WebSphere Business Integration Server
Express and Express Plus

Adapter for SWIFT User Guide

Version 4.3.1

WebSphere Business Integration Server
Express and Express Plus

Adapter for SWIFT User Guide

Version 4.3.1

Note!

Before using this information and the product it supports, read the information in Appendix D, “Notices,” on page 123.

30July2004

This edition of this document applies to IBM WebSphere Business Integration Server Express, version 4.3.1, IBM WebSphere Business Integration Server Express Plus, 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 Business Integration Server Express and IBM(R) WebSphere Business Integration Server Express Plus are made up of the following components: InterChange Server Express, the associated Toolset Express, CollaborationFoundation, and a set of software integration adapters. The tools in Toolset Express help you to create, modify, and manage business processes. You can choose from among the prepackaged adapters for your business processes that span applications. The standard processes template--CollaborationFoundation--allows you to quickly create customized processes.

This document describes configuration, business object development, and troubleshooting for the WebSphere Business Integration Server Express Adapter for SWIFT.

Except where noted, all the information in this guide applies to both IBM WebSphere Business Integration Server Express and IBM WebSphere Business Integration Server Express Plus. The term WebSphere Business Integration Server Express and its variants refer to both products.

Audience

This document is for consultants, developers, and system administrators who support and manage the WebSphereBusiness Integration Server Express Product at customer sites.

Prerequisites for this document

Users of this document should be familiar with

- the WebSphere Business Integration Server Express system
- business object development
- the WebSphere MQ application
- the SWIFT product suite and protocol

Related documents

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

You can download, install, and view the documentation at the following site: http://www.ibm.com/websphere/wbiserverexpress/infocenter

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

Typographic conventions

This document uses the following conventions:

`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.
ProductDir	Represents the directory where the IBM WebSphere Business Integration Server Express for Adapters product is installed. The defaults for each platform are as follows: Windows: IBM\WebSphereServer OS/400: /QIBM/ProdData/WBIServer43/product Linux: /home/${username}/IBM/WebSphereServer
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
`[ ]`	In a syntax line, square brackets surround an optional parameter.
`...`	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, `option[,...]` means that you can enter multiple, comma-separated options.
`< >`	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><`*`connector_name`*`>tmp.log`.
`/, \`	In this document, backslashes (\) are used as the convention for directory paths for Windows. OS/400 and Linux use forward slashes (/) for directory paths. All WebSphere Business Integration Server Express system product path names are relative to the directory where the product is installed on your system.
`%text%` and `$`*`text`*	Text within percent (%) signs indicates the value of the Windows(TM) `text` system variable or user variable. The equivalent notation in a UNIX environment is `$`*`text`*, indicating the value of the *`text`* UNIX environment variable.

New in this release

New in release 4.3.1

This release adds support for the following operating systems:

- Microsoft Windows 2003
- IBM OS/400 V5R2, V5R3
- Red Hat Enterprise Linux AS 3.0, Update 1
- SuSE Linux Enterprise Server 8.1 with SP3

New in release 4.3

This is the first release of this guide.

Chapter 1. Overview

The connector for SWIFT is a runtime component of the WebSphere Business Integration Server Express and Express Plus Adapter for SWIFT. The connector allows the WebSphere InterChange Server Express integration broker to exchange business objects with SWIFT-enabled business processes.

Note: Throughout this document, SWIFT messages denote SWIFT FIN messages unless otherwise explicitly noted.

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

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

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

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

All WebSphere Business Integration Server Express adapters operate with WebSphere InterChange Server Express as their integration broker.

The connector for SWIFT allows InterChange Server Express to exchange business objects with applications that send or receive data in the form of SWIFT messages.

Adapter environment

Before installing, configuring, and using the adapter, you must understand its environment requirements.

Adapter platforms

The adapter runs on the following platforms:

- Microsoft Windows 2000
- Microsoft Windows 2003
- IBM OS/400 V5R2, V5R3
- Red Hat Enterprise Linux AS 3.0, Update 1
- SuSE Linux Enterprise Server 8.1 with SP3

Adapter standards

The adapter supports the following standard.

SWIFTAlliance Access

The SWIFTAlliance Access gateway is a window through which SWIFT messages flow to and from remote financial applications over IP. The connector supports SWIFTAlliance Access 5.0.

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 run-time 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 Server Express system are written in Java. Therefore, when data is transferred between most WebSphere Business Integration Server Express 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 configuration properties, see Appendix A, “Standard configuration properties for connectors,” on page 87.

Connector architecture

The connector allows WebSphere business processes to asynchronously exchange business objects with applications that issue or receive SWIFT messages when changes to data occur. (The connector also supports synchronous acknowledgment.)

SWIFT stands for Society for Worldwide Interbank Financial Telecommunications. It is a United Nations-sanctioned International Standards Organization (ISO) for the creation and maintenance of financial messaging standards.

As shown in Figure 1, the connector interacts with several components (WebSphere components are shown in **bold**) whose collective purpose is to bridge the world of WebSphere business objects with that of SWIFT messages. For IBM WebSphere Business Integration Server Express and Express Plus, the integrator broker in Figure 1, is WebSphere InterChange Server Express.

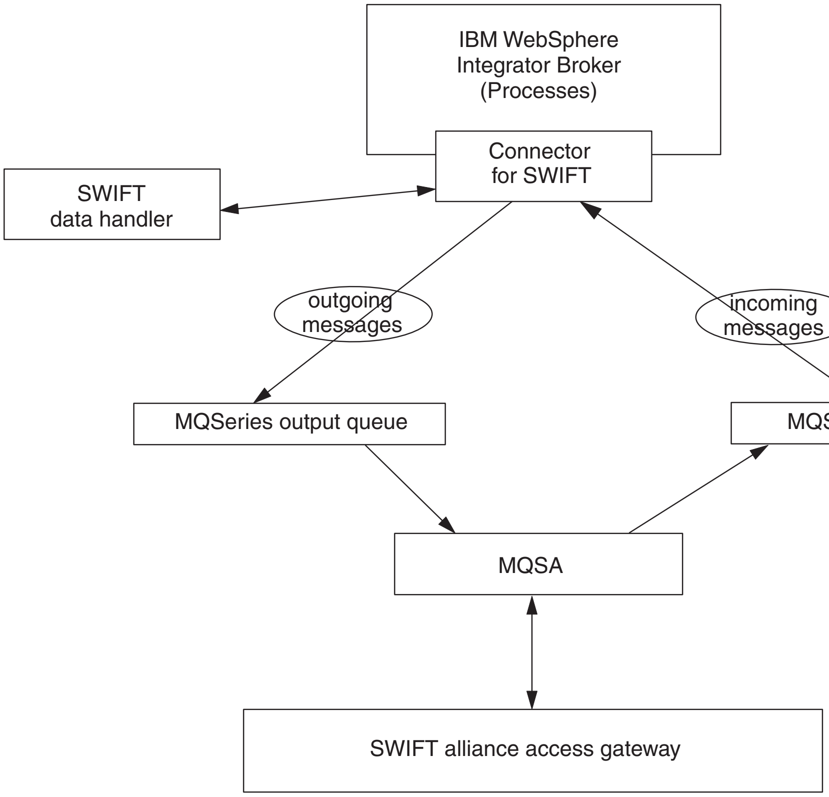


Figure 1. Connector for SWIFT architecture

The SWIFT environment is made up of various components that are described below.

Connector for SWIFT

The connector for SWIFT is metadata-driven. Message routing and format conversion are initiated by an event polling technique. The connector retrieves WebSphere MQ messages from queues, calls the SWIFT data handler to convert messages to their corresponding business objects, and then delivers the objects to the corresponding business processes. In the opposite direction, the connector receives business objects from WebSphere InterChange Server Express, converts them into SWIFT messages using the same data handler, and then delivers the messages to a WebSphere MQ queue.

The type of business object and verb used in processing a message are based on the metadata in the Format field of the WebSphere MQ message header. You construct a meta-object to store the business object name and verb to associate with the WebSphere MQ message header Format field text.

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

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

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

SWIFT data handler

The connector calls the SWIFT data handler to convert business objects into SWIFT messages and vice versa. For more on the SWIFT data handler, see Chapter 4, "SWIFT Data Handler," on page 81.

WebSphere MQ

The connector for SWIFT uses an MQ implementation of the JavaTM Message Service (JMS), an API for accessing enterprise-messaging systems. This makes possible interaction with incoming and outgoing WebSphere MQ event queues.

MQSA

The WebSphere MQ event queues exchange messages with the WebSphere MQ Interface for SWIFTAlliance (MQSA). The MQSA software integrates WebSphere MQ messaging capabilities with SWIFT message types, performing delivery, acknowledgement, queue management, timestamping, and other functions.

SWIFTAlliance Access

The SWIFTAlliance Access gateway is a window through which SWIFT messages flow to and from remote financial applications over IP. The connector supports SWIFTAlliance Access 5.0.

Application-connector communication method

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

Message request

Figure 2 illustrates a message request communication.

1. The connector framework receives a business object representing an ISO 15022 SWIFT message from an integration broker. For IBM WebSphere Business Integration Server Express and Express Plus, the integration broker in Figure 2 is WebSphere InterChange Server Express.
2. The connector passes the business object to the data handler.
3. The data handler converts the ISO 15022 business object into an ISO 15022-compliant SWIFT message.

4. The connector dispatches the ISO 15022 SWIFT message to the WebSphere MQ output queue.
5. The JMS layer makes the appropriate calls to open a queue session and routes the message to the MQSA, which issues the message to the SWIFT Alliance Gateway.

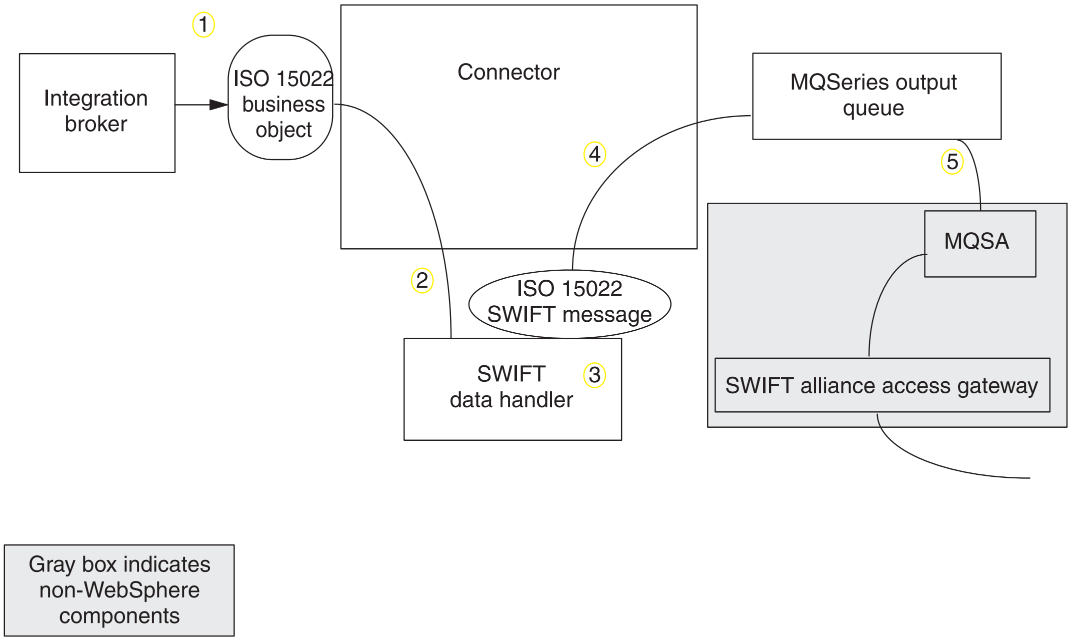


Figure 2. Application-connector communication method: Message request

Event delivery

Figure 3 illustrates the message return communication.

1. The polling method retrieves the next applicable ISO 15022 SWIFT message from the WebSphere MQ input queue.
2. The message is staged in the in-progress queue, where it remains until processing is complete.
3. The data handler converts the message into an ISO 15022 business object.
4. The SWIFT data handler receives the ISO 15022 business object and sets the verb in it to the default verb specified in the data handler-specific meta-object.
5. The connector then determines whether the business object is subscribed to by the integration broker. If so, the connector framework delivers the business object to the integration broker, and the message is removed from the in-progress queue. For IBM WebSphere Business Integration Server Express and Express Plus, the integration broker in Figure 3 is WebSphere InterChange Server Express.

