

WebSphere Business Integration Adapters



Adapter for iSoft Commerce Suite Server User Guide

V14.0

Note!

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

25June2004

This edition of this document applies to the Adapter for iSoft Commerce Server Suite, version 1.4.0.

To send us your comments about IBM WebSphere Business Integration documentattion, 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.

© **Copyright International Business Machines Corporation 2004. All rights reserved.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

New in This Release	v
New in Release 1.4.x.	v
New in Release 1.3.x.	v
New in Release 1.3.0.	v
New in Release 1.2.x	vi
New in Release 1.1.x	vi
New in Release 1.0.x	vi
About this document	vii
Audience	vii
Prerequisites for this document.	vii
Related documents.	vii
Typographic conventions	viii
Chapter 1. Overview	1
Connector architecture	1
Application-connector communication method	3
Event notification and handling	4
Business object requests	6
Verb processing	6
Chapter 2. Installing the adapter	7
Adapter environment	7
Overview of installation tasks	8
Installing the connector and related files.	11
Installed file structure	11
Chapter 3. Configuring the adapter	15
Connector configuration properties	15
Setting up queues	19
Configuring data handlers	22
Setting up request processing	24
Setting up event notification.	25
Configuring connector meta-objects	26
Creating multiple instances of the connector	33
Startup file configuration	35
Starting the connector	35
Stopping the connector	36
Sample dynamic child meta-object.	36
Chapter 4. Creating or modifying business objects.	41
Introduction	41
Connector business object structure	41
Error handling	43
Tracing	44
Chapter 5. Troubleshooting	47
Start-up problems	47
Event processing	47
Appendix A. Standard configuration properties for connectors	49
New and deleted properties	49
Configuring standard connector properties	49
Summary of standard properties	50

Standard configuration properties	55
Appendix B. Connector Configurator.	67
Overview of Connector Configurator	67
Starting Connector Configurator	68
Running Configurator from System Manager	68
Creating a connector-specific property template	69
Creating a new configuration file	71
Using an existing file	72
Completing a configuration file.	73
Setting the configuration file properties	74
Saving your configuration file	79
Changing a configuration file	80
Completing the configuration	80
Using Connector Configurator in a globalized environment	80
Appendix C. Tutorial	83
Prerequisites	83
Setting up your environment	84
Running the scenarios.	86
Notices	91
Programming interface information	92
Trademarks and service marks	93

New in This Release

New in Release 1.4.x

The name Adapter for iSoft Peer-to-Peer Agent has been replaced with Adapter for iSoft Commerce Suite Server.

Connector definition file format has changed from .txt to .xsd.

Two connector-specific properties have been added: `EnableMessageProducerCache` and `SessionPoolSizeForRequests`. For further information, see “Connector-specific properties” on page 16.

When processing a response message to a synchronous request, the connector interprets the feedback code `MQFB_NONE` (the default feedback code if non is set) as `VALCHANGE`. For further information, see the *Adapter for WebSphere MQ User Guide*.

The `MessageFormatProperty` connector property has been added to “Connector-specific properties” on page 16. For use with the Adapter for iSoft Commerce Suite server, this property must be set to `JMSType`.

As of version 1.4.x of the Adapter for iSoft Commerce Suite server, the meta-object ASI property `inputFormat` should no longer be used with this adapter.

New in Release 1.3.x

The adapter can now use WebSphere Application Server as an integration broker.

Solaris 7.0 is no longer supported.

New in Release 1.3.0

February 2004

Some instances of `p2pagent` have been replaced with `icssvr`.

Instances of the queue manager have been renamed to `wbia.queue.manager`.

Note: As of version 3.2, the product previously known as the iSoft Peer-to-Peer Agent has been renamed to the iSoft Commerce Suite server. In this adapter document both names may be used interchangeably to refer to the same application.

December 2003

The adapter for iSoft Peer-to-Peer Agent supports iSoft Commerce Suite version 3.2.

Beginning with version 1.3.0, the adapter is no longer supported on Microsoft Windows NT.

New in Release 1.2.x

The adapter can now use WebSphere Application Server as an integration broker.

The adapter now runs on the following platforms:

- Solaris 7, 8
- AIX 5.1

New in Release 1.1.x

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

New in Release 1.0.x

This is the first release of the iSoft adapter.

The connector supports interoperability with applications via MQSeries 5.1, 5.2, and 5.3.

The connector now has a UseDefaults property for business object processing. For more information, see "UseDefaults" on page 19.

The connector can now apply a default verb when the data handler does not explicitly assign one to a business object. For more information, see "DefaultVerb" on page 17.

The ReplyToQueue can now be dictated via the dynamic child meta-object rather than by the ReplyToQueue connector property. For more information see "JMS headers, WebSphere MQ message properties, and dynamic child meta-object attributes" on page 31.

About this document

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

This document describes the installation, configuration, and business object development for the adapter for iSoft Commerce Suite server.

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 the iSoft Commerce Suite server.

Related documents

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

You can install related documentation from the following sites:

- For general adapter information; for using adapters with WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker); and for using adapters with WebSphere Application Server, see the IBM WebSphere Business Integration Adapters InfoCenter:
<http://www.ibm.com/websphere/integration/wbiadapters/infocenter>
- For using adapters with WebSphere InterChange Server, see the IBM WebSphere InterChange Server InfoCenters:
<http://www.ibm.com/websphere/integration/wicserver/infocenter>
<http://www.ibm.com/websphere/integration/wbicollaborations/infocenter>
- For more information about WebSphere message brokers:
<http://www.ibm.com/software/integration/mqfamily/library/manualsa/>
- For more information about WebSphere Application Server:
<http://www.ibm.com/software/webservers/appserv/library.html>

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

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

Select the component area of interest and browse the Technotes and Flashes sections.

Typographic conventions

This document uses the following conventions:

<code>courier font</code>	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.
<i>italic</i>	Indicates a variable name or a cross-reference.
<i>blue outline</i>	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, <code>option[,...]</code> 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 <code><server_name><connector_name>tmp.log</code> .
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the product is installed on your system.
<code>%text%</code> and <code>\$text</code>	Text within percent (%) signs indicates the value of the Windows text system variable or user variable. The equivalent notation in a UNIX environment is <code>\$text</code> , indicating the value of the <code>text</code> UNIX environment variable.
<code>ProductDir</code>	Represents the directory where the IBM WebSphere Business Integration Adapters product is installed.

Chapter 1. Overview

The connector for iSoft^R Commerce Suite ServerTM is a runtime component of the WebSphere Business Integration Adapter for iSoft Commerce Suite server. The connector allows the WebSphere integration broker to send business data to, and receive business data from, the iSoft Commerce Suite server.

This chapter describes the connector component and the relevant business integration system architecture. It covers the following topics:

- “Connector architecture”
- “Application-connector communication method” on page 3
- “Event notification and handling” on page 4
- “Business object requests” on page 6
- “Verb processing” on page 6

Note: As of version 3.2, the product previously known as the iSoft Peer-to-Peer Agent has been renamed to the iSoft Commerce Suite server.

Connector 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

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 assured event delivery.

The connector allows a WebSphere integration broker (which can be either the InterChange Server or the WebSphere MQ implementation) to asynchronously exchange business objects with iSoft Commerce Suite server.

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

- For messages incoming from Commerce Server: The connector monitors a set of queues that are populated with messages received from Commerce Server. The set includes one or more queues for business data messages originating from trading partners, a queue for notifications that are generated by Commerce Server, and a queue for MDNs sent from the trading partners in response to messages that originated from the integration broker. When Commerce Server 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 the ICS integration broker is being used, the business object is published to those collaborations that have subscribed to it.)

- For business data outgoing to Commerce Server: 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 Commerce Server. Commerce Server 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 Commerce Server, it returns successfully, without waiting to determine whether the message that it delivered to Commerce Server was sent to the Commerce Server trading partner. And when the connector receives an event from Commerce Server, 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 Commerce Server) and for event notification (converting a message from Commerce Server 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 an input queue, the connector looks up the business object name associated with that input queue and with the FORMAT field contained in the message header. The message body, along with a new instance of the appropriate business object, is then passed to the data handler. If a business object is successfully populated with message content, the connector checks to see if it is subscribed, and then delivers it to the InterChange Server using the `gotAppEvents()` 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 in the ICS), 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. Commerce Server monitors the queue, picks up the message, and sends it to the targeted trading partner.

