WebSphere Business Integration Express
and Express Plus for Item Synchronization

Adapter for JText User Guide

Version 5.3.x

WebSphere Business Integration Express
and Express Plus for Item Synchronization

Adapter for JText User Guide

Version 5.3.x

Note!
Before using this information and the product it supports, read the information in "Notices" on page 87.

19December2003

This edition of this document applies to IBM WebSphere Business Integration Express for Item Synchronization, version 4.3.1, IBM WebSphere Business Integration Express Plus for Item Synchronization, version 4.3.1, and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, 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.

Contents

About this document

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

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

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

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.

Prerequisites for this document

You need to be familiar with adapters, business object development, and data handlers. You also need to be familiar with the XML markup language and a schema language, either document type definition (DTD) or XSDL (for schema documents).

Related documents

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

This document contains many references to the *User Guide for WebSphere Business Integration Express for Item Synchronization* and to *Installing WebSphere Business Integration Express for Item Synchronization*. If you chose to print this document, you may wish to print the these other manuals as well. You can download, install, and use the documentation at the following site:

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

Typographic conventions

This document uses the following conventions:

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

New in this release

New in version 5.3.x

In this release, version 5.3 of the Adapter for JText is supported on the IBM WebSphere Business Integration Express and Express Plus for Item Synchronization release.

The Adapter for JText is supported on the following operating systems:

- Microsoft Windows 2000
- OS400 V5R2 (5722-SS1)
- Red Hat Enterprise Linux WS/ES/AS for the Intel 2.1 and 2.4 kernels.
- SuSE Linux Enterprise Server 7.3 for the Intel 2.4 kernel.

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

Chapter 1. Overview of the JText adapter

This chapter describes the JText adapter and its components. The adapter enables the InterChange Server Express integration broker to exchange business objects, files, or messages with other applications. The adapter enables the integration broker to communicate with an application by exchanging text or binary files. This connector facilitates integration of data with applications that lack an API.

Adapters 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 adapter.

Topics included in this chapter are:

For more information about the relationship of the integration broker to the connector, see the *System Administration Guide*.

Use the JText adapter 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.
- String or binary 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 string or binary files through the JText connector.

Adapter components

The JText adapter has the following components:

Figure 1 illustrates the JText connector's architecture when IBM WebSphere InterChange Server is used as the integration broker.

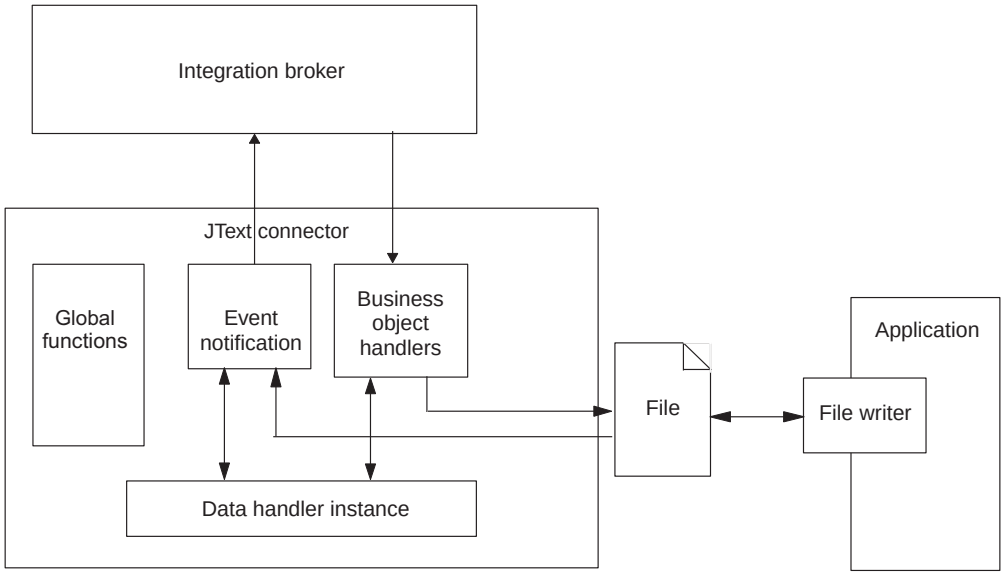

Figure 1. JText connector architecture

Application-specific component

The JText adapter's application-specific component manipulates files and calls a specified data handler to convert data between business objects and strings or byte arrays. 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 the Adapter Framework provides or data handlers that you create to handle specific text-formatting needs. The product provides the following data handlers:

- 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.

For more information, see "How data handler processing works" on page 11. 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 Connector Configurator Express, 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 or byte arrays that it reads from files into business objects, and to format strings or byte arrays 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, and request processing.

The JText adapter 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 23.

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 Data Handler for XML, 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 the following name-value pair:

- `cw_mo_JTextConfig = DynChildMOAttrName`

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 4
- "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 43.

