WebSphere Business Integration Adapters

Adapter for JText User Guide

V5.1.0

WebSphere Business Integration Adapters

Adapter for JText User Guide

V5.1.0

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

20December2002

This edition of this document applies to connector version 5.1.0, and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about IBM CrossWorlds documentation, email 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.

Integration Broker Compatibility

Supported on IBM WebSphere Business Integration Adapter Framework version 2.1.0, IBM CrossWorlds Infrastructure version 4.1.1, WebSphere MQ Integrator version 2.1.0, and WebSphere MQ Integrator Broker, version 2.1.0. See Release Notes for any exceptions.

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 CrossWorlds Lab Director
IBM RTP Laboratory
3039 Cornwallis Road

P.O. BOX 12195
Raleigh, NC 27709-2195
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

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

Lotus, Domino, Lotus Notes, and Notes Mail are trademarks of the Lotus Development Corporation in the United States, other countries, or both.Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

Solaris, 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 CrossWorlds Servers V4.1, IBM CrossWorlds Full Toolset V4.1, IBM CrossWorlds Connectors V4.1, IBM CrossWorlds Collaborations V4.1, WebSphere Business Integration Adapters, V2.1.0

About This Document

The IBM(R) WebSphere(R) Business Integration Adapter Framework is a suite of software integration products that supply connectivity for leading e-business technologies and enterprise applications. The system includes:

- Prebuilt components for common business integration processes
- Tools and templates for customizing and creating components
- A flexible, easy-to-use platform for configuring and managing the components

This document describes the installation, configuration, troubleshooting, and business object development for the JText connector.

Audience

This document is for WebSphere consultants and customers. You should be familiar with the fundamentals of your integration broker, the fundamentals of business object development, and possibly with data handler development.

Related Documents

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

To access the documentation, go to the directory where you installed the product and open the documentation subdirectory. If a `welcome.html` file is present, open it for hyperlinked access to all documentation. If no documentation is present, you can install it or read it directly online at one of the following sites:

- If you are using MQ Integrator as your integration broker:
  http://www.ibm.com/software/websphere/wbiadapters/infocenter
- If you are using InterChange Server as your integration broker:
  http://www.ibm.com/websphere/crossworlds/library/infocenter

The documentation consists primarily of Portable Document Format (PDF) files, with some additional files in HTML format. To read it, you need an HTML browser such as Netscape Navigator or Internet Explorer, and Adobe Acrobat Reader 4.0.5 or higher. For the latest version of Adobe Acrobat Reader for your platform, go to the Adobe website (www.adobe.com).

Typographic Conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *`italic`*	Indicates a variable name or a cross-reference.
blue text	Blue text, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click any blue text to jump to the object of the reference.

{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
[ ]	In a syntax line, square brackets surround an optional parameter.
...	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, `option[,...]` means that you can enter multiple, comma-separated options.
< >	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><connector_name>tmp.log`.
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the product is installed on your system.
UNIX:/Windows:	Paragraphs beginning with either of these indicate notes listing operating system differences.
%text% and *$text*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *$text*, indicating the value of the *text* UNIX environment variable.

New in This Release

New in Release 5.1.0

The release of this document for JText connector version 5.1.0 contains the following new or corrected information:

- Documentation of theStagingDir meta-object attribute, which allows the connector to use a staging directory when writing files. The connector writes files that represent business objects to a staging directory andthen moves them to the configured output directory so that external processes that might manipulate files in the output directory do not receive the files berfore they have been completely written. For more information, see "StagingDir" on page 34.
- Documentation of the IncludeEndBODelimitermeta-object attribute, which enables the connector to either include the value specified for the EndBODelimiter attribute when it writes out files, or to leave the value out of the data stream. For more information, see "IncludeEndBODelimiter" on page 32.
- Documentation of the FTPFileListingFormatmeta-object attribute, which enables the connector to read in files with format information (such as date and timestamps) that differ depending on the locale. For more information, see "FTPFileListingFormat" on page 30.
- Documentation of the FTPKeepConnectionOpenmeta-object attribute, which enables the connector to maintain a persistent connection with an FTP server. For more information, see "FTPKeepConnectionOpen" on page 31.
- Documentation of the FTPOSPlatformmeta-object attribute, which must be set to the value MVS if the FTP server with which the connector communicates is running an MVS platform. For more information, see "FTPOSPlatform" on page 31.
- Documentation of the FTPPollTerminateIfServerDownmeta-object attribute, which allows you to specify whether the connector terminate when an FTP server it is polling for events is unavailable. For more information, see "FTPPollTerminateIfServerDown" on page 31.
- Documentation of the FTPRequestTerminateIfServerDownmeta-object attribute, which allows you to specify whether the connector terminate when performing request processing with an FTP server that is unavailable. For more information, see "FTPRequestTerminateIfServerDown" on page 32.
- Revision of the information regarding the DataEncodingmeta-object attribute. You can set the attribute to any Java-supported encoding. For more information, see "DataEncoding" on page 28.

New in Release 5.0

The connector has been internationalized. For more information, see "Processing Locale-Dependent Data" on page 16 and Appendix A, "Standard Configuration Properties for Connectors", on page 75.

New in Release 4.5.x

WebSphere Business Integration Adapter for JText includes the connector for JText. This adapter operates with both the InterChange Server (ICS) and WebSphere MQ Integrator (WMQI) integration brokers. An integration broker, which is an application that performs integration of heterogeneous sets of applications, provides services that include data routing.

This adapter includes:

- An application component specific to JText
- A sample business object, included in the `\connectors\JText\Samples\` directory
- IBM WebSphere Adapter Framework, which consists of:
  - Connector Framework
  - Development tools (including Business Object Designer and Connector Configurator)
  - APIs (including ODK, JCDK, and CDK)

This manual provides information about using this adapter with both integration brokers: InterChange Server (ICS) and WebSphere MQ Integrator (WMQI).

New in Release 4.4.x

The release of this document for JText connector version 4.4.x contains the following changes:

- Support has been added for configuring the connector to use a remote FTP file system. See "Specifying a Remote FTP File System" on page 51 for details.
- Documentation of the deprecated DataHandlerFormatter and the ByNameValue, ByDelimiter, and BySize formatters has been removed. For documentation on using these formatters, see the JText connector documentation for the 3.0.0 or 2.0.3 releases of the guide for the connector.
- Documentation of dynamic child meta-objects has been clarified, and the manual states that use of the JTextWrapper configuration is being deprecated.
- Documentation of the `BODelimiter` meta-object attribute has been clarified.
- The `MO_JText_DHFormatter.txt` file has been removed from the `\repository\Jtext` directory under the product directory.

New in Release 4.3.x

The release of this document for JText connector version 4.3.x contains minor changes to fix defects and to provide compatibility with the connector agent parallelism feature introduced in CrossWorlds version 4.0.0.

New in Release 4.2.x

The release of this document for JText connector version 4.2.x contains the following new or corrected information:

- Support has been added for dynamic child meta-objects. The connector can now be configured to use wrapper objects or dynamic child meta-objects for dynamic file specification. See "Using a Dynamic Child Meta-Object" on page 4 for details.
- The `EventDataHandler` and `OutputDataHandler` meta-object attributes have been added to allow the connector to directly invoke data handlers rather than invoke

them via the DataHandlerFormatter. You may use these attributes to directly specify a data handler in the JText connector meta-object rather than indirectly within the DataHandlerFormatter's meta-object. See Table 7 on page 28 for details.

- The new user option `None` has been added for configuring the `EndBODelimiter` meta-object attribute when outputting business objects to a file without any `EndBODelimiter` or newline character. See Table 9 on page 49 for details.
- The DataEncoding meta-object attribute has been added to allow users to specify the UTF8 encoding instead of the default UTF7. See Table 7 on page 28 for details.

The release of the document for JText connector version 4.1.x contained the following new information:

- The connector's archiving feature has been redesigned.
- The naming convention for formatter meta-objects has changed.

Important: It is recommended that you use only data handlers and not formatters for business-object processing.

- The `CwJTFormatter.jar` file is no longer distributed.
- The format and use of the `event.log` file has changed.
- The process of specifying a data handler has been clarified and corrected.
- The descriptions of event notification and data handler processing have been clarified and corrected.
- A new section on creating custom meta-objects has been added.
- The sections "Problem with Event Triggering" on page 63 and "JText Failure Handling" on page 64 have been clarified and corrected.

Contents

Chapter 1. Overview of the JText Connector

This chapter describes the connector component of the IBM WebSphere Business Integration Adapter for JText. The connector enables an integration broker to communicate with an application by exchanging text files. This connector facilitates integration of data with applications that lack an API.

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

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

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

For more information about the relationship of the integration broker to the connector, see the *IBM CrossWorlds System Administration Guide*, or the *WebSphere Business Integration Adapters Implementation Guide for MQ Integrator*.

Use the JText connector when:

- An application does not have a C, C++, or Java standard API through which an integration broker can communicate.
- It is not feasible to have an event table for a custom-built application.
- Text files are the most appropriate method for exchanging data.

In these cases, the simplest method for integrating an application into a larger system may be by exchanging text files through the JText connector.

Connector Components

The JText connector has the following components:

- "Application-Specific Component" on page 2
- "Data Handlers" on page 2
- "Meta-Objects" on page 3

Figure 1 illustrates the JText connector's architecture when InterChange Server (ICS) is used as the integration broker.

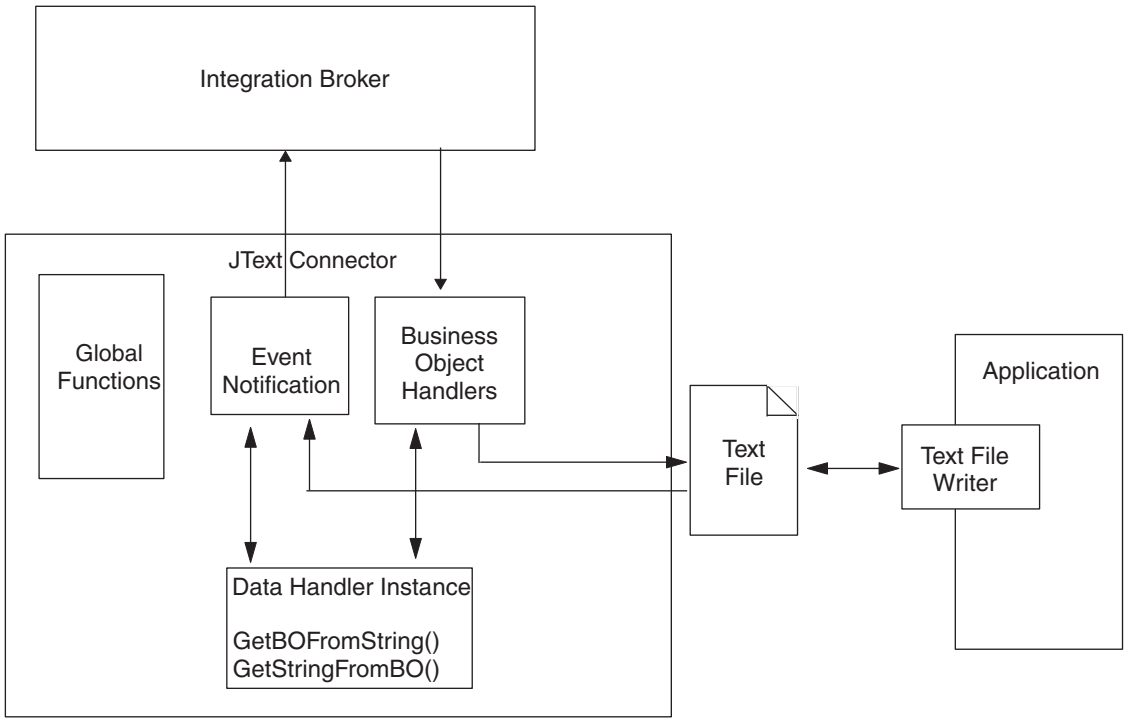


Figure 1. JText Connector Architecture

Application-Specific Component

The JText connector's application-specific component manipulates files and calls a specified data handler to convert data between business objects and strings. It also handles communication with the integration broker.

Data Handlers

The goal of the JText connector is to provide conversion between any existing file format and a business object. To do so, it uses the data handler specified in the connector's meta-object configuration.

The data handler performs the conversion without interacting with the file system in any way, either by reading from or writing to files. All interaction with the text file is handled by other connector components.

To handle data conversion, you can use data handlers that WebSphere Business Integration Adapter Framework provides or data handlers that you create to handle specific text-formatting needs. The product provides the following data handlers:

- XML — Converts business objects to and from XML documents.
- NameValue — Parses text data based on named fields. In this case, the text file contains fields that identify the business object type (`BusinessObject=`*`BOname`*), verb (`Verb=`*`verbName`*), and number of attributes (`AttributeCount=`*`numericValue`*).
- Delimited — Used primarily where the efficiency of machine reading is most important. Parses text data based on a specified delimiter that separates the individual fields of a business object's data.

- FixedWidth — Parses text data by using fixed-length fields. The field lengths are specified by the `MaxLength` property of each business object attribute. The value of this property is stored in the business object definition.

The product provides sample code for the NameValue, Delimited, and FixedWidth data handlers. You can use this code to customize or develop your own data handlers. The sample code is located in:

UNIX:

```
$CROSSWORLDS/DevelopmentKits/edk/DataHandler/Samples
```

Windows:

```
%CROSSWORLDS%\DevelopmentKits\edk\DataHandler\Samples
```

For more information, see "How Data Handler Processing Works" on page 14. For more information about each of the product-delivered data handlers, see the *Data Handler Guide*.

Meta-Objects

In addition to the standard and application-specific connector configuration properties that you set in the appropriate utility (Connector Configurator if your integration broker is WebSphere MQ Integrator, or Connector Designer of IBM CrossWorlds System Manager (CSM) if your integration broker is ICS), the JText connector has a set of configuration properties that enable you to configure the connector to process different business objects differently. You set these properties by using JText meta-objects. A meta-object is a special kind of business object that contains configuration information.

The connector uses the meta-object information to determine what classes to use to transform strings that it reads from files into business objects, and to format strings from business objects into files. The JText meta-objects specify the directories, file extensions, filenames, business object delimiters, and data handlers to use during event, archive, and request processing.

The JText connector uses meta-objects internally. It does not send them through the integration broker. For more information about using meta-objects to configure the connector, see Chapter 3, "Using JText Connector Meta-Objects", on page 25.

Business Objects Used by the JText Connector

Business objects for the JText connector must deliver data in the format required by the data handler specified for conversion. However, the JText connector may not need a set of specially designed business objects comparable to application-specific business objects for an application connector.

For example, the NameValue data handler requires each piece of data to have a string that identifies it (such as `CustomerName=Kumar`, `Region=NE`, and `Department=HR`). Because every generic business object definition contains attributes whose names identify each piece of data, the JText connector can use generic business objects.

However, because generic business objects represent a superset of information required by a multitude of different applications, each generic business object usually contains far more information than is required by any one application.

Therefore, to convert data into a manageable size for each application, a good practice is to create your own business object for each type of data to be processed. In the business object, provide only the data required by the application and the information required by the data handler.

For example, for the FixedWidth data handler, you must ensure that every business object attribute has a value specified for the MaxLength attribute property. For the XML data handler, other specific information is required. On the other hand, for the NameValue and Delimited data handlers, the business object need not contain any information that is not already contained in a generic business object. See the *Data Handler Guide* for information specific to each data handler.

In addition to delivering data, a business object can contain information that enables the connector to dynamically obtain the business object's event filename or to return the output filename to the integration broker. To configure the connector for this dynamic processing, the application-specific information at the business-object level must contain one of the following name-value pairs:

- `cw_mo_JTextConfig` = *`DynChildMOAttrName`*
- `Type` = `JTextWrapper`

  Important: Use of the JTextWrapper configuration has been deprecated.

Only one of these can be present. If both are encountered, the connector throws an exception.

If the business object contains additional application-specific information that is used by the data handler, the name-value pair must appear first in the business object, and must be separated from the additional application-specific information by a semicolon (**;**). The connector reads the name-value pair up to the semicolon to determine whether to use dynamic processing, then passes any information that appears after the semicolon to the data handler.

Using a Dynamic Child Meta-Object

A dynamic child meta-object enables the filename to be exchanged with InterChange Server. This section describes:

- "Why Use a Dynamic Child Meta-Object?"
- "How to Use a Dynamic Child Meta-Object" on page 5
- "Attributes of a Dynamic Child Meta-Object" on page 5

Why Use a Dynamic Child Meta-Object?

Create and use a dynamic child meta-object to cause the connector to do the following:

Service Call Requests

- Dynamically generate an output filename for each type of business object (based on the value inserted into the child's `OutFileName` attribute by the integration broker) or for each individual business object (if the integration broker specifies sequencing).

  Note: The connector uses the child's `FileWriteMode` attribute to determine whether to overwrite or append to the file specified in the child's `OutFileName` attribute.

- Return the name of each connector-generated output filename (if the child's OutFileName attribute does not contain a value). In this case, the connector does the following:
  - derives the name from the parent business object's name
  - writes the object to that file
  - populates the OutFileName meta-object attribute with the derived name
  - passes the derived name back to the integration broker, which obtains the dynamically created output filename without having specified it

Event Processing

The connector populates the child's InFileName attribute with the name of the file from which the business object was read.

How to Use a Dynamic Child Meta-Object

To cause the connector to process the filename dynamically, you must:

1. Create a dynamic child meta-object with specific attributes.
2. In the data business object, add an attribute that represents the dynamic child meta-object.
3. In the data business object, specify the following in the application-specific information at the business-object level:

   `cw_mo_JTextConfig = DynChildMOAttrName`

   where *DynChildMOAttrName* is the name of the attribute in the data business object that represents the dynamic child business object. For an example, see Figure 2.

   Important: The cw_mo_ prefix is required when you use a data handler. If the prefix is missing, the connector writes the dynamic child meta-object to the specified output file as if it were a data business object.
4. In the dynamic child meta-object, specify values for the attributes in the dynamic child meta-object.

Attributes of a Dynamic Child Meta-Object

A dynamic child meta-object must contain the following attributes:

- FileWriteMode — A string attribute whose value specifies whether the connector appends to or overwrites an existing output file. The value of this attribute can be either "a" for append or "o" for overwrite. The connector examines only the first letter and does not consider the value's case.
- InFileName — A string attribute that is populated with the event file name (file and absolute path from which the business object is obtained).
- OutFileName — A string attribute whose value can contain the filename, the absolute path and filename, or an FTP URL for the connector to use when writing to the output file.
  - If this attribute contains only the filename, the connector writes the specified file to the directory from which it was started.
  - If this attribute contains the absolute path and filename, the connector writes the specified file to the specified directory.
  - If this attribute contains only an FTP URL, the connector obtains the login, password, and port values from the EventDir attribute of the top-level JText meta-object.

– If this attribute contains an FTP URL that includes the login, password, and port values, the connector uses the values specified in this attribute and overrides those specified in the `EventDir` attribute of the top-level JText meta-object.

  For more information, see "Specifying a Remote FTP File System" on page 51.

Figure 2 illustrates an example Customer business object that contains a dynamic child meta-object.

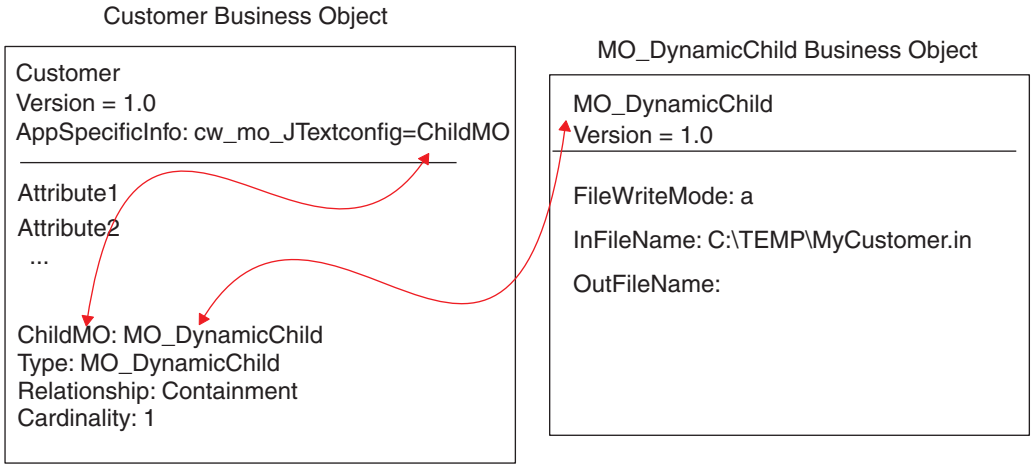


Figure 2. Example of a Dynamic Child Meta-Object

Using a Wrapper Business Object (Deprecated)

A **wrapper** business object also enables the filename to be exchanged with the integration broker. Because the wrapper contains the business object to be written to a file, however, using this format requires:

- subscribing the connector to both the wrapper and the data business object
- if ICS is the integration broker, binding the processing collaboration to the wrapper business object

Because the dynamic child meta-object is contained by the data business object rather than containing it, using a dynamic child meta-object to specify dynamic file information enables you to have the connector subscribe to the data business object and, if your integration broker is InterChange Server, to bind the collaboration only to the data business object.

Create and use a wrapper business object to cause the connector to do the following:

- Dynamically generate an output filename for each type of business object (based on the value inserted into the wrapper's `FileName` attribute by the integration broker) or for each individual business object (if the integration broker specifies sequencing).
- Return the name of each connector-generated output filename.

The wrapper business object must contain the following attributes:

- `FileName` — A string attribute whose value contains the absolute path and filename for the connector to use when writing to the output file. For example:

  UNIX:

  ```
  /tmp/Jtext/item.out
  ```

  Windows:

  ```
  C:\TEMP\MyCustomer.out
  ```

- `FileWriteMode` — A string attribute whose value specifies whether the connector appends to or overwrites an existing output file. The value of this attribute can be either "a" for append or "o" for overwrite. The connector examines only the first letter and does not consider the value's case.
- An attribute that contains the business object being processed. This attribute's `Type` property must specify the type of the child business object, its `Relationship` property must specify `Containment`, and its `Cardinality` property must evaluate to 1. There is no requirement for naming this attribute.

In addition to defining these three attributes, you must specify the following value for the wrapper's `AppSpecificInfo` property at the business-object level (case is not significant):

```
Type = JTextWrapper
```

Figure 3 illustrates a sample wrapper business object instance and the business object it contains.

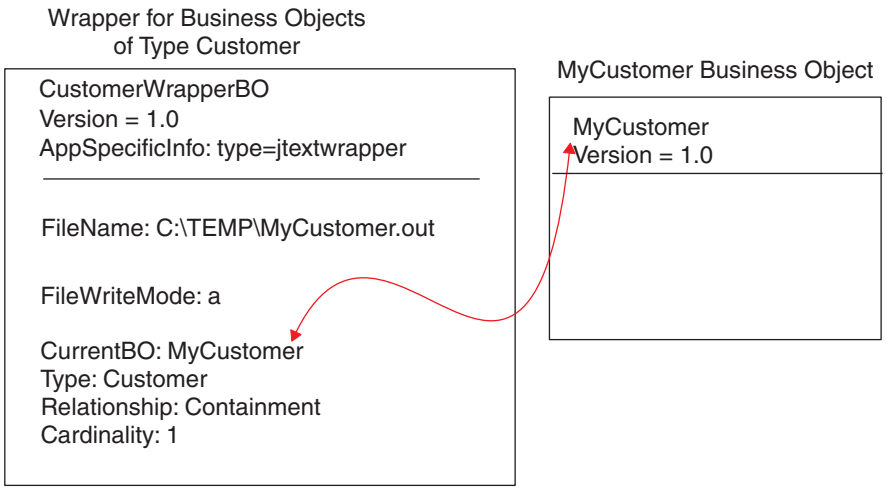


Figure 3. Example of a Wrapper Business Object Instance

For information on how the connector processes the wrapper business object, see "Business Object Request Processing" on page 11. For information on configuring the connector to use the wrapper business object, see "Specifying Request Processing" on page 44.