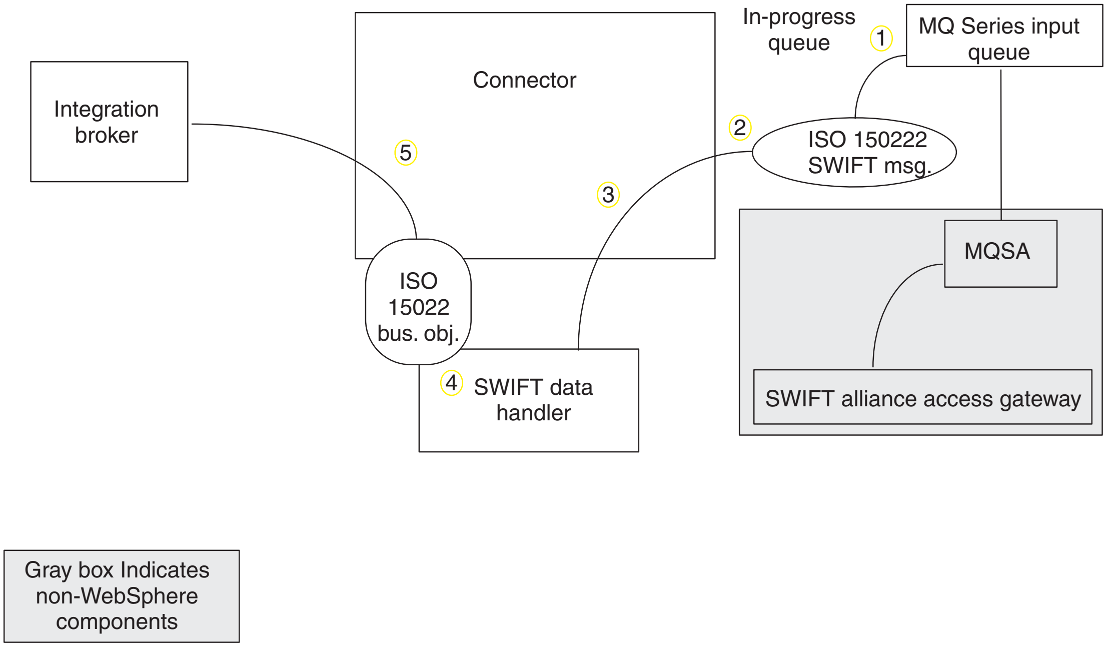


Figure 3. Application-connector communication method: Event delivery

Event handling

For event notification, the connector detects an event written to a queue by an application rather than by a database trigger. An event occurs when SWIFTAlliance generates SWIFT messages and stores them on the WebSphere MQ queue.

Retrieval

The connector uses a polling method to poll the WebSphere MQ input queue at regular intervals for messages. When the connector finds a message, it retrieves it from the WebSphere MQ input queue and examines it to determine its format. If the format has been defined in the connector's static or child meta-objects, the connector uses the data handler to generate an appropriate business object with a verb.

In-progress queue

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

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

Synchronous acknowledgment

To support applications that require feedback on the requests they issue, the connector for SWIFT can issue report messages to the applications detailing the outcome of their requests once they have been processed.

To achieve this, the connector posts the business data for such requests synchronously to InterChange Server Express. If the business object is successfully processed, the connector sends a report back to the requesting application including the return code fromInterChange Server Express and any business object changes. If the connector or InterChange Server Express fails to process the business object, the connector sends a report containing the appropriate error code and error message.

In either case, an application that sends a request to the connector for SWIFT is notified of its outcome.

If the connector for SWIFT receives any messages requesting positive or negative acknowledgment reports (PAN or NAN), it posts the content of the message synchronously to InterChange Server Express and then incorporates the return code and modified business data in to a report message that is sent back to the requesting application.

Table 1 shows the required structure of messages sent to the connector to be processed synchronously.

Table 1. Required structure of synchronous WebSphere MQ messages

MQMD Field (message descriptor)	Description	Supported values (multiple values should be OR'd)
`MessageType`	Message type	`DATAGRAM`
`Report`	Options for report message requested	You can specify one or both of the following: • `MQRO_PAN` The connector sends a report message if the business object can be successfully processed. • `MQRO_NAN`The connector sends a report message if an error occurred while processing the business object. You can specify one of the following to control how the correlation ID of the report message is to be set: • `MQRO_COPY_MSG_ID_TO_CORREL_ID`The connector copies the message ID of the request message to the correlation ID of the report. This is the default action. • `MQRO_PASS_CORREL_ID`The connector copies the correlation ID of the request message to the correlation ID of the report.
`ReplyToQueue`	Name of reply queue	The name of the queue to which the report message should be sent.
`ReplyToQueueManager`	Name of queue manager	The name of the queue manager to which the report message should be sent.

Table 1. Required structure of synchronous WebSphere MQ messages (continued)

MQMD Field (message descriptor)	Description	Supported values (multiple values should be OR'd)
Message Body		A serialized business object in a format compatible with the data handler configured for the connector.

Upon receipt of a message as described in Table 1, the connector:

1. Reconstructs the business object in the message body using the configured data handler.
2. Looks up the business process specified for the business object and verb in the static metadata object.
3. Posts the business object synchronously to the specified process.
4. Generates a report encapsulating the result of the processing and any business object changes or error messages.
5. Sends the report to the queue specified in the `replyToQueue` and `replyToQueueManager` fields of the request.

Table 2 shows the structure of the report that is sent to the requesting application from the connector.

Table 2. Structure of the report returned to the requesting application

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

Recovery

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

Fail on startup

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

Reprocess

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

Ignore

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

Log error

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

Archiving

If the connector property `ArchiveQueue` is specified and identifies a valid queue, the connector places copies of all successfully processed messages in the archive queue. If `ArchiveQueue` is undefined, messages are discarded after processing.

Guaranteed event delivery

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

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

You can configure the guaranteed-event-delivery feature for a JMS-enabled connector with, or without, a JMS event store. To configure the connector for guaranteed event delivery, see the *System Implementation Guide.*

If the connector framework cannot deliver the business object to InterChange Server Express, then the object is placed on a FaultQueue (instead of

UnsubscribedQueue and ErrorQueue) and generates a status indicator and a description of the problem. FaultQueue messages are written in MQRFH2 format.

Business object requests

Business object requests are processed when InterChange Server Express issues a business object.

Message processing

The connector processes business objects passed to it by InterChange Server Express based on the verb for each business object. The connector uses business object handlers to process the business objects that the connector supports.The business object handlers contain methods that interact with an application and that transform business object requests into application operations.

The connector supports the following business object verbs:

- Create
- Retrieve

Create

Processing of business objects with create depends on whether the objects are issued asynchronously or synchronously.

Asynchronous delivery

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

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

Synchronous acknowledgment

If a `replyToQueue` has been defined in the connector properties and a `responseTimeout` exists in the conversion properties for the business object, the connector issues a request in synchronous mode. The connector then waits for a response to verify that appropriate action was taken by the receiving application.

For WebSphere MQ, the connector initially issues a message with a header as shown in Table 3.

Table 3. Request Message Descriptor Header (MQMD)

Field	Description	Value
Format	Format name	Output format as defined in the conversion properties and truncated to 8 characters to meet IBM requirements (example: `MQSTR`).
MessageType	Message type	`MQMT_DATAGRAM`[a]

Table 3. Request Message Descriptor Header (MQMD) (continued)

Field	Description	Value
Report	Options for report message requested.	When a response message is expected, this field is populated as follows: `MQRO_PAN`[a] to indicate that a positive-action report is required if processing is successful. `MQRO_NAN`[a] to indicate that a negative-action report is required if processing fails. `MQRO_COPY_MSG_ID_TO_CORREL_ID`[a] to indicate that the correlation ID of the report generated should equal the message ID of the request originally issued.
ReplyToQueue	Name of reply queue	When a response message is expected, this field is populated with the value of connector property `ReplyToQueue`.
Persistence	Message persistence	`MQPER_PERSISTENT`[a]
Expiry	Message lifetime	`MQEI_UNLIMITED`[a]

[a] Indicates constant defined by IBM.

The message header described in Table 3 is followed by the message body. The message body is a business object that has been serialized using the data handler.

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

When an application receives a synchronous request from the connector, it processes the business object and issues a report message as described in Table 4, Table 5, and Table 6.

Table 4. Response Message Descriptor Header (MQMD)

Field	Description	Value
Format	Format name	Input format of `busObj` as defined in the conversion properties.
MessageType	Message type	`MQMT_REPORT`[a]

[a] Indicates constant defined by IBM.

Table 5. Population of response message

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

Table 6. Feedback codes and response values

WebSphere MQ feedback code	Equivalent WebSphere Business Integration Server Express response[a]
`MQFB_PAN` or `MQFB_APPL_FIRST`	SUCCESS
`MQFB_NAN` or `MQFB_APPL_FIRST + 1`	FAIL
`MQFB_APPL_FIRST + 2`	VALCHANGE
`MQFB_APPL_FIRST + 3`	VALDUPES
`MQFB_APPL_FIRST + 4`	MULTIPLE_HITS
`MQFB_APPL_FIRST + 5`	Not applicable

Table 6. Feedback codes and response values (continued)

WebSphere MQ feedback code	Equivalent WebSphere Business Integration Server Express response[a]
MQFB_APPL_FIRST + 6	Not applicable
MQFB_APPL_FIRST + 7	UNABLE_TO_LOGIN
MQFB_APPL_FIRST + 8	APP_RESPONSE_TIMEOUT (results in immediate termination of connector)
MQFB_NONE	What the connector receives if no feedback code is specified in the response message

[a] See the *System Implementation Guide* for details.

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

Upon retrieval of a response message, the connector by default matches the correlationID of the response to the messageID of a request message. The connector then notifies the thread that issued the request. Depending on the feedback field of the response, the connector either expects a business object or an error message in the message body. If a business object was expected but the message body is not populated, the connector simply returns the same business object that was originally issued by InterChange Server Express for the Request operation. If an error message was expected but the message body is not populated, a generic error message is returned to InterChange Server Express along with the response code. However, you can also use a message selector to identify, filter and otherwise control how the adapter identifies the response message for a given request. This message selector capability is a JMS feature. It applies to synchronous request processing only and is described below.

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

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

response_selector=JMSCorrelationID LIKE *'selectorstring'*

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

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

In the above example, after issuing the request message, the adapter would monitor the ReplyToQueue for a response message with a correlationID equal to

″Oshkosh.″ The adapter would retrieve the first message that matches this message selector and then dispatch it as the response.

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

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

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

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

to identify the response message.

You can also specify multiple substitutions such as the following:

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

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

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

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

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

The connector would resolve this message selector as follows:

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

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

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

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

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

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

Creating custom feedback codes: You can extend the WebSphere MQ feedback codes to override default interpretations shown in Table 6 by specifying the connector property FeedbackCodeMappingMO. This property allows you to create a meta-object in which all WebSphere Business Integration Server Express system-specific return status values are mapped to the WebSphere MQ feedback codes. The return status assigned (using the meta-object) to a feedback code is passed to InterChange Server Express. For more information, see “FeedbackCodeMappingMO” on page 22.

Retrieve

Business objects with the Retrieve verb support synchronous delivery only. The connector processes business objects with this verb as it does for the synchronous delivery defined for create. However, when using a Retrieve verb, the `responseTimeout` and `replyToQueue` are required. Furthermore, the message body must be populated with a serialized business object to complete the transaction.

Table 7 shows the response messages for these verbs.

Table 7. Population of response message

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

Error handling

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

```
Message number

Message text
```

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

Application timeout

The error message ABON_APPRESPONSETIMEOUT is returned when:

- The connector cannot establish a connection to the JMS service provider during message retrieval.
- The connector successfully converts a business object to a message but cannot deliver it to the outgoing queue due to connection loss.

- The connector issues a message but times out waiting for a response from a business object whose conversion property `TimeoutFatal` is equal to `True`.
- The connector receives a response message with a return code equal to APP_RESPONSE_TIMEOUT or UNABLE_TO_LOGIN.