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

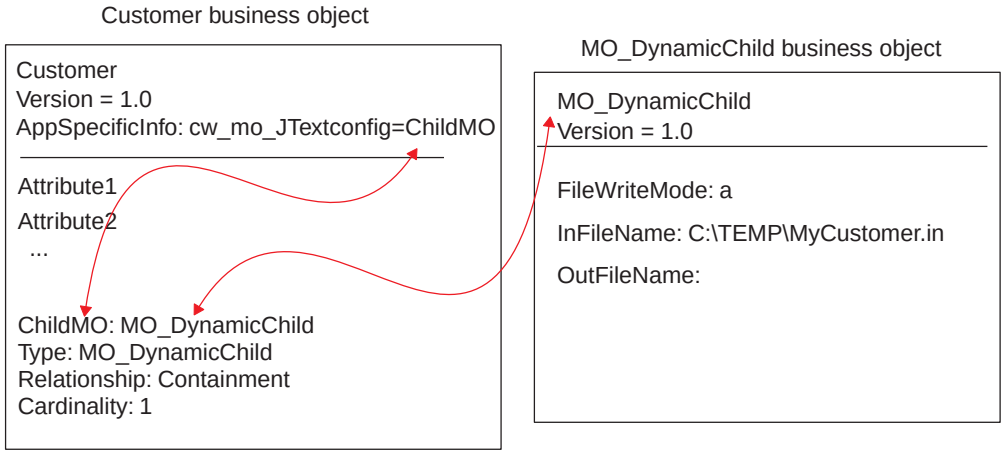

Figure 2. Example of a dynamic child meta-object

How the connector works

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

- Event notification
- 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 a parsing method to determine which subsection represents each business object. For more information, see "Polling for specific business objects" on page 40.
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 `getBO()`or `getBOByteArray()` on the data handler instance, and sends the string or byte array that represents the business object to it. The connector passes each element that represents a business object to the data

handler. When a file represents multiple business objects, the connector sends only an element (that is, a string or byte array representation of a single business object), not the entire file.

4. The data handler converts the string or byte array 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 *Data Handler Guide*.
6. If the data handler encounters any error that prevents it from creating a business object, the connector archives the string or byte array 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 40.

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 objects and the second has 12 business objects. On the first poll call, the connector performs the following steps:

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

On the second poll call, the connector sends the 2nd through 6th business objects from the second file. On the third poll call, the connector sends the 7th through 11th business objects from the second file. On the fourth poll, the connector sends the last business object. 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 52.

- On specifying the event directory and extension, see "Specifying event directories and extensions" on page 34.
- On specifying event processing, see "Specifying event notification" on page 34.

Figure 3 shows an event notification operation (numbers in the graphic do not correlate to the steps outlined above).

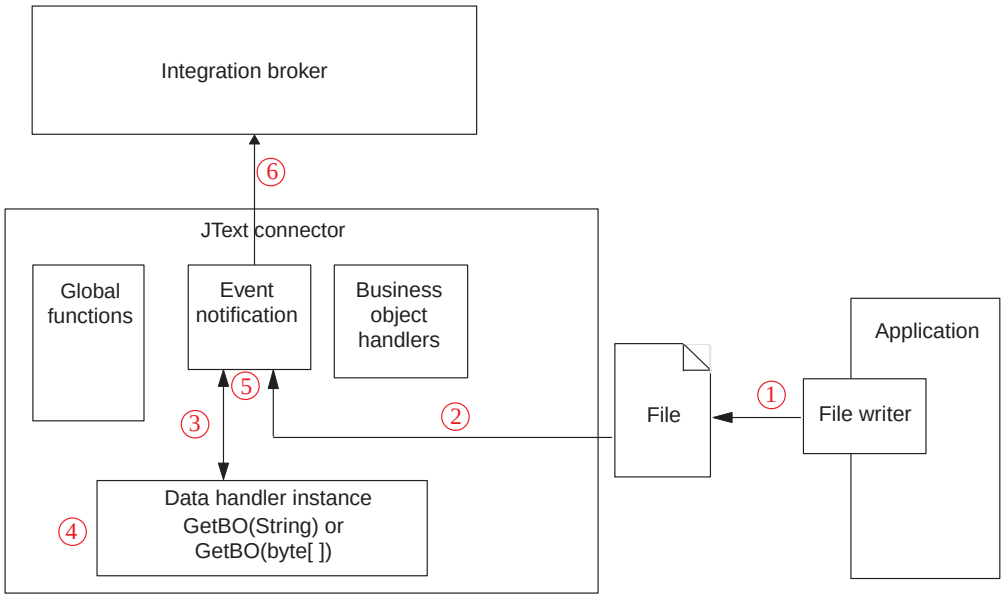

Figure 3. Event notification operation

Event archiving

After it has processed an event, and if it is configured to enable archiving, the JText connector writes the business object string or byte array representation of one business object into a file in the local 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_2003_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 35.

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 the Adapter for JText 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`	`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`	`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`	`C:\temp\JTextConn\Default\Archive`
Archive	unsubscribed	`unsub`	`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`	`C:\temp\JTextConn\Default\Archive`
Archive	fail	`fail`	`C:\temp\JTextConn\Default\Archive`
Output	out	`out`	`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 8 on page 26..

Request processing

When processing a service call request, the connector converts the business object to an output string or byte array, 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. In this case, 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 10

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 must contain the following:

   `cw_mo_JTextConfig = DynChildMOAttrName`
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 string or a byte array, which it returns to the configuration. The data handler also reports errors and provides tracing.
6. The connector writes the string or byte array to a file.