How the Connector Works

The JText connector communicates with an application through the exchange of text files. It performs the following primary tasks when processing business objects:

- Event polling and event notification

- Business object request processing

This section describes these tasks. It also explains how data handler processing works and how the JText connector processes verbs.

Event Notification

The JText connector handles events differently from other connectors. Unlike connectors that depend on third-party applications, the JText connector does not have an event table. Instead, it treats the event directory as an event table.

The following operations occur when the JText connector handles events:

1. The connector polls for events by checking specified directories for files with specified extensions. The presence of a file with the specified extension in the specified directory is considered the equivalent of an event. The connector reads event files directly from the event directory without interpretation. It uses the value specified as the `EndBODelimiter` to determine which substring represents each business object. For more information, see "Polling for Specific Business Objects and the EndBODelimiter" on page 49 and "Recovery from Business Object Delimiter Errors" on page 66.
2. The connector creates an instance of the data handler (based on values specified in the JText meta-object for the data business object).
3. The connector calls `getBOFromString()` on the data handler instance, and sends the text string that represents the business object to it. The connector passes each substring that represents a business object to the data handler. When a file represents multiple business objects, the connector sends only a substring (that is, a text-string representation of a single business object), not the entire file.
4. The data handler converts the text string to a business object and returns it to the connector. The data handler also reports errors and provides tracing.
5. The data handler performs default verb processing. The person who develops the data handler must specify logic for setting the verbs, because the connector does not provide this logic. For information on how the sample data handlers set verbs, see the see the *Data Handler Guide*.
6. If the data handler encounters any error that prevents it from creating a business object, the connector archives the string with the `.fail` extension. If the data handler succeeds, the connector checks for subscriptions to the business object.
   - If the connector does not subscribe to the business object, it writes it to an archive file with the `.unsub` extension.
   - If the connector subscribes to the business object, it sends the business object to the integration broker.
7. If the connector successfully sends the business object to the integration broker, it archives the file with the `.success` or `.partial` extension, depending on whether any business object in the event file has failed processing. If the connector fails to send the business object, it archives the file with the `.fail` extension.

Depending on its configuration, the JText connector can pick up all files in the event directory or pick up only those with a specified extension. For more information, see "Specifying Multiple Event Files or Multiple Event Directories" on page 48.

The JText connector processes event files in the order of their time stamps, from the oldest to the most recent, regardless of their location. In other words, the JText connector processes files located in separate directories in the chronological order of their time stamps.

The `PollQuantity` property specifies the maximum number of business objects that the connector can post to the integration broker in a given poll. For example, assume that the value of `PollQuantity` is set to `5` and that there are two files in a directory in which the connector is polling. The first file has four business object events and the second has 12 events. On the first poll call, the connector performs the following steps:

1. Sends all four business object events from the first file, archiving each business object as it processes it.
2. Sends the first business object event from the second file.

On the second poll call, the connector sends the 2nd through 6th events from the second file. On the third poll call, the connector sends the 7th through 11th events from the second file. On the fourth poll, the connector sends the last event. The connector archives each business object after processing it. If any of the business objects in a file fail processing, the connector archives the entire file with the `.orig` extension.

For more information, see:

- On using the `PollQuantity` property to tune performance, see "Tuning the Performance of the JText Connector" on page 59.
- On specifying the event directory and extension, see "Specifying Event Directories and Extensions" on page 41.
- On specifying event processing, see "Specifying Event Notification" on page 41.

Figure 4 shows an event notification operation (numbers in the graphic do not correlate to the steps outlined above).

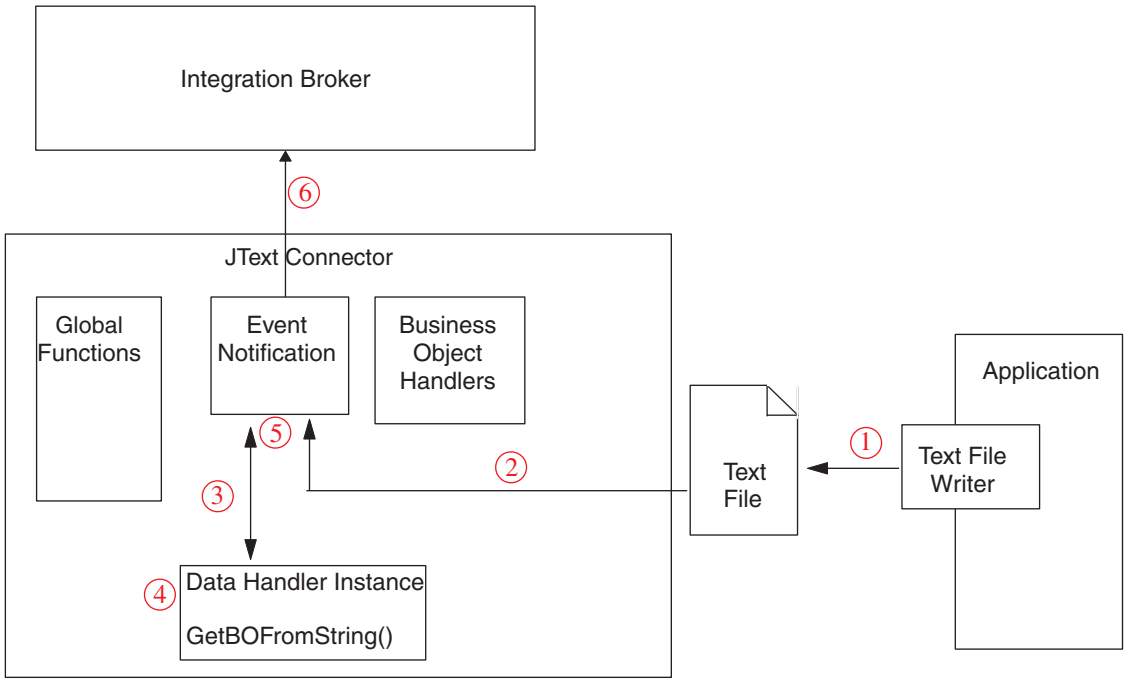


Figure 4. Event Notification Operation

Event Archiving

After it has processed an event, and if it is configured to enable archiving, the JText connector writes the business object string into a file in the archive directory. It names the file with an underscore (_), a time stamp, and a file extension that corresponds to the event status. The delivered default extensions are `success`, `partial`, `unsub`, `orig`, and `fail`. The underscore and time stamp are appended after the filename and before the file extension.

The time stamp is an underscore-separated list that contains the year, month, day, hour, minute, second, and millisecond of the system time. It ensures that archived filenames are unique and that the connector does not overwrite an existing file with the same name. The format of the archived file is:

BOName`_YYYY_MM_DD_HH_MM_SS_sss.[`*extension*`]`

For example, the connector might rename a successfully processed file named `Customer.in` to `Customer_2001_11_15_18_24_59_999.success`.

The JText connector archives a business object to the `.fail` file if a formatting error occurs, or if the connector fails to send the business object to the integration broker. The JText connector archives a business object to a file with an extension of `.unsub` if the connector does not subscribe to it. After you examine these archive files and correct any formatting errors or start the processes that subscribe to the business objects, resubmit the business objects in these files for processing.

For more information on archiving, see “Specifying Event Archiving” on page 42.

Default File Extensions for Event and Archive Files

Because the JText connector does not use event and archive tables, it updates event status by changing file extensions. Table 1 shows the default file extension values that WebSphere Business Integration Adapter delivers for event and archive files.

Table 1. Default File Extensions

File Type	Event Status/Description	Default File Extension	Delivered Default Directory
Event	new	`in`	UNIX:`/tmp/JTextConn/Default/Event` Windows:`C:\temp\JTextConn\Default\Event`
Archive	success (if all the business objects in the event file process successfully, this file contains all the business objects)	`success`	UNIX:`/tmp/JTextConn/Default/Archive` Windows:`C:\temp\JTextConn\Default\Archive`
Archive	success (if some of the business objects in the event file fail processing, this file contains only the successfully processed ones)	`partial`	UNIX:`/tmp/JTextConn/Default/Archive` Windows:`C:\temp\JTextConn\Default\Archive`
Archive	unsubscribed	`unsub`	UNIX:`/tmp/JTextConn/Default/Archive` Windows:`C:\temp\JTextConn\Default\Archive`
Archive	entire original event file (created only if any business object fails processing or is unsubscribed, even if the event file contains only one business object)	`orig`	UNIX:`/tmp/JTextConn/Default/Archive` Windows:`C:\temp\JTextConn\Default\Archive`
Archive	fail	`fail`	UNIX:`/tmp/JTextConn/Default/Archive` Windows:`C:\temp\JTextConn\Default\Archive`
Output	out	`out`	UNIX:`/tmp/JTextConn/Default/Out` Windows:`C:\temp\JTextConn\Default\Out`

Important: The access sequence among multiple applications that access and process the same file at the same time is important. Analyze all operations performed on a given file to avoid issues with file locking and incomplete data.

Note: The connector treats every file in the event directory with the specified extension as an input file. Ensure that the input file extension differs from the archive file extension, or that the input files and archive files are stored in different directories, to prevent the connector from treating an archived file as an event.

For information on specifying your own file extensions, event directory, and output directory, see Table 7 on page 28.

Business Object Request Processing

When processing a service call request, the connector converts the business object to an output string, then writes it to a file.

Before converting the business object, however, the connector determines whether the business object has been configured for dynamic file naming; that is, whether the business object contains a dynamic child meta-object or whether the top-level business object is a wrapper business object. In either of these cases, the connector dynamically names the output file or returns the name of the output file that it generates.

This section describes service call request processing when:

- "Data Business Object Does Not Specify Dynamic File Naming"
- "Data Business Object Contains a Dynamic Child Meta-Object" on page 12
- "A Wrapper Contains the Data Business Object" on page 13

Data Business Object Does Not Specify Dynamic File Naming

When the data business object does not specify dynamic file naming, the connector performs the following operations to handle service call requests:

1. The connector receives a business object request.
2. The connector determines that the AppSpecificInfo property at the business-object level does not contain either of the following:

   ```
   cw_mo_JTextConfig = DynChildMOAttrName
   ```

   or

   ```
   Type = JTextWrapper
   ```

   Note: If the business object's application-specific information contains both of these statements, the connector generates an error.
3. The connector checks the configuration of the top-level JText meta-object to determine which data handler to call. By default, this meta-object specifies the MO_DataHandler_DefaultNameValueConfig data-handler meta-object, which represents the NameValue data handler.
4. The connector creates an instance of the appropriate data handler and sends the business object to it.
5. The data handler converts the business object to a text string, which it returns to the configuration. The data handler also reports errors and provides tracing.
6. The connector writes the text string to a file.

For information on configuring the connector to process requests, see "Specifying Request Processing" on page 44.

Data Business Object Contains a Dynamic Child Meta-Object

When the data business object contains a dynamic child meta-object, the connector performs the following operations to handle service call requests:

1. The connector receives a business object request.
2. The connector determines that the AppSpecificInfo property at the business-object level contains the following text:

   ```
   cw_mo_JTextConfig = DynChildMOAttrName
   ```

   Note: If the business object's application-specific information does not specify a dynamic child meta-object and does not contain such a child, the connector processes the business object as described in "Data Business Object Does Not Specify Dynamic File Naming" on page 12.
3. The connector gets the name of the output file from the dynamic child meta-object's `OutFileName` attribute.
   - If this attribute contains a value, the connector checks whether a file by that name already exists. If the file does not exist, the connector creates a new output file, using the value of the attribute to name the file. If the file already exists, the connector appends to or overwrites the existing file (based on the value of the child meta-object's `FileWriteMode`).

     Important: If the value of the FileWriteMode attribute begins with any value other than an "o", the connector defaults to append mode.

- If this attribute does not contain a value (that is, `OutFileName=CxIgnore`), the connector derives the filename from the name of the parent business object that contains this child meta-object, and uses the configuration of the top-level JText meta-object to determine the output file's location. After writing the business object to the file, the connector returns the file's name and path in this attribute.

4. The connector checks the configuration of the top-level JText meta-object to determine which data handler to call. By default, this meta-object specifies the MO_DataHandler_DefaultNameValueConfig data-handler meta-object, which represents the NameValue data handler.
5. The connector creates an instance of the appropriate data handler and sends the business object to it.
6. The data handler converts the business object to a text string, which it returns to the configuration. The data handler also reports errors and provides tracing.
7. The connector writes the text string to a file whose name it derives in 3.

For information about the wrapper business object, see "Using a Dynamic Child Meta-Object" on page 4.

A Wrapper Contains the Data Business Object

When a wrapper contains the data business object, the connector performs the following operations to handle service call requests:

1. The connector receives a business object request.
2. The connector determines that the AppSpecificInfo property at the business-object level contains the following text:

   ```
   Type = JTextWrapper
   ```

   Note: If the business object's application-specific information does not specify a wrapper business object and is not contained by a wrapper, the connector processes the business object as described in "Data Business Object Does Not Specify Dynamic File Naming" on page 12.
3. The connector gets the name of the output file from the wrapper business object's `FileName` attribute.
   - If this attribute contains a value, the connector checks whether a file by that name already exists. If the file does not exist, the connector creates a new output file, using the value of the attribute to name the file. If the file already exists, the connector appends to or overwrites the existing file (based on the value of `FileWriteMode`).

     Important: If the value of the `FileWriteMode` attribute begins with any value other than an "o", the connector uses append mode.
   - If this attribute does not contain a value (that is, `FileName=CxIgnore`), the connector derives the filename from the name of the business object contained in the wrapper and uses the configuration of the top-level JText meta-object to determine the output file's location. After writing the business object to the file, the connector returns the file's name and path in this attribute.
4. The connector locates the first attribute in the wrapper business object whose `Type` property evaluates to a business object. The connector uses the type specified as the type of business object to process. If the wrapper does not contain an attribute that specifies a business object type or the attribute does not specify a type, the connector processes the wrapper business object as it would a standard business object.

5. The connector checks the configuration of the top-level JText meta-object to determine which data handler to call. By default, this meta-object specifies the `MO_DataHandler_DefaultNameValueConfig` data-handler meta-object, which represents the NameValue data handler.
6. The connector creates an instance of the appropriate data handler and sends the business object to it.
7. The data handler converts the business object to a text string, which it returns to the configuration. The data handler also reports errors and provides tracing.
8. The connector writes the text string to a file whose name it derives in Step 3.

For information about the wrapper business object, see "Using a Wrapper Business Object (Deprecated)" on page 6.

Figure 5 illustrates the JText connector components when the connector processes requests from an integration broker to the destination application.

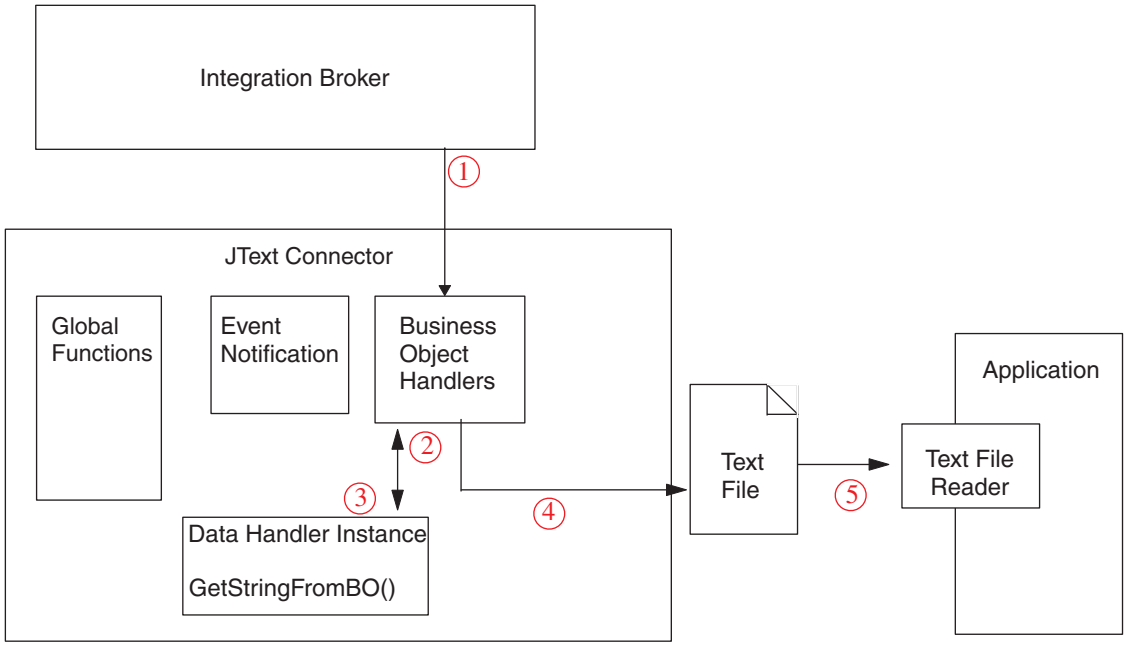


Figure 5. Business Object Request Operation

How Data Handler Processing Works

The connector uses a data handler instance to convert between business objects and strings that are read from event files. The data handler instance also reports errors and provides tracing.

The connector creates an instance of a data handler based on the value of the `EventDataHandler` and `OutputDataHandler` attributes in the top-level JText meta-object. These attributes identify the data-handler meta-object that the connector uses to create the instance of the data handler. The data-handler meta-object can represent a delivered or custom data handler. For more information, see the *Data Handler Guide*.

After receiving the configuration information, the connector performs the following steps:

1. Instantiates a data handler.
2. Calls the data handler's `setOption()` method to set the data handler's `TracingSubSystem` attribute to the connector's name. The data handler uses this value to include the connector's name in the trace messages it writes.

After the data handler has been created and configured, the connector calls the appropriate methods in the data handler to perform the conversion of data to or from a business object.

- For event notification, the connector calls the `getBOFromString()` method on the data handler. The connector passes to the data handler the string from a file that is be converted to a business object. The data handler returns a business object.
- For request processing, the connector calls the `getStringFromBO()` method on the data handler. The connector passes to the data handler the business object to be converted to a string. The data handler returns a serialized version of the business object, in the form of a string.

The `getBOFromString()` and the `getStringFromBO()` methods always send or receive the entire business object hierarchy of a top-level parent and all of its child business objects, respectively.

In either case, the data handler is responsible for filtering out any meta-object data so that it passes only business object-specific data. The product-delivered data handlers provide this functionality. If you use custom data handlers, they must also provide this functionality.

Business Object Verb Processing for Requests

When handling requests, the JText connector does not handle one verb differently from another. It writes to files without performing update, retrieve, or delete operations, regardless of the verb associated with the business object.

When processing requests, the JText connector sets all attributes with a value of `CxIgnore` to their default values if the following conditions are true:

- The verb is Create.
- The connector's `UseDefaults` property is set to `true`.
- The attribute is set to `Required`.
- Default values have been set for the attributes in the business object specification.

Connector Features

Along with event notification and business object request processing, the JText connector provides the following capabilities:

- Varied configurations for different business objects; for example, you can configure different business objects to use different directories and file extensions, or different data formats.
- Configuration capabilities for file extensions, directory location for archive file storage, format type, and file sequencing.
- Configuration capabilities for dynamically determining the output filename for each business object, or for returning the full name of a generated output file.
- Failure recovery.

- Custom data handler capabilities, which means that you can create your own data handler without recompiling the connector code. You need only change the configuration properties to use the new class you have created.
- The ability to exchange data with remote FTP locations as well as local file system directories.

For more information, see Chapter 2, "Installing and Configuring the JText Connector", on page 17, Chapter 3, "Using JText Connector Meta-Objects", on page 25, and the *Data Handler Guide*.

JText Differences from Other Connectors

While the JText connector enables the transfer of data from a source application to a destination application like other connectors, it is unique in the following ways:

- It processes all business objects in the same way. In other words, because it always writes the business object to a file, it performs only Create operations (regardless of the incoming verb).
- It does not interpret the contents of the business objects that it handles. In other words, it reads each business object as a potential string in which key values have no more significance than other text.
- It uses meta-object values for much of its configuration. For more information, see Chapter 3, "Using JText Connector Meta-Objects", on page 25.
- It does not have an event table. Instead, it treats the configured event directory as an event table.

Processing Locale-Dependent Data

The connector has been internationalized so that it can support double-byte character sets, and deliver message text in the specified language. When the connector transfers data from a location that uses one character code set to a location that uses a different code set, it performs character conversion to preserve the meaning of the data. The Java runtime environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encodings for characters in most known character code sets (both single-byte and multibyte). Most components in the IBM CrossWorlds system are written in Java. Therefore, when data is transferred between most IBM CrossWorlds components, there is no need for character conversion. To log error and informational messages in the appropriate language and for the appropriate country or territory, configure the Locale standard configuration property for your environment. For more information on these properties, see ″Standard Configuration Properties for IBM CrossWorlds Connectors.″

Chapter 2. Installing and Configuring the JText Connector

This chapter describes how to install and configure the JText connector.

Prerequisites

Before running the JText connector, create read/write permissions on the event, output, and archive directories that will contain the text files that the connector reads from and writes to.

Installing the Connector

The following subsections describe how to install the JText connector on a UNIX or Windows system.

After your business-integration system is installed, you can install additional connectors from the product CD at any time. To do this, insert the product CD, run the installation program, and choose the connectors that you want to install.

Note: Unless otherwise indicated, the remaining sections in this chapter apply to both UNIX and Windows installations of the JText connector.

Installing on a UNIX System

To install the JText connector on a UNIX system, run the Installer for IBM WebSphere Business Integration Adapter and select the IBM WebSphere Business Integration Adapter for JText.

Table 2 describes the UNIX file structure used by the connector.

Table 2. Installed UNIX File Structure for the JText Connector

Subdirectory of $CROSSWORLDS	Description
`connectors/JText/CWJText.jar`	Contains the connector and the formatters
`connectors/JText/start_JText.sh`	System startup script for the connector. This script is called from the generic connector manager script. When you click Install from Connector Configurator (with WMQI as the integration broker) or Connector Designer (if InterChange Server is the integration broker), the installer creates a customized wrapper for this connector manager script. When the connector works with ICS, use this customized wrapper to start and stop the connector. When the connector works with WMQI, use this customized wrapper only to start the connector; use `mqsiremotestopadapter` to stop the connector.