Unsubscribed business object

The connector delivers a message to the queue specified by the `UnsubscribedQueue` property if:

- The connector retrieves a message that is associated with an unsubscribed business object.
- The connector retrieves a message but cannot associate the text in the Format field with a business object name.

Note: If the `UnsubscribedQueue` is not defined, unsubscribed messages are discarded.

Data handler conversion

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

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

Tracing

Tracing is an optional debugging feature you can turn on to closely follow connector behavior. Trace messages, by default, are written to STDOUT. See the connector configuration properties in Chapter 2, “Installing and configuring the connector,” on page 17, for more on configuring trace messages. For more information on tracing, including how to enable and set it, see the *System Implementation Guide*.

What follows is recommended content for connector trace messages.

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

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

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

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

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

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

Chapter 2. Installing and configuring the connector

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

Overview of installation tasks

To install the connector for SWIFT, you must perform the following tasks.

Confirm adapter prerequisites

Before you install the adapter, confirm that all the environment prerequisites for installing and running the adapter are on your system. For details, see “Adapter environment” on page 1.

Install the integration broker

Installing the integration broker, a task that includes installing and starting WebSphere Business Integration Server Express, is described in the *WebSphere Business Integration Server Express Installation Guide for Windows* or *Linux* or *OS/400*.

Install the adapter for SWIFT and related files

For information on installing the adapter, refer to the *WebSphere Business Integration Server Express Installation Guide* for your platform, located in the WebSphere Business Integration Server Express InfoCenter at the following site:

http://www.ibm.com/websphere/wbiserverexpress/infocenter

Installed file structure

The following subsections describe the installed file structure on Windows, OS/400, and Linux systems.

Installed Windows files

Table 8 describes the Windows file structure used by the connector.

Note: If you are installing a Web release of this connector, see the Release Notes for installation instructions.

Table 8. Installed Windows file structure for the connector

Subdirectory of ProductDir	Description
connectors\SWIFT\CWSwift.jar	Connector jar file
connectors\SWIFT\start_SWIFT.bat	The startup file for the connector.
connectors\messages\SWIFTConnector.txt	Connector message file
repository\SWIFT\CN_SWIFT.txt	Connector definition
DataHandlers\CwDataHandler.jar	The SWIFT data handler
repository\DataHandlers\MO_DataHandler_SWIFT.txt	Meta-object for SWIFT data handler
repository\DataHandlers\MO_DataHandler_Default.txt	Data handler default object
connectors\SWIFT\samples\Sample_SWIFT_MO_Config.txt	Sample configuration object
connectors\SWIFT\samples\MO_SWIFT_MAPSUBSCRIPTIONS.txt	Mapping meta-object
connectors\SWIFT\samples\BO_Definitions\SWIFT_objects.txt	Business object definitions
connectors\SWIFT\samples\Map_Definitions\Map_objects.txt	Map definitions

For more information on installing the connector component, refer to the *WebSphere Business Integration Server Express Installation Guide for Windows*.

Installed OS/400 files

Table 9 describes the OS/400 file structure used by the connector.

Note: If you are installing a Web release of this connector, see the Release Notes for installation instructions.

Table 9. Installed OS/400 file structure for the connector

Subdirectory of ProductDir	Description
connectors/SWIFT/CWSwift.jar	Connector jar file
connectors/SWIFT/start_SWIFT.sh	The startup file for the connector.
connectors/messages/SWIFTConnector.txt	Connector message file
repository/SWIFT/CN_SWIFT.txt	Connector definition
DataHandlers/CwDataHandler.jar	The SWIFT data handler
repository/DataHandlers/MO_DataHandler_SWIFT.txt	Meta-object for SWIFT data handler
repository/DataHandlers/MO_DataHandler_Default.txt	Data handler default object
connectors/SWIFT/samples/Sample_SWIFT_MO_Config.txt	Sample configuration object
connectors/SWIFT/samples/MO_SWIFT_MAPSUBSCRIPTIONS.txt	Mapping meta-object
connectors/SWIFT/samples/BO_Definitions/SWIFT_objects.txt	Business object definitions
connectors/SWIFT/samples/Map_Definitions/Map_objects.txt	Map definitions

For more information on installing the connector component, refer to the *WebSphere Business Integration Server Express Installation Guide for OS/400*.

Installed Linux files

Table 10 describes the Linux file structure used by the connector.

Note: If you are installing a Web release of this connector, see the Release Notes for installation instructions.

Table 10. Installed Linux file structure for the connector

Subdirectory of ProductDir	Description
connectors/SWIFT/CWSwift.jar	Connector jar file
connectors/SWIFT/start_SWIFT.sh	The startup file for the connector. The script is called from the generic connector manager script. When you click Install from the Connector Configuration Express screen of System Manager, the installer creates a customized wrapper for this connector manager script. Use this customized wrapper only to start and stop the connector.
connectors/messages/SWIFTConnector.txt	Connector message file
repository/SWIFT/CN_SWIFT.txt	Connector definition
DataHandlers/CwDataHandler.jar	The SWIFT data handler
repository/DataHandlers/MO_DataHandler_SWIFT.txt	Meta-object for SWIFT data handler
repository/DataHandlers/MO_DataHandler_Default.txt	Data handler default object
connectors/SWIFT/samples/Sample_SWIFT_MO_Config.txt	Sample configuration object
connectors/SWIFT/samples/MO_SWIFT_MAPSUBSCRIPTIONS.txt	Mapping meta-object
connectors/SWIFT/samples/BO_Definitions/SWIFT_objects.txt	Business object definitions
connectors/SWIFT/samples/Map_Definitions/Map_objects.txt	Map definitions

For more information on installing the connector component, refer to the *WebSphere Business Integration Server Express Installation Guide for Linux*.

Connector configuration

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

You use Connector Configurator Express to configure connector properties:

- For a description of Connector Configurator Express and step-by-step procedures, see Appendix B, “Connector Configurator Express,” on page 101.
- For a description of standard connector properties, see “Standard connector properties” and then Appendix A, “Standard configuration properties for connectors,” on page 87.
- For a description of connector-specific properties, see “Connector-specific properties” on page 20.

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

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, “Standard configuration properties for connectors,” on page 87 for documentation of these properties.

Note: When you set configuration properties in Connector Configurator Express, the BrokerType property is set to InterChange Server Express. Properties relevant to InterChange Server Express appear in the Connector Configurator Express window.

Connector-specific properties

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

Note: Always check the values WebSphere MQ provides because they may be incorrect or unknown. If the provided values are incorrect, specify them explicitly.

Table 11 lists the connector-specific configuration properties for the connector for SWIFT. See the sections that follow for explanations of the properties.

Table 11. Connector-specific configuration properties

Name	Possible values	Default value	Required
"ApplicationPassword" on page 21	*Login password*		No
"ApplicationUserID" on page 21	*Login user ID*		No
"ArchiveQueue" on page 21	*Queue to which copies of successfully processed messages are sent*	`queue://CrossWorlds.QueueManager/MQCONN.ARCHIVE`	No
"Channel" on page 21	*MQ server connector channel*		Yes
"ConfigurationMetaObject" on page 21	*Name of configuration meta-object*		Yes
"DataHandlerClassName" on page 21	*Data handler class name*	`com.crossworlds.DataHandlers.swift.SwiftDataHandler`	No
"DataHandlerConfigMO" on page 21	*Data handler meta-object*	`MO_DataHandler_Default`	Yes
"DataHandlerMimeType" on page 22	*MIME type of file*	`swift`	No
"DefaultVerb" on page 22	*Any verb supported by the connector.*	`Create`	
"ErrorQueue" on page 22	*Queue for unprocessed messages*	`queue://crossworlds.Queue.manager/MQCONN.ERROR`	No
"FeedbackCodeMappingMO" on page 22	Feedback code meta-object		No
"HostName" on page 23	*WebSphere MQ server*		No
"InDoubtEvents" on page 23	`FailOnStartup Reprocess Ignore LogError`	`Reprocess`	No
"InputQueue" on page 23	*Poll queues*	`queue://CrossWorlds.QueueManager/MQCONN.IN`	Yes
"InProgressQueue" on page 23	*In-progress event queue*	`queue://CrossWorlds.QueueManager/MQCONN.IN_PROGRESS`	No
"PollQuantity" on page 24	*Number of messages to retrieve from each queue specified in the* `InputQueue` *property*	1	No
"Port" on page 24	*Port established for the WebSphere MQ listener*		No

Table 11. Connector-specific configuration properties (continued)

Name	Possible values	Default value	Required
"ReplyToQueue" on page 24	*Queue to which response messages are delivered when the connector issues requests*	`queue://CrossWorlds.QueueManager/MQCONN.REPLYTO`	No
"UnsubscribedQueue" on page 24	*Queue to which unsubscribed messages are sent*	`queue://CrossWorlds.QueueManager/MQCONN.UNSUBSCRIBE`	No
"UseDefaults" on page 24	true or false	false	

ApplicationPassword

Password used with the ApplicationUserID to log in to WebSphere MQ.

Default = None.

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

ApplicationUserID

User ID used with the ApplicationPassword to log in to WebSphere MQ.

Default=None.

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

ArchiveQueue

Queue to which copies of successfully processed messages are sent.

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

Channel

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

Default=None.

If the value of Channel is left blank or the property is removed, the connector uses the default server channel provided by WebSphere MQ.

ConfigurationMetaObject

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

Default = none.

DataHandlerClassName

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

Default = `com.crossworlds.DataHandlers.swift.SwiftDataHandler`

DataHandlerConfigMO

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

Default = `MO_DataHandler_Default`

DataHandlerMimeType

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

Default = `swift`

DefaultVerb

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

Default= `Create`

ErrorQueue

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

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

FeedbackCodeMappingMO

Allows you to override and reassign the default feedback codes used to synchronously acknowledge receipt of messages to WebSphere InterChange Server Express. This property enables you to specify a meta-object in which each attribute name is understood to represent a feedback code. The corresponding value of the feedback code is the return status that is passed to InterChange Server Express. For a listing of the default feedback codes, see "Synchronous acknowledgment" on page 7. The connector accepts the following attribute values representing WebSphere MQ-specific feedback codes:

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

The connector accepts the following WebSphere Business Integration Server Express system-specific status codes as attribute values in the meta-object:

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

Table 12 shows a sample meta-object.

Table 12. Sample feedback code meta-object attributes

Attribute Name	Default Value
`MQFB_APPL_FIRST`	SUCCESS
`MQFB_APPL_FIRST + 1`	FAIL
`MQFB_APPL_FIRST + 2`	UNABLE_TO_LOGIN

Default = none.

HostName

The name of the server hosting WebSphere MQ.

Default=None.

If the HostName is left blank or removed, the connector allows WebSphere MQ to determine the host.

InDoubtEvents

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

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

Default = Reprocess.

InputQueue

Specifies the message queues that the connector polls for new messages. See the MQSA documentation to configure the WebSphere MQ queues for routing to SWIFTAlliance gateways.

The connector accepts multiple semicolon-delimited queue names. For example, to poll the queues MyQueueA, MyQueueB, and MyQueueC, the value for connector configuration property InputQueue is: MyQueueA;MyQueueB;MyQueueC.

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

With pollQuanity set to 2, the connector retrieves at most 2 messages from each queue per call to pollForEvents. For the first cycle (1 of 2), the connector retrieves the first message from each of MyQueueA, MyQueueB, and MyQueueC. That completes the first round of polling. The connector starts a second round of polling (2 of 2) and retrieves one message each from MyQueueA and MyQueueC—it skips MqQueueB because that queue is now empty. After polling all queues twice, the call to the method pollForEvents is complete. The sequence of message retrieval is:

1. 1 message from MyQueueA
2. 1 message from MyQueueB
3. 1 message from MyQueueC
4. 1 message from MyQueueA
5. Skip MyQueueB because it is empty
6. 1 message from MyQueueC

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

InProgressQueue

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

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

PollQuantity

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

Default =1

Port

Port established for the WebSphere MQ listener.

Default=None.

If the value of Port is left blank or the property is removed, the connector allows WebSphere MQ to determine the correct port.

ReplyToQueue

Queue to which response messages are delivered when the connector issues requests.