For information on configuring the connector to process requests, see "Specifying request processing" on page 36.

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 10.
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 string or a byte array, which it returns to the configuration. The data handler also reports errors and provides tracing.
7. The connector writes the string or byte array to a file whose name it derives in step 3. above.

Figure 4 illustrates the JText connector components when the connector processes requests from an integration broker to the destination application.

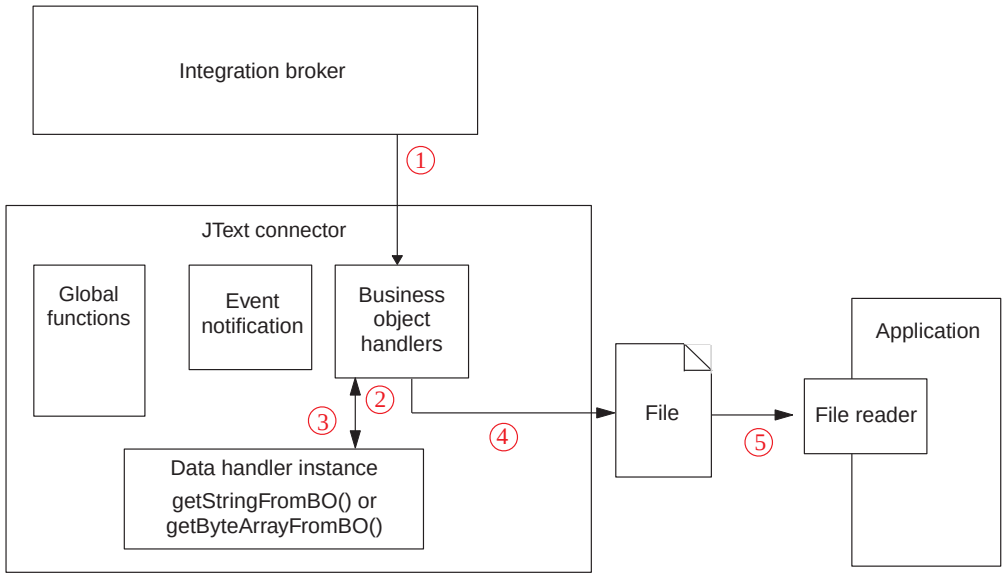

Figure 4. Business object request operation

How data handler processing works

The connector uses a data handler instance to convert between business objects and strings or byte arrays 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*.

The connector determines which interface, string or byte array, based on the setting of the DataProcessingMode meta-object. For additional information on this meta-object, see Table 8 on page 26

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 getBO(String) or getBO(byte[]) 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() or getByteArrayFromBO() method on the data handler. The connector passes to the data handler the business object to be converted to a string or byte array. The data handler returns a serialized version of the business object, in the form of a string or byte array.

The getBO(String) or getBO(byte[])and the getStringFromBO() or getByte ArrayFromBO() 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 adapter," on page 15, Chapter 3, "Using JText connector meta-objects," on page 23, and the *Data Handler Guide*.

JText differences from other adapters

While the JText connector enables the transfer of data from a source application to a destination application like other adapters, 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 or byte array in which key values have no more significance than other data.
- It uses meta-object values for much of its configuration. For more information, see Chapter 3, "Using JText connector meta-objects," on page 23.
- 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 WebSphere business integration system are written in Java. Therefore, when data is transferred between most WebSphere business integration system 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 Appendix A, "Standard configuration properties for connectors," on page 61.

Chapter 2. Installing and configuring the JText adapter

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

Compatibility

The adapter framework that an adapter uses must be compatible with the version of the integration broker (or brokers) with which the adapter is communicating. The 5.3.x version of the adapter for JText is supported on the following adapter framework and integration brokers:

Adapter framework: Adapter Framework, version 2.3.1.

Integration brokers: InterChange Server Express, version 4.3.1.

See the Release Notes for any exceptions.

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. This needs to be done on both the local and remote servers.

In addition, review the following software requirements before you install the adapter for JText.

Software requirements

To implement the adapter for JText, you must install InterChange Server Express, Version 4.3.1 or later.

The Adapter for JText is supported on the following operating systems:

- Microsoft Windows 2000
- OS400 V5R2 (5722-SS1)
- Red Hat Enterprise Linux WS/ES/AS for the Intel 2.1 and 2.4 kernels.
- SuSE Linux Enterprise Server 7.3 for the Intel 2.4 kernel.

Installing the JText adapter and related files

To install adapters for Business Integration Express for Item Sync:

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

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

File structure for the JText adapter

The following three tables describe the file structure used by the e-Mail adapter with the supported operated systems, Microsoft Windows 2000, OS400 V5R2 (5722-SS1), Red Hat Enterprise Linux WS/ES/AS, and SuSE Linux Enterprise Server 7.3. *ProductDir* refers to the base install directory which varies according to which operating system is being used.

Microsoft Windows 2000

On Microsft Windows 2000, the 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.