Table 2. Installed UNIX File Structure for the JText Connector (continued)

Subdirectory of $CROSSWORLDS	Description
connectors/messages	Contains the JTextConnector.txt file, as well as JTextConnector_ll_TT.txt files (message files specific to a language (ll) and a country or territory (TT)).
DataHandlers/CwDataHandler.jar DataHandlers/CustDataHandler.jar DataHandlers/CwXMLDataHandler.jar DataHandlers/CwEDIDataHandler.jar	CwDataHandler.jar contains product-delivered data handlers. CustDataHandler.jar contains custom data handlers. CwEDIDataHandler.jar contains the EDI data handler. CwXMLDataHandler.jar contains the XML data handler.
DevelopmentKits/edk/DataHandler/Samples/delimited.java DevelopmentKits/edk/DataHandler/Samples/fixedwidth.java DevelopmentKits/edk/DataHandler/Samples/namevalue.java	Sample code for data handlers, available as examples for custom development.
connectors/JText/SampleFormatters/DHFormatter.java connectors/JText/SampleFormatters/BySize.java connectors/JText/SampleFormatters/ByNameValue.java connectors/JText/SampleFormatters/ByDelimiter.java	Formatters have been deprecated

After you have installed the connector on a UNIX system, you must do the following:

1. Use the Connector Configuration Tool to generate the customized connector wrapper (connector_manager_JText) required to start the connector. Refer to the *System Installation Guide for UNIX* or the *WebSphere Business Integration Adapters Implementation Guide for MQIntegrator* for more information.
2. Log in as root and run the dbenable.sh script in the JText/ bin directory.

Installing on a Windows System

To install the JText connector on a Windows system, run the Installer for IBM WebSphere Business Integration Adapter and select the IBM WebSphere Business Integration Adapter for JText. The Installer installs standard files associated with the JText connector as well as files that can be used to develop custom components for the JText connector.

Table 3 describes the Windows file structure used by the JText connector.

Table 3. Installed Windows File Structure for the JText Connector

Subdirectory of %CROSSWORLDS%	Description
connectors\JText\CWJText.jar	Contains the connector and the formatters
connectors\JText\start_JText.bat	Starts the connector on Windows
connectors\messages	Contains the JTextConnector.txt file, as well as JTextConnector_ll_TT.txt files (message files specific to a language (ll) and a country or territory (TT)).

Table 3. Installed Windows File Structure for the JText Connector (continued)

Subdirectory of %CROSSWORLDS%	Description
DataHandlers/CwDataHandler.jar DataHandlers/CustDataHandler.jar DataHandlers/CwXMLDataHandler.jar DataHandlers/CwEDIDataHandler.jar	CwDataHandler.jar contains product-delivered data handlers. CustDataHandler.jar contains custom data handlers. CwEDIDataHandler.jar contains the EDI data handler. CwXMLDataHandler.jar contains the XML data handler.
DevelopmentKits\edk\DataHandler\Samples\delimited.java DevelopmentKits\edk\DataHandler\Samples\fixedwidth.java DevelopmentKits\edk\DataHandler\Samples\namevalue.java	Sample code for data handlers, available as examples for custom development.
connectors\JText\SampleFormatters\DHFormatter.java connectors\JText\SampleFormatters\BySize.java connectors\JText\SampleFormatters\ByNameValue.java connectors\JText\SampleFormatters\ByDelimiter.java	Formatters have been deprecated

Installer adds an icon for the connector file to the business-integration product menu. For a fast way to start the connector, create a shortcut to this file on the desktop.

Note: For more information on the Installer, refer to the *System Installation Guide for Windows* or the *WebSphere Business Integration Adapters Implementation Guide for MQIntegrator*.

Verifying Installation of the Data Handler

In addition to the files loaded specifically for the JText connector when it is installed, the installation program automatically selects the Data Handlers installation option to load the product-delivered data handlers. Ensure that these files have also been loaded.

Table 4 describes the file structure used by the data handlers.

Table 4. Data Handler Files

Subdirectory of %CROSSWORLDS% or $CROSSWORLDS	Description
DataHandlers	Contains CWDataHandler.jar file for compiled versions of the product-provided data handlers. Also contains the empty CustDataHandler.jar file for custom data handlers. Also contains the CwXMLDataHandler.jar data handler for XML and the CwEDIDataHandler.jar data handler for EDI.

Configuring the Connector

If you are using WMQI as the integration broker, you configure connector properties from Connector Configurator. If you are using ICS as the integration broker, you configure connector properties from Connector Designer, which you access from the IBM CrossWorlds System Manager (CSM). Regardless of the integration broker or the configuration tool used, you also configure meta-objects to enable the connector to process different business objects differently.

Note: It is inadvisable to run the JText Connector with the Parallel Process Degree Resource set to a value greater than 1. For more information on Parallel Process Degree, see the *System Administration Guide*.

Configuring Connector Properties

A connector obtains its configuration values at startup. During a runtime session, you might want to change the values of one or more connector properties. Changes to connector properties can be:

- Dynamic—These changes take effect immediately after they are made.
- Static—These changes require either connector component restart or system restart before they take effect.

To determine whether a property is dynamic or static, refer to the configuration utility for your integration broker.

Connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

You must set the values of some of these properties before running the connector.

Configuring Business Object Processing

You use meta-objects to configure the following aspects of connector behavior:

- Which data handler to use.
- From what directory to poll for event files.
- What file extension to use for event files.
- If archiving, what directory to use for archiving files.
- If archiving, what file extension to use when archiving files that have been processed.
- What format type to use for event and output files.
- Whether to pick up events from different objects in different directories or pick up multiple files in the same event directory.

For information on meta-objects, see Chapter 3, "Using JText Connector Meta-Objects", on page 25.

Standard Configuration Properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard Configuration Properties for Connectors", on page 75 for documentation of these properties.

Table 5 provides information specific to this connector about configuration properties in the appendix.

Table 5. Property Information Specific to This Connector

Property	Note
`CharacterEncoding`	Because this connector is Java-based, it does not use this property.

Table 5. Property Information Specific to This Connector (continued)

Property	Note
Locale	Because this connector has been internationalized, you can change the value of this property. See release notes for the connector to determine currently supported locales.

Connector-Specific Properties

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

Table 6 lists the connector-specific configuration properties for the connector. See the sections that follow for explanations of the properties.

Table 6. Connector-Specific Configuration Properties

Name	Possible Values	Default Value	Required?
ArchivingEnabled	true or false	true	Yes
EventLog	*Name and location of file*	event.log	No
EventRecovery	abort or retry	retry	Yes
FTPPollFrequency	*number of poll cycles*		No
GenerateTemplate	*BOName*		No
OutputLog	*File that registers the next sequence number for each incoming business object during request processing*	Output.Log	No
PollQuantity	*Number of events processed at each poll*	25	No

ArchivingEnabled

Turns on archiving. If this property is set to true, the event file is archived in the archive directory with the specified extension. If this property is set to false, the event file is not archived. In this case, the connector deletes the file after sending all events to the integration broker. For more information, see "Specifying Event Archiving" on page 42.

The default value is true.

EventLog

Provides file storage location for events that are generated by the connector. This file is located in the JText subdirectory in the connectors directory where the product is installed.

The default value is event.log.

EventRecovery

Specifies recovery behavior. If this property is set to retry, the connector uses the event.log file to recover failed events. If this property is set to abort, the connector terminates when it encounters a failed event. For more information, see "Event Log File" on page 64.

The default value is retry.

FTPPollFrequency
Determines how frequently the connector polls an FTP server measured in the number of standard poll cycles. For example, if PollFrequency is set to `10000`, and `FTPPollFrequency` is set to `6`, the connector polls the local event directory every 10 seconds and polls the remote directory every 60 seconds. The connector performs FTP polling only if you specify a value for this property. If `FTPPollFrequency` evaluates to `0` or blank, the connector does not perform FTP polling. By default it does not.

There is no default value for this property.

GenerateTemplate
Enables the connector to generate a template for each supported business object after connector startup. The syntax for this property is *BOName*;*BOName* where the name of a specific business object is substituted for *BOName*. For example, to generate two templates, one for a Customer business object and one for an Item business object, specify `Customer;Item`. For more information, see "Generating Sample Business Objects for Testing" on page 61.

There is no default value for this property.

OutputLog
Specifies the name of the file that stores the sequence number that the connector uses to create unique output files for each type of business object during request processing. The format of the file is:

```
BusinessObjectName = NextSequenceNumber
```

where *BusinessObjectName* is the name of the request business object, and *NextSequenceNumber* represents the sequence number of the most recently received business object, incremented by one. For example, if the connector is processing Customer and Item business objects, the output log file might contain the following:

```
Customer = 12
Item = 2
```

This file indicates that the connector has already processed 11 Customers and 1 Item. The next Customer and Item business objects will be written to the `Customer_12.out` and `Item_2.out` files, respectively. When it receives a request Order business object, the connector adds a new row to the output log file and writes the business object to the `Order_1.out` file.

If `FileSeqEnabled` is set to `true`, the connector uses this sequence number to uniquely name the output files that it creates for each business object. The connector names each output file by appending an underscore (_) and the sequence number to the business object's name or to a file whose name is specified in the `OutputFileName` meta-object attribute. Because the output log is stored in user-readable format, you can use a standard text editor to read the file or to reset its value.

For more information on the OutputFileName attribute, see "Specifying the Name of the Output File" on page 34. For more information about the output log, see "Specifying Request Processing" on page 44. For information on returning the generated file's name, see "Returning a File's Name" on page 45.

The default is `Output.Log`.

PollQuantity

Specifies the number of events to process for each poll. The connector poll method retrieves the specified number of event records and processes them in a single poll. Processing multiple events per poll can improve performance when the application generates large numbers of events. However, because integration-broker requests are blocked while the poll method is processing events, do not set the number of events too high. If each poll call takes a long time, it delays integration-broker request operations. For more information, see "Tuning the Performance of the JText Connector" on page 59.

The default value is `25`.

Adding Supported Business Objects

By default, the JText connector supports the `MO_JTextConnector_Default` meta-object. To fully configure the connector, use Connector Configurator (if WMQI is the integration broker) or Connector Designer (if ICS is the integration broker) to add other required business objects to its list of supported business objects. Depending on how you use the connector, you may need to add all or many of the following business objects:

- The meta-object for the data handler (which is specified in the `EventDataHandler` and `OutputDataHandler` attributes of the `MO_JTextConnector_Default` meta-object). By default, these attributes specify the MO_DataHandler_DefaultNameValueConfig data-handler meta-object, which represents the NameValue data handler. For more information, see "Specifying a Data Handler" on page 57.
- `MO_JTextConnector_`*`BusObjName`* — if you create meta-objects for specific business objects. For more information, see "Creating a JText Meta-Object for a Specific Business Object" on page 57.
- A wrapper business object that contains the business object being processed—if you use the deprecated format to configure the connector for dynamically file naming. For more information, see "Dynamic File Naming" on page 44 and "Returning a File's Name" on page 45.
- Business objects that are to be read from or written to a file. For more information, see "Business Objects Used by the JText Connector" on page 3.

Connector Startup

For information on starting a connector, stopping a connector, and the connector's temporary startup file, see the *WebSphere Business Integration Adapters Implementation Guide for MQ Integrator*, or, for ICS, see the IBM CrossWorlds *System Administration Guide*.

Chapter 3. Using JText Connector Meta-Objects

A **meta-object** is a WebSphere Business Integration Adapter business object that contains configuration information used by a connector or a data handler. The JText connector requires each of its supported business objects to have an associated JText meta-object for that business object type. This top-level meta-object contains at least one child meta-object.

- The connector uses the top-level JText meta-object to obtain configuration information such as the data handler to use for data conversion, the paths of the business object's event, archive, and output directories, the file extensions for its event, archive, and output files, information that is required if the connector is processing files on an FTP system, and whether the connector generates unique file identifiers for its output files.
- The connector uses a child meta-object to specify configuration values for the data handler to use when converting data between the business object and a string. By default, the top-level meta-object specifies the NameValue data handler to convert data.

To provide different configuration information for each business object that the connector supports, you can create a custom top-level JText meta-object for each one. Because each top-level meta-object specifies its own data-handler meta-object, the connector can process each type of business object in a different format. The data-handler meta-object eliminates the need to edit a business object definition or to modify the connector itself when you introduce new text formats or make changes to existing formats.

Meta-objects are loaded into memory at startup, making their configuration information available to the connector. Note that meta-objects are not sent to the integration broker for processing. They affect the behavior only of the connector.

This chapter describes how to configure the JText connector by using JText meta-objects. For information on using data-handler meta-objects, see the *Data Handler Guide*.

JText Meta-Object Naming Conventions

The name of a top-level JText meta-object has three components, as illustrated by the name of the default top-level meta-object, `MO_JTextConnector_Default`. The components of a top-level JText meta-object name are as follows:

- MO_ is a prefix that indicates a meta-object.

- `ConnectorInstanceName_` specifies the name of the connector instance, such as JText. This name is configurable to support the use of multiple connector instances. For example, a connector named `JText2` might have a meta-object named `MO_JText2Connector_Default`.
- `Default` specifies the name of the associated business object. To create a meta-object for a specific business object, change the string *`Default`* to the name of the business object, as in `MO_JTextConnector_Customer` for a business object named *Customer*. You can include additional components and underscores in the meta-object name. For example, the Oracle_Customer business object would be associated with the `MO_JTextConnector_Oracle_Customer` meta-object. The connector uses default meta-objects whenever corresponding business object-specific meta-objects do not exist.

For information on creating meta-objects for a specific business object, see "Creating a JText Meta-Object for a Specific Business Object" on page 57.

JText Meta-Object Structure

A JText meta-object has a hierarchical structure. The default top-level meta-object is named `MO_JTextConnector_Default`. Two attributes of the top-level meta-object, `EventDataHandler` and `OutputDataHandler`, represent child meta-objects that provide configuration information for the data handler that the connector uses. The connector uses the data handler to convert data between business objects and strings.

By default, both of these attributes specify the same data-handler meta-object (MO_DataHandler_DefaultNameValueConfig). This data-handler meta-object calls the NameValue data handler to actually convert the data. In other words, the delivered default configuration specifies that event and output file conversion use the same data handler. For information on instantiating a data handler, see the *Data Handler Guide*.

Note: Because formatter usage has been deprecated in favor of data handler usage, the `EventFormat` and `OutputFormat` attributes that formerly represented a formatter have been removed from the `MO_JTextConnector_Default` meta-object. To use a formatter, you must:

- add the `EventFormat` and `OutputFormat` attributes to the top-level meta-object
- specify the appropriate business object as the Type of these attributes
- change the Type of the `EventDataHandler` and `OutputDataHandler` attributes to `String`

For information on using a formatter, see the documentation for the 3.0.0 or 2.3.0 release of the JText connector.

Figure 6 shows the hierarchical structure for the default JText meta-objects and each attribute name and type.

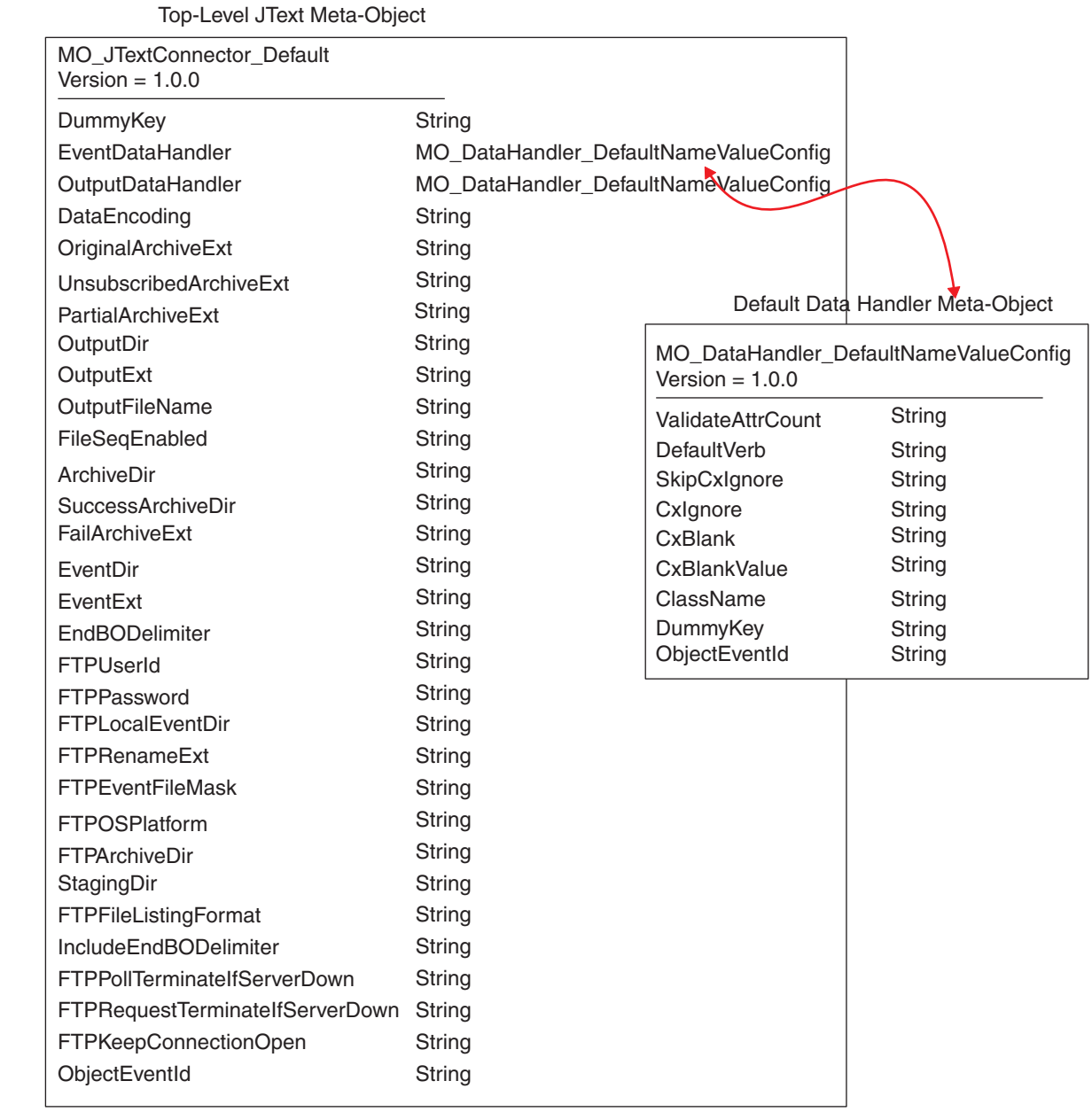


Figure 6. Hierarchical Structure of the JText Meta-Object

Creating Custom Meta-Objects

`MO_JTextConnector_Default`, as the top-level JText meta-object, contains configuration information and child meta-objects for the connector. You can create separate top-level meta-objects for each type of business object that the connector handles. These custom meta-objects can contain the same or different child meta-objects to configure the type of data handler. For example, to configure processing differently for the Customer and Item business objects, create the `MO_JTextConnector_Customer` and `MO_JTextConnector_Item` meta-objects, and design these top-level meta-objects to contain different data-handler meta-objects.

At initialization, the connector retrievesa list of its supported meta-objects and business objects from the integration broker . From the names of these objects, the connector determines which business objects have their own associated top-level meta-objects. At runtime, the connector matches the name of a request business object with one of its supported meta-objects to locate the appropriate configuration information.

For example, assume that the connector supports the following meta-objects:

- MO_JTextConnector_Default
- MO_JTextConnector_Customer
- MO_JTextConnector_Item

and the following business objects:

- Customer
- Item
- Order

When the integration broker sends a request Customer business object, the connector uses the configuration information specified in the MO_JTextConnector_Customer meta-object. When the integration broker sends a request Order business object, the connector uses the configuration information specified in the MO_JTextConnector_Default meta-object.

MO_JTextConnector_Default Attributes

This section describes the attributes in the MO_JTextConnector_Default meta-objects.

Note: All values in an attribute's DefaultValue property are case-sensitive. Directory information must specify the absolute path of a directory.

Table 7 and the following sections describe the functionality of each attribute in the MO_JTextConnector_Default meta-object. Among other information, this table includes the value provided for each simple attribute's DefaultValue property. You can replace the product-delivered value with your own value.

Table 7. Attributes in the MO_JTextConnector_Default Meta-object Definition

Attribute Name	Description
ArchiveDir	Specifies the absolute path of the Archive directory. The directory must already exist. The delivered default values are: UNIX: /tmp/JTextConn/Default/archive Windows:C:\temp\JTextConn\Default\Archive
DataEncoding	DataEncoding is the encoding to be used to read and write business object strings. If this property is not specified in the static meta-object, the connector tries to read or write the business object string without using any specific encoding. You can specify any Java-supported encoding set for this attribute.
DummyKey	This attribute exists to satisfy the requirement that one attribute in every business object definition have the Key property enabled.
EndBODelimiter	Specifies a delimiter that separates business objects within an input file. For more information on the EndBODelimiter attribute, see "Polling for Specific Business Objects and the EndBODelimiter" on page 49. If you do not provide a default value during configuration, the connector inserts the following value: <EndBO:*BOName*>.

Table 7. Attributes in the MO_JTextConnector_Default Meta-object Definition (continued)

Attribute Name	Description
EventDataHandler	Represents a child meta-object whose attributes provide configuration values for the data handler to be used for event processing (business object string converted to business object). The delivered default value is MO_DataHandler_DefaultNameValueConfig
EventDir	Specifies the absolute path of the Event directory. The directory must already exist. If you create separate meta-objects for different business objects, and you specify the same EventDir path for both, you must specify unique values for the EventExt attribute in each meta-object. For more information, see "Specifying Event Directories and Extensions" on page 41. To configure the connector to use a remote FTP file system for event processing, specify the FTP URL in this attribute. Optionally, you can use this attribute to specify the following additional information in the URL: • the id and password of a user with privileges to connect to the FTP server and perform FTP operations; if not specified in EventDir, must be specified in FTPUserId and FTPPassword. • the FTP port; if not specified in EventDir, the connector uses the default FTP port. • the remote event directory; if not specified in EventDir, the connector polls the event files from the directory to which the connection is established to the FTP server. Syntax for specifying FTP information in the EventDir attribute is: ftp://[*UserId*:*password*@]*FTPserver*[:*port*][*RemoteEventDirectory*] For more information, see "Remote Event Processing" on page 51. To specify local file information in the EventDir attribute, use the full path of the file. Alternately, you can use a FILE URL, which uses the following format: [file://]*FullPathname* The delivered default values are: UNIX: /tmp/JTextConn/Default/event Windows:C:\temp\JTextConn\Default\Event
EventExt	Specifies the extension of the file used for event notification. If no value is specified, the JText connector polls for files with no file extension. For more information, see "Specifying Multiple Event Files or Multiple Event Directories" on page 48. **Note:** The use of an asterisk (*) for this attribute to specify that the connector poll for all files in a single event directory regardless of their extension is no longer supported. The delivered default value is in.
FailArchiveExt	Specifies the file extension used to archive business objects that were not successfully processed. For more information, see "Specifying Event Archiving" on page 42. The delivered default value is fail.
FileSeqEnabled	Specifies filename sequencing, which outputs each business object to a separate file. The file's name includes a unique sequence number. For more information, see "Specifying Request Processing" on page 44. The delivered default value is true.