Default = `queue://crossworlds.Queue.manager/MQCONN.REPLYTO`

UnsubscribedQueue

Queue to which messages about business objects that are not subscribed to are sent.

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

UseDefaults

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

Queue Uniform Resource Identifiers (URI)

A URI uniquely identifies a queue. A URI for a queue begins with the sequence `queue://` followed by:

- The name of the queue manager on which the queue resides
- A forward slash (/)
- The name of the queue
- Optionally, a list of name-value pairs to set the remaining queue properties.

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

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

Table 13 shows property names for queue URIs.

Table 13. SWIFT-specific connector property names for queue URIs

Property name	Description	Values
expiry	Lifetime of the message in milliseconds.	0 = unlimited. positive integers = timeout (in ms).
priority	Priority of the message.	0-9, where 1 is the highest priority. A value of -1 means that the property is determined by the configuration of the queue. A value of -2 means that the connector can use its own default value.
persistence	Whether the message should be retained in persistent memory.	1 = non-persistent 2 = persistent A value of -1 means that the property is determined by the configuration of the queue. A value of -2 means that the connector uses its own default value.
CCSID[1]	Character set of the destination.	Integers - valid values listed in base WebSphere MQ documentation.
targetClient	Whether the receiving application is JMS compliant or not.	1 = MQ (MQMD header only) This value must be set to 1 for SWIFTAlliance.
encoding	How to represent numeric fields.	An integer value as described in the base WebSphere MQ documentation.

Notes:

1. The connector has no control over the character set (CCSID) or encoding attributes of data in MQMessages. For the connector to work properly, WebSphere MQ queues require an ASCII character set, and must be configured accordingly in MQSA. Because data conversion is applied as the data is retrieved from or delivered to the message buffer, the connector relies on the IBM WebSphere MQ implementation of JMS to convert data (see the IBM WebSphere MQ Java client library documentation). Accordingly, these conversions should be bi-directionally equivalent to those performed by the native WebSphere MQ API using option MQGMO_CONVERT. The connector has no control over differences or failures in the conversion process. It can retrieve message data of any CCSID or encoding supported by WebSphere MQ without additional modifications (such as those imposed by MQSA). To deliver a message of a specific CCSID or encoding, the output queue must be a fully qualified URI and specify values for CCSID and encoding. The connector passes this information to WebSphere MQ, which (via the JMS API) uses the information when encoding data for MQMessage delivery. Often, lack of support for CCSID and encoding can be resolved by downloading the most recent version of the IBM WebSphere MQ Java client library from the IBM website. For further information on MQSA requirements, see MQSA documentation. If problems specific to CCSID and encoding persist, contact IBM Technical Support to discuss the possibility of using another Java Virtual Machine to run the connector.

Meta-object attributes configuration

The connector for SWIFT can recognize and read two kinds of meta-objects:

- Static connector meta-object
- Dynamic child meta-object

The attribute values of the dynamic child meta-object duplicate and override those of the static meta-object.

Static meta-object

The static meta-object consists of a list of conversion properties defined for different business objects. To define the conversion properties for a business object, first create a string attribute and name it using the syntax busObj_verb. For example, to define the conversion properties for a Customer object with the verb

Create, create an attribute named Swift_MT502_Create. In the application-specific text of the attribute, you specify the actual conversion properties.

Additionally, a reserved attribute named Default can be defined in the meta-object. When this attribute is present, its properties act as default values for all business object conversion properties.

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

Table 14 describes the meta-object properties.

Table 14. Static meta-object properties

Property name	Description
CollaborationName	The collaboration name must be specified in the application-specific text of the attribute for the business object/verb combination. For example, if you expect to handle synchronous requests for the business object Customer with the Create verb, the static metadata object must contain an attribute named Swift_MT*nnn_Verb*, where *nnn* is the Swift message type, for example, Swift_MT502_Create. The Swift_MT502_Create attribute must contain application-specific text that includes a name-value pair. For example, CollaborationName=MyCustomerProcessingCollab. See the "Application-specific information" on page 27 section for syntax details. Failure to do this results in runtime errors when the connector attempts to synchronously process a request involving the Customer business object. **Note:** This property is available only for synchronous requests.
DoNotReportBusObj	Optionally, you can include the DoNotReportBusObj property. By setting this property to true, all PAN report messages issued have a blank message body. This is recommended when you want to confirm that a request has been successfully processed but does not need notification of changes to the business object. This does not affect NAN reports. If this property is not found in the static meta-object, the connector defaults to false and populates the message report with the business object. **Note:** This property is available only for synchronous requests.
InputFormat	The input format is the message format to associate with the given business object. When a message is retrieved and is in this format, it is converted to the given business object if possible. If this format is not specified for a business object, the connector does not handle subscription deliveries for the given business object. In the static MO, the InputQueue property and the InputFormat property can serve as criteria for the adapter to map a given message to a specific business object. This feature is not used by the adapter for the SWIFT protocol.

Table 14. Static meta-object properties (continued)

Property name	Description
OutputFormat	The output format is set on messages created from the given business object. If a value for the OutputFormat property is not specified, the input format is used, if available. An OutputFormat property value defined in a dynamic child meta-object overrides the value defined in the static meta-object.
InputQueue	The input queue that the connector polls to detect new messages. The InputQueue property in the connector-specific properties defines which queues the adapter polls. This is the only property that the adapter uses to determine which queues to poll. In the static MO, the InputQueue property and the InputFormat property can serve as criteria for the adapter to map a given message to a specific business object. This feature is not used by the adapter for the SWIFT protocol.
OutputQueue	The output queue is the queue to which messages derived from the given business object are delivered. An OutputQueue property value defined in a dynamic child meta-object overrides the value defined in the static meta-object.
ResponseTimeout	The length of time in milliseconds to wait for a response before timing out. The connector returns SUCCESS immediately without waiting for a response if this property is undefined or has a value less than zero. A ResponseTimeout property value defined in a dynamic child meta-object overrides the value defined in the static meta-object.
TimeoutFatal	If this property is defined and has a value of true, the connector returns APP_RESPONSE_TIMEOUT when a response is not received within the time specified by ResponseTimeout. All other threads waiting for response messages immediately return APP_RESPONSE_TIMEOUT to InterChange Server Express. This causes InterChange Server Express to terminate the connection to the connector. A TimeoutFatal property defined in a dynamic child meta-object overrides the value defined in the static meta-object.

Note: The InputQueue property in the connector-specific properties defines which queues the adapter polls. This is the only property that the adapter uses to determine which queues to poll. In the static MO, the InputQueue property and the InputFormat property can serve as criteria for the adapter to map a given message to a specific business object. For the adapter for SWIFT, do not use this feature.

Application-specific information

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

```
InputFormat=ORDER_IN;OutputFormat=ORDER_OUT
```

You can use application-specific information to map a data handler to an input queue.

Mapping data handlers to InputQueues

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

1. Use connector-specific properties (see "InputQueue" on page 23) to configure one or more input queues.
2. For each input queue, specify the queue manager and input queue name as well as data handler class name and mime type in the application-specific information.

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

```
[Attribute]
Name = Swift_MT502_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputQueue=//queue.manager/CompReceipts;
  DataHandlerClassName=com.crossworlds.
DataHandlers.swift.disposition_notification;
  DataHandlerMimeType=message/
disposition_notification
IsRequiredServerBound = false
[End]
```

Overloading input formats

When retrieving a message, the connector normally matches the input format to one specific business object and verb combination. The connector then passes the business object name and the contents of the message to the data handler. This allows the data handler to verify that the message contents correspond to the business object that the user expects.

If, however, the same input format is defined for more than one business object, the connector cannot determine which business object the data represents before passing it to the data handler. In such cases, the connector passes the message contents only to the data handler and then looks up conversion properties based on the business object that is generated. Accordingly, the data handler must determine the business object based on the message content alone.

If the verb on the generated business object is not set, the connector searches for conversion properties defined for this business object with any verb. If only one set of conversion properties is found, the connector assigns the specified verb. If more properties are found, the connector fails the message because it is unable to distinguish among the verbs.

A sample static meta-object

The static meta-object shown below configures the connector to convert `SWIFT_MT502` business objects using verbs Create and Retrieve. Note that attribute `Default` is defined in the meta-object. The connector uses the conversion properties of this attribute:

```
OutputQueue=CustomerQueue1;ResponseTimeout=5000;
TimeoutFatal=true
```

as default values for all other conversion properties. Thus, unless specified otherwise by an attribute or overridden by a dynamic child meta-object value, the connector issues all business objects to queue `CustomerQueue1` and then waits for a response message. If a response does not arrive within 5000 milliseconds, the connector terminates immediately.

Business object with verb create: Attribute `Swift_MT502_Create` indicates to the connector that any messages of format `NEW` should be converted to a business object with the verb Create. Because an output format is not defined, the connector sends messages representing this object-verb combination using the format defined for input (in this case `NEW`).

Business object with verb retrieve: Attribute `Swift_MT502_Retrieve` specifies that business objects with verb Retrieve should be sent as messages with format `RETRIEVE`. Note that the default response time has been overridden so that the connector can wait up 10000 milliseconds before timing out (it still terminates if a response is not received).

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

[Attribute]
Name = Default
Type = String
Cardinality = 1
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
AppSpecificInfo = OutputQueue=CustomerQueue1;ResponseTimeout=5000;TimeoutFatal=true
IsRequiredServerBound = false
[End]
[Attribute]
Name = Swift_MT502_Create
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = InputFormat=NEW
IsRequiredServerBound = false
[End]
[Attribute]
Name = Swift_MT502_Retrieve
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = OutputFormat=RETRIEVE;ResponseTimeout=10000
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
```

```
IsRequired = false
IsRequiredServerBound = false
[End]

[Verb]
Name = Create
[End]

[Verb]
Name = Retrieve
[End]

[End]
```

Dynamic child meta-object

If it is difficult or unfeasible to specify the necessary metadata through a static meta-object, the connector can optionally accept metadata specified at runtime for each business object instance.

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

Because dynamic child meta-object properties override those found in static meta-objects, if you specify a dynamic child meta-object, you need not include a connector property that specifies the static meta-object. Accordingly, you can use either a dynamic child meta-object or a static meta-object, or both.

Table 15 shows sample static meta-object properties for business object `Swift_MT502_Create`. Note that the application-specific text consists of semicolon-delimited name-value pairs

Table 15. Static meta-object structure for Swift_MT502_Create

Attribute name	Application-specific text

Swift_MT502_Create	`InputFormat=ORDER_IN;`
	`OutputFormat=ORDER_OUT;`
	`OutputQueue=QueueA;`
	`ResponseTimeout=10000;`
	`TimeoutFatal=False`

Table 16 shows a sample dynamic child meta-object for business object `Swift_MT_Create`.

Table 16. Dynamic child meta-object Structure for Swift_MT502_Create

Property name	Value
`OutputFormat`	`ORDER_OUT`
`OutputQueue`	`QueueA`
`ResponseTimeout`	`10000`
`TimeoutFatal`	`False`

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

Population of the dynamic child meta-object during polling

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

Table Table 17 shows how a dynamic child meta-object might be structured for polling.

Table 17. JMS dynamic child meta-object structure for polling

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

As shown in Table 17, you can define an additional property, InputQueue, in a dynamic child meta-object. This property contains the name of the queue from which a given message has been retrieved. If this property is not defined in the child meta-object, it will not be populated.

Example scenario:

- The connector retrieves a message with the format ORDER_IN from the WebSphere MQ queue.
- The connector converts this message to an order business object and checks the application-specific text to determine if a meta-object is defined.
- If so, the connector creates an instance of this meta-object and populates the InputQueue and InputFormat properties accordingly, then publishes the business object to available processes.

Sample dynamic child meta-object

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

[Attribute]
Name = OutputFormat
Type = String
MaxLength = 1
IsKey = true
IsForeignKey = false
IsRequired = false
DefaultValue = ORDER
IsRequiredServerBound = false
[End]
[Attribute]
Name = OutputQueue
Type = String
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
```

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

[Verb]
Name = Create
[End]


[Verb]
Name = Retrieve
[End]

[End]
[BusinessObjectDefinition]
Name = Swift_MT502
Version = 1.0.0
AppSpecificInfo = cw_mo_conn=MyConfig

[Attribute]
```

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

[Verb]
Name = Create
[End]

[Verb]
Name = Retrieve
[End]

[End]
```

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

You can add attributes to a dynamic meta-object to gain more information about, and more control over, the message transport. Adding such attributes allows you to modify JMS properties, to control the ReplyToQueue on a per-request basis (rather than using the default ReplyToQueue specified in the adapter properties), and to re-target a message CorrelationID. This section describes these attributes and how they affect event notification and request processing in both synchronous and asynchronous modes.