Table 2. Installed file structure for the JText adapter on Windows 2000

Subdirectory of%*ProductDir*%	Description
ProductDir\connectors\JText\CWJText.jar	Contains the connector and the formatters
ProductDir\connectors\JText\start_JText.bat	System startup script for the connector. This script is called from the generic connector manager script.
ProductDir\repository\JText\	Contains the MO_JText_Default.txt file .
ProductDir\connectors\JText\Dependencies\	Contains the license agreement., README files, and third party dependency files for FTP and NetComponents
ProductDir\connectors\messages\	Contains the JTextConnector.txt file.

OS400

All product pathnames are relative to the directory where the product is installed on your system. The base install directory (*ProductDir*) as shipped for OS400 is /QIBM/UserData/WebBIICS.

Table 3. Installed file structure for the JText adapter on OS/400

Subdirectory of%*ProductDir*%	Description
ProductDir/connectors/JText/CWJText.jar	Contains the connector and the formatters
ProductDir/connectors/JText/start_JText.sh	System startup script for the connector. This script is called from the generic connector manager script.
ProductDir/repository/JText/	Contains the MO_JText_Default.txt file.
ProductDir/connectors/JText/Dependencies/	Contains the license agreement., README files, and third party dependency files for FTP and NetComponents
ProductDir/connectors/messages/	Contains the JTextConnector.txt file.

Linux

All product pathnames are relative to the directory where the product is installed on your system. The base install directory (*ProductDir*) as shipped for Linux is `/IBM/WebSphereItemSync`.

Table 4. Installed file structure for the JText adapter on Linux

Directory	Description
ProductDir/bin/Data/App/	Contains the JText file.
ProductDir/connectors/JText	Contains the adapter application-specific component class file CWEJText.jar and the adapter start file start_JText.sh, and a ReadMe.htm informational file.
ProductDir/connectors/JText/Dependencies/	Contains the license agreement, ReadMe.htm files and Netcomponents subdirectory.
ProductDir/connectors/JText/Dependencies/NetComponents/1.3.8/src/java/com/oroinc/net/ftp/	Contains dependency files for FTP functionality, including the following open source files: • .java • FTPFileListParser.java • FTPClient.java
ProductDir/connectors/messages	Contains the JTextConnector.txt file.
ProductDir/repository	Contains the MO_JText_Default.txt file.

For more information on installing the connector component, refer to *Installing WebSphere Business Integration Express and Express Plus for Item Synchronization*.

Verifying installation of the data handler

In addition to the files loaded specifically for the JText Adapter 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 5 describes the file structure used by the data handlers.

Table 5. Data handler files

Subdirectory of %*ProductDir*% or $*ProdDir*	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.

Configuring the connector

You configure connector properties from Connector Configurator Express. Regardless of the integration broker, 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.

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.
- Which mode JText process in, binary or text.
- 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.
- 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 23.

Standard configuration properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors," on page 61 for documentation of these properties.

Table 6 provides information specific to this connector about configuration properties in Appendix A.

Table 6. Property Information Specific to This Connector

Property	Note
`CharacterEncoding`	Because this connector is Java-based, it does not use this property.
`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 7 lists the connector-specific configuration properties for the connector. See the sections that follow for explanations of the properties.

Table 7. 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 35.

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 56.

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 53.

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 32. For more information about the output log, see "Specifying request processing" on page 36. For information on returning the generated file's name, see "Returning a file's name" on page 38.

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 52.

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 Express 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 49.
- 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 50.
- 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 *User Guide for WebSphere Business Integration Express and Express Plus for Item Synchronization*.

Chapter 3. Using JText connector meta-objects

A **meta-object** is a 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 which 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 or byte array. 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 data 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*. Topics included in this chapter include:

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 50.

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 or byte arrays.

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.

Figure 5 shows the hierarchical structure for the default JText meta-objects and each attribute name and type.