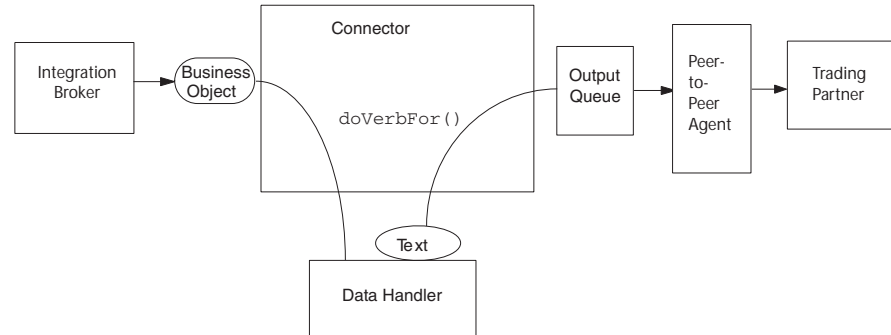


Figure 1. Application-Connector Communication Method: Message Request

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 `gotAppEvents()` method delivers the business object to InterChange Server, and the message is removed from the in-progress queue.

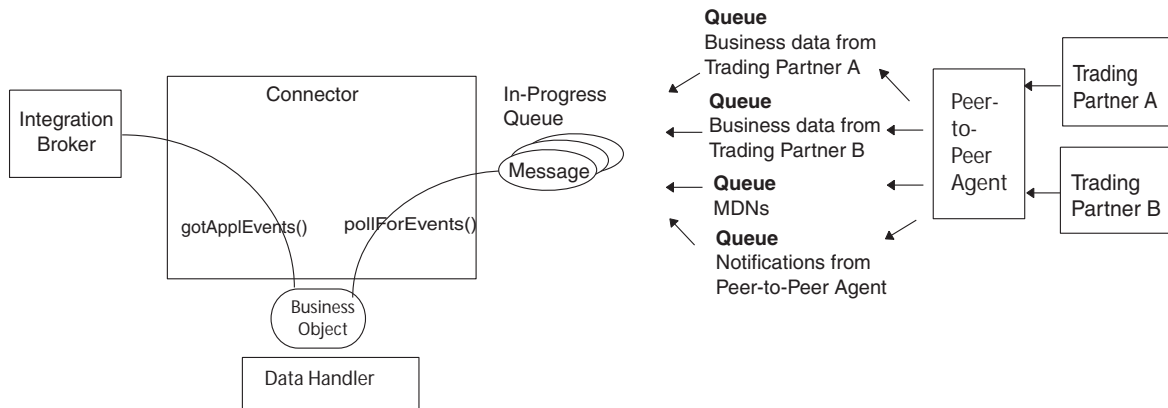


Figure 2. Application-Connector Communication Method: Event Delivery

Event notification and handling

The connector monitors one or more queues that Commerce Server populates with these types of messages:

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

An event occurs whenever Commerce Server 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 43 for event failure scenarios.

The connector processes messages by first opening a transactional session to the input queue. This transactional approach allows for a small chance that a business

object could be delivered to a collaboration twice due to the connector successfully submitting the business object but failing to commit the transaction in the queue. To avoid this problem, the connector moves all messages to an in-progress queue. There, the message is held until processing is complete. If the connector shuts down unexpectedly during processing, the message remains in the in-progress queue instead of being reinstated to the original input queue.

Note: Transactional sessions with a JMS service provider require that every requested action on a queue be performed and committed before events are removed from the queue. Accordingly, when the connector retrieves a message from the queue, it does not commit to the retrieval until three things occur: 1) The message has been converted to a business object; 2) the business object is delivered to InterChange Server by the `gotAppEvents()` 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 43 in Chapter 4, “Creating or modifying business objects,” on page 41

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 the adapter

This chapter describes how to install the adapter. It covers the following topics:

- “Adapter environment”
- “Overview of installation tasks” on page 8
- “Installing the connector and related files” on page 11
- “Installed file structure” on page 11

Adapter environment

Before installing, configuring, and using the adapter, you must understand its environment requirements. They are listed in the following section.

- “Broker compatibility”
- “Adapter platforms”
- “Globalization” on page 8

Broker compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. Version 1.4.0 of the adapter for iSoft Commerce Suite server is supported on the following adapter framework and integration brokers:

- **Adapter framework:**
WebSphere Business Integration Adapter Framework versions 2.1, 2.2, 2.3.x, and 2.4.
- **Integration brokers:**
 - WebSphere InterChange Server, versions 4.1.1, 4.2, 4.2.1, 4.2.2
 - WebSphere MQ Integrator, version 2.1.0
 - WebSphere MQ Integrator Broker, version 2.1.0
 - WebSphere Business Integration Message Broker, version 5.0
 - WebSphere Application Server Enterprise, version 5.0.2, with WebSphere Studio Application Developer Integration Edition, version 5.0.1

See *Release Notes* for any exceptions.

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

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

For WebSphere message brokers, see *Implementing Adapters with WebSphere Message Brokers*.

For WebSphere Application Server, see *Implementing Adapters with WebSphere Application Server*.

Adapter platforms

The adapter is supported on the following software.

Operating systems:

- AIX 4.3.3, AIX 5.1, AIX 5.2
- Solaris 8.0

- Windows 2000

Databases:

- DB2
- Oracle 8.1.7.4, Oracle 9.2
- Microsoft Access

Third-party software:

- iSoft Commerce Suite version 3.1

Adapter dependencies

The adapter for iSoft requires the following software.

- For interoperability with iSoft Commerce Suite server, 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.

- You must have the IBM WebSphere MQ Java client libraries, version 5.2.2.

Globalization

This adapter is not DBCS (double-byte character set)-enabled or translated.

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 in the installation documentation for your broker and operating system.

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 11.

Configure the Commerce Server application

Configuration of iSoft Commerce Suite server is done through an XML configuration file. By default, Commerce Server looks for the file `icssvr.cfg` in the local directory and executes any commands found inside. An iSoft Commerce Suite server 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.

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.

In the following scenario, trading partner B sends an order to trading partner A. The iSoft Commerce Suite server 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 `icssvr.cfg_a` route these messages to the `COMPAXML` queue. iSoft Commerce Suite server A can also route any messages with a MIME type set to `text/plain` to `COMPATXT`. 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, `COMPAXML`, for instance, or a delimited data handler could be configured to process all messages in a text queue, like `COMPATXT`. Use the following commands in the `icssvr.cfg` file.

- To transport a file from `icssvrB` to `icssvrA` and have `icssvrA` place the file in the `COMPAXML` queue.
`send http B A -cX -fNoutboxB\toA\Order1.txt`
- To transport a file from `icssvrB` to `icssvrA` and have `icssvrA` place the file in the `COMPATXT` 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 Commerce Server is named `wbia.queue.manager`.
- The integration broker is acting on behalf of trading partner **A**, to exchange XML documents with Commerce Server, 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 Commerce Server to Trading Partner **B**.
- Queue `"b_to_a"` is where the Commerce Server 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 Commerce Server will put any MDN's it receives from Trading Partner **B**, for delivery to Trading Partner **A**
- Queue `"notifications"` is where the Commerce Server will put any notifications it generates for Trading Partner **A**

```
<xml>
```

```
<command>set -cnBT1</command>
```

```
<command>set -bhmq://wbia.queue.manager</command>
```

```
/* Configures Commerce Server to put all notifications in queue 'notifications'*/
```