Table 7. Attributes in the MO_JTextConnector_Default Meta-object Definition (continued)

Attribute Name	Description
FTPArchiveDir	Specifies the absolute path of the archive directory on the FTP server. The directory must already exist. There are several options for using this attribute to specify archiving: • Specifying a value for this attribute but no value for the FTPRenameExt attribute causes the connector to append a timestamp to the event file and move it to the FTP server archive directory specified in this attribute. • Specifying a value both for this attribute and the FTPRenameExt attribute causes the connector to rename the processed event file with a timestamp and the value specified in FTPRenameExt, and move it to the FTP server archive directory specified in this attribute. • Specifying no value either for this attribute or the FTPRenameExt attribute causes the connector to delete the processed event file without archiving it. • Specifying no value for this attribute but specifying a value for the FTPRenameExt attribute causes the connector to rename the processed event file with a timestamp and the value specified in FTPRenameExt, and move it to the directory specified in the EventExt attribute. • Specifying / (slash) for this attribute but no value for the FTPRenameExt attribute causes the connector to move the processed event file to the root directory on the FTP server. • Specifying / (slash) for this attribute and a value for the FTPRenameExt attribute causes the connector to rename the processed event file with the extension specified in FTPRenameExt, and move it to the root directory on the FTP server. For more information, see "Specifying Event Archiving" on page 42. There is no delivered default value for this attribute.
FTPEventFileMask	Uses embedded wildcard characters to specify the mask or prefix of remote FTP files for event processing. Specify a value for this attribute only to identify the file mask on a mainframe that does not adhere to the same naming standards that apply to UNIX or Windows systems. Using wildcard characters in the file name enables you to specify multiple files for event processing. For example, you can use the following format to specify multiple event files: ACT.Z1UC.INPT*For more information, see "Identifying Files on a Mainframe: Optional Configuration" on page 54. There is no delivered default value.
FTPFileListingFormat	Specifies the format in which the JText connector should expect file information to appear when reading in files. This enables the connector to read in files in different locales where date and time information may be stored in different orders within the file format information. To configure the connector to use the format for your locale, specify a semicolon-delimited series of characters that represent the order in which file attributes occur; below is a list that associates the possible characters with the file attributes they represent. P Permission L Links U User G Group S Size D Date M Month T Time N Name A suitable value for this attribute, then, might be P;L;U;G;S;D;M;T;N.

Table 7. Attributes in the MO_JTextConnector_Default Meta-object Definition (continued)

Attribute Name	Description
FTPKeepConnectionOpen	Set the Default Value property of this attribute to the value true to cause the JText connector to maintain its connection with an FTP site. If this attribute is set to the value true then the connector only closes the connection when the connector terminates or if the FTP server closes the connection itself (due to a configured timeout, for instance). The connector checks to make sure that the connection is still alive each time it performs a remote operation in order to handle the situation when the FTP server might have closed the connection due to a timeout. If the connection has been closed then the connector re-establishes it. Set the Default Value property of this attribute to the value false to cause the JText connector to open a connection with the FTP server each time it performs an operation and to close the connection when it is finished. Configuring the connector to keep the connection alive can improve the performance of the connector when performing request processing on FTP sites.
FTPLocalEventDir	Specifies the local system directory into which the connector downloads event files from the FTP site. You must specify a value for this attribute to enable the connector to process events using FTP. For more information, see “Specifying the Local Directory” on page 52. There is no delivered default value.
FTPOSPlatform	Use this attribute only if configuring the connector to use a remote FTP file system where the remote FTP server is an MVS platform. In this case, specify the value of this attribute as MVS. Case is not significant. For more information, see “Specifying a Remote FTP File System” on page 51. There is no delivered default value.
FTPPassword	Specifies the password of the user who has privileges to connect to the FTP server and perform FTP operations. You need not specify a value for this attribute if the password is included in the URL specified in the EventDir or OutputDir attribute. For more information, see “Specifying the FTP URL and Login Information” on page 52. There is no delivered default value for this attribute.
FTPPollTerminateIfServerDown	Specifies the behavior of the connector when configured to poll the FTP site for events and the FTP site is unavailable. If the Default Value property of the FTPPollTerminateIfServerDown attribute is set to the value true and the FTP site is unavailable when the connector attempts a poll call, then the connector terminates. If the Default Value property of the FTPPollTerminateIfServerDown attribute is set to the value false and the FTP site is unavailable when the connector attempts a poll call, then the connector does not terminate. There is no delivered default value.
FTPRenameExt	Specifies the file extension or suffix that the connector uses to rename the remote FTP file after the connector has polled for it. Renaming the file prevents the connector from polling the same file in the next poll cycle. Alternatively, you can configure the connector to rename the processed event file and move it to an archive directory. For more information, see the FailArchiveExt attribute. For more information, see “Identifying Files on a Mainframe: Optional Configuration” on page 54. There is no delivered default value.

Table 7. Attributes in the MO_JTextConnector_Default Meta-object Definition (continued)

Attribute Name	Description
FTPRequestTerminateIfServerDown	Specifies the behavior of the connector when configured to perform request processing and communicate with an FTP site, and the FTP site is unavailable. If the Default Value property of the FTPRequestTerminateIfServerDown attribute is set to the value true and the FTP site is unavailable when the connector attempts to perform request processing, then the connector terminates. If the Default Value property of the FTPRequestTerminateIfServerDown attribute is set to the value false and the FTP site is unavailable when the connector attempts to perform request processing, then the connector does not terminate. There is no delivered default value.
FTPUserId	Specifies the name of the user who has privileges to connect to the FTP server and perform FTP operations. You need not specify a value for this attribute if the UserId is included in the URL specified in the EventDir or OutputDir attribute. The connector ignores this attribute if it does not find an FTP URL in the EventDir attribute (during event processing) or OutputDir attribute (during request processing). For more information, see "Specifying the FTP URL and Login Information" on page 52. There is no delivered default value for this attribute.
IncludeEndBODelimiter	Specifies whether or not the value specified for the EndBODelimiter meta-object attribute is included in the string written to a file by the JText connector. If the Default Value property of this attribute is set to true then the connector includes the value specified for the EndBODelimiter attribute when it writes files. If the Default Value property of this attribute is set to false then the connector does not include the value specified in the EndBODelimiter attribute when it writes files.
ObjectEventID	Placeholder not used by the connector in a meta-object but required by the integration broker. This attribute must be the last attribute in the meta-object. There is no delivered default value.
OriginalArchiveExt	Specifies the file extension used to archive the original event file, which preserves the entire event file for reference in case any of its business objects fail processing or are unsubscribed. For more information, see "Specifying Event Archiving" on page 42. The delivered default value is orig.
OutputDataHandler	Represents a child meta-object whose attributes provide configuration values for the data handler to be used for service call requests (business object converted to business object string). The delivered default value is MO_DataHandler_DefaultNameValueConfig

Table 7. Attributes in the MO_JTextConnector_Default Meta-object Definition (continued)

Attribute Name	Description
OutputDir	Specifies the absolute path of the Output directory. The directory must already exist. To configure the connector to use a remote FTP file system for request processing, specify the FTP URL in this attribute. Optionally, you can use this attribute to specify the following additional information in the URL: • the UserId and password of a user with privileges to connect to the FTP server and perform FTP operations; if not specified in EventDir, must be specified in FTPUserId and FTPPassword. • the FTP port; if not specified in OutputDir, the connector uses the default FTP port. • the remote output directory; if not specified in OutputDir, the connector loads request files into the default connection directory (the directory on the FTP server to which the connection is established). Syntax for specifying FTP information in the OutputDir attribute is:ftp://[*UserId*:*password*@]*FTPserver*[:*port*]For more information, see "Remote Request Processing" on page 55. To specify local file information in the OutputDir attribute, use the full path of the file. Alternately, you can use a FILE URL, which uses the following format: [file://]*FullPathname* The delivered default values are: UNIX: /tmp/JTextConn/Default/out Windows:c:\temp\JTextConn\Default\Out
OutputExt	Specifies the extension of the file used for request processing. The delivered default value is out. **Note:** If OutputFileName contains no extension, but the OutputExt attribute does contain an extension, the output file is generated with both the file name and the extension. If neither contain an extension, the output file is generated without one.
OutputFileName	Specifies the name and path of the output file into which the connector writes the incoming business object during request processing. If the OutputDir attribute contains a valid output directory, the output file is generated into the specified directory. For more information, see "Specifying the Name of the Output File" on page 34. **Note:** If OutputFileName and OutputExt attributes do not contain an extension, the output file is generated without an extension. The delivered default value is Native.
PartialArchiveExt	Specifies the file extension used to archive the successfully processed business objects (when the event file contains multiple business objects, not all of which process successfully). For more information, see "Specifying Event Archiving" on page 42. The delivered default value is partial.

Table 7. Attributes in the MO_JTextConnector_Default Meta-object Definition (continued)

Attribute Name	Description
StagingDir	Specifies a directory in which the connector should write files to before moving them into the directory specified by the OutputDir attribute. This is designed to handle environments where other software processes might be monitoring and manipulating the directory into which the JText connector outputs files (such as an FTP process that detects files created by the connector and moves them to another location). In situations such as this, there is a risk that the external process could move the file before it has been completely written. You can specify a staging directory in theStagingDir attribute, therefore, so that the connector writes the file completely to the staging directory and then moves it to the output directory when it is finished, eliminating the risk of the external process picking up an incomplete file. It is recommended that the staging directory and output directory be on the same file system or drive to accommodate different operating systems' approaches to file moving operations. There is no delivered default value.
SuccessArchiveExt	Specifies the file extension used to archive all successfully processed business objects. For more information, see "Specifying Event Archiving" on page 42. The delivered default value is success.
UnsubscribedArchiveExt	Specifies the file extension used to archive all unsubscribed business objects. For more information, see "Specifying Event Archiving" on page 42. The delivered default value is unsub.

Specifying the Name of the Output File

There are three ways to specify the name of the output file:

- Use the OutputFileName attribute

  Use this attribute when you want the connector to write each business object of the same type to separate files with unique sequence numbers, or to append multiple business objects to a single file with a specified name.
- Use a dynamic child meta-object

  Use a dynamic child meta-object when you want to dynamically generate an output filename for each type of business object or to return the name of a connector-generated output file. See "Using a Dynamic Child Meta-Object" on page 4 for details.
- Use a wrapper business object (deprecating)

  Provided only for backward compatibility, use a wrapper business object for the same purposes as a dynamic child meta-object. See "Using a Wrapper Business Object (Deprecated)" on page 6 for details.

There are several ways to use the OutputFileName attribute to specify the name of the output file:

- If OutputFileName is set to the string Native and the FileSeqEnabled attribute is set to true, the connector sends the business object string to a unique file whose name is derived from the name of the incoming business object, whose extension is derived from the OutputExt attribute, and whose path is derived from the OutputDir attribute. In this case, the connector's default behavior is to write each business object of the same type to separate files with unique sequence numbers. To cause the connector to overwrite the output file each time it receives business objects of the same type, set the FileSeqEnabled attribute to false.

- If OutputFileName is set to a string other than Native and the FileSeqEnabled attribute is set to true, the connector handles the value of the output file in one of the following ways:
  - If OutputFileName contains an absolute path (including the filename and the extension of the output file, for example, OutputFileName= C:\temp\Out\test.out), the connector uses only this attribute to generate the output file. In this case, the connector's default behavior is to write each business object of the same type to separate files with the specified name and with unique sequence numbers.
  - If OutputFileName contains the full path and the filename, but not the extension, and the OutputExt attribute contains a value, (for example, OutputFileName= C:\temp\Out\test and OutputExt=out), the connector uses the value of both attributes to generate the output file. In this case, the connector generates a file named C:\temp\Out\test_1.out.
  - If OutputFileName contains the full path and the filename, but not the extension, and the OutputExt attribute does not contain a value, the connector generates the output file without any extension. In this case, the connector generates a file named C:\temp\Out\test_1.
  - If OutputFileName contains only the filename, and not the path or extension, and the OutputDir attribute contains a value, the connector generates the output file in the directory specified by OutputDir. If OutputExt contains a value, the connector also uses that value. If not, it creates the filename without any extension.

Note: If the connector is processing more than one type of business object and OutputFileName is set to a string other than Native, each business object must have its own top-level meta-object, which specifies a unique output filename. For example, the meta-object used by the Customer business object might be MO_JTextConnector_Customer, and the meta-object used by Item might be MO_JTextConnector_Item. Set the value of the OutputFileName attribute in each of these meta-objects to a unique value.

- To cause the connector to append multiple business objects to a single file with the specified name, specify a value for OutputFileName and set the FileSeqEnabled attribute to false.
- To cause the connector to overwrite the output file each time it receives business objects of the same type, use a dynamic child meta-object. Specify its absolute path and filename in the InFileName attribute and set the FileWriteMode attribute to "o". For more information on using a dynamic child meta-object, see "Using a Dynamic Child Meta-Object" on page 4.

Native is a reserved word.

For more information, see "Specifying Request Processing" on page 44.

Example of a Meta-object

This section provides an example of a default meta-object. The example includes only those properties that indicate each attribute's name, type, and delivered default value.

The directory paths in the example are specific to the Windows platform. For UNIX platforms, use the following directory paths:

```
ArchiveDir
   DefaultValue = /tmp/JTextConn/Default/Archive
EventDir
```

```
   DefaultValue = /tmp/JTextConn/Default/Event
OutputDir
   DefaultValue = /tmp/JTextConn/Default/Out
```

All the directories in this example meta-object show absolute paths, such as `c:\temp\JTextConn\Default\Out`. Note that the directories must be writable.

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

   [Attribute]
   Name = DummyKey
   Type = String
   Cardinality = 1
   IsKey = true
   IsForeignKey = false
   IsRequired = true
   DefaultValue = dummykey
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = EventDataHandler
   Type = MO_DataHandler_DefaultNameValueConfig
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 0
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = OutputDataHandler
   Type = MO_DataHandler_DefaultNameValueConfig
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 0
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = DataEncoding
   Type = String
   Cardinality = 1
   MaxLength = 1
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = OriginalArchiveExt
   Type = String
   Cardinality = 1
   MaxLength = 1
   IsKey = false
   IsForeignKey = false
   IsRequired = true
   DefaultValue = org
   IsRequiredServerBound = false
```

```
[End]

[Attribute]
Name = UnsubscribedArchiveExt
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = unsub
IsRequiredServerBound = false
[End]

[Attribute]
Name = PartialArchiveExt
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = part
IsRequiredServerBound = false
[End]

[Attribute]
Name = OutputDir
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = C:\temp\JTextConn\Default\Out
IsRequiredServerBound = false
[End]

[Attribute]
Name = OutputExt
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = out
IsRequiredServerBound = false
[End]

[Attribute]
Name = OutputFileName
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = Native
IsRequiredServerBound = false
[End]

[Attribute]
Name = FileSeqEnabled
Type = String
Cardinality = 1
MaxLength = 1
```

```
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = true
IsRequiredServerBound = false
[End]

[Attribute]
Name = ArchiveDir
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = C:\temp\JTextConn\Default\archive
IsRequiredServerBound = false
[End]

[Attribute]
Name = SuccessArchiveExt
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = success
IsRequiredServerBound = false
[End]

[Attribute]
Name = FailArchiveExt
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = fail
IsRequiredServerBound = false
[End]

[Attribute]
Name = EventDir
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = C:\temp\JTextConn\Default\event
IsRequiredServerBound = false
[End]

[Attribute]
Name = EventExt
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
DefaultValue = in
IsRequiredServerBound = false
[End]
```

```
[Attribute]
Name = EndBODelimiter
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPUserId
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPPassword
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPLocalEventDir
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPRenameExt
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPEventFileMask
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

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

```
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPArchiveDir
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = StagingDir
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPFileListingFormat
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]


[Attribute]
Name = IncludeEndBODelimiter
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPPollTerminateIfServerDown
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPRequestTerminateIfServerDown
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = FTPKeepConnectionOpen
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]

[End]
...
```

Common Configuration Tasks

This section describes the most common configuration tasks.

Specifying Event Notification

This section describes the following:

Specifying Event Directories and Extensions

If you send more than one type of business object to the connector for processing, and each business object type has its own top-level meta-object, the combination of values you specify for the `EventDir` and `EventExt` attributes must be unique for each directory/extension pair for each business object.

In other words, if you specify the same event directory for two business object types, you must specify different event extensions for these business objects. If you

specify the same extension for two business object types, you must specify different event directories for these business objects.

For example, assume you have created the MO_JTextConnector_Customer and MO_JTextConnector_Item meta-objects to provide configuration values for the Customer and Item business objects, respectively. If you instruct the connector to locate the input files for both business objects in the same directory (by specifying the same path in the EventDir attribute), you must uniquely identify the input files by specifying different values for the EventExt attribute.

Therefore, if the EventDir attribute evaluates to C:\temp\event for both Customers and Items, the value of the EventExt attributes for these two business objects must be different (such as in for Customer input files and inp for Items).

Configuring Polling Behavior

To configure polling behavior, perform the following steps:

1. Configure the following attributes of the MO_JTextConnector_Default meta-object:
   - EventDir—Specify the absolute path of an existing directory whose files trigger event notification.
   - EventExt—The connector looks for files with the delivered-default extension of in. If you use this attribute to specify a different extension, the connector looks for the specified extension. If you leave this attribute empty, the connector polls for files with no extension.
   - EventDataHandler—Specify the data handler to use for data conversion during event notification.
2. Use the appropriate configuration utility (Connector Configurator if WebSphere MQ Integrator is the integration broker, or Connector Designer, which you access from CSM, if InterChange Server is the integration broker) to configure the following connector properties:
   - PollFrequency—Specify the interval frequency.
   - PollQuantity—Specify the number of events for each polling interval.
   - PollEndTime—Specify the time to complete the polling of events.
   - PollStartTime—Specify the time to begin the polling of events.
3. Establish read permissions on the event directory.

Specifying Event Archiving

Depending on whether all or some of the business objects in the event file process successfully, the JText connector uses different extensions when it creates the archive file for successfully processed business objects. The connector also writes business objects that fail processing and those that are unsubscribed to differently named archive files.

This section describes the following:

Archive Filenames

If you retain the delivered default values for the archive extension attributes, the connector creates archive files named as shown below:

- Event file has a single business object

After the JText connector processes an event file that contains a single business object, it creates one of following files in the archive directory:

- *filename_timestamp*.success, to archive a successfully processed business object
- *filename_timestamp*.fail, to archive a business object that was not successfully processed
- filename_timestamp.unsub, to archive a business object to which it does not subscribe

If the business object fails processing or is unsubscribed, the connector also creates the *filename_timestamp*.orig file, which preserves the event file as the connector originally received it.

- Event file has multiple business objects, all of which process successfully

  After the JText connector successfully processes an event file with multiple business objects, it creates *filename_timestamp*.success in the archive directory.

- Event file has multiple business objects, some of which are unsubscribed or fail processing

  After the JText connector processes an event file that contains multiple business objects, it may create all of the following files in the archive directory:

  - filename_timestamp.partial, to archive all business objects whose processing was successful
  - filename_timestamp.fail, to archive all business objects whose processing was unsuccessful
  - filename_timestamp.unsub, to archive all business objects to which the connector does not subscribe
  - *filename_timestamp*.orig, to preserve the event file as the connector originally received it

  For example, assume that the LegacyApp.in file contains four business objects:

  - Contract, which is successfully processed
  - Customer, which fails formatting
  - Order, which is successfully processed
  - Item, to which the connector does not subscribe

  In such a case, the connector creates the following files in the archive directory:

  - LegacyApp_*timestamp*.partial, which contains Contract and Order
  - LegacyApp_*timestamp*.fail, which contains Customer
  - LegacyApp_*timestamp*.unsub, which contains Item
  - LegacyApp_*timestamp*.orig, which contains Contract, Customer, Order, and Item

Configuring Archiving

To configure the connector for archiving, follow these steps:

1. Configure the following attributes of the MO_JTextConnector_Default meta-object:
   - ArchiveDir or FTPArchiveDir—Specify the absolute path of an existing local or FTP server directory into which the connector is to place events (with file extensions that indicate processing status) after they are processed.
   - SuccessArchiveExt—Specify the extension for the file that contains the successfully processed business objects (when all business objects process successfully).

- PartialArchiveExt—Specify the extension for the file that contains all the successfully processed business objects (when some of the business objects in the event file do not process successfully).
- UnsubscribedArchiveExt—Specify the extension for the file that contains the business objects to which the connector does not subscribe.
- OriginalArchiveExt—Specify the extension for the file that preserves all the business objects that were contained in the event file.
- FailArchiveExt—Specify the extension for the file that contains the business objects that failed processing.

2. Use Connector Configurator (if the integration broker is WebSphere MQ Integrator) or Connector Designer (if the integration broker is InterChange Server) to configure the ArchivingEnabled connector property.
3. Establish write permissions on the archive directory.

Specifying Request Processing

You can cause the JText connector to write business objects to files whose names are specified dynamically (in each business object instance) or statically (through meta-objects). You can also cause the connector to return each filename that it generates statically; this feature is useful to obtain filenames generated with a unique sequence number. This section contains the following subsections:

Dynamic File Naming

To cause the connector to dynamically generate an output filename for each type of business object, create a dynamic child meta-object. Use the child meta-object:

- either to specify the name of the output file or to receive the name of the generated filename
- to specify whether to append to or overwrite the output file

Note: Alternatively, you can create a wrapper business object that contains the business object you want written to the file and that provides the same information as the dynamic child meta-object. If you use a wrapper business object, you must add it to the list of business objects that the connector supports.

Important: In addition to creating the dynamic child meta-object (or the wrapper business object) to enable the connector to generate or return the output filename, if you are using ICS as the integration broker, you must also modify your maps or collaboration logic to insert into the dynamic child meta-object's InFileName attribute (or the wrapper business object's FileName attribute) a path and filename for each business object, and, if required, unique sequence numbers.

For more information, see "Using a Dynamic Child Meta-Object" on page 4 or "Using a Wrapper Business Object (Deprecated)" on page 6.

For information about how the connector processes the meta-object or wrapper business object, see "Business Object Request Processing" on page 11.

For information on configuring the connector to use a dynamically generated output filename, see "Configuring the Output File" on page 46.

Static File Naming

When you use meta-objects to specify the name of output files, you must restart the connector for any changes to take effect. You can specify whether the connector appends all business objects of a given type to a single file or creates a separate file for each business object.

When it uses the delivered default configuration, the connector creates an output file for each business object it processes. It names the output file for the incoming business object and adds a sequence number to make the name unique; it gives it the extension of `.out`. For example, if it receives the Customer and Item business objects, the connector writes their data to the `Customer_1.out` and `Item_1.out` output files. For information on obtaining the names of generated output files, see "Returning a File's Name" on page 45. For information on the file that stores the sequence numbers, see "OutputLog" on page 22.

To use the meta-object to configure the name of output files, do the following:

1. Configure the following attributes of the `MO_JTextConnector_Default` meta-object:
   - `OutputDir`—Specify the absolute path of an existing directory to which the connector is to write files when it processes requests. The connector uses the value of this attribute only when the business object is not contained by a wrapper business object, or its wrapper business object does not contain a value for its `FileName` attribute. For more information, see "Configuring the Output File" on page 46.
   - `FileExt`—Use this attribute to specify your own extension if you want to change the delivered default configuration, which causes the connector to create files with the out extension.
   - `FileSeqEnabled`—Keep set to `true` to cause the connector to output one business object per file, each with a unique sequence number. Set to `false` to cause the connector to output all business objects of a given type to a single file, either by overwriting or appending. For information on configuring overwrite or append behavior, see Table 8 on page 47.
   - `OutputFileName`—To the cause the connector to append business objects to a single output file rather than overwrite the data in the file or generate unique files for each business object, specify the output file's full path and filename.