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

Table 18. Dynamic meta-object header attributes

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

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

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

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

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

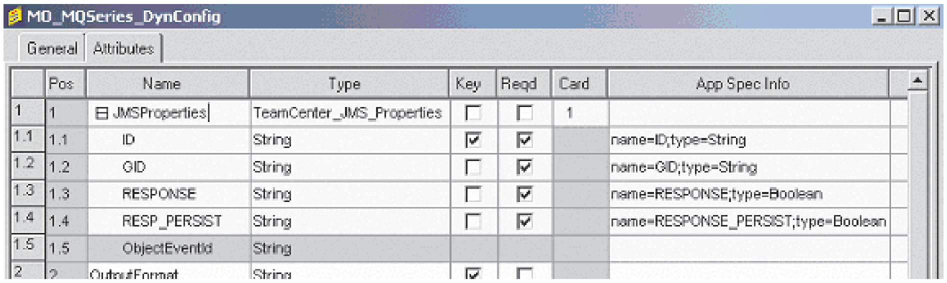
1. The name of the attribute has no semantic value.
2. The type of the attribute should always be String regardless of the JMS property type.
3. The application-specific information of the attribute must contain two name-value pairs defining the name and format of the JMS message property to which the attribute maps.

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

Table 19. Application-specific information for JMS property attributes

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

The figure below shows attribute JMSProperties in the dynamic meta-object and definitions for four properties in the JMS message header: ID, GID, RESPONSE and RESPONSE_PERSIST. The application-specific information of the attributes defines the name and type of each. For example, attribute ID maps to JMS property ID of type String).



	Pos	Name	Type	Key	Reqd	Card	App Spec Info
1	1	⊟ JMSProperties	TeamCenter_JMS_Properties	□	□	1	
1.1	1.1	ID	String	☑	☑		name=ID;type=String
1.2	1.2	GID	String	□	☑		name=GID;type=String
1.3	1.3	RESPONSE	String	□	☑		name=RESPONSE;type=Boolean
1.4	1.4	RESP_PERSIST	String	□	☑		name=RESPONSE_PERSIST;type=Boolean
1.5	1.5	ObjectEventId	String				
2	2	OutputFormat	String	☑	□		

Figure 4. JMS properties attribute in a dynamic meta-object

Asynchronous event notification: If a dynamic meta-object with header attributes is present in the event business object, the connector performs the following steps (in addition to populating the meta-object with transport-related data):

1. Populates the CorrelationId attribute of the meta-object with the value specified in the JMSCorrelationID header field of the message.
2. Populates the ReplyToQueue attribute of the meta-object with the queue specified in the JMSReplyTo header field of the message. Since this header field is represented by a Java object in the message, the attribute is populated with the name of the queue (often a URI).

3. Populates the DeliveryMode attribute of the meta-object with the value specified in the JMSDeliveryMode header field of the message.
4. Populates the Priority attribute of the meta-object with the JMSPriority header field of the message.
5. Populates the Destination attribute of the meta-object with the name of the JMSDestination header field of the message. Since the Destination is represented by an object, the attribute is populated with the name of the Destination object.
6. Populates the Expiration attribute of the meta-object with the value of the JMSExpiration header field of the message.
7. Populates the MessageID attribute of the meta-object with the value of the JMSMessageID header field of the message.
8. Populates the Redelivered attribute of the meta-object with the value of the JMSRedelivered header field of the message.
9. Populates the TimeStamp attribute of the meta-object with the value of the JMSTimeStamp header field of the message.
10. Populates the Type attribute of the meta-object with the value of the JMSType header field of the message.
11. Populates the UserID attribute of the meta-object with the value of the JMSXUserID property field of the message.
12. Populates the AppID attribute of the meta-object with the value of the JMSXAppID property field of the message.
13. Populates the DeliveryCount attribute of the meta-object with the value of the JMSXDeliveryCount property field of the message.
14. Populates the GroupID attribute of the meta-object with the value of the JMSXGroupID property field of the message.
15. Populates the GroupSeq attribute of the meta-object with the value of the JMSXGroupSeq property field of the message.
16. Examines the object defined for the JMSProperties attribute of the meta-object. The adapter populates each attribute of this object with the value of the corresponding property in the message. If a specific property is undefined in the message, the adapter sets the value of the attribute to CxBlank.

Synchronous event notification: For synchronous event processing, the adapter posts an event and waits for a response from InterChange Server Express before sending a response message back to the application. Any changes to the business data are reflected in the response message returned. Before posting the event, the adapter populates the dynamic meta-object just as described for asynchronous event notification. The values set in the dynamic meta-object are reflected in the response-issued header as described below (all other read-only header attributes in the dynamic meta-object are ignored.):

- **CorrelationID** If the dynamic meta-object includes the attribute CorrelationId, you must set it to the value expected by the originating application. The application uses the CorrelationID to match a message returned from the connector to the original request. Unexpected or invalid values for a CorrelationID will cause problems. It is helpful to determine how the application handles correlating request and response messages before using this attribute. You have four options for populating the CorrelationID in a synchronous request.
  1. Leave the value unchanged. The CorrelationID of the response message will be the same as the CorrelationID of the request message. This is equivalent to the WebSphere MQ option MQRO_PASS_CORREL_ID.

2. Change the value to CxIgnore. The connector by default copies the message ID of the request to the CorrelationID of the response. This is equivalent to the WebSphere MQ option MQRO_COPY_MSG_ID_TO_CORREL_ID.
3. Change the value to CxBlank. The connector will not set the CorrelationID on the response message.
4. Change the value to a custom value. This requires that the application processing the response recognize the custom value.

If you do not define attribute CorrelationID in the meta-object, the connector handles the CorrelationID automatically.

- **ReplyToQueue** If you update the dynamic meta-object by specifying a different queue for attribute ReplyToQueue, the connector sends the response message to the queue you specify. This is not recommended. Having the connector send response messages to different queues may interfere with communication because an application that sets a specific reply queue in a request message is assumed to be waiting for a response on that queue.
- **JMS properties** The values set for the JMS Properties attribute in the dynamic meta-object when the updated business object is returned to the connector are set in the response message.

Asynchronous request processing: The connector uses the dynamic meta-object, if present, to populate the request message prior to issuing it. The connector performs the following steps before sending a request message:

1. If attribute CorrelationID is present in the dynamic meta-object, the connector sets the CorrelationID of the outbound request message to this value.
2. If attribute ReplyToQueue is specified in the dynamic meta-object, the connector passes this queue via the request message and waits on this queue for a response. This allows you to override the ReplyToQueuevalue specified in the connector configuration properties. If you additionally specify a negative ResponseTimeout (meaning that the connector should not wait for a response), theReplyToQueue is set in the response message, even though the connector does not actually wait for a response.
3. If attribute DeliveryMode is set to 2, the message is sent persistently. If DeliveryMode is set to 1, the message is not sent persistently. Any other value may fail the connector. If DeliveryMode is not specified in the MO, then the JMS provider establishes the persistence setting.
4. If attribute Priority is specified, the connector sets the value in the outgoing request. The Priority attribute can take values 0 through 9; any other value may cause the connector to terminate.
5. If attribute JMSProperties is specified in the dynamic meta-object, the corresponding JMS properties specified in the child dynamic meta-object are set in the outbound message sent by the connector.

Note: If header attributes in the dynamic meta-object are undefined or specify CxIgnore, the connector follows its default settings.

Synchronous request processing: The connector uses the dynamic meta-object, if present, to populate the request message prior to issuing it. If the dynamic meta-object contains header attributes, the connector populates it with corresponding new values found in the response message. The connector performs the following steps (in addition to populating the meta-object with transport-related data) after receiving a response message:

1. If attribute CorrelationID is present in the dynamic meta-object, the adapter updates this attribute with the JMSCorrelationID specified in the response message.
2. If attribute ReplyToQueue is defined in the dynamic meta-object, the adapter updates this attribute with the name of the JMSReplyTo specified in the response message.
3. If attribute DeliveryMode is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSDeliveryMode header field of the message.
4. If attribute Priority is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSPriority header field of the message.
5. If attribute Destination is defined in the dynamic meta-object, the adapter updates this attribute with the name of the JMSDestination specified in the response message.
6. If attribute Expiration is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSExpiration header field of the message.
7. If attribute MessageID is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSMessageID header field of the message.
8. If attribute Redelivered is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSRedelivered header field of the message.
9. If attribute TimeStamp is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSTimeStamp header field of the message.
10. If attribute Type is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSType header field of the message.
11. If attribute UserID is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXUserID header field of the message.
12. If attribute AppID is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXAppID property field of the message.
13. If attribute DeliveryCount is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXDeliveryCount header field of the message.
14. If attribute GroupID is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXGroupID header field of the message.
15. If attribute GroupSeq is present in the dynamic meta-object, the adapter updates this attribute with the value of the JMSXGroupSeq header field of the message.
16. If attribute JMSProperties is defined in the dynamic meta-object, the adapter updates any properties defined in the child object with the values found in the response message. If a property defined in the child object does not exist in the message, the value is set to CxBlank.

Note: Using the dynamic meta-object to change the CorrelationID set in the request message does not affect the way the adapter identifies the response message—the adapter by default expects that the CorrelationID of any response message equals the message ID of the request sent by the adapter.

Error handling: If a JMS property cannot be read from or written to a message, the connector logs an error and the request or event fails. If a user-specified `ReplyToQueue` does not exist or cannot be accessed, the connector logs an error and the request fails. If a `CorrelationID` is invalid or cannot be set, the connector logs an error and the request fails. In all cases, the message logged is from the connector message file.

Startup file configuration

Before you start the connector for SWIFT, you must configure the startup file. The sections below describe how to do this for Windows, Linux, and OS/400 systems.

Windows

To complete the configuration of the connector for Windows 2000, you must modify the `start_SWIFT.bat` file:

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

Linux

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

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

OS/400

On the OS/400 it is not necessary to configure the startup file. The start_SWIFT.sh script can be used as installed to start the adapter.

Running multiple instances of the adapter

Note: When you create an additional instance of this adapter (or any adapter that is supplied with WebSphere Business Integration Server Express or Express Plus), that instance of the adapter will be counted as a separate adapter by the licensing function that limits the total number of adapters that can be deployed.

You can set your system up to create and run multiple instances of a connector by following the steps below. You must:

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

Create a new directory

You must create a connector directory for each connector instance.

- For Windows platforms, the connector directory should be named:

  `ProductDir\connectors\connectorInstance`

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

```
ProductDir\repository\connectorInstance
```

You can specify the InterChange Server Express server name as a parameter of `startup.bat`, for example:

```
start_Swift.bat connName serverName
```

- For OS/400 platforms, the connector directory should be named:

  ```
  /QIBM/UserData/WBIServer43/WebSphereICSName/connectors
  /connectorInstance
  ```

  where `connectorInstance` uniquely identifies the connector instance and where *`WebSphereICSName`* is the name of the InterChange Server Express server instance with which the connector runs.

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

  ```
  /QIBM/UserData/WBIServer43/WebSphereICSName/repository/connectorInstance
  ```

  where *`WebSphereICSName`* is the name of the InterChange Server Express server instance with which the connector runs.

  You can specify the InterChange Server Express server name as a parameter of `startup.sh`, for example:

  ```
  start_Swift.sh connName serverName
  ```

- For Linux platforms, the connector directory should be named:

  ```
  ProductDir/connectors/connectorInstance
  ```

  where `connectorInstance` uniquely identifies the connector instance

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

  ```
  ProductDir/repository/connectorInstance
  ```

  You can specify the InterChange Server Express server name as a parameter of `connector_manager`, for example

  ```
  start_Swift.sh connName WebSphereICSName [-cConfigFile ]
  ```

Create business object definitions

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

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

     ```
     ProductDir\repository\initialconnectorInstance
     ```

     Any additional files you create should be in the appropriate `connectorInstance` subdirectory of `ProductDir\repository`
   - For OS/400:

     ```
     /QIBM/UserData/WBIServer43/WebSphereICSName/repository
     /initialConnectorInstance
     ```

where *WebSphereICSName* is the name of the InterChange Server Express server instance with which the connector runs.