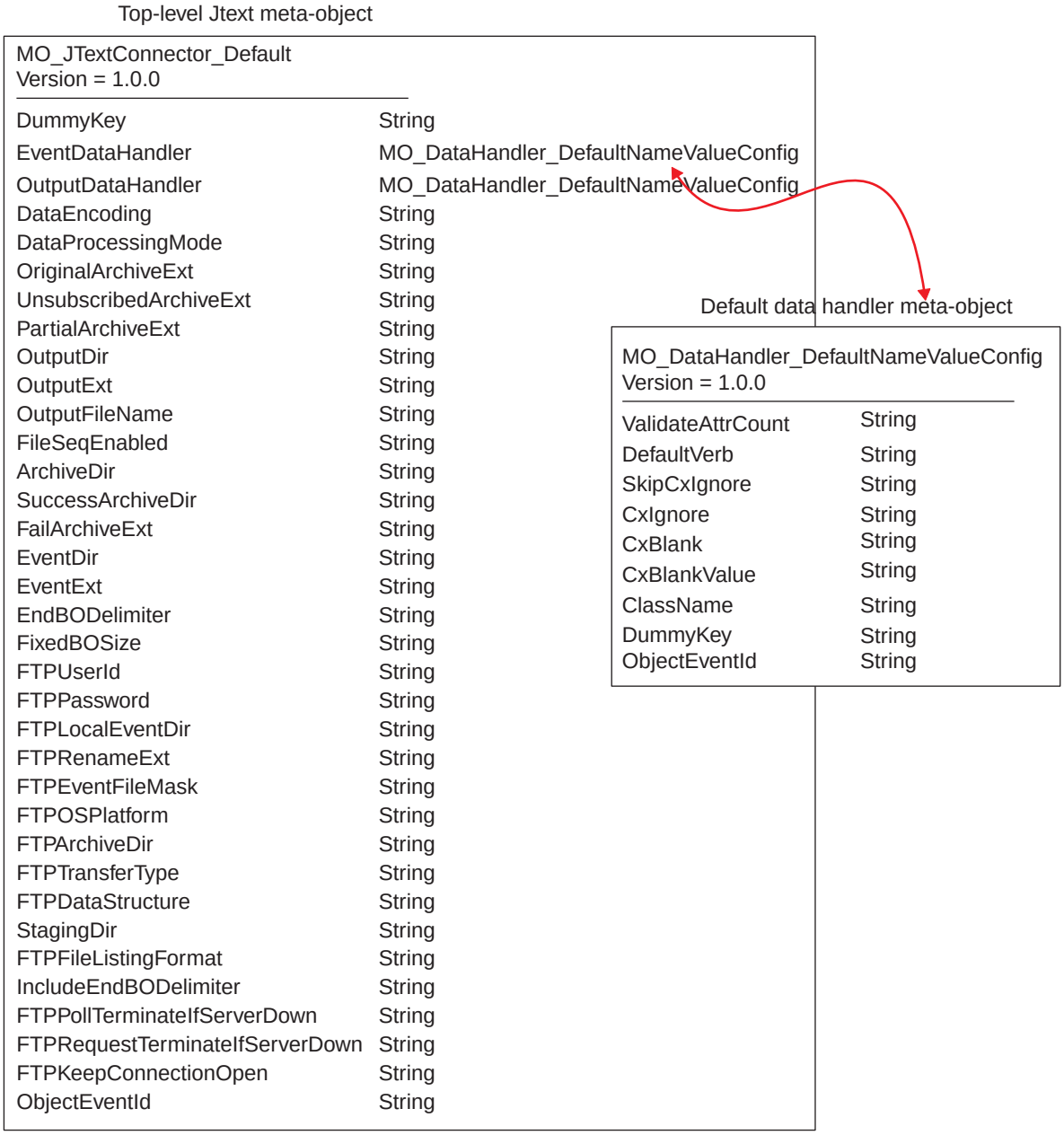

Figure 5. Hierarchical Structure of the JText meta-object

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 retrieves a 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 8 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 8. Attributes in the MO_JTextConnector_Default meta-object definition

Attribute name	Description
ArchiveDir	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.
DataProcessing Mode	This attribute provides new flexibility for reading and writing binary files. When set to Binary, this MO property enables JText to read and write binary files from the file system while calling the appropriate data handler interface for BO to byte array and vice versa transformations. The traditional setting for this is Text. In Text mode, the BO to String and vice versa data handler interface is used. When the property is not set, it defaults to Text. Binary mode should only be used with a data handler that appropriately implements the getBO(byte[]) and getByteArrayFromBO() methods.
DummyKey	This attribute exists to satisfy the requirement that one attribute in every business object definition have the Key property enabled.

Table 8. Attributes in the MO_JTextConnector_Default meta-object definition (continued)

Attribute name	Description
`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" on page 40. If you do not provide a default value during configuration and the `DataProcessingMode` is set to `Text`, the property defaults to the following value: `<EndBO:BOName>`. When `DataProcessingMode` is binary, the property defaults to the following value: `FF01`.
`EndOfFileDelimiter`	When `DataProcessingMode` is binary, and `FTPDataStructure` is `Record`, both `EndBODelimiter` and `EndOfFileDelimiter` are used. This property is set to the hexadecimal byte that is used for the end of file marker in the record file. If it is not set, the default, `FF02`, is used.
`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 34.
	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 43. 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 value is:
	`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 40. **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 35. The delivered default value is `fail`.

Table 8. Attributes in the MO_JTextConnector_Default meta-object definition (continued)

Attribute name	Description
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 36. The delivered default value is true.
FixedBOSize	When present with a valid value, this meta-object property overrides the EndBODelimiter property, and provides users an alternative to the traditional delimiter based BO parsing.
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 filename 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 filename 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, adding 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 filename 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 35. There is no delivered default value for this attribute.
FTPDataStructure	This attribute is of type String. The user can specify the FTP data structure (either File or Record) to get or put files from or to the remote site. If nothing is specified, Jtext will use 'File as default value
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 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 46. There is no delivered default value.

Table 8. Attributes in the MO_JTextConnector_Default meta-object definition (continued)

Attribute name	Description
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`.
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 44. 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 43. 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 44. 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.

Table 8. Attributes in the MO_JTextConnector_Default meta-object definition (continued)

Attribute name	Description
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 46. There is no delivered default value.
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 .
FTPTransferType	This JText meta-object property is used during both event and request processing. The possible values for this property are Binary and ASCII. The property dictates the transfer type JText will use when remotely placing or retrieving files from an FTP server. When the property does not exist, the adapter behavior defaults to Binary.
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 44. 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 35. 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 8. 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 47. 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 value is: 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 32. **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 35. The delivered default value is partial.

Table 8. 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 the StagingDir 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 35. 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 35. The delivered default value is unsub.

