generated depends solely on MQSeries .
	Connection lost on request processing	Full	
	Connection lost while idle	No	

Category	Feature	Support	Notes
Connector Properties	ApplicationPassword	No	MQSeries does not use password for authentication.
	ApplicationUserName	Partial	Connector sends message under authority of specified local user.
	UseDefaults	Partial	Depends on the data handler established for the connector.
Message Tracing	General messaging	Full	
	generateMsg()	Full	
	Trace level 0	Full	
	Trace level 1	Full	
	Trace level 2	Full	
	Trace level 3	Full	
	Trace level 4	Full	
	Trace level 5	Full	
Misc.	CDK method LogMsg	Full	
	Java Package Names	Full	
	Logging messages	Full	
	NT service compliance	Full	
	Transaction support	Full	
Special Value	CxBlank processing	Partial	Depends on the data handler chosen for the connector
	CxIgnore processing	N/A	Behavior is entirely dependent upon how Peer-to-Peer Agent and the target trading partner interpret the request.

Notices

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

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

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

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

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

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

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

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

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

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

Burlingame, CA 94010
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

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

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

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

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

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

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