Any additional files you create should be in the appropriate `connectorInstance` subdirectory of `/QIBM/UserData/WBIServer43/`*`WebSphereICSName`*`/repository`

- For Linux:

  ```
  ProductDir/repository/initialconnectorInstance
  ```

  Any additional files you create should be in the appropriate `connectorInstance` subdirectory of `ProductDir/repository`

Create a connector definition

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

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

Create a startup script

To create a startup script:

1. Copy the initial connector's startup script and name it to include the name of the connector directory:

   dirname

   (For Linux only) Change the startup script CONJAR from CONJAR=${CONDIR}/CW${CONNAME}.jar to CONJAR=${CONDIR}/CWSWIFT.jar
2. Put this startup script in the connector directory you created in "Create a new directory" on page 39.
3. (For Windows only) Create a startup script shortcut.
4. (For Windows only) Copy the initial connector's shortcut text and change the name of the initial connector (in the command line) to match the name of the new connector instance.
5. (For OS/400 only) Create a job description for the connector using the following information:

   ```
   CRTDUPOBJ OBJ(QWBISWIFT) FROMLIB(QWBISVR43) OBJTYPE(*JOBD)
   TOLIB(QWBISVR43) NEWOBJ(newSWIFTCname)
   ```

   where *newSWIFTCname* is a 10-character name to be used for the job description for your new Swift connector.
6. (For OS/400 only) Add the new connector to the WebSphere Business Integration Console. For information about the Console, see the online help provided with the Console.

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

Starting the connector

A connector must be explicitly started using its connector start-up script. The startup script should reside in the connector's run-time directory. For example, for Windows, use:

```
ProductDir\connectors\connName
```

where *connName* identifies the connector. The name of the startup script depends on the operating-system platform, as Table 20 shows.

Table 20. Startup scripts for a connector

Operating system	Startup script
Windows	`start_connName.bat`
OS/400	`start_connName.sh`
Linux	Before executing the startup script you must set environmental variables. The following command will set these variables and run the startup script `start_connName.sh` automatically: `connector_manager -start connName WebSphereICSName [-cConfigFile ]`

For more information on how to start a connector, including the command line start-up options, see the *System Administration Guide*.

Invoking the startup script on Windows

On Windows platforms, you can invoke the connector startup script in the following ways:

- From the **Start** menu:
  - Select **Programs>IBM WebSphere Business Integration Express>Adapters>Connectors>*your_connector_name***

    By default, the program name is ″IBM WebSphere Business Integration Express.″ However, it can be customized. Alternatively, you can create a desktop shortcut to your connector.
  - On Windows systems, you can configure the connector to start as a Windows service. In this case, the connector starts when the Windows system boots (for an Auto service) or when you start the service through the Windows Services window (for a Manual service).
- From the command line:

  ```
  start_connName connName WebSphereICSName [-cConfigFile ]
  ```

  where *connName* is the name of the connector and *WebSphereICSName* is the name of the InterChange Server Express instance. By default, the name of the InterChange Server Express instance is Web SphereICS.

Invoking the startup script on OS/400

On OS/400 platforms, you can invoke the connector startup script in the following ways:

- From the Windows system where the WebSphere Business Integration Server Express Console is installed:

  Select **Programs>IBM WebSphere Business Integration Console >Console**. Then specify the OS/400 system name or IP address and a user profile and password that has *JOBCTL special authority. Select the *connName* adapter from the list of adapters and select the **Start Adapter** button.

- From the OS/400 command line:
  - In batch mode:

    Run CL Command QSH and from the QSHELL environment, run

    ```
    /QIBM/ProdData/WBIServer43/bin/submit_adapter.sh connName
    WebSphereICSName pathToConnNameStartScript jobDescriptionName
    ```

    where *connName* is the connector name, *WebSphereICSName* is the InterChange Server Express server name (default is QWBIDFT), *pathToConnNameStartScript* is the full path to the connector start script, and *jobDescriptionName* is the name of the job description to use in the QWBISVR43 library.
  - In interactive mode:

    Run CL Command QSH and from the QSHELL environment, run

    ```
    /QIBM/UserData/WBIServer43/WebSphereICSName/connectors/connName/
    start_connName.sh connName WebSphereICSName [-cConfigFile ]
    ```

    where *connName* is the name of your connector and *WebSphereICSName* is the name of the InterChange Server Express instance.

Note: To start with TCP/IP servers, use the command:

```
/QIBM/ProdData/WBIServer43/bin/add_autostart_adapter.sh
connName WebSphereICSName pathToConnNameStartScript
jobDescriptionName
```

where *connName* is the name of your connector, *WebSphereICSName* is the name of the InterChange Server Express instance, *pathToConnNameStartScript* is the full path to the connector start script, and *jobDescriptionName* is the name of the job description for the adapter.

Invoking the startup script on Linux

On Linux platforms, you can invoke the connector startup script in the following way:

- From the command line, use

  ```
  connector_manager -start connName WebSphereICSName [-cConfigFile ]
  ```

  where *connName* is the name of the connector and *WebSphereICSName* is the name of the InterChange Server Express instance.

Stopping the connector

The way to stop a connector depends on the way that the connector was started.

Stopping the connector from Windows

On Windows platforms, you can stop the connector in the following ways:

- Activate the connector window, and type "q" and then press Enter.
- If the connector started as a Windows service, you can stop it through **Control Panel (Control panel>Administrative Tools> Services>CWConnectorWBISwiftAdapter)**.

Stopping the connector from OS/400

On OS/400 platforms, you can stop the connector in the following ways:

- From the Console or command line:

  If you started the connector using the Console or using the "submit_adapter.sh" script in QSHELL, from the OS/400 command entry, use the CL Command

WRKACTJOB SBS(QWBISVR43) to display the jobs to the Server Express product. Scroll the list to find the job with a jobname that matches the job description for the connector. For example, for the Swift connector the jobname is QWBISWIFTC.

Select option 4 on this job, and press F4 to get the prompt for the ENDJOB command. Then specify *IMMED for the Option parameter and press enter.

- If you started the connector using the start_connName.sh script from QSHELL, press F3 in the QSHELL environment to end QSHELL and the connector.

Stopping the connector from Linux

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

```
connector_manager -stop serverName connName
```

where *serverName* is WebSphere ICS and *connName* is the name of the connector.

Chapter 3. Business objects

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

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

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

Connector business object requirements

The business object requirements for the connector reflect the way the SWIFT data handler converts a SWIFT message into a WebSphere business object, and vice versa.

The sections below discuss the requirements for WebSphere business objects as well as the SWIFT message structure. For a step-by-step description of how the SWIFT data handler interacts with WebSphere business objects and SWIFT messages, see Chapter 4, "SWIFT Data Handler," on page 81.

A review of the Business Object Development Guide is strongly recommended.

Business object hierarchy

WebSphere business objects can be flat or hierarchical. All the attributes of a **flat** business object are **simple** (that is, each attribute represents a single value, such as a `String` or `Integer` or `Date`).

In addition to containing simple attributes, a **hierarchical** business object has attributes that represent a child business object, an array of child business objects, or a combination of both. In turn, each child business object can contain a child business object or an array of business objects, and so on.

Important: A business object array can contain data whose type is a business object. It cannot contain data of any other type, such as `String` or `Integer`.

There are two types of relationships between parent and child business objects:

- **Single-cardinality—**When an attribute in a parent business object represents a single child business object. The attribute is of the same type as the child business object.
- **Multiple-cardinality—**When an attribute in the parent business object represents an array of child business objects. The attribute is an array of the same type as the child business objects.

WebSphere uses the following terms when describing business objects:

- **hierarchical—**Refers to a complete business object, including the top-level business object and its the child business objects at any level.
- **parent—**Refers to a business object that contains at least one child business object. A top-level business object is also a parent.
- **individual—**Refers to a single business object, independent of any child business objects it might contain or that contain it.
- **top-level—**Refers to the individual business object at the top of the hierarchy, which does not itself have a parent business object.
- **wrapper—**Refers to a top-level business object that contains information used to process its child business objects. For example, the XML connector requires the wrapper business object to contain information that determines the format of its child data business objects and routes the children.

Business object attribute properties

Business object architecture defines various properties that apply to attributes. This section describes how the connector interprets several of these properties. For further information on these properties, see Business Object Attributes and Attribute Properties in Chapter 2 of the *Business Object Development Guide.*

Name property

Each business object attribute must have a unique name within the business object. The name should describe the data that the attribute contains.

For an application-specific business object, check the connector or data handler guide for specific naming requirements.

The name can be up to 80 alphanumeric characters and underscores. It cannot contain spaces, punctuation, or special characters.

Type property

The Type property defines the data type of the attribute:

- For a simple attribute, the supported types are `Boolean`, `Integer`, `Float`, `Double`, `String`, `Date`, and `LongText`.
- If the attribute represents a child business object, specify the type as the name of the child business object definition (for example, `Type = MT502A`) and specify the cardinality as 1.
- If the attribute represents an array of child business objects, specify the type as the name of the child business object definition and specify the cardinality as `n`.

Note: All attributes that represent child business objects also have a `ContainedObjectVersion` property (which specifies the child's version number) and a `Relationship` property (which specifies the value `Containment`).

Cardinality property

Each simple attribute has cardinality 1. Each business object attribute that represents a child or array of child business objects has cardinality `1` or `n`, respectively.

Note: When specified for a required attribute, cardinality `1` indicates a child business object must exist, and cardinality `n` indicates zero to many instances of a child business object.

Key property

At least one attribute in each business object must be specified as the key. To define an attribute as a key, set this property to `true`.

When you specify as key an attribute that represents a child business object, the key is the concatenation of the keys in the child business object. When you specify as key an attribute that represents an array of child business objects, the key is the concatenation of the keys in the child business object at location `0` in the array.

Note: Key information is not available in the collaboration mapping process.

Foreign key property

The Foreign Key property is typically used in application-specific business objects to specify that the value of an attribute holds the primary key of another business object, serving as a means of linking the two business objects. The attribute that holds the primary key of another business object is called a **foreign key**. Define the Foreign Key property as `true` for each attribute that represents a foreign key.

You can also use the Foreign Key property for other processing instructions. For example, this property can be used to specify what kind of foreign key lookup the connector performs. In this case, you might set Foreign Key to `true` to indicate that the connector checks for the existence of the entity in the database and creates the relationship only if the record for the entity exists.

Required property

The Required property specifies whether an attribute must contain a value. If a particular attribute in the business object that you are creating must contain a value, set the Required property for the attribute to `true`.

For information on enforcing the Required property for attributes, see the section on `initAndValidateAttributes()` in *Connector Reference: C++ Class Library* and *Connector Reference: Java Class Library*.

AppSpecificInfo

The AppSpecificInfo property is a `String` no longer than 255 characters that is specified primarily for an application-specific business object.

Note: Application-specific text is not available in the collaboration mapping process .

Max length property

The Max Length property is set to the number of bytes that a `String`-type attribute can contain. Although this value is not enforced by the WebSphere system, specific connectors or data handlers may use this value. Check the guide for the connector or data handler that will process the business object to determine minimum and maximum allowed lengths.

Note: The Max Length property is very important when you use a fixed width data handler. Attribute length is not available in the collaboration mapping process.

Default value property

The Default Value property can specify a default value for an attribute.

If this property is specified for an application-specific business object, and the UseDefaults connector configuration property is set to `true`, the connector can use the default values specified in the business object definition to provide values for attributes that have no values at runtime.

For more information on how the Default Value property is used, see the section on `initAndValidateAttributes()` in *Connector Reference: C++ Class Library* and *Connector Reference: Java Class Library*.

Comments property

The Comments property allows you to specify a human-readable comment for an attribute. Unlike the AppSpecificInfo property, which is used to process a business object, the Comments property provides only documentation information.

Special attribute value

Simple attributes in business objects can have the special value, `CxIgnore`. When it receives a business object from an integration broker, the connector ignores all attributes with a value of `CxIgnore`. It is as if those attributes were invisible to the connector.