Note: AttributesFTPTransferType, FTPDataStructure, DataProcessingMode, EndOfFileDelimiter, and FixedBOSizeare not part of the JText meta-object as delivered. To use these attributes, they need to be explicitly added to the meta-object and their default values must be set.

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.

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 36.

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 Connector Configurator Express 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:

- "Local archive filenames"
- "Configuring local archiving" on page 36

Local 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 local archiving

To configure the connector for archiving, follow these steps:

1. Configure the following attributes of the MO_JTextConnector_Default meta-object:
   - ArchiveDir—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 Express 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

Important: In addition to creating the dynamic child meta-object to enable the connector to generate or return the output filename, if you are using InterChange Server Express as the integration broker, you must also modify your maps or collaboration logic to insert into the dynamic child meta-object's In`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.

For information about how the connector processes the meta-object, see "Request processing" on page 9.

For information on configuring the connector to use a dynamically generated output filename, see "Configuring the output file" on page 38.

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 38. For information on the file that stores the sequence numbers, see "OutputLog" on page 20.

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. For more information, see "Configuring the output file" on page 38.
   - `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 9 on page 39.
   - `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 9 on page 39.
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 37.
- Use a dynamic child meta-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: In addition to creating the dynamic child meta-object to enable the connector to generate or return the output filename, if you are using InterChange Server Express as the integration broker, you must also modify your maps or collaboration logic to insert into the dynamic child meta-object's In`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 `FTPArchiveDir` and `FTPRenameExt` attributes. These attributes enable you to rename and move the remotely archived file independently of the success of the processing.

For further information see "Specifying remote archiving" on page 44

Configuring the output file

Table 9 on page 39 illustrates the possible configuration options for the output file:

Table 9. 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	
	`AppSpecificInfo` (at business-object level)	`cw_mo_JTextConfig = DynChildMOName`
	For dynamic child meta-object: `OutFileName`	*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	
	`AppSpecificInfo` (at business-object level)	`cw_mo_JTextConfig = DynChildMOName`
	For dynamic child meta-object:`OutFileName`	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	
	`AppSpecificInfo` (at business-object level)	`cw_mo_JTextConfig = DynChildMOName`
	`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`
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 32.	`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`

Table 9. Specifying output files (continued)

Desired output condition	Attributes/property requiring configuration	Attribute/property value
	`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 `AppSpecificInfo` (at business-object level)	`cw_mo_JTextConfig = DynChildMOName`
	`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 50.
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

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:

Using EndBODelimiter parsing method

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*>.

Important: If DataProcessingMode is set to binary and if there is no value specified for EndBODelimiter, JText will set the default EndBODelimiter to FF01 (2 bytes) and EndOfFileDelimiter to FF02 (2 bytes).

Table 10 illustrates delimiter options.

Table 10. 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 EOLs as there are new lines between business object strings. • Specify a custom delimiter in conjunction with EOLs. 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 EOLs as there are new lines between business object strings. • Specify a user-specified delimiter in conjunction with EOLs 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	
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 for an EndBODelimiter: 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 50.

  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 50.

Using FixedBOSize parsing method

This meta-object property is only valid in the following instances:

1. When performing event processing.
2. When DataProcessingMode is set to Binary.

When present with a valid value, this meta-object property overrides the EndBODelimiter property, and provides users an alternative to the traditional delimiter based business object parsing. This property enables the connector to correlate a fixed number of bytes with a single business object. For example, if a file consisted of 300 bytes, and the FixedBOSize property was set to 100, the JText Adapter would convert these three 100 byte length packets through a binary enabled data handler and send them to the InterChange Server Express.

If both FixedBOSize and EndBODelimiter have a value set, then Jtext will take FixedBOSize for file parsing and it will ignore EndBODelimiter.

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 47
- "Notes on configuring the connector for FTP transfer" on page 49

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 44
- "Specifying the local directory" on page 44
- "Specifying remote archiving" on page 44
- "Specifying remote polling" on page 45

- "How the connector processes events from a remote site" on page 46
- "Identifying files on a mainframe: Optional configuration" on page 46
- "Summary of configuration operations for event processing" on page 47

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.companyA.com
```

URL with optional username and port number values:

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

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

```
ftp://companyA:admin@ftp.companyA.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 filename and move it to the remote 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 filename, adding a timestamp and ignoring the FTPRenameExt, then move it to 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 filename with the value specified in FTPRenameExt, and move it to the directory specified in the EventDir attribute.

File naming with timestamping for remote FTP servers: Support for host file systems (MVS) using Sequential datasets has been enhanced by providing for time stamping to avoid duplicate file names. MVS doesn't support special characters, such as "_", in a dataset or recordset name. On Windows, we use a time stamp in the original filename while archiving the file This avoids duplicate filenames in an archive folder, thereby preventing the overwriting of an existing file.