     To cause the connector to overwrite the output file each time it receives the same type of business object, do not specify a value for `OutputFileName`.

     In each of these cases, set `FileSeqEnabled` to `false`.

     For information on configuring overwrite or append behavior, see Table 8 on page 47.
2. Establish write permissions on the output directory.

Note: You must create meta-objects for specific business objects if the connector is to use different data formats or file naming conventions for different business objects.

Returning a File's Name

To cause the connector to return the names of the files it generates, do the following:

- Use meta-objects to specify path and filenames and to cause the connector to generate a unique sequence number for each output file. For more information, see "Static File Naming" on page 45.
- Use a dynamic child meta-object (or a wrapper business object) to cause the connector to return the name of each file it generates. Follow the steps in "Using a Dynamic Child Meta-Object" on page 4, but do not specify a value for its `InFileName` attribute. When the connector receives a business object whose dynamic child meta-object specifies `OutFileName=CxIgnore`, it creates a filename based on the configuration of its top-level meta-object, and returns the full path and filename as a value in the `InFileName` attribute.

Note: The connector populates the `InFileName` attribute only with a local path, even when processing files over an FTP server.

Important: If you use a wrapper business object, you must add it to the list of business objects that the connector supports.

Important: In addition to creating the dynamic child meta-object (or the wrapper business object) to enable the connector to generate or return the output filename, if you are using ICS as the integration broker, you must also modify your maps or collaboration logic to insert into the dynamic child meta-object's `InFileName` attribute (or the wrapper business object's `FileName` attribute) a path and filename for each business object, and, if required, unique sequence numbers.

Differences Between Local and Remote Processing

The connector processes files remotely in much the same way that it processes them locally. There are, however, a few differences:

- When processing events and generating filenames dynamically, the connector populates the `InFileName` attribute of the dynamic child meta-object only with a local path name and not with a remote path.
- When processing requests, if the connector is not configured for dynamic file naming and `FileSeqEnabled` evaluates to `false` and the output file already exists:
  - If processing locally, the connector overwrites the existing file.
  - If processing remotely, the connector throws an exception.
- In addition to configuring the standard archive extension attributes for local event processing, when using the connector to process files remotely over an FTP server, you can also configure the `FTPArchiveExt` attribute. This attribute enables you to rename the remotely archived file independently of the success of the processing.

Configuring the Output File

Table 8 on page 47 illustrates the possible configuration options for the output file:

Table 8. Specifying Output Files

Desired Output Condition	Attributes/Property Requiring Configuration	Attribute/Property Value
Each business object of a given type is appended to a file whose absolute path and filename is derived at runtime from an attribute in the business object.	Use a dynamic child meta-object (alternatively, use a wrapper business object)	
	`AppSpecificInfo` (at business-object level)	`cw_mo_JTextConfig = DynChildMOName` Alternatively (in wrapper):`Type=JTextWrapper`
	For dynamic child meta-object: `OutFileNameAlternatively` (in wrapper): `FileName`	*user-specified pathname and filename*
	`FileWriteMode`	`a` or `append`
Each business object of a given type overwrites the output file whose absolute path and filename is derived at runtime from an attribute in the business object.	Use a dynamic child meta-object (alternatively, use a wrapper business object)	
	`AppSpecificInfo` (at business-object level)	`cw_mo_JTextConfig = DynChildMOName`Alternatively (in wrapper):`Type=JTextWrapper`
	For dynamic child meta-object:`OutFileNameAlternatively` (in wrapper): `FileName`	user-specified pathname and filename
	`FileWriteMode`	`o` or `overwrite`
Each business object of a given type is written to its own unique file whose name is derived from the business object's name and a generated unique sequence number.	`OutputDir`	*user-specified pathname*
	`FileSeqEnabled`	`true`
	`OutputFileName`	`Native`
The connector returns the name of each file it generates. Each business object of a given type is written to its own unique file whose name is derived from the business object's name and a generated unique sequence number.	Use a dynamic child meta-object (alternatively, use a wrapper business object)	
	`AppSpecificInfo` (at business-object level)	`cw_mo_JTextConfig = DynChildMOName` Alternatively (in wrapper):`Type=JTextWrapper`
	`InFileName` (in dynamic child meta-object)	`CxIgnore`
	`FileWriteMode` (in dynamic child meta-object)	N/A
	Use meta-object configuration: `MO_JTextConnector_businessobjectname`:	
	`OutputDir`	*user-specified pathname*
	`FileSeqEnabled`	`true`
	`OutputFileName`	`Native`
All business objects of a given type are appended to a single file whose name is user-specified.	`FileSeqEnabled`	`false`
	`OutputFileName`	*user-specified pathname and filename*
Each business object of a given type is written to its own unique file whose name is user-specified plus a unique sequence number.	`FileSeqEnabled`	`true`

Table 8. Specifying Output Files (continued)

Desired Output Condition	Attributes/Property Requiring Configuration	Attribute/Property Value
If the connector is processing more than one type of business object and `OutputFileName` is set to a string other than `Native`, each business must have its own top-level meta-object. For more information, see "Specifying the Name of the Output File" on page 34.	`OutputFileName`	*user-specified pathname and filename*
Each business object of a given type overwrites the output file, whose name is derived from the business object's name.	`OutputDir`	*user-specified pathname*
	`FileSeqEnabled`	`false`
	`OutputFileName`	`Native`
The connector returns the name of each file it generates. Each business object of a given type is written to its own unique file whose name is user-specified plus a unique sequence number.	Use a dynamic child meta-object (alternatively, use a wrapper business object)	
	`AppSpecificInfo` (at business-object level)	`cw_mo_JTextConfig = DynChildMOName` Alternatively (in wrapper):`Type=JTextWrapper`
	`InFileName` (in dynamic child meta-object)	`CxIgnore`
	`FileWriteMode` (in dynamic child meta-object)	N/A
	Use meta-object configuration: `MO_JTextConnector_businessobjectname`:	
	`FileSeqEnabled`	`true`
	`OutputFileName`	*user-specified pathname and filename*

Specifying Multiple Event Files or Multiple Event Directories

You can configure the connector to pick up only files with a specified extension. You can also configure the connector to pick up files from multiple directories.

Important: The use of an asterisk (*) for the `EventExt` attribute to specify that the connector poll for all files in a single event directory regardless of their extension is no longer supported.

To specify a separate event directory for each business object type, perform the following steps:

1. Create a separate meta-object for each supported business object; for example, create `MO_JTextConnector_Customer` and `MO_JTextConnector_Item`. For more information, see "Creating a JText Meta-Object for a Specific Business Object" on page 57.
2. Specify the appropriate directory in each meta-object's `EventDir` attribute.

Note: The JText connector processes event files in the order of their time stamps, from the earliest to the most recent, regardless of their location. In other words, the JText connector processes files located in separate directories in the chronological order of their time stamps.

Polling for Specific Business Objects and the EndBODelimiter

Configuration of the JText connector differs depending on whether all your event files are in a single directory, they all have the same extension, they contain a single business object or multiple business objects, they contain business objects of one type or multiple types, and they represent each business object on a single line or on multiple lines.

This section explains the following:

- "Specifying EndBODelimiters"
- "Using Non-printable Characters as EndBODelimiters" on page 50

Specifying EndBODelimiters

If no value is specified for the `EndBODelimiter` meta-object attribute, the connector:

- expects the event file to delimit business object strings with `<EndBO:BOName>`
- specifies `<EndBO:`*`BOName`*`>` as the delimiter when it writes business object strings to output files.

If an event file contains only one business object, you can specify `EOF` (end of file) for this attribute.

If you set the value of the `EndBODelimiter` attribute to a non-empty string, the string is assumed to be the business object delimiter for every file. If the value is not set or is cleared, the connector assumes the delimiter is `<EndBO:`*`BOName`*`>`.

Table 9 illustrates delimiter options.

Table 9. Using the EndBODelimiter Attribute

Conditions	Delimiter	Notes
File contains one or more business object strings with one or more types of business object or File contains multiple business object strings of the same type of business object; each string runs over several lines.	`<EndBO:`*`BOName`*`>`or `EOL` or user-specified value	• Specify as many semicolon-separated `EOL`s as there are new lines between business object strings. • Specify a custom delimiter in conjunction with `EOL`s. A custom delimiter must always be the first element when used with `EOL`. The following example is valid: `customEndBO;EOL;EOL`. The following example is **not** valid: `EOL;customEndBO;EOL`.
Each file contains only one business object string	`EOL` For user-specified value	• Specify as many semicolon-separated `EOL`s as there are new lines between business object strings. • Specify a user-specified delimiter in conjunction with `EOL`s and `EOF` if required by the input strings. A custom delimiter must always be the first element when used with `EOL`. The following example is valid: `customEndBO;EOL;EOL`. The following example is **not** valid: `EOL;customEndBO;EOL`
File contains multiple business object strings, one per line	`EOL`	

Table 9. Using the EndBODelimiter Attribute (continued)

Conditions	Delimiter	Notes
File contains multiple business object strings of the same type of business object; each string runs over several lines without any separators between business-object strings	`None`	Can use the delivered default meta-object or a custom meta-object **Note:** This option is available only during service call requests and not for event notification. Do not use this delimiter in conjunction with any other delimiter.

Note: If the source file contains empty lines, the connector ignores them.

Using Non-printable Characters as EndBODelimiters

To poll for files in multiple directories, you must create a meta-object for each supported business object. The value you specify for each meta-object's `EndBODelimiter` attribute depends on whether your source file contains a single business object or multiple business objects.

- Files that contain a single business object

  You can specify `EOF` as the `EndBODelimiter` if the entire data file contains only one business object string.

- Files that contain multiple business objects

  If your input file contains multiple business objects that have only a new line as the business object delimiter, specify the string `EOL` in the `EndBODelimiter` attribute. In this case, the source file contains strings representing multiple business objects of the same type.

  Important: To poll from a file that contains multiple business object types, you must use the `MO_JTextConnector_Default` meta-object, and must ensure that its `EventExt` and `EventDir` attributes correctly point to the directory where this event file is located. To poll for business object types that are represented in separate event files or whose event files are located in different directories, you must create a separate top-level meta-object for each type. Use the `EventExt` and `EventDir` attributes to point to the appropriate directory.

  To use a custom data handler when polling files that contain multiple business objects of different types, see "Reading Multiple Business Objects of Different Types from the Same File" on page 58.

  If using a name/value format, you cannot specify the `EOL` business object delimiter if the event file splits business object data over multiple lines. For more information, see the *Data Handler Guide*.

  The following examples illustrate the delimiter to use for different event file formats:

- File contains four business object strings and uses the non-printable character `EOL` as the end of business object delimiter:

```
Sample_BO~Create~1~TableGenKey5~strange~TextConnector_924055528_0
Sample_BO~Create~2~TableGenKey5~strange~TextConnector_924055528_0
Sample_BO~Create~3~TableGenKey5~strange~TextConnector_924055528_0
Sample_BO~Create~4~TableGenKey5~strange~TextConnector_924055528_0
```

- File contains four business object strings and uses a user-specified value and the non-printable character `EOL` as the end of business object delimiter, that is `CustomEndBO;EOL`:

```
Sample_BO~Create~1~TableGenKey5~strange~TextConnector_924055528_0CustomEndBO
Sample_BO~Create~2~TableGenKey5~strange~TextConnector_924055528_0CustomEndBO
Sample_BO~Create~3~TableGenKey5~strange~TextConnector_924055528_0CustomEndBO
Sample_BO~Create~4~TableGenKey5~strange~TextConnector_924055528_0CustomEndBO
```

- File that contains four business object strings and uses the non-printable character `EOL;EOL` as the end of business object delimiter:

```
Sample_BO~Create~1~TableGenKey5~strange~TextConnector_924055528_0

Sample_BO~Create~2~TableGenKey5~strange~TextConnector_924055528_0

Sample_BO~Create~3~TableGenKey5~strange~TextConnector_924055528_0

Sample_BO~Create~4~TableGenKey5~strange~TextConnector_924055528_0
```

- File that contains four business object strings and uses `None` as the end of business object delimiter:

```
Sample_BO~Create~1~TableGenKey5~strange~TextConnector_924055528_0Sample_BO
~Create~2~TableGenKey5~strange~TextConnector_924055528_0Sample_BO~Create~3
~TableGenKey5~strange~TextConnector_924055528_0Sample_BO~Create~4
~TableGenKey5~strange~TextConnector_924055528_0
```

Note: The connector is case-sensitive to the string that you specify, except for the `EOL` and `EOF` delimiters.

For more information on creating your own meta-objects, see "Creating a JText Meta-Object for a Specific Business Object" on page 57.

Specifying a Remote FTP File System

This section describes how to configure the JText Connector to use a remote FTP file system for event and request processing.

Important: To enable the connector to use a remote FTP file system, you must specify an FTP URL in the `EventDir` attribute (for event processing) or `OutputDir` attribute (for request processing). You must also resolve all firewall issues before using the connector to perform FTP operations.

This section describes the following:

- "Remote Event Processing"
- "Remote Request Processing" on page 55
- "Notes on Configuring the Connector for FTP Transfer" on page 56

Remote Event Processing

To configure the connector to use a remote FTP file system for event processing, you must specify the FTP URL, FTP login information, a local directory into which the connector downloads the event files from the remote directory, archiving information, and information related to how the connector behaves when the FTP server is unavailable. This section describes all of these configurations as well as additional optional configurations.

- "Specifying the FTP URL and Login Information" on page 52
- "Specifying the Local Directory" on page 52
- "Specifying Remote Archiving" on page 52
- "Specifying Remote Polling" on page 53
- "How the Connector Processes Events from a Remote Site" on page 53
- "Identifying Files on a Mainframe: Optional Configuration" on page 54
- "Summary of Configuration Operations for Event Processing" on page 54

Specifying the FTP URL and Login Information: The connector polls for events from the directory specified in the EventDir meta-object attribute. To configure the connector to use a remote FTP file system for event processing, specify the FTP URL as the value of this attribute. The FTP URL must conform to IETF standards.

In addition to specifying the FTP server in the URL, you can optionally specify the following information in the EventDir meta-object attribute:

- Name of a user with privileges to connect to the FTP server and perform FTP operations—If you do not specify the username in EventDir, specify it in the FTPUserId meta-object attribute.
- Password of a user with privileges to connect to the FTP server and perform FTP operations—If you do not specify the password in EventDir, specify it in the FTPPassword meta-object attribute.
- Port number—If the port is not specified in EventDir, the connector uses the default port.
- Remote event directory—If you do not specify the remote event directory in EventDir, the connector polls the event files from the directory to which the connection is established to the FTP server.

Important: You can specify the FTP values either in a static top-level meta-object or in a dynamic child meta-object. If the username and password are not specified in any meta-object attribute, the connector terminates when attempting to connect to the FTP server. For more information, see "Using a Dynamic Child Meta-Object" on page 4.

The examples below illustrate three different formats for EventDir attribute values:

URL only with required values:

```
ftp://ftp.crossworlds.com
```

URL with optional username and port number values:

```
ftp://crossworlds:admin@ftp.crossworlds.com:1433
```

URL with optional username, port number, and remote event directory values:

```
ftp://crossworlds:admin@ftp.crossworlds.com:1433/temp/JTextConn/Default/Event
```

Specifying the Local Directory: In addition to specifying the FTP URL and related login information, you must specify the location of the local directory into which the connector downloads the event files from the remote directory. To specify the local directory, use the FTPLocalEventDir meta-object attribute.

Important: If the connector finds a proper FTP URL in EventDir, but does not find the FTPLocalEventDir meta-object attribute or finds an invalid or a blank value for this attribute, the connector does not start. The connector does not evaluate the FTPLocalEventDir attribute when configured to run locally.

Specifying Remote Archiving: You have several options in specifying how the connector handles remote archiving. To specify a remote archive directory, use the FTPArchiveDir meta-object attribute. This attribute specifies the absolute path of the archive directory on the FTP server. The directory must already exist. There are several options for using this attribute to specify archiving:

- Specifying a value for the FTPArchiveDir attribute but no value for the FTPRenameExt attribute causes the connector to append a timestamp to the event file and move it to the FTP server archive directory specified in the FTPArchiveDir attribute.
- Specifying a value both for the FTPArchiveDir attribute and the FTPRenameExt attribute causes the connector to rename the processed event file with a timestamp and the value specified in FTPRenameExt, and move it to the the FTP server archive directory specified in the FTPArchiveDir attribute.
- Specifying no value either for the FTPArchiveDir or the FTPRenameExt attributes causes the connector to delete the processed event file without archiving it.
- Specifying no value for the FTPArchiveDir attribute but specifying a value for the FTPRenameExt attribute causes the connector to rename the processed event file with a timestamp and the value specified in FTPRenameExt, and move it to the directory specified in the EventDir attribute.

Specifying Remote Polling: You can use the "FTPPollFrequency" on page 22 configuration property to set how frequently the connector polls an FTP server measured in the number of standard poll cycles. This setting is useful if the connector is still reading files from the local event directory when it starts the next polling cycle.

For example, if PollFrequency is set to 10000, and FTPPollFrequency is set to 6, the connector polls the local event directory every 10 seconds and polls the remote directory every 60 seconds. The connector performs FTP polling only if you specify a value for this property. If FTPPollFrequency evaluates to 0 or blank, the connector does not perform FTP polling.

For more information, see "Tuning the Performance of the JText Connector" on page 59.

How the Connector Processes Events from a Remote Site: When polling for events from a remote site, the connector performs the following steps:

1. Obtains the server name, port number, username, password, and remote event directory from meta-object attributes or default values.
2. Establishes a connection to the remote FTP site to get event files from the remote event directory.
3. Downloads the event files from the remote directory to the local directory specified in the FTPLocalEventDir meta-object attribute.

   Note: To enable the connector to process events using FTP, this attribute must have a value.
4. Polls the local directory.

Figure 7 illustrates local and remote event processing.

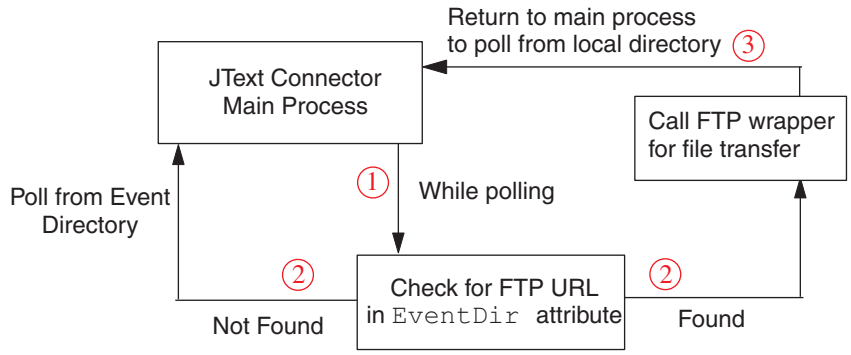


Figure 7. Local and Remote Event Notification Operation

Identifying Files on a Mainframe: Optional Configuration: Use the `FTPEventFileMask` attribute to identify file extensions on a mainframe that do not adhere to the same naming standards that apply to UNIX or Windows systems. If no value is provided for this attribute, the connector uses the value specified in the `EventExt` attribute.

When specifying a value for `FTPEventFileMask`, you can include wildcard characters. The following example illustrates one possible format for this attribute:

```
ACT.Z1UC.INPT*
```

If the connector finds more than one file at the remote site that meets the criteria specified for `FTPEventFileMask`, it does the following:

1. Downloads all specified remote event files to the directory specified in the `FTPLocalEventDir` attribute.
2. Renames the extension of the remote files with the value specified in the `FTPRenameExt` meta-object attribute. Renaming the files prevents the connector from polling the same file in the next poll cycle.
3. Disconnects from the FTP server.
4. Processes the files locally in the directory specified in the `FTPEventFileMask` meta-object attribute.

Summary of Configuration Operations for Event Processing: To configure the connector to use a remote FTP file system for event processing, specify the following configuration values:

- Specify the FTP URL in the `EventDir` meta-object attribute. Optionally, specify the name and password of a user with privileges to connect to the FTP server and perform FTP operations.
- If you do not specify the login name and password in the `EventDir` meta-object attribute, do so in the `FTPUserId` and `FTPPassword` meta-object attributes.
- If you do not specify the port in the `EventDir` meta-object attribute, the connector uses the default FTP port.
- Use the `FTPLocalEventDir` meta-object attribute to specify the local system directory into which the connector downloads event files from the FTP site.
- On a mainframe that does not adhere to the same naming standards that apply to UNIX or Windows systems, use the `FTPEventFileMask` meta-object attribute to identify files to be polled.
- To configure the connector to work with an MVS FTP server when the remote system is MVS, specify `MVS` in the `FTPOSPlatform` attribute.

Remote Request Processing

To configure the connector to use a remote FTP file system for request processing, you must specify the FTP URL, FTP login information, and a remote directory into which the connector uploads the request files from the local directory. This section describes all of these configurations as well as additional optional configurations.

Specifying the FTP URL and Login Information: The connector uploads service call request files into the directory specified in the `OutputDir` meta-object attribute. To configure the connector to use a remote FTP file system for request processing, specify the FTP URL as the value of this attribute. The FTP URL must conform to IETF standards.

In addition to the FTP URL, you can optionally specify the following information in the `OutputDir` meta-object attribute:

- Name of a user with privileges to connect to the FTP server and perform FTP operations—If you do not specify the username in `OutputDir`, specify it in the `FTPUserId` meta-object attribute.
- Password of a user with privileges to connect to the FTP server and perform FTP operations—If you do not specify the password in `OutputDir`, specify it in the `FTPPassword` meta-object attribute.
- Port number—If the port is not specified in `EventDir`, the connector uses the default port.
- Remote output directory—If you do not specify the remote output directory in `OutputDir`, the connector loads the request files into the default connection directory (the directory on the FTP server to which the connection is established).

Important: You can specify the FTP values either in a static top-level meta-object or in a dynamic child meta-object. If the username and password are not specified in any meta-object attribute, the connector terminates by throwing an exception. For more information, see "Using a Dynamic Child Meta-Object" on page 4.

The examples below illustrate three different formats for `OutputDir` attribute values:

URL only with required values:

```
ftp://ftp.crossworlds.com
```

URL with optional username and port number values:

```
ftp://crossworlds:admin@ftp.crossworlds.com:1433
```

URL with optional username, port number, and remote output directory values:

```
ftp://crossworlds:admin@ftp.crossworlds.com:1433/temp/JTextConn/Default/Out
```

How the Connector Processes Service Call Requests to a Remote Site: When the connector is configured for FTP processing and it receives a service call request, it performs the following steps:

1. Obtains the server name, port number, username, and password from meta-object attributes or default values.

2. Establishes a connection to the remote FTP site to place service call request files from the local directory.
3. Uploads the request files from the local directory to the remote directory.
4. Disconnects from the remote server.

Figure 8 illustrates local and remote request processing.