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

```
/* Configures Commerce Server to put all MDNs in queue 'mdns'*/
```

```
<command>set -rpmq://wbia.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://wbia.queue.manager/b_to_a</command>
```

```
/*Establishes a persistent send command so that Commerce Server 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 Commerce Server.

A typical implementation of the connector requires the creation of an outbound queue for delivering messages to Commerce Server, 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 19.

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 24, “Setting up event notification” on page 25, and “Configuring connector meta-objects” on page 26.

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 15.

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 22

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 35

Installing the connector and related files

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

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

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 script.

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 <i>ProductDir</i>	Description
<code>connectors\ISoft\CWISoft.jar</code>	Contains classes used by the iSoft connector
<code>connectors\ISoft\start_ISoft.bat</code>	The startup script for the connector (Windows)
<code>connectors\messages\ISoftConnector.txt</code>	Message file for the connector
<code>repository\ISoft\CN_ISoft.txt</code>	Repository definition for the connector
<code>connectors\ISoft\samples\WebSphereICS\Company_A_Fruit_Catalog.B0</code>	A sample instance of a business object
<code>connectors\ISoft\samples\WebSphereICS\icssvr\icssvr.cfg_a</code>	Sample Commerce Server configuration file.
<code>connectors\ISoft\samples\WebSphereICS\icssvr\icssvr.cfg_b</code>	Sample Commerce Server configuration file.
<code>connectors\ISoft\samples\WebSphereICS\icssvr\outboxB\ toA\Order1.txt</code>	Sample document for orders

Subdirectory of <i>ProductDir</i>	Description
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	Sample collaboration templates
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	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\clearQueues.txt	See the tutorial (Appendix C, "Tutorial," on page 83) for a description of the \samples\WebSphereMQIntegratorBroker files.
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\icssvr\icssvr.cfg_a	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\icssvr\icssvr.cfg_b	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\icssvr\icssvr.cfg_c	
connectors\ISoft\samples\WebSphereMQIntegratorBroker\icssvr\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	
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.

UNIX 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.

The table below describes the UNI 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 <i>ProductDir</i>	Description
connectors/ISoft/CWISoft.jar	Contains classes used by the iSoft connector
connectors/ISoft/start_ISoft.sh	The startup script for the connector (UNIX)
connectors/messages/ISoftConnector.txt	Message file for the connector

Subdirectory of ProductDir

repository/ISoft/CN_ISoft.txt
connectors/ISoft/samples/WebSphereICS/Company_A_Fruit_Catalog.BO
connectors/ISoft/samples/WebSphereICS/icssvr/icssvr.cfg_a
connectors/ISoft/samples/WebSphereICS/icssvr/icssvr.cfg_b
connectors/ISoft/samples/WebSphereICS/icssvr/outboxB/toA/Order1.txt
connectors/ISoft/samples/WebSphereICS/Sample_ISoft_BOS.in

connectors/ISoft/samples/WebSphereICS/Sample_ISoft_Collabs.in

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
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/icssvr\icssvr.cfg_a
connectors/ISoft/samples/WebSphereMQIntegratorBroker/icssvr/icssvr.cfg_b
connectors/ISoft/samples/WebSphereMQIntegratorBroker/icssvr/icssvr.cfg_c
connectors/ISoft/samples/WebSphereMQIntegratorBroker/icssvr/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
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

Description

Repository definition for the connector
A sample instance of a business object
Sample Commerce Server configuration file.
Sample Commerce Server configuration file.
Sample document for orders
Repository definition for sample iSoft
business objects
Repository definition for sample
collaboration templates
Sample collaboration templates

See the tutorial (Appendix C, "Tutorial," on
page 83) for a description of the
/samples/WebSphereMQIntegratorBroker
files.

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

For information on configuring the adapter, refer to the next chapter.

Chapter 3. Configuring the adapter

This chapter describes how to configure the connector component of the adapter, and how to configure the message flows to work with the connector. It covers these topics:

- “Connector configuration properties”
- “Setting up queues” on page 19
- “Queue Uniform Resource Identifiers (URI)” on page 21
- “Configuring data handlers” on page 22
- “Setting up request processing” on page 24
- “Setting up event notification” on page 25
- “Configuring connector meta-objects” on page 26
- “Creating multiple instances of the connector” on page 33
- “Startup file configuration” on page 35
- “Starting the connector” on page 35
- “Stopping the connector” on page 36
- “Sample dynamic child meta-object” on page 36

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 to configure connector properties:

- For a description of Connector Configurator and step-by-step procedures, see Appendix B, “Connector Configurator,” on page 67.
- For a description of standard connector properties, see “Standard connector properties” and then Appendix A, “Standard configuration properties for connectors,” on page 49.
- For a description of connector-specific properties, see “Connector-specific properties” on page 16.

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

Standard connector properties

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

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

Connector-specific properties

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

The table below lists the connector-specific configuration properties for the 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	
EnableMessageProducerCache	true or false	true	No
ErrorQueue	Queue for unprocessed messages	queue://isoft. queuemanager/ISOFT.ERROR	No
HostName	WebSphere MQ server		Yes
InDoubtEvents	FailOnStartup IgnoreLogError	Reprocess	No
InputQueue	Poll queues	queue://isoft. queuemanager/ISOFT.IN	No
InProgressQueue	In-progress event queue	queue://isoft. queuemanager/ISOFT.IN_PROGRESS	No
MessageFormatProperty	Property name specifying message format	JMSType	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
SessionPoolSizeForRequests	Maximum pool size for caching the sessions used during request processing	10	No
UnsubscribedQueue	Queue to which unsubscribed messages are sent	queue://isoft. queuemanager/ISOFT.UNSUBSCRIBE	No
UseDefaults	true or false	false	
Name	Possible values	Default value	Required

ArchiveQueue

Queue to which copies of successfully processed messages are sent.

Default = `queue://wbia.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 21.

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`

EnableMessageProducerCache

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

Default = `true`

ErrorQueue

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

Default = `queue://wbia.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 *pollQuantity* of 2, the connector will retrieve at most 2 messages from each queue per call to *pollForEvents*. For the first cycle (1 of 2), the connector retrieves the first message from each of MyQueueA, MyQueueB, and MyQueueC. That completes the first round of polling and if we had a *pollQuantity* of 1, the connector would stop. Since we have a *pollQuantity* of 2, the connector starts a second round of polling (2 of 2) and retrieves one message each from MyQueueA and MyQueueC—it skips MyQueueB since it is now empty. After polling all queues two times 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://wbia.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://wbia.queue.manager/ISOFT.IN_PROGRESS

MessageFormatProperty

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

Default =JMSType

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.

SessionPoolSizeForRequests

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

Default = 10

UnsubscribedQueue

Queue to which messages that are not subscribed are sent.

Default = queue://wbia.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 iSoft Commerce Suite server.

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

- Inbound from Commerce Server to Integration Broker

This is the queue in which Commerce Server will place messages that contain data documents received from the trading partner that Commerce Server 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 Commerce Server 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 Commerce Server 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 Commerce Server 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 26.

- Outbound from Integration Broker to iSoft Commerce Suite server

This is the queue in which the connector places data document messages that are to be sent to Commerce Server 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. Commerce Server 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 Commerce Server.

For implementations in which Commerce Server 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 Commerce Server to poll multiple queues. You can also have the Commerce Server 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 Commerce Server will use it when routing the document. For more information, see “Configuring connector meta-objects” on page 26.

- MDNs

This is the queue in which Commerce Server 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 Commerce Server 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 Commerce Server. The default name for this queue is `wbia.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 10 for 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 `wbia.queue.manager` and causes all messages to be sent as WebSphere MQ messages with priority 5.

```
queue://wbia.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.

Property name	Description	Values
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 17

Note: The connector has no control of the character set (CCSID) or encoding attributes of data in MQ Messages. 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 MQ Message 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 26 and “A Sample connector meta-object” on page 28.

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 Commerce Server 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

Commerce Server 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 Commerce Server 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 Commerce Server handles the task of creating and sending MDNs to the trading partner.

Property Name	Description	Default value
<code>BOName</code>	Name of the business object that will store the MDN data. Required.	<code>Sample_ISoft_MDN</code>

Property Name	Description	Default value
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 attribute 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 ICS. 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 43 later in this guide.

Notification data handler

The Commerce Server 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 Commerce Server 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 attribute whose name matches the specified MIME type and whose type is the Notification child meta-object. Required.	com.crossworlds. DataHandlers.isoftp2p
DummyKey	Key attribute required by ICS. 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 42 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 iSoft Commerce Suite server. 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 Commerce Server 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 Commerce Server

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 Commerce Server 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" on page 26 "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 Commerce Server, 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 the ICS integration broker) or message flows (for a WMQI integration broker) 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 16.

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 messages 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 use this property with the adapter for iSoft Commerce Suite server.
InputQueue	The InputQueue is the input queue that the connector will poll to discover new messages received from Commerce Server. Do not set this property using default conversion properties; its value is used to match incoming messages to business objects.
OutputFormat	The OutputFormat is set on messages created from the given business object. If the OutputFormat is not specified, the input format is used, if available. An OutputFormat defined in a dynamic child meta-object overrides the value defined in the static meta-object.
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.

Mapping data handlers to InputQueues

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

1. Use connector-specific properties (see “`InputQueue`” on page 18) to configure one or more input queues.
2. For each input queue, specify the queue manager and input queue name as well as data handler class name and mime type in the application-specific information.