We use the following format for MVS systems to overcome this limitation:
Event File: Test.in
Archived file: Test.TSyyyyMM.TSDDHHMM.TSSsSss
Where: yyyy -- year
MM -- month
DD -- date
HH -- hour
MM -- minutes
Ss -- seconds
Sss -- milliseconds
On MVS platforms the dataset or recordset separator is "." (dot) and maximum number of '.' (dots) allowed in a dataset or recordset is 6 (six) The dataset or recordset name must not exceed 8 characters per "." (dot) and the total number of characters must not exceed 44 characters. Here is an example of a file name in this format:
FTPRenameExt -- ARCHIVE
Archived File -- (SAMPLE).ARCHIVE.TS200304.TS290535.TS42234

Note: The JText adapter does not support PDS in MVS as the members of PDS can not be renamed with time stamps while archiving.

Specifying remote polling: You can use the "FTPPollFrequency" on page 19 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 52.

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 6 illustrates local and remote event processing.

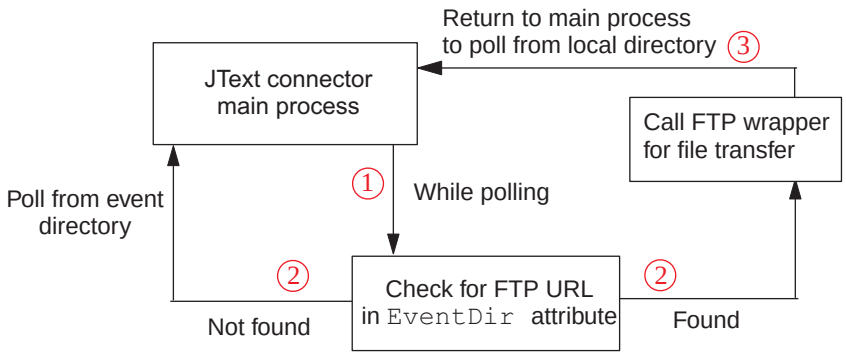

Figure 6. Local and remote event notification operation

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 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 several possible formats for this attribute:

```
ACT.Z1UC.*
ACT.*.INPT
*.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 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"
- "How the connector processes service call requests to a remote site" on page 48
- "Summary of configuration operations for request processing" on page 48

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.companyA.com
```

URL with optional username and port number values:

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

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

```
ftp://companyA:admin@ftp.companyA.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 7 illustrates local and remote request processing.

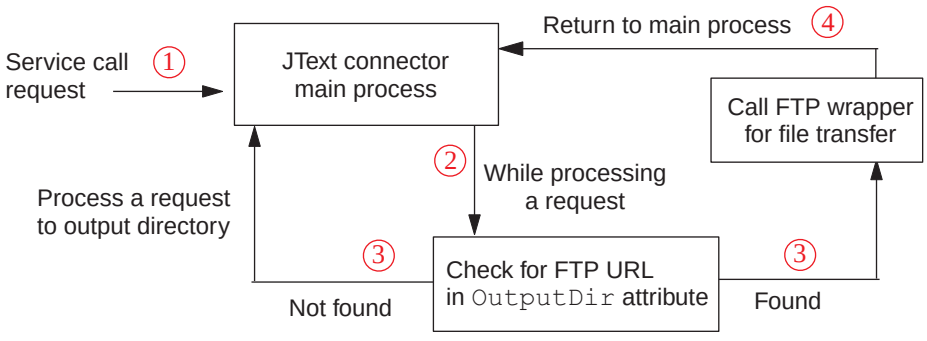

Figure 7. Local and remote request operations

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.

Specifying a data handler

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

1. Determine the 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 or byte array 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 or byte array conversion).

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 Express to copy the meta-objects one at a time.

     Important: Before you can manipulate a business object definition in Business Object Designer Express, 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 Express, 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 Express to copy business object definitions into the repository, select the Save To Server submenu from the File menu. Alternatively, for InterChange Server Express, 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 -s`*`ServerName`* `-i`*`FileName`*
5. Refresh the integration broker's administration utility to verify the new business objects.
6. 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 WebSphere MQ messaging queue for the new connector.
8. Restart the integration broker.

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.

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.

For InterChange Server Express, 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 Express 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 56.

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 11. 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 Linux:

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

or the following on OS400:

```
/QIBM/UserData/WebBIICS/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).

Appendix A. Standard configuration properties for connectors

This appendix describes the standard configuration properties for the connector component of the adapters in WebSphere Business Integration Express for Item Synchronization, running on WebSphere InterChange Server Express.

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

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

Configuring standard connector properties

Adapter connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

Using Connector Configurator Express

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

Setting and updating property values

The default length of a property field is 255 characters.

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

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

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

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

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

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

Summary of standard properties

Table 12 provides a quick reference to the standard connector configuration properties.

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

Table 12. Summary of standard configuration properties

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

Table 12. Summary of standard configuration properties (continued)

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

Table 12. Summary of standard configuration properties (continued)

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

Table 12. Summary of standard configuration properties (continued)

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

Standard configuration properties

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

AdminInQueue

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

The default value is `CONNECTORNAME/ADMININQUEUE`.

AdminOutQueue

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

The default value is `CONNECTORNAME/ADMINOUTQUEUE`.

AgentConnections

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

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

AgentTraceLevel

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

ApplicationName

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

CharacterEncoding

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

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

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

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

The default value is ascii.

ConcurrentEventTriggeredFlows

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

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

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

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

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

ContainerManagedEvents

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

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

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

- PollQuantity = 1 to 500
- SourceQueue = SOURCEQUEUE

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

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

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

ControllerStoreAndForwardMode

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

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

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

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

The default is true.