If no value is required, the connector sets the value of that attribute to `CxIgnore` by default.

Application-specific text at the attribute level

Note: Business object level application-specific text is not used by the connector.

For business object attributes, the application-specific text format consists of name-value parameters. Each name-value parameter includes the parameter name and its value. The format of attribute application-specific text is as follows:

name=*value*`[:`*name_n*=*value_n*`][...]`

Each parameter set is separated from the next by a colon (:) delimiter.

Table 21 describes the name-value parameters for attribute application-specific text.

Table 21. Name-value parameters in AppSpecificText for attributes

Parameter	Required	Description
`block`	Yes for top-level object only	The number of the block in the SWIFT message. Values range from 0-5. For information on the SWIFT message blocks, see "Overview of SWIFT message structure" on page 49.
`parse`	Yes for attributes of the top-level object only	Describes whether, and how, to parse the SWIFT message block. Values are: `fixlen`—parse as fixed length `delim`—parse as delimited text `field`—Block 4 only `no`—Do not parse; treat as a single string.
`tag`	Yes for attributes of type tag business object	The tag number of the field. For more on SWIFT message tags, see Appendix C, "SWIFT message structure," on page 115. For further information on sequence and field business objects, see "Block 4 business object structure" on page 62.
`letter=`*a*	Yes for each attribute that points to a tag business object	One or more supported letters appended to the tag in the SWIFT message format. For example 20`A` or `[A|B|NULL]` (A or B or null). Note that `NULL` must be specified for tags where no letter is a possibility, or for tags that do not have a letter option at all. For example, tag 59.
`content`	No	The qualifier in the SWIFT message format. For example, in a SWIFT message MT502, tag20C, the qualifier = `SEME`.

Overview of SWIFT message structure

SWIFT messages consist of five blocks of data. In addition, the MQSA component adds two blocks that are used for queue management. The high-level structure of a SWIFT message is as follows:

MQSA UUID

- SWIFT 1:Basic Header Block
- SWIFT 2: Application Header Block
- SWIFT 3:User Header Block
- SWIFT 4: Text Block
- SWIFT 5: Trailer

MQSA S Block

Note: The MQSA component adds the UUID (User Unique Message Identifier) and S blocks. Neither are parsed by the SWIFT data handler. The S block has the same structure as SWIFT block 5, except that field tags consist of three `char` strings. For example, `{S:{COP:P}}`.

For further information on SWIFT message structure, see Appendix C, "SWIFT message structure," on page 115, and *All Things SWIFT: the SWIFT User Handbook*.

Overview of business objects for SWIFT

As shown in Figure 5 there are five kinds of business objects for SWIFT:

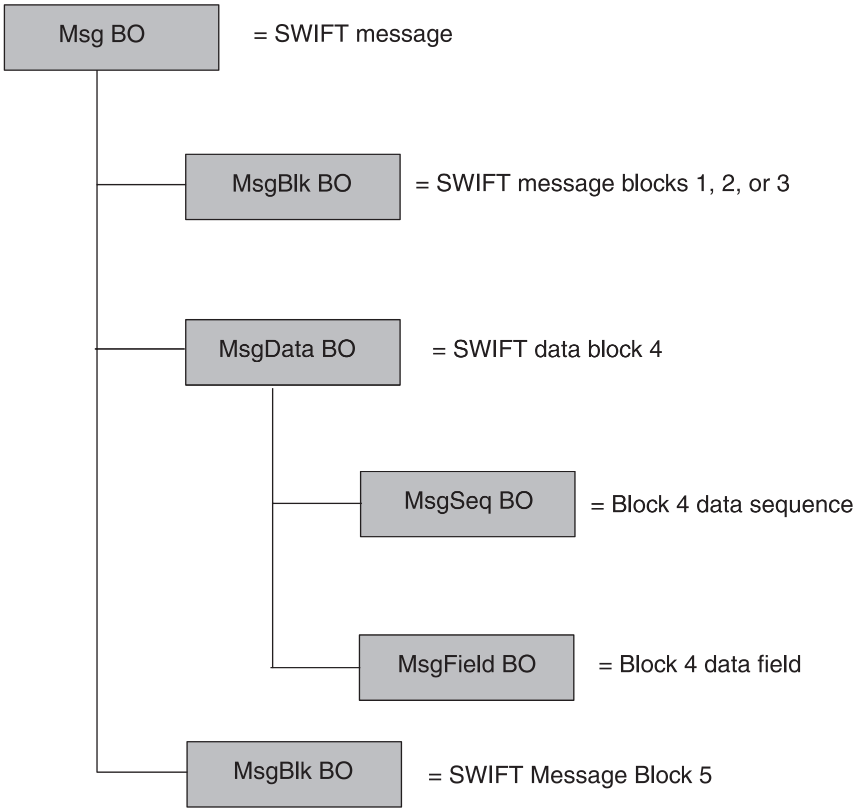


Figure 5. Business objects map to SWIFT message components

- **Message business object (Msg BO)** This is the top-level business object whose attributes correspond to the blocks in a SWIFT message. For further information, see "Top-level business object structure" on page 51.
- **Message block business object (MsgBlk BO)** A child object of the Msg BO that can represent blocks 1, 2, 3, or 5 in a SWIFT message. For further information, see "Block 1 business object structure" on page 54.
- **Message data business object (MsgData BO)** A child object of the Msg BO that represents block 4 of the SWIFT message. For further information, see "Block 4 business object structure" on page 62.
- **Message sequence business object (MsgSeq BO)** A child object of a MsgData BO or of another MsgSeq BO. A MsgSeq BO represents a sequence of fields occurring in block 4 of the SWIFT message. For further information, see "Sequence business object structure" on page 67.
- **Message field business object (MsgField BO)** A child object of the MsgData BO or of a MsgSeq BO that contains the content of a field. Fields correspond to tags in SWIFT messages. For further information, see "Field business object definitions" on page 70.

Each of these business objects consist of the following:

- **Name** The name of the business object consists of a SWIFT Message name, a SWIFT message sequence name, or a SWIFT field name. More detailed naming conventions, if any, are provided in the sections for each kind of business object listed below. For example:
  - `Swift_MT502` is the name of the Msg BO. For further information, see "Top-level business object structure" on page 51.

  - `Swift_ApplicationHeader` is the name of a MsgBlk BO. For further information, see “Block 1 business object structure” on page 54, “Block 2 business object structure” on page 57, and “Block 3 business object structure” on page 60.
  - `Swift_MT502Data` is the name of a MsgData BO. For further information, see “Block 4 business object structure” on page 62.
  - `Swift_MT502_B1` is the name of a MsgSeq BO. For further information, see “Sequence business object structure” on page 67.
  - `Swift_Tag_22` is the name of a MsgField BO. For further information, see “Field business object definitions” on page 70.
- **Version** The version of the business object is set to 1.1.0. For example:

  ```
  Version = 1.1.0
  ```
- **Attributes** Each business object contains one or more attributes. For more information see “Business object attribute properties” on page 46 and the sections below on each kind of business object.
- **Verbs** Each business object supports the following standard verbs:
  - Create
  - Retrieve

SWIFT message and business object data mapping

The IBM WebSphere Business Integration Server Express Adapter for SWIFT supports the following type of mapping:

- **SWIFT-message-to-WebSphere-business-object** The sections below describe the data mapping that occurs between SWIFT messages and WebSphere business objects.

Top-level business object structure

The structure of the top-level business object for a SWIFT message, or Msg BO, reflects that of the SWIFT message. WebSphere requires a business object for each SWIFT block. As shown in Table 22, the top-level business object must have at least 5 attributes, one for each SWIFT block.

Note: Only attribute properties of consequence are shown in Table 22. For a listing of all attribute properties, see “Sample top-level Business Object (Msg BO) definition” on page 53.

Table 22. Top-level business object structure

Name	Type	Key	Required	Application specific info
UUID (MQSA prepended)	String	Yes	No	block=0;parse=no
Swift_01Header	Swift_BasicHeader	No	Yes	block=1; parse=fixlen
Swift_02Header	Swift_Application Header	No	No	block=2; parse=fixlen
Swift_03Header	Swift_UserHeader	No	No	block=3; parse=delim
Swift_Data	Swift_Text	No	No	block=4;parse=field
Swift_05Trailer	String	No	No	block=5;parse=no
Swift_BlockS (MQSA appended)	String	No	No	block=6;parse=no

The following rules apply to the top-level business object:

- The name of the top-level object must be constructed in the following way:

  *BOPrefix*_MT*MessageType*

  where:

  BOPrefix = an attribute of the meta-object (MO). For further information on the meta-object, see “Static meta-object” on page 25.

  _MT = a constant string.

  MessageType = an attribute of block 2 of the SWIFT message. For further information, see *All Things SWIFT: the SWIFT User Handbook*

  An example of a top-level business object name is Swift_MT502.
- UUID, prepended to the message by the MQSA, is represented with a String attribute
- Blocks 1-4 are represented with single-cardinality containers
- Block 5 is a string attribute, and is not extracted from the message by the SWIFT data handler.

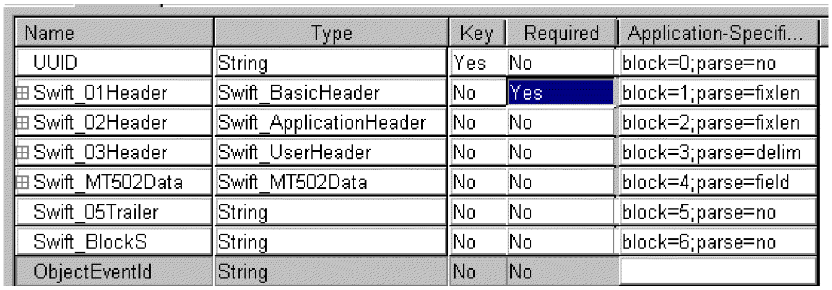
Note: It is possible to create business objects for block 5 and block S using block 3 as a template.

See Table 21 for the attribute application-specific information.

Figure 6 shows a business object definition for a top-level business object of a SWIFT message. This Msg BO definition was created in the WebSphere development environment.

The application-specific information contains the block number and parsing parameters for each attribute. For further information on attribute application-specific text, see Table 21. The Swift_ attributes correspond to child business objects discussed in the following sections. For a full specification of this sample business object definition, see “Sample top-level Business Object (Msg BO) definition” on page 53. Of special note is the type for the data block attribute, Swift_MT502Data, which indicates SWIFT message type 502, an order to buy or sell. This attribute corresponds to a child object of the top-level Msg BO that represents block 4 of the SWIFT message. The child object is a message data business object (MsgData BO).

All SWIFT top-level business object definitions are identical to that shown in Figure 6 with one exception: Block 4, shown as Swift_MT502Data, reflects the actual data definition of a specific SWIFT message.



Name	Type	Key	Required	Application-Specifi...
UUID	String	Yes	No	block=0;parse=no
⊞ Swift_01Header	Swift_BasicHeader	No	Yes	block=1;parse=fixlen
⊞ Swift_02Header	Swift_ApplicationHeader	No	No	block=2;parse=fixlen
⊞ Swift_03Header	Swift_UserHeader	No	No	block=3;parse=delim
⊞ Swift_MT502Data	Swift_MT502Data	No	No	block=4;parse=field
Swift_05Trailer	String	No	No	block=5;parse=no
Swift_BlockS	String	No	No	block=6;parse=no
ObjectEventId	String	No	No	

Figure 6. Definition for top-level business object of a SWIFT message

Note: To create a top-level business object definition for a SWIFT message, you must start Business Object Designer Express and then create all the child objects first.