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

```
[Attribute]
Name = 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 Commerce Server
- `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://wbia.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://wbia.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://wbia.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://wbia.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
JMSProperties	The iSoft Commerce Suite server 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 <code>type=String;name=to</code> and <code>type=String;name=from</code> corresponding to the fields in the MQ RFH2 header that the iSoft Commerce Suite server 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

Table 1. Dynamic Meta-Object Header Attributes (continued)

Header attribute name	Mode
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 Commerce Suite server.

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 Commerce Suite server recognizes as containing trading partner information. Currently, the iSoft Commerce Suite server only recognizes these two fields in the MQ RFH2 header. In the future, if the iSoft Commerce Suite server is enhanced to recognized additional fields, they can be defined here in the `JMSProperties` object.

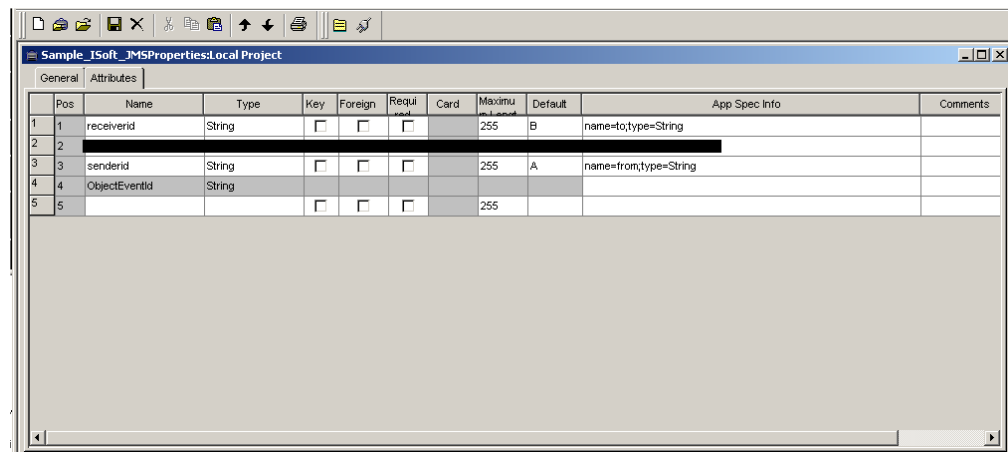


Figure 3. JMSProperties attribute in a dynamic meta-object

Creating multiple instances of the connector

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

- Create a new directory for the connector instance

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

Create a new directory

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

`ProductDir\connectors\connectorInstance`

where `connectorInstance` uniquely identifies the connector instance.

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

`ProductDir\Repository\connectorInstance`

Create business object definitions

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

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

`ProductDir\Repository\initialConnectorInstance`

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

Create a connector definition

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

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

Create a start-up script

To create a startup script:

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

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

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

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.

Starting the connector

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

`ProductDir\connectors\connName`

where `connName` identifies the connector. The name of the startup script depends on the operating-system platform, as Table 3 shows.

Table 3. Startup scripts for a connector

Operating system	Startup script
UNIX-based systems	<code>connector_manager_connName</code>
Windows	<code>start_connName.bat</code>

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

- On Windows systems, from the **Start** menu
Select **Programs>IBM WebSphere Business Integration Adapters>Adapters>Connectors**. By default, the program name is “IBM WebSphere Business Integration Adapters”. However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.

- From the command line

- On Windows systems:

`start_connName connName brokerName [-cconfigFile]`

- On UNIX-based systems:

`connector_manager_connName -start`

where `connName` is the name of the connector and `brokerName` identifies your integration broker, as follows:

- For WebSphere InterChange Server, specify for `brokerName` the name of the ICS instance.
- For WebSphere message brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) or WebSphere Application Server, specify for `brokerName` a string that identifies the broker.

Note: For a WebSphere message broker or WebSphere Application Server on a Windows system, you *must* include the `-c` option followed by the name of the connector configuration file. For ICS, the `-c` is optional.

- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).

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

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

Stopping the connector

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

- If you started the connector from the command line, with its connector startup script:
 - On Windows systems, invoking the startup script creates a separate “console” window for the connector. In this window, type “Q” and press Enter to stop the connector.
 - On UNIX-based systems, connectors run in the background so they have no separate window. Instead, run the following command to stop the connector:
`connector_manager_connName -stop`
 where *connName* is the name of the connector.
- From Adapter Monitor (WebSphere Business Integration Adapters product only), which is launched when you start System Manager
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- From System Monitor (WebSphere InterChange Server product only)
You can load, activate, deactivate, pause, shutdown or delete a connector using this tool.
- On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector stops when the Windows system shuts down.

Sample dynamic child meta-object

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

```
[Attribute]
Name = OutputFormat
Type = String
MaxLength = 1
```

```

IsKey = true
IsForeignKey = false
IsRequired = false
DefaultValue = CUST
IsRequiredServerBound = false
[End]
[Attribute]
Name = OutputQueue
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = OUT
IsRequiredServerBound = false
[End]
[Attribute]
Name = ResponseTimeout
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = -1
IsRequiredServerBound = false
[End]
[Attribute]
Name = TimeoutFatal
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
DefaultValue = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = InputFormat
Type = String
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = InputQueue
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Verb]
Name = Create
[End]

```

```

[Verb]
Name = Delete
[End]

[Verb]
Name = Retrieve
[End]

[Verb]
Name = Update
[End]
[End]
[BusinessObjectDefinition]
Name = Customer
Version = 1.0.0
AppSpecificInfo = cw_mo_conn=MyConfig

[Attribute]
Name = FirstName
Type = String
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = LastName
Type = String
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = Telephone
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = MyConfig
Type = MO_Sample_Config
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

```