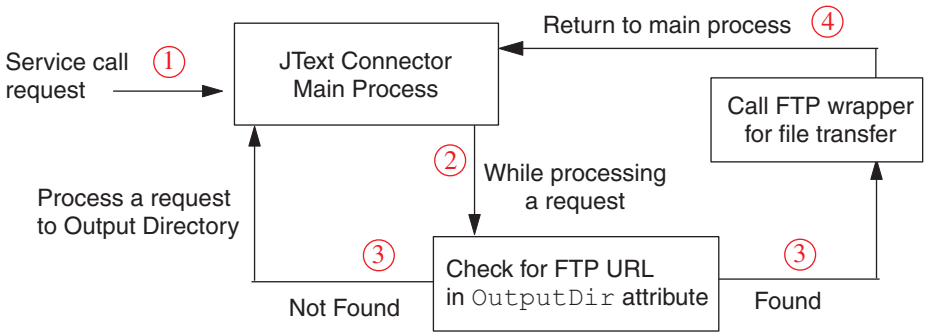


Figure 8. Local and Remote Request Operations

Summary of Configuration Operations for Request Processing: To configure the connector to use a remote FTP file system for request processing, specify the following configuration values:

- Specify the FTP URL in the `OutputDir` meta-object attribute. Optionally, specify the name and password of a user with privileges to connect to the FTP server and perform FTP operations.
- If you do not specify the login name and password in the `OutputDir` meta-object attribute, do so in the `FTPUserId` and `FTPPassword` meta-object attributes.
- If you do not specify the port in the `OutputDir` meta-object attribute, the connector uses the default port.
- To configure the connector to work with an MVS FTP server when the remote system is MVS, specify `MVS` in the `FTPOSPlatform` attribute.

Notes on Configuring the Connector for FTP Transfer

The following features apply to FTP transfer of data:

- The connector uses Binary mode of data transfer when doing FTP operations.
- The connector does not use FTP transfer of data if the value of the `EventDir` or `OutputDir` meta-object attribute does not begin with `ftp://` .
- During event processing, if the event business object contains a dynamic child meta-object with an `InFileName` attribute, the connector populates this attribute with the full path of the file specified in the `FTPLocalEventDir`, but not the path on the remote system.
- Values entered in the `EventExt` and `FTPRenameExt` meta-object attributes cannot be same; if they were the same, the connector would continuously pick up files that it had already picked up earlier.
- The connector does not support file sizes that are not supported by FTP.
- You must consider case sensitivity for file names, extensions, and other components in accordance with the platform of the FTP site.
- Transferring files from a remote FTP site might impact the connector's performance.

- When data is exchanged to or from the remote FTP site, there is a chance that data can be corrupted or lost due to loss of network connection or similar problems.
- The integration broker does not maintain any type of connection cache or pool. Connections are opened and closed for each polling cycle and request processing. Network latency and other configuration outside the control of the connector can impact its performance.
- The value specified for the `FTPLocalEventDir` meta-object attribute can not be specified as the value of the `EventDir` meta-object attribute of any meta-object that does not specify FTP values. This restriction prevents the connector from using values specified in different types of business objects in same directory that require totally different types of processing.
- If the remote event directory or output directory specified at the end of the FTP URL does not exist, the connector shuts down when it interacts with the FTP site. It does not shut down at the time of connector startup.
- When configuring the connector for processing files over an FTP server, you must configure the FTP server to use the UNIX settings required by the NetComponents API.

Specifying a Data Handler

To specify a data handler to be used by the JText Connector, perform the following steps:

1. Determine the type of text format used by the application with which the JText connector communicates. Note that only one data handler class can be registered for any given format type.
2. Configure the following child objects of the top-level JText meta-object:
   - `EventDataHandler`—To specify the data handler meta-object to be used for event processing (business object string to business object conversion).
   - `OutputDataHandler`—To specify the data handler meta-object to be used for the request processing (business object to business object string conversion).

Important: To prevent the connector from returning an error, include the top-level data-handler meta-object in the list of business objects supported by the connector.

Changing the Specified Data Handler

To change the data handler from the delivered default (either to a different delivered one or to a custom data handler), do the following:

- Verify that the connector supports the business object specified as the default value in the `EventDataHandler` and `OutputDataHandler` attributes.
- Verify that the class or jar file that contains the data handler is included in the class path when the connector is started. If you use a delivered data handler, or you add a custom data handler to the `CustDataHandler.jar` file (as recommended in the *Data Handler Guide*), the file is included in the delivered startup script (`start_JText.bat` or `connector_manager_JText.sh`).

For information on creating a data handler, see the *Data Handler Guide*.

Creating a JText Meta-Object for a Specific Business Object

When you create a JText meta-object for a specific business object, rename the meta-object to identify the particular business object. For example, to create meta-objects for the Customer and Item business objects, you might name the meta-objects `MO_JTextConnector_Customer` and `MO_JTextConnector_Item`.

Tip: Use default meta-objects when all business objects to be written to files have exactly the same configuration. In other words, all text files reside in the same event directory and are written to the same output directory, use the same data handler, and have the same file extension (or should be put into the same file). Create your own meta-objects if the connector must use different processing for different business objects on requests, or if specific processing instructions are required for polling. If you create separate meta-objects for specific business objects, the connector uses your meta-objects for both integration-broker requests and subscription delivery operations.

Any business object for which you do not create a meta-object is configured by the values in the default `MO_JTextConnector_Default` meta-object. For the business object definition for this default meta-object, see the `\repository\JText` directory.

Reading Multiple Business Objects of Different Types from the Same File

If a text file contains multiple business objects of different types, you must use the `MO_JTextConnector_Default` meta-object, and must ensure that its `EventExt` and `EventDir` attributes correctly point to the directory where this event file is located. Each business object in the file must be separated by the same delimiter.

The delivered data handlers can determine the name of each business object from the input string. In other words, when using the default top-level JText meta-object and the delivered data handlers, you need not use the <EndBO:*BOName*> delimiter to identity each type of business object in a file that contains multiple types.

If you develop a custom data handler to convert business object strings to business objects, ensure that it can interpret the business object's type from the input string.

Specifying Values for ObjectEventID Attributes

You do not have to add `ObjectEventId` attributes to business object strings. For event notification business objects, the connector framework populates these business object attributes if the IDs are not populated by the connector.

For service call request business objects, `ObjectEventId` attributes are either ignored or included in the string written to a file. Whether `ObjectEventId` attributes are included in the output file depends on the data handler that is used.

Setting Up a Second Instance of a JText Connector

To set up a second instance of the JText connector, follow these steps:

1. Make a copy of the JText connector directory and its repository directory and rename them. For example, assume you name the second connector definition `JText2`. After you create the second directories, your directory structure looks like the following:

   \connectors\JText \connectors\JText2 \repository\JText \repository\JText2

2. Copy all the meta-objects for the JText connector (there should be at least two of them) and modify the name of the business objects. For example, for the JText2 connector, change the names from `MO_JTextConnector_`*`BOName`* to `MO_JText2Connector_`*`BOName`*.

   There are two ways you can copy the meta-objects:

   - Create a text file that contains the `MO_JText2Connector_`*`BOName`* meta-object and its children. Use a text editor's search and replace option to replace `MO_JTextConnector_` with `MO_JText2Connector_`.

- Use Business Object Designer to copy the meta-objects one at a time.

  Important: Before you can manipulate a business object definition in Business Object Designer, you must copy the text from the top of the `\repository\ReposVersion.txt` file to the top of every definition file.

3. In Connector Configurator (if WebSphere MQ Integrator is the integration broker) or Connector Designer (if InterChange Server is the integration broker), copy the connector's definition and rename it to `JText2Connector`. Change the supported meta-objects and business objects.
4. Copy any new definition files into the repository. To use Business Object Designer to copy business object definitions into the repository, select the Save To Server submenu from the File menu. Alternatively, on ICS, perform the following steps to copy business object definitions into the repository from the operating system:
   a. Copy the text from the top of the `\repository\ReposVersion.txt` file to the top of every definition file.
   b. Use the following `repos_copy` command to copy in the new meta-objects:

      ```
      repos_copy -sServerName -iFileName
      ```
5. Refresh the integration broker's administration utility to verify the new business objects.
6. For UNIX, make a copy of the existing connector manager script for the JText connector and change the parameters to refer to `JText2`. For Windows, make a copy of the existing shortcut for the JText connector and change the parameters to refer to `JText2`, and modify it to point to the `JText2` directory rather than the `JText` directory.
7. Add a new MQ queue for the new connector. See the *WebSphere Business Integration Adapters Implementation Guide* or System Administration Guide for information on doing this.
8. Restart the integration broker.
9. In UNIX, run the connector manager script. In Windows, click on the new shortcut.

Tuning the Performance of the JText Connector

To tune the polling performance of the JText connector, set the following connector configuration properties as described below.

- `PollQuantity` – This property sets the maximum number of business objects that the connector can deliver to the integration broker in a single call to poll for events. If you set `PollQuantity` to a high value, the connector tries to submit more business objects in one poll. This can improve performance and helps to clear up internal queues and memory usage.

  Enabling the connector to post large quantities of business objects to the integration broker, however, can affect other business-integration components. For example, if the message queuing system has been set up with default values, the queues can fill up quickly if the JText connector sends many large business objects through the system. Therefore, when tuning performance, keep in mind that there is an optimal performance setting for `PollQuantity`.
- `PollFrequency` – This connector configuration property specifies the amount of time between polling actions. Setting this property to a longer time slows down the connector during event processing. Setting it to a shorter time ensures that events are picked up, converted to business objects, and delivered quickly.

In other words, the connector picks up new files during each poll call. If the connector does not poll often, it takes longer for it to deliver the files that accrue in the event directory. If the connector polls frequently, it picks up the files more often and delivers them more frequently.

The more frequently the connector polls for events, however, the less time it has for processing requests. If you use the connector primarily for request processing, set `PollFrequency` to a lower value than if you use the connector primarily for event processing.

As with the `PollQuantity` configuration property discussed above, setting `PollFrequency` to an extreme value, such as a very long or short time, can affect the performance of other business-integration components.

- `FTPPollFrequency` – This connector configuration property specifies how frequently the connector polls an FTP server measured in the number of standard poll cycles. For example, if `PollFrequency` is set to `10000`, and `FTPPollFrequency` is set to `6`, the connector polls the local event directory every 10 seconds and polls the remote directory every 60 seconds. The connector performs FTP polling only if you specify a value for this property. If `FTPPollFrequency` evaluates to `0` or blank, the connector does not perform FTP polling.

In summary, the best approach to improving performance in polling is to set PollQuantity, `PollFrequency`, and `FTPPollFrequency` so that they complement each other.

Specifying Multithreaded Processing

It is not possible to configure the multithreaded behavior of the JText connector. JText uses multithreading for two types of processing:

- Reading multiple files
- Formatting a file content into multiple business objects

You cannot configure how many files JText reads at once and how the connector formats file content into business objects.

Generating Sample Files for Testing

You might want to generate a file that looks like the input file that the JText connector expects. This file can assist you in setting up the output formats in the source application. A sample file can also be used for testing.

On ICS, the easiest way to generate a file similar to the input file is as follows:

1. Create a pass-through collaboration that takes as input and sends to the destination the business object that is to be written out to a file.
2. Bind the source port to a connector that supports that business object and can be emulated by Test Connector.
3. Bind the destination port to the JText connector.
4. Input sample values for the business object into Test Connector, and send that business object to the JText connector. The JText connector writes the values to the output file in the configured format.

This process enables you to see multiple business objects written to a single file, which you can use as input during testing.

Generating Sample Business Objects for Testing

You might want to generate business objects that look like ones the JText connector expects. You can populate the business objects with values to use during testing.

To cause the connector to automatically generate business object templates, use the `GenerateTemplate` configuration property. You can generate a definition for each business object that the connector supports.

The connector uses the value of the `GenerateTemplate` property to create an instance of a serialized business object when the connector starts up. A **serialized** business object is the string representation of the business object that the data handler creates. Use Connector Configurator (if WebSphere MQ Integrator is the integration broker) or Connector Designer (if InterChange Server is the integration broker) to specify the names of the business objects for this property.

The syntax for this property is *BOName;BOName*, where the name of a specific business object name is substituted for *BOName*. Case is significant. To specify more than one business object, separate the names with a semicolon, as in `Customer;Item`. Ending punctuation is not required. Templates for these business objects are created the next time you start the connector.

The generated templates contain the delivered default values that are set for the attributes of the business objects in the business object's definition. If there is no delivered default value for an attribute, it is either ignored (using `CxIgnore`) or left blank (using `CxBlank`). One child business object is created for each single-cardinality child business object and two identical instances of a child business object are created for multiple-cardinality business objects.

To begin generating templates for a specified business object, start the connector. The connector writes the template to the same file as the output file. If you do not want to use this feature, leave the `GenerateTemplate` property empty.

Chapter 4. Troubleshooting the JText Connector

This chapter includes the following information to help you diagnose problems with the JText connector.

Error Message Logging

Error messages are logged to the standard connector log file, `STDOUT`, or to the file specified by the `LogFileName` standard connector property.

Errors are also logged to the event log file. For more information on the event log file, see "Event Log File" on page 64.

Problem with Meta-Object Naming

During connector startup, the following error message means that the meta-object name does not correspond to the connector instance name.

```
Wrong subscription: JText_Customer doesn't have supporting MO:
this BO is unsubscribed."
```

If the meta-object name does not match the name of the connector instance, the meta-object does not recognize the business objects supported by the connector. To prevent this, name the meta-object to correspond with the connector instance. For example, a meta-object named `MO_JText2Connector_Default` recognizes business objects supported by the JText2 connector.

Problem with Event Triggering

The connector ignores event files with the following delimiter problems:

- The `EndBODelimiter` attribute in the top-level meta-object is set to a valid value, such as the plus sign (+) or the pipe symbol (|), but the event file does not contain the specified delimiter at the end of each business object.
- The connector is configured to look for the `EndBO:BOName` business object delimiter, but the event file does not contain this delimiter. The connector logs a warning message that states:

```
Unable to create Workunits from file filename. Check EndBODelimiter in the file.
```

In both of the above cases, the file remains in the event directory without any change.

The connector also keeps the file in the event directory without change when device failures occur while a file is being accessed, opened, or closed. For example, if the system tries to access a file when it is out of memory, the connector ignores the file.

JText Failure Handling

For the JText connector, the following types of errors can occur:

Table 10. JText Error Types

Type of Error	Description
Business object delimiter failures	Business object delimiter failures occur when the `EndBODelimiter` attribute in the top-level meta-object is set to a valid value, and the event file contains the specified delimiter at the end of each business object, but the data itself uses the delimiter value in its text. When the connector encounters the delimiter value in the text, it sends a partial business object string to the formatter, which fails processing. In this case, the connector writes the event to the *filename_timestamp*`.fail` file, which contains records for all business objects that encountered delimiter failures.
Subscription errors	Can occur if the connector can find the business object delimiter and retrieve the business object name, but the business object is not subscribed. In this case, an event is sent to the *filename_timestamp*`.unsub` file, which contains records for all unsubscribed business objects.
Formatting errors	Can occur if the connector finds the delimiter with a business object name that does not match the input business object name, or the format in the business object file does not match the format of the meta-object. An event is sent to the *filename_timestamp*`.fail` file, which contains records for all business objects that failed formatting.
Sending errors	Can occur if the connector tries to send a business object when the integration broker is down. If the Send operation fails, an event is sent to the *filename_timestamp*`.fail` file, which contains records for all business objects that were not successfully sent.

Event Log File

The connector logs information about successfully processed business objects to the `event.log` file. If the connector goes down before it processes all business objects in an event file, it uses this log file during recovery to ensure that it sends each business object only once to the integration broker.

The format of the log file is:

EventFileName `::1,2,n`

where `EventFileName` is the name of the current event file, and each number represents the sequence number of a successfully processed business object in that file.

For example, assume that the connector has successfully processed three of the first four business objects in the `Customer.in` file, and that the second business object failed processing. Assume also that the connector has not yet finished processing `Customer.in`. In this case, the `event.log` file might look like the following on UNIX:

```
$CROSSWORLDS/JText/Event/Customer.in :: 1,3,4
```

and like the following on Windows:

```
C:\JText\Event\Customer.in :: 1,3,4
```

If the connector went down before processing the entire Customer.in file, at startup the connector uses the information in the log file to resume processing the event file at the point where it had stopped processing. The connector reads the log to get the name of the event file to be recovered and the latest business-object sequence number. Then the connector begins sending to the integration broker all business objects in the event file whose sequence number is greater than the last number in the log file. For example, given the file above, the connector begins processing the fifth business object in the Customer.in file.

The connector keeps the contents of the log file in memory to enhance performance. It accesses the file on disk only to update it with a new entry. The connector reads the log file only at recovery time.

For information on how the connector uses the event.log file in the recovery process, see "Failure Recovery".

Failure Recovery

Note: The following recovery steps do not apply if a disk failure occurs or a disk is full.

To recover from failures during event notification, the connector does the following:

1. The connector processes business object strings from the event file. When it successfully processes an entry, the connector logs the entry in the event.log file. It also writes it to a file in the archive directory (specified in the ArchiveDir meta-object attribute).
   - If none of the business objects in the event file have failed processing, the connector archives the successfully processed ones in an archive file with the extension specified in the SuccessArchiveExt attribute.
   - If any of the business objects in the event file have failed processing, the connector archives the successfully processed ones in an archive file with the extension specified in the PartialArchiveExt attribute.
   - After it has written business objects to the file specified in the SuccessArchiveExt attribute, if any business object fails processing, the connector changes the extension of this file to the one specified in PartialArchiveExt.

   The delivered default values for these extensions are .success and .partial.
2. If errors occur, the connector does the following:
   - Subscription errors— the connector creates the archive file in the archive directory with the extension specified in the UnsubscribedArchiveExt meta-object attribute. The delivered default value for this extension is .unsub.
   - Formatting errors or sending errors—the connector creates the archive file in the archive directory with the extension specified in the FailArchiveExt meta-object attribute. The delivered default value for this extension is .fail.
   - Business object delimiter errors—the connector creates the archive file in the archive directory with the extension specified in the FailArchiveExt attribute. It also backs up the event file by moving it to the archive directory and changing its extension to the one specified in OriginalArchiveExt.

   The connector does not log the failed business objects to event.log.

3. After the connector processes all business objects in an event file, it clears the `event.log` file and begins writing entries to it from the next event file.
4. If the connector goes down before it processes all business objects in an event file, it uses the information in `event.log` to determine where to begin processing during the recovery process. When it comes back up, the connector checks whether there are any entries in the log file.
   - If there are no entries, the connector sends all business objects in the event file to the integration broker.
   - If there are entries, the connector uses this information to resume processing an event file at the point where it had stopped processing. The connector reads the log to get the name of the event file to be recovered and the latest business-object sequence number. Then the connector sends to the integration broker all business objects in the event file whose sequence number is greater than the last number in the log file. For example, if the event file contains 15 business objects and the last sequence number in the log file is 8, the connector sends the last seven business objects to the integration broker.

     Using the log file prevents the connector from sending the same event multiple times to the integration broker. The connector keeps the log file in memory to enhance performance. The connector accesses the file on disk only to update it with a new entry, and reads the log file only at recovery time.

   If you set the `EventRecovery` configuration property to `retry`, the connector at startup automatically recovers outstanding events from a previously processed file. However, if you set this property to `abort`, the connector terminates during startup if there are any events to be recovered.
5. To recover from errors that occurred during the event notification process, you must restart the connector. Before doing this, however, do the following:
   - Examine the files that the connector created for failed and unsubscribed business objects. Make appropriate corrections so that the business object strings can be successfully processed when the connector starts.
   - Copy appropriate files from the archive directory to the event directory and change all `.fail` or `.unsub` extensions to the extension specified in the `EventExt` attribute (by default, `.in`). To facilitate record-keeping, rename these files in a meaningful way. For example, rename `Customer.unsub` to `Customer_unsub_resubmit.in`.
   - You may need to perform additional steps manually to recover, depending on the type of failure that has occurred.

The following guidelines can help you determine what recovery steps to take, based on the type of error that occurred.

Recovery from Business Object Delimiter Errors

The connector writes the business object to an archive file in the archive directory, giving it the extension specified in the `FailArchiveExt` meta-object attribute. To handle recovery for such a failure, do the following:

1. Ensure that the event file contains the business object delimiter, that the delimiter is correct, and that it does not contain the delimiter value in the data itself as text. If the use of the delimiter is not correct, correct it.
2. Review the connector's log file (specified in the `LogFileName` configuration attribute) to determine other reasons why the process failed.
3. Copy the file from the archive directory to the event directory and change the `.fail` extension to the extension specified in the `EventExt` attribute (by default,

.in). To facilitate record-keeping, rename the file in a meaningful way. For example, rename Customer.fail to Customer_delimiter_error.in.

Recovery from Subscription Errors

The connector writes the business object to a file located in the archive directory, giving it the extension specified in the UnsubscribedArchiveExt meta-object attribute. To handle recovery for such a failure, do the following:

1. Open the archived file, find that business object string, and verify that the business object name and verb are subscribed. Make appropriate corrections if necessary.
2. Ensure that the integration broker is running.
3. Copy the file from the archive directory to the event directory and change the .unsub extension to the extension specified in the EventExt attribute (by default, .in). To facilitate record-keeping, rename the file in a meaningful way. For example, rename Customer.unsub to Customer_unsub_resubmit.in.

Recovery from Formatting Errors

The connector writes the business object to a file located in the archive directory, giving it the extension specified in the FailArchiveExt meta-object attribute. To handle recovery for such a failure, do the following:

1. Open the archived file and verify that:
   - The business object string format matches the expected format in the meta-object. If there is a mismatch, either change the format type in the meta-object or in the business object string.
   - The formatting syntax of the business object string is correct. If it is incorrect, correct it.
2. Copy the file from the archive directory to the event directory and change the .fail extension to the extension specified in the EventExt attribute (by default, .in). To facilitate record-keeping, rename the file in a meaningful way. For example, rename Customer.fail to Customer_fail_formatting.in.

Recovery from Sending Errors

The connector writes the business object to a file located in the archive directory, giving it the extension specified in the FailArchiveExt meta-object attribute. To handle recovery for such a failure, do the following:

1. Verify that all components of the business-integration system are running.
2. Copy the file from the archive directory to the event directory and change the .fail extension to the extension specified in the EventExt attribute (by default, .in). To facilitate record-keeping, rename the file in a meaningful way. For example, rename Customer.fail to Customer_fail_sending.in.
3. Restart the connector.

Data Handlers and Supported Business Objects

If the connector returns an error stating that the data handler has not been configured, verify that the meta-object for the data handler is included in the list of supported business objects. The most common error returned by the connector states that the BOPrefix is not set.

The list of supported business objects for the DHFormatter should include the following:

- MO_JTextConnector_Default
- MO_JTextConnector_*BusObjName* (meta-objects created for specific business objects)

- Business objects that are to be read from or written to a file.
- The meta-object for the data handler (which is specified in the `DataHandlerConfigMO` attribute of the `MO_JTextConnector_Default` meta-object).

Chapter 5. Migrating to or Upgrading the JText Connector

This chapter describes how to upgrade to the 4.4.x version of the JText connector from the 4.0.0 version, and to the 4.0.x version of the JText connector from the 3.2.0 version. It also describes how to upgrade to the JText connector from the Text connector.