ControllerTraceLevel

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

DeliveryQueue

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

The default value is DELIVERYQUEUE.

DeliveryTransport

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

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

WebSphere MQ and IDL

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

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

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

JMS

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

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

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

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

DuplicateEventElimination

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

This property can also be set to `false`.

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

FaultQueue

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

The default value is `CONNECTORNAME/FAULTQUEUE`.

JvmMaxHeapSize

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

The default value is 128m.

JvmMaxNativeStackSize

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

The default value is 128k.

JvmMinHeapSize

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

The default value is 1m.

jms.FactoryClassName

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

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

jms.MessageBrokerName

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

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

ListenerConcurrency

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

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

Locale

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

A locale name has the following format:

ll_TT.codeset

where:

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

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

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

LogAtInterchangeEnd

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

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

MaxEventCapacity

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

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

MessageFileName

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

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

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

MonitorQueue

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

The default value is CONNECTORNAME/MONITORQUEUE

OADAutoRestartAgent

The Repository Directory must be set to <REMOTE>.

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

The default value is false.

OADMaxNumRetry

The Repository Directory must be set to <REMOTE>.

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

The default value is 1000.

OADRetryTimeInterval

The Repository Directory must be set to <REMOTE>.

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

The default is 10.

PollEndTime

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

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

PollFrequency

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

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

The default is 10000.

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

PollQuantity

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

PollStartTime

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

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

RequestQueue

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

The default value is `REQUESTQUEUE`.

RepositoryDirectory

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

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

ResponseQueue

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

RestartCount

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

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

RestartRetryCount

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

The default is `3`.

RestartRetryInterval

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

The default is 1.

SourceQueue

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see "ContainerManagedEvents" on page 66.

The default value is `SOURCEQUEUE`.

SynchronousRequestQueue

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

SynchronousResponseQueue

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

SynchronousRequestTimeout

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

The default value is 0.

WireFormat

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

Appendix B. Connector Configurator Express

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

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

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

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

You use Connector Configurator Express to:

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

The topics covered in this appendix are:

Overview of Connector Configurator Express

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

You use Connector Configurator Express to:

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

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

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

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

Starting Connector Configurator Express

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

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

Running Configurator Express in stand-alone mode

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

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

If you are creating a configuration file, you may prefer to run Connector Configurator Express independently to generate the file, and then connect to System Manager to save it in an InterChange Server Express project (see "Completing a configuration file" on page 81.)

Running Configurator Express from System Manager

You can run Connector Configurator Express from System Manager.

To run Connector Configurator Express:

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

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 77.
- To use an existing file, simply modify an existing template and save it under the new name.

Note: Connector-specific templates are provided for the iSoft, JText, and e-Mail connectors only. If you are configuring one of these connectors, see the *Quick Start Guide*, or skip this section and go to "Creating a new configuration file" on page 79.

Creating a new template

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

To create a template:

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

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

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

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

Specifying general characteristics

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

- **Edit properties**

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

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

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

- Flags

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

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

Specifying values

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

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

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

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

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

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

Setting dependencies

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

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

< (less than)

>= (greater than or equal to)

<=(less than or equal to)

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

Creating a new configuration file

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

Creating a configuration file from a connector-specific template

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

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

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

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

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

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

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

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

Using an existing file

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

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

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

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

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

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

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

Opening an existing file from System Manager

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

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

To edit an existing configuration file:

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

Completing a configuration file

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

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

- "Setting standard connector properties"
- "Setting connector-specific configuration properties" on page 82
- "Specifying supported business object definitions" on page 82
- "Associated maps" on page 84
- "Setting trace/log file values" on page 85
- "Configuring messaging" on page 85

Note: For connectors that use JMS messaging, an additional category may display, for special configuration of data handlers that convert the data to business objects. For further information, see "Data handlers" on page 85.

Setting the configuration file properties

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

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

Setting standard connector properties

To change the value of a standard property:

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

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

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

Setting connector-specific configuration properties

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

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property or **Add Child** to add a child property to a property.
2. Enter a value for the property or child property.
   - To set values for connector-specific property values for your connector, **see the connector-specific properties section of this guide**.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for "Setting standard connector properties" on page 81.

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

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

Encryption for connector properties

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

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

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

Update method

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

Specifying supported business object definitions

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

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

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

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

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

Business object name

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

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

To delete a business object from the supported list:

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

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

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

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

Maximum transaction level

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

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

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

Associated maps

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

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

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

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

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

- **Business Object Name**

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

- **Associated Maps**

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

- **Explicit**

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

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

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

To explicitly bind a map:

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

Configuring messaging

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

Setting trace/log file values

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

To change the logging and tracing values:

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

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

Data handlers

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

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

Saving your configuration file

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

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

Completing the configuration

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

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

Notices

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

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

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

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

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

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

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

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

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

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

Burlingame, CA 94010
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

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

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

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

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

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

Adapter for JText includes software developed by the Eclipse Project. (http://www.eclipse.org/)

WebSphere Business Integration Express for Item Synchronization V4.3.1, WebSphere Business Integration Express Plus for Item Synchronization V.4.3.1

Printed in USA