```
[Verb]  
Name = Create  
[End]
```

```
[Verb]  
Name = Delete  
[End]
```

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

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

Chapter 4. Creating or modifying 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 chapter covers these topics:

- “Introduction”
- “Connector business object structure”
- “Error handling” on page 43
- “Tracing” on page 44

Introduction

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.

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. 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 definitions 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 Commerce Server can be configured to generate notification messages to inform the user of retry attempts being exhausted. The Commerce Server 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 "YYYYMMDDHHMSSRRRRRRRR"
  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:
icssvrNotificationMessage
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 Commerce Server 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 Commerce Server 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 `LogFile` 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 `gotAppEvent()` method, the connector sends the message to the queue specified by the UnsubscribedQueue property and continues processing other events.

Connector not active

When the `gotAppEvent()` 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 the adapter," on page 7 for more on configuring trace messages. For more information on tracing, including how to enable and set it, see the *Connector Development Guide*.

What follows is recommended content for connector trace messages.

- | | |
|---------|--|
| Level 0 | This level is used for trace messages that identify the connector version. |
| Level 1 | Use this level for trace messages that provide key information on each business object processed or record each time a polling thread detects a new message in an input queue. |

- Level 2 Use this level for trace messages that log each time a business object is posted to InterChange Server, either from `gotAppEvent()` 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 5. Troubleshooting

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

Start-up problems

Problem

The connector shuts down unexpectedly during initialization and the following message is reported:
Exception in thread "main"
java.lang.NoClassDefFoundError:
javax.jms/JMSEException...

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...

The connector shuts down unexpectedly during initialization and the following message is reported:
Exception in thread "main"
java.lang.NoClassDefFoundError:
javax.naming/Referenceable...

The connector shuts down unexpectedly during initialization and the following exception is reported:
java.lang.UnsatisfiedLinkError: no mqjbnd01 in shared library path

The connector reports MQJMS2005: failed to create MQQueueManager for ':'

Potential solution / explanation

Connector cannot find file `jms.jar` from the IBM MQSeries Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM MQSeries Java client library folder.

Connector cannot find file `com.ibm.mqjms.jar` from the IBM MQSeries Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM MQSeries Java client library folder.

Connector cannot find file `jndi.jar` from the IBM MQSeries Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM MQSeries Java client library folder.

Connector cannot find a required run-time library (`mqjbnd01.dll [NT]` or `libmqjbnd01.so [Solaris]`) from the IBM MQSeries Java client libraries. Ensure that your path includes the IBM MQSeries Java client library folder. Explicitly set values for the following properties: `HostName`, `Channel`, and `Port`.

Event processing

Problem

The connector delivers all messages with an `MQRFH2` header.

The connector truncates all message formats to 8-characters upon delivery regardless of how the format has been defined in the connector meta-object.

Potential solution / explanation

To deliver messages with only the `MQMD` MQSeries 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 the adapter," on page 7 for more information.

This is a limitation of the MQSeries `MQMD` message header and not the connector.

Appendix A. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of WebSphere Business Integration adapters. The information covers connectors running on the following integration brokers:

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

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

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

Note: In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes and follow the conventions for each operating system.

New and deleted properties

These standard properties have been added in this release.

New properties

- XMLNameSpaceFormat

Deleted properties

- RestartCount

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

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

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

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

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 (only if WebSphere InterChange Server is the integration broker)
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. If the connector is working in stand-alone mode (independently of System Manager), for example with one of the WebSphere message brokers, you can only change properties through the configuration file. In this case, a dynamic update is not possible.
- **Agent restart (ICS only)**
The change takes effect only after you stop and restart the application-specific component.
- **Component restart**
The change takes effect only after the connector is stopped and then restarted in System Manager. You do not need to stop and restart the application-specific component or the integration broker.
- **Server restart**
The change takes effect only after you stop and restart the application-specific component and the integration broker.

To determine how a specific property is updated, refer to the **Update Method** column in the Connector Configurator window, or see the Update Method column in Table 4 on page 51 below.

Summary of standard properties

Table 4 on page 51 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on RepositoryDirectory.

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

Note: In the "Notes" column in Table 4 on page 51, the phrase "Repository directory is REMOTE" indicates that the broker is the InterChange Server. When the broker is WMQI or WAS, the repository directory is set to LOCAL

Table 4. 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> (broker is ICS)
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS, WMQI, WAS		Component restart	
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> (broker is ICS)
ContainerManagedEvents	No value or JMS	No value	Component restart	Delivery Transport is JMS
ControllerStoreAndForwardMode	true or false	true	Dynamic	Repository directory is <REMOTE> (broker is ICS)
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE> (broker is ICS)
DeliveryQueue		CONNECTORNAME/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	MQ, IDL, or JMS	JMS	Component restart	If Repository directory is local, then value is JMS only

Table 4. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
DuplicateEventElimination	true or 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> (broker is ICS)
JvmMaxNativeStackSize	Size of stack in kilobytes	128k	Component restart	Repository directory is <REMOTE> (broker is ICS)
JvmMinHeapSize	Heap size in megabytes	1m	Component restart	Repository directory is <REMOTE> (broker is ICS)
ListenerConcurrency	1- 100	1	Component restart	Delivery Transport must be MQ
Locale	en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR Note: This is a subset of the supported locales.	en_US	Component restart	

Table 4. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
LogAtInterchangeEnd	true or false	false	Component restart	Repository Directory must be <REMOTE> (broker is ICS)
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
MessageFileName	Path or filename	CONNECTORNAMEConnector.txt	Component restart	
MonitorQueue	Any valid queue name	CONNECTORNAME/MONITORQUEUE	Component restart	JMS transport only: DuplicateEvent Elimination must be true
OADAutoRestartAgent	true or false	false	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
OADMaxNumRetry	A positive number	1000	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
OADRetryTimeInterval	A positive number in minutes	10	Dynamic	Repository Directory must be <REMOTE> (broker is ICS)
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	Agent restart	JMS transport only: Container Managed Events is specified
PollStartTime	HH:MM(HH is 0-23, MM is 0-59)	HH:MM	Component restart	

Table 4. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
RepositoryDirectory	Location of metadata repository		Agent restart	For ICS: set to <REMOTE> For WebSphere MQ message brokers and WAS: set to C:\crossworlds\repository
RequestQueue	Valid JMS queue name	CONNECTORNAME/REQUESTQUEUE	Component restart	Delivery Transport is JMS
ResponseQueue	Valid JMS queue name	CONNECTORNAME/RESPONSEQUEUE	Component restart	Delivery Transport is JMS: required only if Repository directory is <REMOTE>
RestartRetryCount	0-99	3	Dynamic	
RestartRetryInterval	A sensible positive value in minutes: 1 - 2147483547	1	Dynamic	
RHF2MessageDomain	mrm, xml	mrm	Component restart	Only if Delivery Transport is JMS and WireFormat is CwXML.
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 (milliseconds)	0	Component restart	Delivery Transport is JMS
SynchronousResponseQueue		CONNECTORNAME/ SYNCHRONOUSRESPONSEQUEUE	Component restart	Delivery Transport is JMS
WireFormat	CwXML, CwBO	CwXML	Agent restart	CwXML if Repository Directory is not <REMOTE>: CwBO if Repository Directory is <REMOTE>
WsifSynchronousRequest Timeout	0 - any number (milliseconds)	0	Component restart	WAS only
XMLNamespaceFormat	short, long	short	Agent restart	WebSphere MQ message brokers and WAS only

Standard configuration properties

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

AdminInQueue

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

The default value is `CONNECTORNAME/ADMININQUEUE`.

AdminOutQueue

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

The default value is `CONNECTORNAME/ADMINOUTQUEUE`.

AgentConnections

Applicable only if `RepositoryDirectory` is `<REMOTE>`.

The `AgentConnections` property controls the number of ORB (Object Request Broker) connections opened by `orb.init[]`.

The default value of this property is set to 1. You can change it as required.

AgentTraceLevel

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

ApplicationName

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

BrokerType

Identifies the integration broker type that you are using. The options are ICS, WebSphere message brokers (WMQI, WMQIB or WBIMB) or WAS.

CharacterEncoding

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

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

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

ConcurrentEventTriggeredFlows

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

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

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

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

ContainerManagedEvents

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

There is no default 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 (data handler class), and DataHandlerConfigMOName (the meta-object name, which is optional) properties. To set those values, use the **Data Handler** tab in Connector Configurator.

These properties are adapter-specific, but **example** values are:

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

The fields for these values in 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

Applicable only if `RepositoryDirectory` is `<REMOTE>`.

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

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

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

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

The default is `true`.

ControllerTraceLevel

Applicable only if `RepositoryDirectory` is `<REMOTE>`.

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

DeliveryQueue

Applicable only if `DeliveryTransport` is `JMS`.

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

The default value is `CONNECTORNAME/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.

- If the `RepositoryDirectory` is remote, the value of the `DeliveryTransport` property can be `MQ`, `IDL`, or `JMS`, and the default is `IDL`.
- If the `RepositoryDirectory` is a local directory, the value may only be `JMS`.

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. 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 in the following environment:

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

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

- The `LDR_CNTRL` environment variable in the `CWSharedEnv.sh` script.

This script resides in the `\bin` directory below the product directory. With a text editor, add the following line as the first line in the `CWSharedEnv.sh` script:

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

This line restricts heap memory usage to a maximum of 768 MB (3 segments * 256 MB). If the process memory grows more than this limit, page swapping can occur, which can adversely affect the performance of your system.

- The `IPCCBaseAddress` property to a value of 11 or 12. For more information on this property, see the *System Installation Guide for UNIX*.

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`. Use the default when connecting to a local message broker.

When you connect to a remote message broker, this property takes the following (mandatory) values:

`QueueMgrName:<Channel>:<HostName>:<PortNumber>`,

where the variables are:

`QueueMgrName`: The name of the queue manager.

`Channel`: The channel used by the client.

`HostName`: The name of the machine where the queue manager is to reside.

`PortNumber`: The port number to be used by the queue manager for listening.

For example:

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

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 when ICS is the integration broker. 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:

<i>ll</i>	a two-character language code (usually in lower case)
<i>TT</i>	a two-letter country or territory code (usually in upper case)
<i>codeset</i>	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-down list. To add other supported values to the drop-down list, you must manually modify the `\Data\Std\stdConnProps.xml` file in the product directory. For more information, see the appendix on Connector Configurator.

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

<http://www.ibm.com/software/websphere/wbiadapters/infocenter>, or
<http://www.ibm.com/websphere/integration/wicsserver/infocenter>

LogAtInterchangeEnd

Applicable only if RepositoryDirectory is <REMOTE>.

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

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

MaxEventCapacity

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

Valid only when the RepositoryDirectory is <REMOTE>.

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

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

The default value is false.

OADMaxNumRetry

Valid only when the RepositoryDirectory is <REMOTE>.

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

The default value is 1000.

OADRetryTimeInterval

Valid only when the RepositoryDirectory is <REMOTE>.

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

The default is 10.

PollEndTime

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

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

PollFrequency

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

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

Set PollFrequency to one of the following values:

- The number of milliseconds between polling actions (an integer).
- 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. Where they exist, these restrictions are documented in the chapter on installing and configuring the adapter.

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.

FIX

An email message is also considered an event. The connector behaves as follows when it is polled for email.

Polled once - connector goes to pick 1. the body of the message as it is also considered an attachment also. Since no DH was specified for this mime type, it will ignore the body. 2. connector process first PO attachment. DH is available for this mime type so it sends the business object to the Visual Test Connector. If the 3. accept in VTC again no BO should come thru Polled second time 1. connector process second PO attachment. DH is available for this mime type so it sends the BO to VTC2. accept in VTC again now the third PO attachment should come through. This is the correct behaviour.

PollStartTime

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

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

RequestQueue

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

The default value is `CONNECTOR/REQUESTQUEUE`.

RepositoryDirectory

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

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

When the integration broker is a WebSphere message broker or WAS, this value must be set to `<local directory>`.

ResponseQueue

Applicable only if `DeliveryTransport` is JMS and required only if `RepositoryDirectory` is `<REMOTE>`.

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

RestartRetryCount

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

The default is 3.

RestartRetryInterval

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

The default is 1.

RHF2MessageDomain

WebSphere message brokers and WAS only.

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

A sample header would look like this:

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

The default value is `mrm`, but it may also be set to `xml`. This property only appears when `DeliveryTransport` is set to `JMSand` and `WireFormat` is set to `CwXML`.

SourceQueue

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

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see “`ContainerManagedEvents`” on page 56.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is `JMS`.

Delivers request messages that require a synchronous response from the connector framework to the broker. This queue is necessary only if the connector uses synchronous execution. With synchronous execution, the connector framework

sends a message to the SynchronousRequestQueue and waits for a response back from the broker on the SynchronousResponseQueue. The response message sent to the connector bears a correlation ID that matches the ID of the original message.

The default is CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE

SynchronousResponseQueue

Applicable only if DeliveryTransport is JMS.

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

The default is CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE

SynchronousRequestTimeout

Applicable only if DeliveryTransport is JMS.

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

The default value is 0.

WireFormat

Message format on the transport.

- If the RepositoryDirectory is a local directory, the setting is CwXML.
- If the value of RepositoryDirectory is <REMOTE>, the setting is CwB0.

WsifSynchronousRequest Timeout

WAS integration broker only.

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.

XMLNameSpaceFormat

WebSphere message brokers and WAS integration broker only.

A strong property that allows the user to specify short and long name spaces in the XML format of business object definitions.

The default value is short.

Appendix B. Connector Configurator

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

You use Connector Configurator to:

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

Note:

In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes and follow the conventions for each operating system.

The topics covered in this appendix are:

- “Overview of Connector Configurator” on page 67
- “Starting Connector Configurator” on page 68
- “Creating a connector-specific property template” on page 69
- “Creating a new configuration file” on page 71
- “Setting the configuration file properties” on page 74
- “Using Connector Configurator in a globalized environment” on page 80

Overview of Connector Configurator

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

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

You use Connector Configurator to:

- Create a **connector-specific property template** for configuring your connector.
- Create a **connector configuration file**; you must create one configuration file for each connector you install.
- Set properties in a configuration file.

You may need to modify the default values that are set for properties in the connector templates. You must also designate supported business object definitions and, with ICS, maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