Note: The JText connector versions 4.4.x and 4.3.x contain no specific configuration changes. They contain only optional configuration changes, none of which is necessary unless you want to take advantage of the new options. See the "New in This Release" on page xi section for details.

Upgrade Scenarios

If you are upgrading to the 4.1.x release of the JText connector from the 4.0.x release, follow the instructions in "Upgrade Scenarios" on page 69.

If you are upgrading to the 4.1.x release of the JText connector from the 3.2.0 release, follow the instructions in "Upgrading to Version 4.0.x" on page 71 and "Upgrade Scenarios" on page 69.

If you are upgrading to the JText connector from the Text connector, follow the instructions in "Upgrading to the JText Connector" on page 72.

Upgrading to Version 4.1.0 from Version 4.0.x

This section explains:

- "Meta-object Changes" on page 69
- "Architecture Changes" on page 70
- "Jar File Changes" on page 70

Meta-object Changes

The `MO_JTextConnector_Default` meta-object contains three new attributes (`PartialArchiveExt`, `UnsubscribedArchiveExt`, and `OriginalArchiveExt`) that expand the flexibility of archive processing.

You must add these attributes to the JText top-level meta-object, and configure values for them. Use Business Object Designer to add the new attributes, provide your own default values for them, and save the changes to the definition.

Add the three new attributes to each of the meta-objects that you have customized from `MO_JTextConnector_Default`. For example, if you have created your own meta-object for the Customer and Item business objects, add the new attributes to these meta-objects, provide your own default values for them, and save the changes to the repository.

For more information, see Table 7 on page 28.

Architecture Changes

This section explains:

- "Naming Convention Changes"
- "Configuration Property Changes"

Naming Convention Changes

Changes to the connector's architecture require the name of your formatter meta-objects to **not** conform to the naming convention for JText top-level meta-objects.

Some previous releases of the connector delivered a DHFormatter meta-object named `MO_JTextConnector_DHFormatter`. This name is no longer valid. An acceptable name for this meta-object must not have `Connector` specified in the second position; for example, `MO_JText_Default_DHFormatter` is a valid name for this meta-object.

If your formatter meta-objects use the same naming convention as JText top-level meta-objects, you must change their name and change the type of every business-object attribute that represents this meta-object. To make this change:

1. Rename the existing formatter meta-object to the new meta-object name.
2. In the top-level meta-object, change the type of every attribute that represents the formatter meta-object from the previous meta-object name to the new name.
3. Edit the connector's configuration to remove the old meta-object from its list of supported business objects, and add the new formatter meta-object to the list.
4. Remove the old formatter meta-object from the integration broker repository.

For more information, see "JText Meta-Object Naming Conventions" on page 25.

Configuration Property Changes

The `InRecoveryWindow`, `OutRecoveryWindow`, and `EventRecoveryEnabled` connector-specific configuration properties no longer exist. A new property, `EventRecovery`, has been added.

To take advantage of the connector's new archiving feature:

1. Edit the JText connector in Connector Designer to remove the old properties and add the new property.
2. Set the value of the new property to `"abort"` or `"retry"`.

For more information about this property, see "EventRecovery" on page 21.

Jar File Changes

The `CwJTFormatter.jar` file is no longer delivered. All product-delivered formatters have been moved to the `CwJText.jar` file.

If you use custom formatters, which you have placed in the `CwJTFormatter.jar` file, do one of the following:

- Migrate the custom formatters to a new jar file. Include that jar within the class path specified in your `start_JText.bat` or `start_JText.sh` file.
- Place your old `CwJTFormatter.jar` file in your classpath. Ensure that `CwJTFormatter.jar` follows `CwJText.jar` in the classpath.

Reasons to Upgrade to Version 4.0.x from Version 3.2.0

With version 4.0.x of the JText connector, the structure of the meta-objects required to configure the connector was dramatically simplified, thus simplifying the configuration process.

Whereas earlier versions of the connector used a meta-object structure that included three levels of hierarchy and at least ten different meta-objects, the 4.0.x version uses only two meta-objects and only two levels of hierarchy. This new version changes the way that you configure the connector but does not change the connector's functionality.

Upgrading to Version 4.0.x

Because the new meta-objects use the same configuration data as previous versions, upgrading does not require changing any configuration values. However, because the new meta-objects store the data in differently named attributes in far fewer meta-objects, upgrading does require the following operations:

- Create new meta-objects.
- Replace the value of each attribute's `DefaultValue` property in each new meta-object with the customized default values in your existing meta-objects.
- Remove all obsolete meta-objects from the repository.

The IBM CrossWorlds Exchange provides a utility that automates the above operations. To execute these operations manually, perform the following steps:

1. Make a backup of the repository by using the `repos_copy` utility. For example, the following command backs up the entire contents in the Server1 repository to the output file, `InterChangeRepository.out`:

   ```
   repos_copy -oInterChangeRepository.out -sServer1 -pmypassword
   ```

2. For each of your existing top-level meta-objects, create a new meta-object with the same attributes as the new delivered top-level meta-object, `MO_JTextConnector_Default`. For example, if you have created your own meta-object for the Customer business object named according to the previous naming convention (`MO_JText_Customer_Connector`), create a new meta-object for Customer that uses the new naming convention (`MO_JTextConnector_Customer`).
3. Set the default values of the new meta-objects based on the values in the original meta-objects. See Table 11 for the correspondence between the attributes in the original meta-objects and the new ones.
4. Use CSM to delete the original set of meta-object definitions from the repository, keeping only the ones just created and `MO_JText_Default_DHFormatter`.

Table 11 illustrates the correspondence between the original and new attributes, including the names of the original meta-objects. Whereas multiple meta-objects contained the original attributes, `MO_JTextConnector_Default` meta-object contains all of the new attributes.

Table 11. Correspondence of Original Meta-Objects and Attributes to New Attributes

Original Meta-Object Name	Original Attribute Name	New Attribute Name
MO_JText_*BOName*_Connector	DummyKey	N/A
MO_JText_*BOName*_ArchiveDir	Directory	ArchiveDir

Table 11. Correspondence of Original Meta-Objects and Attributes to New Attributes (continued)

Original Meta-Object Name	Original Attribute Name	New Attribute Name
MO_JText_*BOName*_ArchiveFileExt	Success	SuccessArchiveExt
		PartialArchiveExt
	Fail	FailArchiveExt
		UnsubscribedArchiveExt
MO_JText_*BOName*_EventDir	Directory	EventDir
	FileExt	EventExt
MO_JText_*BOName*_OutputDir	Directory	OutputDir
	FileExt	OutputExt
	FileSequencingEnabled	FileSeqEnabled
MO_JText_*BOName*_FormatType		N/A
MO_JText_*BOName*_ServicePolicy	OutputFileName	OutputFileName
	EndBODelimiter	EndBODelimiter
MO_JText_*BOName*_FormatService	EventService	EventFormat
	OutputService	OutputFormat

Table 11 on page 71 does not include the following information

- Attribute(s) corresponding to the `MO_JText_BOname_FormatType` meta-object

  Previous versions of the connector required you to list all possible formats in the `MO_JText_BOname_FormatType` meta-object before configuring the formats to use for event and output files. In the new meta-object structure, you need only configure the formats to be used for event and output files. This change is indicated in Table 11 on page 71 by the absence of corresponding attribute(s) for the `MO_JText_BOname_FormatType` meta-object.
- Meta-objects for the individual Formatters

  The top-level meta-object has two attributes that contain a Formatter meta-object. The Formatter meta-objects have the same attributes as in the original structure, and are used in the same way. Because three of the four Formatters have been deprecated, the only relevant Formatter meta-object is `MO_JText_BOname__DHFormatter`.

Reasons to Upgrade from the Text Connector

If your site currently uses the Text connector to communicate between an application and integration broker, consider upgrading to the JText connector for the following reasons:

- Performance. The Text connector processes only one file at a time, which can hinder performance when processing large files or a great number of files.
- Format availability. The Text connector handles few format types.
- Ease of modification. The Text connector is more difficult to modify.

In contrast, the JText connector can be configured to:

- Process multiple files at one time.
- Search multiple locations for specific business objects, thereby increasing performance.
- Accommodate a wider range of format types.

Upgrading to the JText Connector

To upgrade to the JText connector from the Text connector:

1. From the product CD, copy the JText directory to the %CROSSWORLD%connectors directory.
2. Open a Command Prompt window and use repos_copy to add the following two files to the repository: CN_JText.txt and MO_JText_Default.txt.
3. Ensure that the specified directories for the following meta-object attributes have been created: ArchiveDir, EventDir, and OutputDir. If these directories have not been created, create them.
4. Configure the meta-object attributes.
5. Subscribe to the desired business objects.

Appendix A. Standard Configuration Properties for Connectors

Connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

This chapter describes standard configuration properties, applicable to all connectors. For information about properties specific to the connector, see the installing and configuring chapter of its adapter guide.

The connector uses the following order to determine a property's value (where the highest numbers override the value of those that precede):

1. Default
2. Repository (relevant only when InterChange Server is the integration broker)
3. Local configuration file
4. Command line

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

Configuring Standard Connector Properties for IBM CrossWorlds InterChange Server

This section describes standard configuration properties applicable to connectors whose integration broker is IBM CrossWorlds InterChange Server (ICS). Standard configuration properties provide information that is used by a configurable component of InterChange Server called the **connector controller**. Like the connector framework, the code for the connector controller is common to all connectors. However, you configure a separate instance of the controller for each connector.

A connector, which consists of the connector framework and the application-specific component, has been referred to historically as the **connector agent**. When a standard configuration property refers to the agent, it is referring to both the connector framework and the application-specific component.

For more information on configuring connectors that work on InterChange Server, refer to information on the connector controller in:

- *Technical Introduction to IBM CrossWorlds*
- *IBM CrossWorlds System Administration Guide*
- *IBM CrossWorlds System Implementation Guide*

Important: Not all properties are applicable to all connectors that use InterChange Server. For information specific to an connector, see its adapter guide.

You configure connector properties from Connector Designer, which you access from IBM CrossWorlds System Manager.

Note: Connector Designer and CrossWorlds System Manager run only on the Windows system. Even if you are running the connector on a UNIX system, you must still have a Windows machine with these tools installed. Therefore, to set connector properties for a connector that runs on UNIX, you must start up CrossWorlds System Manager on the Windows machine, connect to the UNIX InterChange Server, and bring up Connector Designer for the connector.

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a runtime session, the property's update semantics determine how and when the change takes effect. There are four different types of update semantics for standard connector properties:

- Dynamic—The change takes effect immediately after it is saved.
- Component restart—The change takes effect only after the connector is stopped and then restarted in CrossWorlds System Manager. This does not require stopping and restarting the application-specific component or InterChange Server.
- Server restart—The change takes effect only after you stop and restart the application-specific component and InterChange Server.
- Agent restart—The change takes effect only after you stop and restart the application-specific component.

To determine the update semantics for a specific property, refer to the Update Method column in the Connector Designer window, or see the Update Method column of the table below.

The following table provides a quick reference to the standard connector configuration properties. You must set the values of some of these properties before running the connector. See the sections that follow for explanations of the properties.

Property Name	Possible Values	Default Value	Update Method	Notes
AgentConnections	1-4	1	server restart	multi-threaded connector only
AgentTraceLevel	0-5	0	dynamic	

Property Name	Possible Values	Default Value	Update Method	Notes
Agent URL	*URL for connector agent gateway*		server restart	HTTP transport only
ApplicationName	*application name*	*the value that is specified for the connector name*	component restart	value required
Anonymous Connections	`true` or `false`	`false`	server restart	HTTPS transport only
CA Certificate Location	*path/filename*		server restart	HTTPS transport only
CharacterEncoding	`ascii7` or `ascii8`	`ascii7`	component restart	HTTPS transport only
ConcurrentEventTriggeredFlows	`1` to `32,767`	`1`	server restart	
ContainerManagedEvents	`JMS` or `no value`	`JMS`		guaranteed event delivery
ControllerStoreAndForwardMode	`true` or `false`	`true`	dynamic	
ControllerTraceLevel	`0-5`	`0`	dynamic	
DeliveryTransport	`MQ`, `IDL`, `JMS`, or `HTTP`	`MQ`	system restart	
GW Name	*gateway name*		server restart	HTTP transport only
jms.BrokerName	If FactoryClassName is `IBM`, use `crossworlds.queue.manager`. If FactoryClassName is `Sonic`, use `localhost:2506`.	`crossworlds.queue.manager`	server restart	JMS transport only
jms.FactoryClassName	`CxCommon.Messaging.jms.IBMMQSeriesFactory` or `CxCommon.Messaging.jms.SonicMQFactory` or any Java class name	`CxCommon.Messaging.jms.IBMMQSeriesFactory`	server restart	JMS transport only
jms.Password	Any valid password		server restart	JMS transport only
jms.UserName	Any valid name		server restart	JMS transport only
Listener Port	*port number*	`80` for HTTP `443` for HTTPS	server restart	HTTP transport only
Locale	`en_US` , `ja_JP`, `ko_KR`, `zh_C`, `zh_T`, `fr_F`, `de_D`, `it_I`, `es_E`, `pt_BR` **Note:** These are only a subset of supported locales.	`en_US`	component restart	
LogAtInterchangeEnd	`true` or `false`	`false`	component restart	
LogFileName	*filename* or `STDOUT`	`C:\CrossWorlds\InterChangeSystem.log`	component restart	
MessageFileName	*path/filename*	`ConnectorNameConnector.txt` or `InterchangeSystem.txt`	component restart	
OADAutoRestartAgent	`true` or `false`	`false`	dynamic	
OADMaxNumRetry	*a positive number*	`1000`	dynamic	
OADRetryTimeInterval	*a positive number in minutes*	`10`	dynamic	
PollEndTime	`HH:MM`	`HH:MM`	component restart	

Property Name	Possible Values	Default Value	Update Method	Notes
PollFrequency	-1 *to 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	
PollStartTime	HH:MM*(HH is 0-23, MM is 0-59)*	HH:MM	component restart	
RestartRetryCount	0-99	3	dynamic	
RestartRetryInterval	*a sensible positive value in minutes*	1	dynamic	
SourceQueue	*valid MQSeries queue name*	SourceQueue		designates event source queue in support of guaranteed event delivery
TraceFileName	*path/filename*	STDOUT	component restart	

AgentConnections

The AgentConnections property controls the number of IIOP connections opened for request transport between an application-specific component and its connector controller. By default, the value of this property is set to 1, which causes InterChange Server to open a single IIOP connection.

This property enhances performance for a multi-threaded connector by allowing multiple connections between the connector controller and application-specific component. When there is a large request/response workload for a particular connection, the IBM WebSphere administrator can increase this value to enhance performance. Recommended values are in the range of 2 to 4. Increasing the value of this property increases the scalability of the Visigenic software, which establishes the IIOP connections. You must restart the application-specific component and the server for a change in property value to take effect.

Important: If a connector is single-threaded, it cannot take advantage of the multiple connections. Increasing the value of this property causes the request transport to bottleneck at the application-specific component. To determine whether a specific connector is single- or multi-threaded, see the installing and configuring chapter of its adapter guide.

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.

Agent URL

Used when HTTP or HTTPS is the delivery transport, specifies the URL of the application-specific component's gateway. You set this value on the controller to allow it to connect to its application-specific component's gateway by providing that gateway's URL and the port number on which that gateway listens for communications from the controller.

Use the format *Protocol*://*host_name*:*port_number*. For example, `http://www.`*`othercompany`*`.com:80`. Specify `https` protocol for secure transport. If you set the protocol to `https`, two additional properties, "Anonymous Connections" and "CA Certificate Location", display in the Connector Designer. These properties are required for HTTPS.

The default port is `80` for HTTP, and `443` for HTTPS. If you do not specify a port in the `Agent URL` property, the controller uses the default port for the selected protocol. However, it is recommended that you explicitly state the port number on both the controller and application-specific component side, even if you use the default port.

The port number you specify in this property's value must match the port number configured for the application-specific component's gateway. That port number is set for the application-specific component in the connector's configuration file. Note that you can change only controller configuration properties using the Connector Definitions screen. To change application-specific component properties, you must directly edit the connector's configuration file.

This property is only displayed when `HTTP` is chosen as the value for the DeliveryTransport property.

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.

Anonymous Connections

Used when HTTPS is the delivery transport, determines whether the controller allows data exchange in the event that the application-specific component gateway cannot be validated. Failure to validate may occur for a number of reasons. For example, if certification authority (CA) certificates are missing at the controller end, or identity certificates or private key stores are missing at the application-specific component gateway end, the server may not be able to validate the gateway. Data exchange is secure in an anonymous connection. However, without validating the application-specific component, the system is vulnerable to unauthenticated access.

The default value is `false`.

This property is only displayed when `HTTP` is chosen as the value for the DeliveryTransport property.

CA Certificate Location

Used when HTTPS is the delivery transport, specifies the directory path where certification authority (CA) certificates are stored. When the application-specific component gateway presents its identity certificates, the connector controller authenticates them with the CA certificates stored in this directory. If the connector controller cannot authenticate the identity certificates, it either terminates the connection or sets up an anonymous connection, depending on the value of the "Anonymous Connections" property.

This property is only displayed when `HTTP` is chosen as the value for the DeliveryTransport property.

CharacterEncoding

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

Note: Java-based connectors do not use this property. A C++ connector currently uses the value ASCII for this property. If you previously configured the value of this property to `ascii7` or `ascii8`, you must reconfigure the connector to use either `ASCII` or one of the other supported values. To determine whether a specific connector is written in Java or C++, see the installing and configuring chapter of its adapter guide.

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

Attention: Do not run a non-internationalized connector against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed.

The default value is `ascii`.

ConcurrentEventTriggeredFlows

Determines how many business objects can be concurrently processed by the connector controller 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 controller for a source application to simultaneously map multiple event business objects, and to simultaneously deliver them to multiple collaboration instances. Setting this property to enable concurrent mapping of multiple business objects can speed delivery of business objects to a collaboration, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

Note: To implement concurrent processing for an entire flow (from a source application to a destination application) also requires that the collaboration be configured to use multiple threads and that the destination application's application-specific component be able to process requests concurrently. To configure the collaboration, set its `Maximum number of concurrent events` property high enough to use multiple threads. For an application-specific component to process requests concurrently, it must be either multi-threaded, or be capable of using Connector Agent Parallelism and be configured for multiple processes (setting the `Parallel Process Degree` configuration property greater than `1`). For information on setting these properties and resources, see the *System Administration Guide*.

Important: To determine whether a specific connector is single- or multi-threaded, see the installing and configuring chapter of its adapter guide.

The ConcurrentEventTriggeredFlows property has no effect on connector polling, which is single-threaded and performed serially. For more information about using concurrent flow processing, see the *System Administration Guide*.

ContainerManagedEvents

Setting this property to JMS enables the connector to remove a message from the source queue and place it on the destination queue as a single transaction. This property can also be set to no value.

Note: To enable guaranteed event delivery when ContainerManagedEvents is set to JMS, you must also configure the PollQuantity property to a value between 1 and 500, and must set the value of the SourceQueue property to the name of the queue on which messages are to be placed.

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

The default value is JMS.

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

ControllerStoreAndForwardMode

Sets the behavior of the connector controller after it detects that the destination application-specific component is unavailable. If this property is set to true and the destination application-specific component is unavailable when an event reaches InterChange Server, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forward the request to it.

Important: If the destination application's application-specific component becomes unavailable after the connector controller forwards a service call request to it, the connector controller fails the request.

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

The default is true.

ControllerTraceLevel

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

DeliveryTransport

Specifies the transport mechanism for the delivery of events. Possible values are MQ for MQSeries, IDL for CORBA IIOP, JMS for Java Messaging Service, HTTP, and WMQI-JMS for Java Messaging Service used with WebSphere MQ Integrator. The default is MQ.

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

Note: Setting the value of the DeliveryTransport property to WMQI-JMS configures the connector for communication with the WebSphere MQ Integrator broker. Setting the DeliveryTransport property to any value other than WMQI-JMS configures the connector for communication with the InterChange Server broker.

MQSeries and IDL

It is recommended using MQSeries rather than IDL for event delivery transport, unless you have compelling reasons not to license and maintain two separate products. MQSeries offers the following advantages over IDL:

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

If you select JMS as the delivery transport, four additional properties, ″jms.BrokerName,″ ″jms.FactoryClassName,″ ″jms.Password,″ and ″jms.UserName,″ display in Connector Designer. 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
- MQSeries 5.1 or 5.2
- InterChange Server (ICS) as the Integration broker

In this environment, you may experience difficulty starting the both the connector controller (on the server side) and the connector (on the client side) due to memory use within the MQSeries 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 Administration Guide* and the *System Installation Guide for UNIX*.

Notes:

- If your installation uses more than 768M of process heap size, this resolution would adversely affect product performance.
- If you run on AIX 4.3.3, you do not need to set the LDR_CNTRL environment variable. However, you must set IPCCBaseAddress to a value of 11 or 12.

HTTP

Enables HTTP communication between the connector controller and application-specific component through the WebSphere business integration system web gateway. HTTP allows communication between InterChange Server and an application-specific component residing on a remote machine across the Internet or an intranet. If you set the DeliveryTransport property to `HTTP`, the "Agent URL" on page 78, "GW Name", and "Listener Port" on page 84 properties are required.

GW Name

Used when HTTP or HTTPS is the delivery transport, specifies the CORBA object name of the controller gateway for InterChange Server. The gateway name must be unique among all other gateways on the local network.

jms.BrokerName

Specifies the broker name to use for the JMS provider.

The default is `crossworlds.queue.manager`.

jms.FactoryClassName

Specifies the class name to instantiate for a JMS provider.

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

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

Locale

Specifies the language code, country or territory, and, optionally, the associated character code set. The value of this property determines such cultural conventions as collation and sort order of data, date and time formats, and the symbols used in monetary specifications. For more information, see the overview chapter of the connector guide for an internationalized connector.

A locale name has the following format:

ll_TT.codeset

where:

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

The default is en_US.

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

Attention: If the connector has not been internationalized, the only valid value for this property is en_US. Do not run a non-internationalized C++ connector against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed. To determine whether a specific connector has been internationalized, see the installing and configuring chapter of its connector guide.

Listener Port

Used when HTTP or HTTPS is the delivery transport, this property specifies the port number on which the controller gateway listens on behalf of the connector controller. If you do not specify a port number, the controller gateway listens on the default port for the selected protocol. The default port is 80 for HTTP, and 443 for HTTPS. If a number other than the default is used, the Remote URL property in the connector configuration file must reflect this.

LogAtInterchangeEnd

Specifies whether to log errors to InterChange Server's log destination, in addition to the location specified in the "LogFileName" property. Logging to the server's log destination also turns on email notification, which generates email messages for the MESSAGE_RECIPIENT specified in the InterchangeSystem.cfg file when errors or fatal errors occur. As an example, when a connector loses its connection to its application, if "LogAtInterchangeEnd" is set to true, an email message is sent to the specified message recipient. The default is false.

LogFileName

The name of the file where the application-specific component logs messages. Specify the file name in an absolute path. To log to the command prompt window that opens when the application-specific component starts, change the value of this property to STDOUT. The default value is STDOUT.

MessageFileName

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

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

Important: To determine whether a specific connector has its own message file, see the installing and configuring chapter of its adapter guide.