The mode in which you run Connector Configurator, and the configuration file type you use, may differ according to which integration broker you are running. For example, if WMQI is your broker, you run Connector Configurator directly, and not from within System Manager (see “Running Configurator in stand-alone mode” on page 68).

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

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

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

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in “Creating a new template” on page 69 to set up a new one.

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

Starting Connector Configurator

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

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

Running Configurator in stand-alone mode

You can run Connector Configurator independently and work with connector configuration files, irrespective of your broker.

To do so:

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

You may choose to run Connector Configurator independently to generate the file, and then connect to System Manager to save it in a System Manager project (see “Completing a configuration file” on page 73.)

Running Configurator from System Manager

You can run Connector Configurator from System Manager.

To run Connector Configurator:

1. Open the System Manager.

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

To edit an existing configuration file:

- In the System Manager window, select any of the configuration files listed in the Connector folder and right-click on it. Connector Configurator opens and displays the configuration file with the integration broker type and file name at the top.
- From Connector Configurator, select **File>Open**. Select the name of the connector configuration file from a project or from the directory in which it is stored.
- Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

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

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

- To create a new template, see “Creating a new template” on page 69.
- To use an existing file, simply modify an existing template and save it under the new name. You can find existing templates in your `\WebSphereAdapters\bin\Data\App` directory.

Creating a new template

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

To create a template in Connector Configurator:

1. Click **File>New>Connector-Specific Property Template**.
2. The **Connector-Specific Property Template** dialog box appears.
 - Enter a name for the new template in the **Name** field below **Input a New Template Name**. You will see this name again when you open the dialog box for creating a new configuration file from a template.
 - To see the connector-specific property definitions in any template, select that template’s name in the **Template Name** display. A list of the property definitions contained in that template appears in the **Template Preview** display.
3. You can use an existing template whose property definitions are similar to those required by your connector as a starting point for your template. If you do not see any template that displays the connector-specific properties used by your connector, you will need to create one.

- If you are planning to modify an existing template, select the name of the template from the list in the **Template Name** table below **Select the Existing Template to Modify: Find Template**.
- This table displays the names of all currently available templates. You can also search for a template.

Specifying general characteristics

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

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

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

Specifying values

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

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.
2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, enter your values.

To create a new property value:

1. Select the property in the **Edit properties** list and right-click on it.
2. From the dialog box, select **Add**.
3. Enter the name of the new property value and click OK. The value appears in the **Value** panel on the right.

The **Value** panel displays a table with three columns:

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

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

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

After a value has been created and appears in the grid, it can be edited from within the table display.

To make a change in an existing value in the table, select an entire row by clicking on the row number. Then right-click in the **Value** field and click **Edit Value**.

Setting dependencies

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

A dependent property is a property that is included in the template and used in the configuration file *only if* the value of another property meets a specific condition. For example, `PollQuantity` appears in the template only if `JMS` is the transport mechanism and `DuplicateEventElimination` is set to `True`.

To designate a property as dependent and to set the condition upon which it depends, do this:

1. In the **Available Properties** display, select the property that will be made dependent.
2. In the **Select Property** field, use the drop-down menu to select the property that will hold the conditional value.
3. In the **Condition Operator** field, select one of the following:
 - == (equal to)
 - != (not equal to)
 - > (greater than)
 - < (less than)
 - >= (greater than or equal to)
 - <=(less than or equal to)
4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator stores the information you have entered as an XML document, under `\data\app` in the `\bin` directory where you have installed Connector Configurator.

Creating a new configuration file

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

- In the System Manager window, right-click on the **Connectors** folder and select **Create New Connector**. Connector Configurator opens and displays the **New Connector** dialog box.
- In stand-alone mode: from Connector Configurator, select **File>New>Connector Configuration**. In the New Connector window, enter the name of the new connector.

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

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

Creating a configuration file from a connector-specific template

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

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

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

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

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

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

Select the template you want to use and click **OK**.
3. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector name. You can fill in all the field values to drop-down the definition now, or you can save the file and complete the fields later.
4. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.

If you save as a file, the **Save File Connector** dialog box appears. Choose *.cfg as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator indicates that the configuration file was successfully created.

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

Using an existing file

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

- A connector definition file.

This is a text file that lists properties and applicable default values for a specific connector. Some connectors include such a file in a `\repository` directory in their delivery package (the file typically has the extension `.txt`; for example, `CN_XML.txt` for the XML connector).
- An ICS repository file.

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

Such a file typically has the extension `*.cfg`.

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

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

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

1. In Connector Configurator, click **File>Open>From File**.
2. In the **Open File Connector** dialog box, select one of the following file types to see the available files:
 - Configuration (*.cfg)
 - ICS Repository (*.in, *.out)
Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will appear when you open the file.
 - All files (*.*)
Choose this option if a *.txt file was delivered in the adapter package for the connector, or if a definition file is available under another extension.
3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

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

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

Completing a configuration file

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

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

1. Under the **Standard Properties** tab, select the value field for the BrokerType property. In the drop-down menu, select the value ICS, WMQI, or WAS.
2. The Standard Properties tab will display the properties associated with the selected broker. You can save the file now or complete the remaining configuration fields, as described in “Specifying supported business object definitions” on page 76..
3. When you have finished your configuration, click **File>Save>To Project** or **File>Save>To File**.

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

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

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

Setting the configuration file properties

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

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

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

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

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

- Associated Maps
- Resources
- Messaging (where applicable)

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

Standard properties differ from connector-specific properties as follows:

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

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

- A field with a grey background indicates a standard property. You can change the value but cannot change the name or remove the property.

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

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.
3. After entering all the values for the standard properties, you can do one of the following:
 - To discard the changes, preserve the original values, and exit Connector Configurator, click **File>Exit** (or close the window), and click **No** when prompted to save changes.
 - To enter values for other categories in Connector Configurator, select the tab for the category. The values you enter for **Standard Properties** (or any other category) are retained when you move to the next category. When you close the window, you are prompted to either save or discard the values that you entered in all the categories as a whole.
 - To save the revised values, click **File>Exit** (or close the window) and click **Yes** when prompted to save changes. Alternatively, click **Save>To File** from either the File menu or the toolbar.

Setting application-specific configuration properties

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

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

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

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

Encryption for connector properties

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

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

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

Update method

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under “Setting and updating property values” on page 50.

Specifying supported business object definitions

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

Note: Some connectors require that certain business objects be specified as supported in order to perform event notification or additional configuration (using meta-objects) with their applications. For more information, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

If ICS is your broker

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

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

1. Click an empty field in the **Business Object Name** list. A drop-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 window, click **Save to Project**. The revised connector definition, including designated support for the added business object definition, is saved to an ICL (Integration Component Library) project in System Manager.

To delete a business object from the supported list:

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

Deleting a business object from the supported list changes the connector definition and makes the deleted business object unavailable for use in this implementation

of this connector. It does not affect the connector code, nor does it remove the business object definition itself from System Manager.

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

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

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

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

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

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

If a WebSphere Message Broker is your broker

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

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

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

If WAS is your broker

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

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

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

Associated maps (ICS only)

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

The list of business objects contains the application-specific business object which the agent supports and the corresponding generic object that the controller sends

to the subscribing collaboration. The association of a map determines which map will be used to transform the application-specific business object to the generic business object or the generic business object to the application-specific business object.

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

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

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

- **Business Object Name**

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

- **Associated Maps**

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

- **Explicit**

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

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

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

To explicitly bind a map:

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

Resources (ICS)

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

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

Messaging (ICS)

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

Setting trace/log file values

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

To change the logging and tracing values:

1. Click the **Trace/Log Files** tab.
2. For either logging or tracing, you can choose to write messages to one or both of the following:

- To console (STDOUT):
Writes logging or tracing messages to the STDOUT display.

Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.

- To File:
Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. Logging or tracing message are written to the file and location that you specify.

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

Data handlers

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

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

Saving your configuration file

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

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

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

- In stand-alone mode, as a file with a *.cfg extension in a directory folder. By default, the file is saved to \WebSphereAdapters\bin\Data\App.
- You can also save it to a WebSphere Application Server project if you have set one up.

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

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

Changing a configuration file

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

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

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

- Open the existing configuration file in Connector Configurator.
- Select the **Standard Properties** tab.
- In the **BrokerType** field of the Standard Properties tab, select the value that is appropriate for your broker.

When you change the current value, the available tabs and field selections on the properties screen will immediately change, to show only those tabs and fields that pertain to the new broker you have selected.

Completing the configuration

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

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

Using Connector Configurator in a globalized environment

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

Connector Configurator supports non-English characters in:

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

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

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

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

Appendix C. Tutorial

This appendix is a tutorial. It shows how you can use the iSoft adapter to send business objects to and receive business objects from an application communicating via the iSoft Commerce Suite server. The iSoft Commerce Suite server and the iSoft adapter communicate via WebSphere MQ Message queues. This scenario is designed to be simple and show the basic points of the adapter's functionality.

Note: As of version 3.2, the product previously known as the iSoft Commerce Suite server has been renamed to the iSoft Commerce Suite server. In this adapter document both names may be used interchangeably to refer to the same application.

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

Prerequisites

Before using this tutorial you must install and be familiar with the IBM WebSphere Business Integration product.

About the tutorial

The tutorial consists of a request processing and an event notification scenario. The scenarios feature companies A, B and C. Trading relationships exist between companies A and C, and A and B. The companies have standardized communication using AS/2 protocol. All companies use the iSoft Commerce Suite server to implement AS/2. Company A has deployed the WebSphere adapter for iSoft, which facilitates communication with the iSoft Commerce Suite server and, consequently, its trading partners.

In the scenario, Company A publishes a catalog to Companies B and C. Companies B and C place orders using the information in the catalog. More specifically, an object representing the catalog, `Sample_ISoft_Catalog` is passed to the iSoft adapter, translated to an XML message, and placed in a queue. The iSoft Commerce Suite server deployed by Company A gets the message from the queue and transports it to Company B's iSoft Commerce Suite server, which persists the message in the file system. (In a real world scenario, an application on Company B's network would pick up and process the file at this time.) When Companies B or C place an order, their respective iSoft Commerce Suite servers pass an order to Company A's iSoft Commerce Suite server. Company A's iSoft Commerce Suite server places the order in queue, the iSoft adapter de-queues the order, translates it to a business object and passes it to an integration broker. In this tutorial all iSoft Commerce Suite servers run on the same machine and use the same queue manager. However, by updating the information in the iSoft Commerce Suite server config files (`icssvr.cfg_a` and `icssvr.cfg_b`), you could modify the scenarios to use agents running on separate machines and relying on different queue managers.

Before you begin

Before proceeding with this tutorial, make sure that:

- You installed and are experienced with the IBM WebSphere product.
- You installed and are experienced with WebSphere MQ 5.3.

- You installed the WebSphere MQ client libraries for Java.
- You installed the WebSphere Business Integration Adapter for iSoft.
- Your WebSphere MQ queue manager is named `wbia.queue.manager` (the default value during installation). Otherwise, whenever this document refers to `wbia.queue.manager` substitute your queue manager name.

Setting up your environment

This section describes how to prepare your environment to work with the tutorial. In what follows, *sample_folder* refers to the `samples/WebSphereMQIntegratorBroker` folder in which the samples for this MQ Integrator Broker tutorial reside, and *WBI_folder* refers to the folder containing your current IBM WebSphere Business Integration installation.

1. **Create the queues** Create and start a WebSphere MQ queue manager with a running channel initiator and listener. Then create the queues shown below using the `createQueues.txt` script. To use the script, execute the following:

```
runmqsc wbia.queue.manager < createQueues.txt
```

Create the following queues:

- ADMININQUEUE
- ADMINOUTQUEUE
- DELIVERYQUEUE
- FAULTQUEUE
- REQUESTQUEUE
- RESPONSEQUEUE
- COMPA.TXT
- COMPA.XML
- COMPA.IN
- COMPA.OUT
- COMPA.NOTICES
- COMPA.RECEIPTS
- ISOFT.ARCHIVE
- ISOFT.ERROR
- ISOFT.IN_PROGRESS
- ISOFT.REPLY
- ISOFT.UNSUBSCRIBED
- ISOFTCONNECTOR/MONITORQUEUE
- ISOFTCONNECTOR/REQUESTQUEUE
- ISOFTCONNECTOR/RESPONSEQUEUE
- ISOFTCONNECTOR/SOURCEQUEUE
- ISOFTCONNECTOR/SYNCHRONOUSREQUESTQUEUE
- ISOFTCONNECTOR/SYNCHRONOUSRESPONSEQUEUE
- PORTCONNECTOR/SYNCHRONOUSREQUESTQUEUE
- PORTCONNECTOR/SYNCHRONOUSRESPONSEQUEUE

2. **Configure the standard properties for the iSoft and Port connectors** Open both `sample_folder\ISoftConnector.cfg` and `sample_folder\PortConnector.cfg` using Connector Configurator. In both windows set the following Standard Properties:

- `jms.MessageBrokerName` Set this property to the name of your queue manager (default = `wbia.queue.manager`)
 - `RepositoryDirectory` Set this property to `WBI_folder\connectors\ISoft\samples\WebSphereMQIntegratorBroker\repos`
 - `PollFrequency` Set this value to 10000 (UNIX only - for Windows, leave value key)
3. **Configure connector-specific properties** Open both `sample_folder\ISoftConnector.cfg` and `sample_folder\PortConnector.cfg` using Connector Configurator. In both windows set the following connector-specific Properties:
- Replace all instances of `wbia.queue.manager` with the name of your queue manager (for both iSoft and Port connectors)
 - Set the following properties in the `ISoftConnector.cfg` file only
 - Set the `hostname` property to the name of your machine.
 - Set the `port` property to the port your queue manager is listening on (default = 1414)
 - Set the `channel` property to `CHANNEL1`
4. **Creating or updating connector start scripts**
- FOR Windows:
 - Create a new shortcut named `START ISOFT CONNECTOR`. Right-click the shortcut and select the `Shortcut` tab.
 - Set the `Target` text box to execute the adapter.
 - The shortcut should execute the `start_ISoft.bat` file and pass it three Command Line arguments:
 - The name of the connector, which in this case is `ISoft`
 - The name of the broker, which you may create. For example, `iSoftSTA`. In this example the adapter is running in `sandalone` mode. No broker is used. Accordingly, this parameter's value may contain any combination of alphanumeric characters.
 - The string literal `-c` followed by the full path to the `XMLConnector.cfg` file. For example:


```
c\WBI_folder\connectors\ISoft\samples\WebSphereMQIntegratorBroker\ISoftConnector.cfg
```
- Here is a full example of a correctly set `Target` text box:
- ```
WBI_folder\connectors\ISoft\
start_ISoft.bat ISoft WMQIrm -c\WBI_folder\connectors\ISoft\samples\
WebSphereMQIntegratorBroker\ISoftConnector.cfg
```
- Set the `Start In` text box value to `WBI_folder\connectors\ISoft`
  - Click `OK`.
- For UNIX
  - Open the file: `WBI_folder/bin/connector_manager_ISoft`. Set the value of the `AGENTCONFIG_FILE` property to `-c` and the full path and filename for the `ISoftConnector.cfg` file. For example:
 

```
AGENTCONFIG_FILE=-c\WBI_folder/connectors/ISoft/samples/
WebSphereMQIntegratorBroker/ISoftConnector.cfg
```
5. **Configure the meta-object** If your WebSphere MQ Queue Manager is not named `wbia.queue.manager`, you will need to update URIs in attribute `AppSpecificInfo` fields in the `Sample_ISoft_M0_Config` business object.
- a. Open the `Sample_ISoft_M0_Config` business object in Business Object Designer. The file is located in the `WBI_folder/repos` directory

- b. Scroll to the right and examine values in the AppSpecificInfo column. Replace instances of `wbia.queue.manager` with the name of your queue manager.
6. **Configure the iSoft Commerce Suite server**
    - a. Open all files that start with `icssvr.cfg` (including such files as `icssvr.cfg_a`).
    - b. Replace `wbia.queue.manager` with the name of your queue manager.
    - c. For UNIX, update the file separators in the configuration files. Replace `\` with `/` in `addpair` commands.
    - d. Make sure that the following directories exist in the `sample_folder\icssvr`:
      - `inboxB`
      - `inboxB/fromA`
      - `inboxC`
      - `inboxC/fromA`
      - `noticesB`
      - `noticesC`
      - `receiptsB`
      - `receiptsC`

If these directories do not exist, please create them.

---

## Running the scenarios

Before you run the scenarios:

1. **Start the adapter for iSoft** if it is not already running.
  - For Windows:
    - Double-click on `sample_folder\START ISOFT CONNECTOR`
  - For UNIX:
    - Execute `connector_manager_ISoft -start` from the command line. Update the permissions of this file as needed. Make sure that the file is in your execution PATH.
2. **Start the Visual Test Connector** if it is not already running.
  - a. Click File->Create/Select Profile. A new window pops up. If it's the first time executing the sample, then click File->New Profile.
  - b. Select `PortConnector.cfg` from the `sample_folder` directory.
  - c. Set the Connector Name field to `PortConnector`.
  - d. Set the Broker Type name to `WMQI`.
  - e. Select `PortConnector` from the list.
  - f. Click OK.
  - g. Click File->Connect.
3. **Start iSoft components**
  - a. Copy instances of `icsmqd` and `icssvr` to the `sample_folder\icssvr` directory
  - b. Set the following environment variables:

```
P2PMBSERVER_INPUTURI=http://127.0.0.1:3081
P2PMBSERVER_MBNAME=wbia.queue.manager
```

These variables will be read by the iSoft WebSphere MQ driver (`icsmqd`) when it is started. If `wbia.queue.manager` is not the name of your queue manager, substitute your queue manager's name.

- c. Start one instance of the iSoft WebSphere MQ driver.
- d. Open three instances of the iSoft Commerce Suite server.
- e. In the first window enter the following:

```
batch icssvr.cfg_a
```

This instance will be referred to as Commerce Server A

- f. In the second window, enter the following:

```
batch icssvr.cfg_b
```

This instance will now be referred to as Commerce Server B

- g. In the third window, enter the following:

```
batch icssvr.cfg_c
```

This instance will now be referred to as Commerce Server C

## Request processing scenario

This part of the tutorial describes a request processing scenario.

### 1. Load the business object

- a. Using the Test Connector, load a copy of Company\_A\_Fruit\_Catalog\_with\_dynamic\_mo.B0 by selecting Edit->Load BO. If the Load BO menu option is unfocused, click in the left grid pane and try selecting Edit->Load BO again.
- b. When you are prompted for a name, enter A\_send. Examine the business object, expanding its child objects. Notice that when you expand JMSProperties the attribute receiverid has the value B. This information is included in the RFH2 header. It tells the iSoft Commerce Suite server to forward the message to a trading partner named B.

### 2. Send the business object to trading partner B

- a. Click the Send BO button. This posts a message to the COMPA.OUT queue. Commerce Server A must now get the message and send it to its trading partner.
- b. In the Commerce Server A window, enter the following:

```
send http MAILBOX MAILBOX -de -dsMAILBOXID=COMPA.OUT -r
```

Alternatively, to start a persistent send command that continually polls the queue every 5 seconds until April 15th 2003 invoke the following command

```
send http MAILBOX MAILBOX -de -dsMAILBOXID=COMPA.OUT -r
-tC5s -tE20030415000000
```

After performing a successful send you'll notice that both the Commerce Server A and Commerce Server B windows contain new log messages.

### 3. Send the business object to trading partner C

- a. Edit the business object instance A\_send. Set the value of MOConfig.JMSProperties.receiverid to C.
- b. Click the Send Bo button.
- c. In the Commerce Server A window, enter the following:

```
send http MAILBOX MAILBOX -de -dsMAILBOXID=COMPA.OUT -r
```

The message is delivered to trading partner C.

### 4. Confirm message delivery

- a. Check the file system. There should be new files in *sample\_folder\icssvr\inboxB\fromA* and *sample\_folder\icssvr\inboxC\fromA*.

- b. New messages should exist in the COMPA.RECEIPTS and COMPA.NOTICES queues.

## Event notification scenario

In the following scenario Trading Partner B sends an order to Trading Partner A. The iSoft Commerce Suite server has the ability to route messages to different queues based on a messages MIME type. In this scenario you create messages with a text/xml MIME type. The addroute commands in icssvr.cfg\_a route these messages to the COMPA.XML queue. Commerce Server A could 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 on a specific queue (such as COMPA.XML). The delimited data handler could process all messages on another queue (for example, COMPA.TXT). The following sample demonstrates polling XML messages only

### 1. Process incoming trading partner messages

- a. Invoke the following send command in Commerce Server B.

```
send http B A -cX -fNoutboxB\toA\Order1.txt
```

This command gets a file from the file system and transports it from Commerce Server B to Commerce Server A. Commerce Server A puts the message in the COMPA.XML queue.

- b. Alternatively, to send information to Commerce Server A and deposit it in the COMPA.TXT queue, you could have entered the following command:

```
send http B A -cT -fNoutboxB\toA\Order1.txt
```

### 2. Confirm message arrival

- Using MQSeries Explorer or a similar application, open queue queue://wbia.queue.manager/COMPA.XML to see if a new message has arrived from Commerce Server B.

### 3. Poll for the message

- Go to the iSoft adapter window, enter p, and hit return. Alternatively, you can set up the adapter to poll for messages at a specified interval by setting the PollFrequency iSoft Connector property. The adapter is polling the COMPA.XML, COMPA.NOTICES, and COMPA.RECEIPTS queues. A message is added to the COMPA.NOTICES queue every time Commerce Server A performs an action. When Commerce Server A receives an MDN (message receipt) it deposits the receipt in the COMPA.RECEIPTS queue. Assuming that you executed all of the previous steps (including those in “Request processing scenario” on page 87), each of these queues should contain at least one message prior to polling.

### 4. Check the Test Connector for the processed objects

- a. You should see three objects in the right hand pane:
  - a Sample\_iSoft\_Notification object
  - a Sample\_iSoft\_MDN object
  - a Sample\_iSoft\_Order object
- b. Double click each object to inspect it.
- c. Select each object separately and click the Reply Success button.

**Note:** Be sure to click Reply Success or Reply Fail for each object before shutting down the connector.



If you have performed all the above steps successfully, you should have a working sample scenario that enables the iSoft adapter to send and receive messages from the iSoft Commerce Suite server.



---

## 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.

#### COPYRIGHT LICENSE

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

---

## 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 Adapter Framework V2.4.