OADAutoRestartAgent

Specifies whether the Object Activation Daemon (OAD) automatically attempts to restart the application-specific component after an abnormal shutdown. The properties "OADMaxNumRetry" and "OADRetryTimeInterval" are related to this property. This property is required for automatic restart. For more information, see the*IBM CrossWorlds System Administration Guide*.

The default value is `false`.

OADMaxNumRetry

Specifies the maximum number of times that the OAD automatically attempts to restart the application-specific component after an abnormal shutdown. For more information, see the *IBM CrossWorlds System Administration Guide*.

The default value is `1000`.

OADRetryTimeInterval

Specifies the number of minutes of the retry time interval that the OAD automatically attempts to restart the application-specific component after an abnormal shutdown. If the application-specific component does not start within the specified interval, the OAD repeats the attempt as many times as specified in "OADMaxNumRetry". For more information, see the *System Administration Guide*.

The default is `10`.

PollEndTime

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

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

PollFrequency

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

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

The default is `10000`.

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

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.

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. For more information, see the *IBM CrossWorlds System Administration Guide*.

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. For more information, see the *IBM CrossWorlds System Administration Guide*.

The default is `1`.

SourceQueue

Designates the source queue for the connector framework in support of guaranteed event delivery. For further information, see "ContainerManagedEvents" on page 81.

The default value is `SourceQueue`.

TraceFileName

The name of the file where the application-specific component writes trace messages. Specify the filename in an absolute path. The default is `STDOUT`.

Configuring Standard Connector Properties for WebSphere MQ Integrator

This section describes standard configuration properties applicable to adapters whose integration broker is WebSphere MQ Integrator (WMQI). For information on using WMQI, see the *Implementation Guide for MQ Integrator*.

Important: Not all properties are applicable to all connectors that use WMQI. For information specific to a connector, see its adapter guide.

You configure connector properties from Connector Configurator.

Note: Connector Configurator runs only on the Windows system. Even if you are running the connector on a UNIX system, you must still have a Windows machine with this tool installed. Therefore, to set connector properties for a connector that runs on UNIX, you must execute Connector Configurator on the Windows computer and copy the configuration files to the UNIX

computer using FTP or some other file transfer mechanism. For more information about Connector Configurator, see Appendix B, "Connector Configurator."

A connector obtains its configuration values at startup. If you change the value of one or more connector properties during a runtime session, you must restart the connector. Standard configuration properties provide information that is used by the adapter framework and connector framework, and is common to all connectors.

Standard Connector Properties

The following table provides a quick reference for standard connector configuration properties. See the sections that follow for explanations of the properties.

Name	Possible Values	Default Value
AdminInQueue	*valid MQSeries queue name*	ADMININQUEUE
AdminOutQueue	*valid MQSeries queue name*	ADMINOUTQUEUE
AgentTraceLevel	0-5	0
ApplicationName	*application name*	*AppName*Connector
BrokerType	WMQI	WMQI
CharacterEncoding	ASCII, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 **Note:** These are only a subset of supported values.	ASCII
ConcurrentRequests	1-10	10
ContainerManagedEvents	JMS *or no value*	JMS
DeliveryQueue	*valid MQSeries queue name*	DELIVERYQUEUE
DeliveryTransport	WMQI-JMS	WMQI-JMS
FaultQueue	*valid MQSeries queue name*	FAULTQUEUE
Locale	en_US , ja_JP, ko_KR, zh_C, zh_T, fr_F, de_D, it_I, es_E, pt_BR **Note:** These are only a subset of supported locales.	en_US
MessageFileName	*path/filename*	*InterchangeSystem.txt*
PollEndTime	*HH:MM*	HH:MM
PollFrequency	*milliseconds*/key/no	10000
PollStartTime	*HH:MM*	HH:MM
QueueManager	*valid MQSeries queue manager name*	crossworlds.queue.manager
QueueManagerLogin	*user name for MQSeries queue manager*	crossworlds
QueueManagerPassword	*password for MQSeries queue manager user name*	WMQI
RepositoryDirectory	*path/directory name*	C:\crossworlds\Repository
RequestQueue	*valid MQSeries queue name*	REQUESTQUEUE
RestartRetryCount	0-99	3

Name	Possible Values	Default Value
RestartRetryInterval	*an appropriate integer indicating the number of minutes between restart attempts*	`1`
SourceQueue	*valid MQSeries queue name*	`SourceQueue`
SynchronousRequestQueue	*valid MQSeries queue name*	
SynchronousResponseQueue	*valid MQSeries queue name*	
Timeout	*an appropriate integer indicating the number of minutes the connector waits for a response to a synchronous request*	`0`
WireFormat	CwXML	`CwXML`

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

The default value is `ADMININQUEUE`.

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

AgentTraceLevel
Level of trace messages for the connector's 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 connection to the application. This name is used by the system administrator to monitor the connector's environment. When you create a new connector definition, this property defaults to the name of the connector; when you work with the definition for an IBM WebSphere-delivered connector, the property is also likely to be set to the name of the connector. Set the property to a value that suggests the program with which the connector is interfacing, such as the name of an application, or something that identifies a file system or website in the case of technology connectors.

BrokerType
This property is set to the value `WMQI` for connectors that are configured to use WebSphere MQSeries Integrator as the integration broker.

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

Note: Java-based connectors do not use this property. A C++ connector currently uses the value ASCII for this property. If you previously configured the value of this property to `ascii7` or `ascii8`, you must reconfigure the connector to use either `ASCII` or one of the other supported values. To determine whether a specific connector is written in Java or C++, see the installing and configuring chapter of its adapter guide.

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

Attention: Do not run a non-internationalized connector against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed.

The default value is ascii.

ConcurrentRequests
ConcurrentRequests is 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.

ContainerManagedEvents
Setting this property to JMS enables the connector to remove a message from the source queue and place it on the destination queue as a single transaction. This property can also be set to no value.

Note: When ContainerManagedEvents is set to JMS, you must also configure the following properties to enable guaranteed event delivery: PollQuantity = 1 to 500, SourceQueue = SOURCEQUEUE

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

The default value is JMS.

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

The default value is DELIVERYQUEUE.

DeliveryTransport
Specifies the transport mechanism for the delivery of events. The property defaults to the value WMQI-JMS, indicating that the Java Messaging Service is used for communication with WebSphere MQ Integrator. Although the list of possible values in the drop-down menu also includes MQ, IDL, JMS, and HTTP, this property must be set to WMQI-JMS when WMQI is the integration broker or the connector cannot start.

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 FAULTQUEUE.

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. For more information, see the overview chapter of the connector guide for an internationalized connector.

A locale name has the following format:
ll_TT.codeset

where:

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

The default is en_US.

Important: By default only a subset of supported locales display in the drop list. To add other supported values to the drop list, you must manually modify the \Data\stdConnProps.xml file in the product directory.

Attention:

- WebSphere MQ Integrator supports only one locale at a time. Ensure that every component of the installation (for example, all adapters, applications, and the integration broker itself) is set to the same locale.
- If the connector has not been internationalized, the only valid value for this property is en_US. Do not run a non-internationalized C++ connector against InterChange Server version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed. To determine whether a specific connector has been internationalized, see the installing and configuring chapter of its connector guide.

MessageFileName

The name of the connector message file. The standard location for the message file is \connectors\messages. Specify the message filename in an absolute path if the message file is not located in the standard location. This property defaults to the value InterchangeSystem.txt for new connector definitions and should be changed to the name of the message file for the specific connector.

PollEndTime

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

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

PollFrequency

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

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

The default is 10000.

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

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.

QueueManager
The queue manager that controls the various queues used for communication between the connector and the integration broker.

The default value is crossworlds.queue.manager.

QueueManagerLogin
A valid user name to log in to the MQSeries queue manager specified for the "QueueManager" property.

If local binding is being used to connect to the queue manager, then this property is not necessary. If client mode is being used to connect to the queue manager over TCP/IP, however, then this property must be set to a valid queue manager user name. For more information on local binding and client mode communication, see the *WebSphere Business Integration Adapter Implmentation Guide for MQ Integrator*.

The default value is crossworlds.

QueueManagerPassword
The password for the MQSeries queue manager user name specified for the "QueueManagerLogin" property.

If local binding is being used to connect to the queue manager, then this property is not necessary. If client mode is being used to connect to the queue manager over TCP/IP, however, then this property must be set to the proper password. For more information on local binding and client mode communication, see the *WebSphere Business Integration Adapter Implmentation Guide for MQ Integrator*.

The default value is admin.

RepositoryDirectory
The path and name of the directory from which the connector reads the XML schema documents that store the meta-data of business object definitions.

The default value is C:\crossworlds\repository.

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

The default value is REQUESTQUEUE.

RestartRetryCount
Specifies the number of times the connector attempts to restart itself. The default value is 3, indicating that the connector tries to restart 3 times. For instance, if a

connector is unable to log in to an application it fails to start, but with this property set to the value 3 the connector tries a total of three times to start. When used in conjunction with the "RestartRetryInterval" property, this behavior enables a connector to make several attempts at communicating with an application that might not reliably have a connection available all the time.

RestartRetryInterval

Specifies the interval in minutes at which the connector attempts to restart itself. The default value is 1, indicating that the connector waits 1 minute in between its restart attempts.

SourceQueue

Designates the source queue for the connector framework in support of guaranteed event delivery. For further information, see "ContainerManagedEvents" on page 81.

The default is `SOURCEQUEUE`.

SynchronousRequestQueue

Delivers request messages that require a synchronous response from the connector framework to WMQI. 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 WMQI on the SynchronousResponseQueue. The response message sent to the connector bears a correlation ID that matches the ID of the original message.

SynchronousResponseQueue

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

Timeout

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

The data format for messages exchanged by the connector. The default value `CwXML` is the only valid value, and directs the connector to compose the messages in XML.

Appendix B. Connector Configurator

Before you can use a connector, you must create or modify a connector configuration file (`*.cfg` file) that sets the properties for the connector, designates the business objects and any meta-objects that it supports, and sets logging and tracing values that the connector will use at runtime.

Use Connector Configurator to create and modify the configuration file for your connector. If a configuration file has previously been created for your connector, you can use Connector Configurator to open the file and modify its settings. If no configuration file has yet been created for your connector, you can use Connector Configurator to both create the file and set its properties. This appendix describes how to use Connector Configurator to:

- Create a new configuration file
- Set properties in a configuration file
- Complete the configuration tasks for the connector

Note: The Server menu selection of the Connector Configurator screen is not used for configuring connectors for use with MQ Integrator as the broker; it is reserved for use with InterChange Server (ICS). If you are using MQ Integrator as the broker, do not attempt to connect to InterChange Server.

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

Note: Some properties in the connector configuration file use directory paths, and these paths default to the Windows convention for directory paths. If you use the connector configuration file in a UNIX environment, revise any directory path constructs in the configuration properties to match the UNIX convention for directory paths.

Using Connector Configurator in an Internationalized Environment

Because Connector Configurator is internationalized, it handles character conversion between the configuration file and the integration broker. The tool 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\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>
```

Creating a New Configuration File

You can create a connector configuration (`*.cfg`) file in either of the following ways:

- Create a completely new connector configuration file from within Connector Configurator
- Load a file that contains preliminary settings for your connector (referred to as a connector definition file) and save it as a connector configuration file

Creating a File within Configurator

1. Choose File > New.
2. The New Connector dialog appears, with the following fields:
   - Name

     Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, must end with the word "connector", and must be consistent with the file name for a connector that is installed on the system; for example, enter `ClarifyConnector` if the connector file name is `Clarify.dll`.

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

     Choose WMQI connectivity.
3. The configuration screen displays, showing the broker that you are using and the name that you have given to the connector. You can fill in all the field values to complete the definition now (see "Setting the Configuration File Properties" on page 96), or you can save the file and complete the fields later.
4. To save the file, choose File > Save > to File. The Save File Connector dialog displays. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the the correct case, navigate to the directory where you want to locate the file, and choose Save. The status display in the lower 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 in "Setting the Configuration File Properties" on page 96.

Loading Settings from a Connector Definition File

The connector definition file 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).

Note: If you have previously used an IBM WebSphere connector in an environment that uses InterChange Server (ICS) as the integration broker, the definitions for that connector may be available to you. Check for a repository file that was used in the IBM CrossWorlds configuration of that connector. Such a file typically has the extension `.in` or `.out`. However, the definition is not complete until you perform the steps described below.

To complete the connector configuration:

- Designate the supported business objects and, if required, the meta-objects.
- Set values for the trace and log files.
- Set appropriate values for the connector under the Standard and Application Config Properties tabs.
- Set required Standard or Application Config properties that have no existing value (only for some connectors).
- Using the Application Config Properties tab, add an application-specific property and set its value (only for some connectors).

Although some of these values are preset in the connector definition file, you may wish to change their values.

To use a connector definition file to configure a connector, you must open the definition file in Connector Configurator, revise the configuration, and then save the file as a configuration file (`*.cfg` file). To do this, follow these steps:

1. In Connector Configurator, choose File > Open > From File.
2. In the Open File Connector dialog, choose one of the following:
   - 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 display 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 choose Open.
4. The Connector Configurator window displays the configuration screen, with the attributes and values that Connector Configurator finds in the connector definition file.

   The title of the configuration screen displays the type of the broker and the name of the connector as specified in the connector definition file. If you use a connector definition file or repository file that retains its original values, the

title on the configuration screen may display the broker type as "ICS." You must change this value before you can configure the connector for use with MQ Integrator as the broker. To do so:

a. Under the Standard Properties tab, select the value field for the Delivery Transport property. In the drop-down menu, select the value `WMQI-JMS`.

b. The Standard Properties tab refreshes to display a new property, BrokerType, with `WMQI` as the value. This value indicates that MQ Integrator has been selected as the broker type. When you save the file, you retain this broker selection. You can save the file now or proceed to complete the remaining configuration fields, as described in "Setting the Configuration File Properties".

c. When you have finished making entries in the configuration fields, choose File > Save > To File, choose `*.cfg` as the extension, choose the correct location for the file in the directory structure, and choose Save. (If multiple connector configurations are open, as they might be if you have opened a repository file, choose Save All to save all of the configurations.)

   Before it saves the file, Connector Configurator validates 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.

   Important: Connector Configurator saves the configuration file to the specified location and file name. When you start the connector, the name and final location of its configuration file must match exactly (including case) the name and path specified in its startup file.

Setting the Configuration File Properties

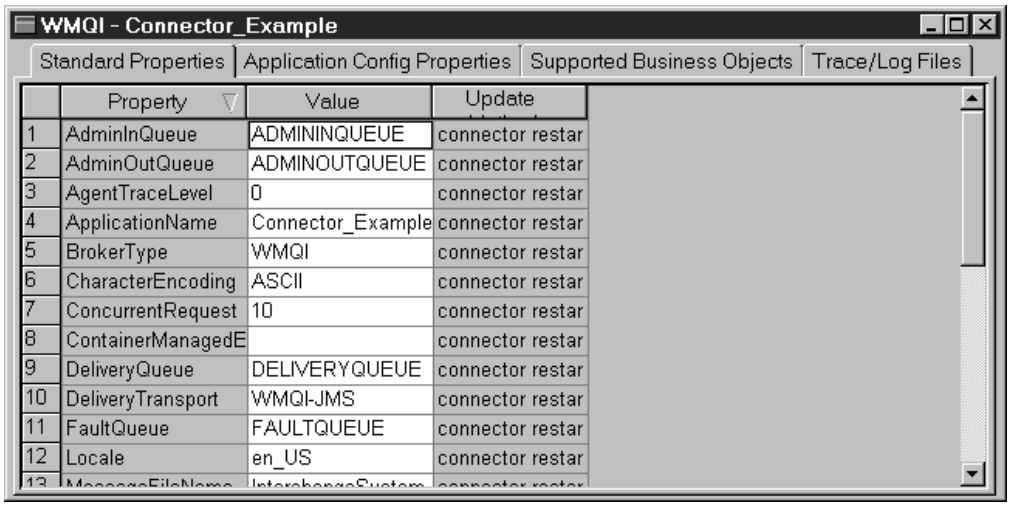
When you create and name a new connector configuration file, or when you open an existing connector configuration file, Connector Configurator displays a configuration screen with tabs for the categories of required configuration values.

Connector Configurator requires values for properties in all of these categories:

1. Standard Properties
2. Application Config Properties
3. Supported Business Objects
4. Trace/Log File values

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 information about the values to use in the Data Handlers category, see "Using Guaranteed Event Delivery" in the *Connector Development Guide*.

The following screen displays the values of the Standard Properties tab:



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

Standard properties differ from application configuration 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-configuration (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 chapter of each adapter guide describes the application-specific properties and the recommended values.

The fields for Standard Properties and Application Config 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 specific needs of the application or connector. You can change the value and delete these properties.
- Value fields are configurable.
- The Update Method field is informational and not configurable. This field specifies the action required to activate a property whose value has changed.

Setting Standard Connector Properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or choose from the drop-down menu if one appears.
3. After entering all values for the standard properties, you can do one of the following:

- To discard the changes, preserve the original values, and exit Connector Configurator, choose File > Exit (or close the window), and choose No when prompted to save changes.
- To enter values for other categories in Connector Configurator, choose the tab for the category. The values you enter for Standard Properties (or 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 of the categories as a whole.
- To save the revised values, choose File > Exit (or close the window) and choose Yes when prompted to save changes. Alternatively, choose Save > To File from either the File menu or the toolbar.

Setting Application-Configuration Properties

For application-specific configuration properties, you can add or change property names, configure values, delete a property, and encrypt a property:

1. Click in the field whose name or value you want to set.
2. Enter a name or value.
3. To encrypt a property, click 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 server 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 clicking the Encrypt checkbox in the Edit Property window. To decrypt a value, click to clear the Encrypt check box, enter the correct value in the Verification dialog box, and choose OK. If the entered value is correct, the value is decrypted and displays. The adapter guide for each connector contains a list and description of each property and its default value.

Update Method

When MQ Integrator is the integration broker, connector properties are static. The Update Method is always Connector Restart. In other words, for changes to take effect, you must restart the connector after saving the revised connector configuration file.

Specifying Supported Business Object Definitions

The procedures in this section assume that you have already created:

- Business object definitions
- MQ message set files (`*.set` files)

The `*.set` files contain message set IDs that Connector Configurator requires for designating the connector's supported business objects. See the *WebSphere Business Integration Adapters Implementation Guide for MQ Integrator* for information about creating the MQ message set files

Each time that you add business object definitions to the system, you must use Connector Configurator to designate those business objects as supported by the connector.

Important: If the connector requires meta-objects, you must create message set files for each of them and load them into Connector Configurator, in the same manner as for business objects.

To specify supported business objects:

1. Select the Supported Business Objects tab and choose Load. The Open Message Set ID File(s) dialog displays.
2. Navigate to the directory where you have placed the message set file for the connector and select the appropriate message set file (*.set) or files.
3. Choose Open. The Business Object Name field displays the business object names contained in the *.set file; the numeric message set ID for each business object is listed in its corresponding Message Set ID field. Do not change the message set IDs. These names and numeric IDs are saved when you save the configuration file.
4. When you add business objects to the configuration, you must load their message set files. If you attempt to load a message set that contains a business object name that already exists in the configuration, or if you attempt to load a message set file that contains a duplicate business object name, Connector Configurator detects the duplicate and displays the Load Results dialog. The dialog shows the business object name or names for which there are duplicates. For each duplicate name shown, click in the Message Set ID field, and choose the Message Set ID that you wish to use.

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

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.

Appendix C. JText Connector Feature List

This appendix details the features supported by the JText connector. For descriptions of these features, see "Appendix A: Connector Feature Checklist" in the *Connector Development Guide*.

Business Object Request Handling Features

Table 12 details the business object request handling features supported by the connector.

Table 12. Business Object Request Handling Features

Category	Feature	Support	Notes
Create	Create verb	N/A	
Delete	Delete verb	N/A	
	Logical delete	N/A	The connector does not delete files from disk.
Exist	Exist verb	N/A	The connector deals with business objects and files. It does not check for the existence of an entity in an application database.
Misc	Attribute names	Full	
	Business object names	No	The connector uses business object names; they dictate how the connector should format them.
Retrieve	Ignore missing child object	N/A	
RetrieveByContent	Ignore missing child object	N/A	
	Multiple results	N/A	
	RetrieveByContent verb	N/A	
Update	After-image support	N/A	
	Delta support	N/A	
	KeepRelations	N/A	
Verbs	Retrieve verb	N/A	The connector does not support the Retrieve verb. It always returns FAIL, to be consistent with other connectors.
	Subverb support	N/A	
	Verb stability	N/A	The connector does not return any business objects during a service call request operation. Business objects are written to disk. The JText connector ignores verbs as long as they are not null.

Event Notification Features

Table 13 details the event notification features supported by the connector.

Table 13. Event Notification Features

Category	Feature	Support	Notes
Connector Properties	Event distribution	No	Not supported.
	PollQuantity	Full	The connector submits UPTO PollQuantity business object each poll cycle.
Event Table	Event status values	N/A	
	Object key	No	
	Object name	Full	

Table 13. Event Notification Features (continued)

Category	Feature	Support	Notes
	Priority	N/A	
Misc.	Archiving	Full	If a file is processed successfully, it is archived with the `.success` extension and moved into the Archive directory. If any of the records in a file are not processed successfully then `.fail`, `.unsub`, `.partial`, `.orig`, or `.success` archive files may be created.
	CDK method gotApplEvent	Full	Only Poll thread makes this call.
	Delta event notification	No	
	Event sequence	Partial	
	Future event processing	No	
	In-Progress event recovery	Partial	
	Physical delete event	No	
	RetrieveAll	N/A	
	Smart filtering	Partial	Duplication is checked using the filename and time stamp. Files matching both properties in the event table are not picked up. But there is no checking for event duplication at the business object level.
	Verb stability	Full	

General Features

Table 14 details the general features supported by the connector.

Table 14. General Features

Category	Feature	Support	Notes
Business Object Attributes	Foreign key	No	
	Foreign Key attribute property	N/A	
	Key	No	
	Max Length	Partial	This attribute applies when formatting business objects to fixed-sized records.
	Meta-data-driven design	Full	The JText connector supports any business objects supported by WebSphere Business Integration Adapter Framework. However, some formatters and data handlers have specific requirements for application text, business object structure, and so on. If these formatters or data handlers are used, only business objects that meet those requirements can be successfully processed by the JText connector.
	Required	Partial	
Connection Lost	Connection lost on poll	Partial	
	Connection lost on request processing	Partial	
	Connection lost while idle	Partial	In certain conditions (such as an IO error), JText sends `APPRESPONSE_TIMEOUT`.
Connector Properties	ApplicationPassword	No	
	ApplicationUserName	No	
	UseDefaults	Full	

Table 14. General Features (continued)

Category	Feature	Support	Notes
Message Tracing	General messaging	Full	
	generateMsg()	Full	All trace messages are hard-coded.
	Trace level 0	Partial	
	Trace level 1	Partial	
	Trace level 2	Partial	
	Trace level 3	Partial	
	Trace level 4	Partial	
	Trace level 5	Partial	
Misc.	CDK method LogMsg	No	
	Java Package Names	Full	
	Logging messages	Full	All known error conditions are taken care of.
	NT service compliance	Full	
	Transaction support	N/A	
Special Value	CxBlank processing	No	
	CxIgnore processing	Full	

Printed in U.S.A.