Sample top-level Business Object (Msg BO) definition

This section presents a sample definition of a top-level business object, or Msg BO, for a SWIFT message of type MT502—an order to buy or sell.

```
[BusinessObjectDefinition]
Name = Swift_MT502
Version = 1.1.0

   [Attribute]
   Name = UUID
   Type = String
   Cardinality = 1
   MaxLength = 255
   IsKey = true
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = block=0;parse=no
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = Swift_01Header
   Type = Swift_BasicHeader
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 1
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = block=1;parse=fixlen
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = Swift_02Header
   Type = Swift_ApplicationHeader
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 1
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = block=2;parse=fixlen
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = Swift_03Header
   Type = Swift_UserHeader
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 1
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = block=3;parse=delim
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = Swift_MT502Data
   Type = Swift_MT502Data
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
```

```
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = block=4;parse=field
IsRequiredServerBound = false
[End]
[Attribute]
Name = Swift_05Trailer
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = block=5;parse=no
IsRequiredServerBound = false
[End]
[Attribute]
Name = Swift_BlockS
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = block=6;parse=no
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Verb]
Name = Create
[End]

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

Block 1 business object structure

The MsgBlck BO, `Swift_BasicHeader`, has the format and attributes shown in Table 23. The SWIFT data handler converts each of the SWIFT fields in this block into attributes in the `Swift_BasicHeader` business object. Note that there is no attribute application-specific information for this business object.

Note: Only attribute properties of consequence are shown in Table 23. For a listing of all attribute properties, see "Sample block 1 business object definition" on page 55.

Table 23. Block 1 business object structure

Name	Type	Key	Foreign key	Required	Cardinality	Default	Max length
BlockIdentifier	String	Yes	No	Yes	1	1:[a]	2
ApplicationIdentifier	String	No	No	Yes	1		1
ServiceIdentifier	String	No	No	Yes	1		2

Table 23. Block 1 business object structure (continued)

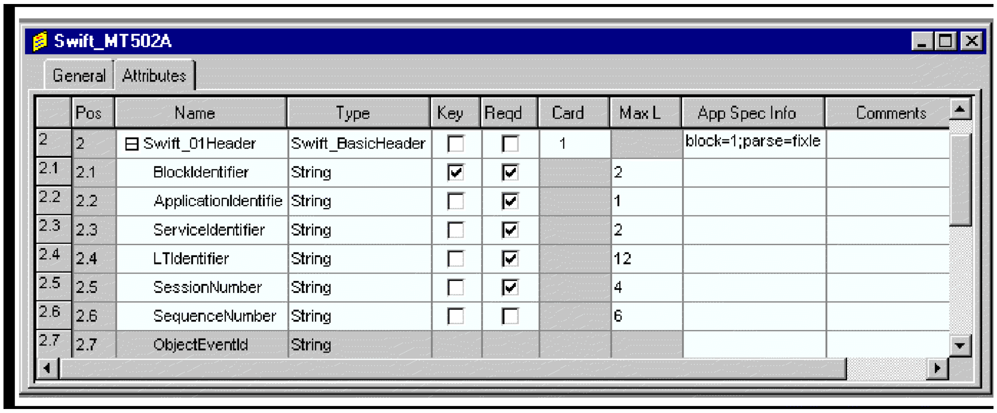
Name	Type	Key	Foreign key	Required	Cardinality	Default	Max length
LTIdentifier	String	No	No	Yes	1		12
SessionNumber	String	No	No	Yes	1		4
SequenceNumber	String	No	No	No	1		4

[a] The BlockIdentifier attribute includes the delimiter ":" as in "1:".

See Table 21 for the attribute application-specific information.

Figure 7 shows a block 1 business object definition that has been manually created in a WebSphere development environment. Each attribute name (`ApplicationIdentifier`, `ServiceIdentifier`, and so on) corresponds to a field in this SWIFT message block. For further information on this SWIFT message block, see Appendix C, "SWIFT message structure," on page 115, and *All Things SWIFT: the SWIFT User Handbook*. Specify Type `String` for each named attribute. Note that there is no attribute application-specific information for the components of this business object.

Note: Be sure to specify the correct MaxLength values for the attribute names in this fixed-length block business definition.



	Pos	Name	Type	Key	Reqd	Card	Max L	App Spec Info	Comments
2	2	⊟ Swift_01Header	Swift_BasicHeader	□	□	1		block=1;parse=fixle	
2.1	2.1	BlockIdentifier	String	☑	☑		2		
2.2	2.2	ApplicationIdentifie	String	□	☑		1		
2.3	2.3	ServiceIdentifier	String	□	☑		2		
2.4	2.4	LTIdentifier	String	□	☑		12		
2.5	2.5	SessionNumber	String	□	☑		4		
2.6	2.6	SequenceNumber	String	□	□		6		
2.7	2.7	ObjectEventId	String						

Figure 7. Block 1 business object definition

Note: To create a block 1 business object definition for a SWIFT message, start Business Object Designer Express and then enter values for the attributes shown in "Sample block 1 business object definition" on page 55.

Sample block 1 business object definition

This section presents a sample definition of a block 1 business object for a SWIFT message of type MT502—an order to buy or sell.

```
[BusinessObjectDefinition]
Name = Swift_BasicHeader
Version = 1.1.0

   [Attribute]
   Name = BlockIdentifier
   Type = String
   Cardinality = 1
   MaxLength = 2
   IsKey = true
   IsForeignKey = false
   IsRequired = true
```

```
DefaultValue = 1:
IsRequiredServerBound = false
[End]
[Attribute]
Name = ApplicationIdentifier
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
IsRequiredServerBound = false
[End]
[Attribute]
Name = ServiceIdentifier
Type = String
Cardinality = 1
MaxLength = 2
IsKey = false
IsForeignKey = false
IsRequired = true
IsRequiredServerBound = false
[End]
[Attribute]
Name = LTIdentifier
Type = String
Cardinality = 1
MaxLength = 12
IsKey = false
IsForeignKey = false
IsRequired = true
IsRequiredServerBound = false
[End]
[Attribute]
Name = SessionNumber
Type = String
Cardinality = 1
MaxLength = 4
IsKey = false
IsForeignKey = false
IsRequired = true
IsRequiredServerBound = false
[End]
[Attribute]
Name = SequenceNumber
Type = String
Cardinality = 1
MaxLength = 6
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
Cardinality = 1
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

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

Block 2 business object structure

The block 2 MsgBlk BO, `Swift_ApplicationHeader`, has the format and attributes shown in Table 24.. The SWIFT data handler converts each of the SWIFT fields in this block into attributes in the `Swift_ApplicationHeader` business object. Note that there is no attribute application-specific information for this business object.

Note: Only attribute properties of consequence are shown in Table 24. For a listing of all attribute properties, see "Sample block 2 business object definition" on page 58.

Table 24. Block 2 business object structure

Name	Type	Key	Required	Cardinality	Default	Max length
Block Identifier	String	No	Yes	1	2:[a]	2
IOIdentifier	String	No	Yes	1		1
MessageType	String	No	Yes	1		3
I_ReceiverAddress	String	No	Yes	1		12
I_MessagePriority	String	No	Yes	1		1
I_DeliveryMonitoring	String	No	No	1		1
I_ObsolescencePeriod	String	No	No	1		3
O_InputTime	String	No	Yes	1		4
O_MessageInputReference	String	No	Yes	1		28
O_OutputDate	String	No	No	1		6
O_OutputMessagePriority	String	No	No	1		6

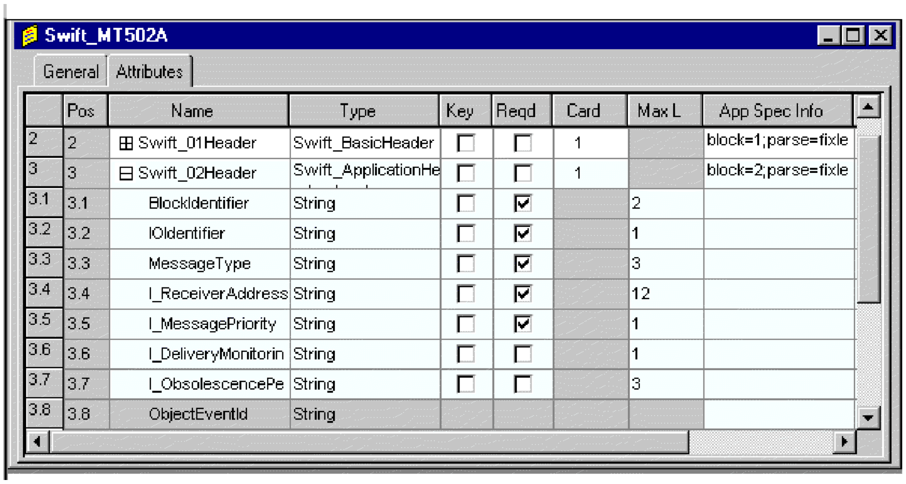
[a] The BlockIdentifier attribute includes the delimiter ":" as in "`2:`".

The first three attributes in Table 24 are I/O attributes. Attributes that start with I_ are input attributes and are populated during SWIFT-to-business-object conversion. Attributes that start with O_ are output attributes and are populated in business-object-to-SWIFT conversions. The CxIgnore property must be set for business-object-to-SWIFT conversions.

See Table 21 for the attribute application-specific information.

Figure 8 shows a block 2 business object definition that has been manually created in a WebSphere development environment. Each attribute name (`BlockIdentifier,` `IOIdentifier`, and so on) corresponds to a field in this SWIFT message block. The definition shown is for the input attributes (`I_`) populated during SWIFT-to-business-object conversion. For further information on this SWIFT message block, see Appendix C, "SWIFT message structure," on page 115, and *All Things SWIFT: the SWIFT User Handbook.* Specify type `String` for each named attribute. Note that there is no attribute application-specific information for the components of this business object.

Note: Be sure to specify the correct MaxLength values for the attribute names in this fixed-length block business definition.



	Pos	Name	Type	Key	Reqd	Card	Max L	App Spec Info
2	2	⊞ Swift_01Header	Swift_BasicHeader	□	□	1		block=1;parse=fixle
3	3	⊟ Swift_02Header	Swift_ApplicationHe	□	□	1		block=2;parse=fixle
3.1	3.1	BlockIdentifier	String	□	☑		2	
3.2	3.2	IOIdentifier	String	□	☑		1	
3.3	3.3	MessageType	String	□	☑		3	
3.4	3.4	I_ReceiverAddress	String	□	☑		12	
3.5	3.5	I_MessagePriority	String	□	☑		1	
3.6	3.6	I_DeliveryMonitorin	String	□	□		1	
3.7	3.7	I_ObsolescencePe	String	□	□		3	
3.8	3.8	ObjectEventId	String					

Figure 8. Block 2 business object definition

Note: To create a block 2 business object definition for a SWIFT message, start Business Object Designer Express and then enter values for the attributes shown in “Sample block 2 business object definition” on page 58.

Sample block 2 business object definition

This section presents a sample definition of a block 2 business object for a SWIFT message of type MT502—an order to buy or sell.

```
[BusinessObjectDefinition]
Name = Swift_ApplicationHeader
Version = 1.1.0

   [Attribute]
   Name = BlockIdentifier
   Type = String
   Cardinality = 1
   MaxLength = 2
   IsKey = false
   IsForeignKey = false
   IsRequired = true
   DefaultValue = 2:
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = IOIdentifier
   Type = String
   Cardinality = 1
   MaxLength = 1
   IsKey = false
   IsForeignKey = false
   IsRequired = true
   DefaultValue = O
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = MessageType
   Type = String
   Cardinality = 1
   MaxLength = 3
   IsKey = false
   IsForeignKey = false
   IsRequired = true
   IsRequiredServerBound = false
   [End]
   [Attribute]
```

```
Name = O_InputTime
Type = String
Cardinality = 1
MaxLength = 4
IsKey = false
IsForeignKey = false
IsRequired = true
IsRequiredServerBound = false
[End]
[Attribute]
Name = O_MessageInputReference
Type = String
Cardinality = 1
MaxLength = 28
IsKey = false
IsForeignKey = false
IsRequired = true
IsRequiredServerBound = false
[End]
[Attribute]
Name = O_OutputDate
Type = String
Cardinality = 1
MaxLength = 6
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = O_OutputTime
Type = String
Cardinality = 1
MaxLength = 4
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = O_OutputMessagePriority
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = I_ReceiverAddress
Type = String
Cardinality = 1
MaxLength = 12
IsKey = false
IsForeignKey = false
IsRequired = true
IsRequiredServerBound = false
[End]
[Attribute]
Name = I_MessagePriority
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
```

```
IsRequiredServerBound = false
[End]
[Attribute]
Name = I_DeliveryMonitoring
Type = String
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = I_ObsolescencePeriod
Type = String
Cardinality = 1
MaxLength = 3
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
Cardinality = 1
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

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

Block 3 business object structure

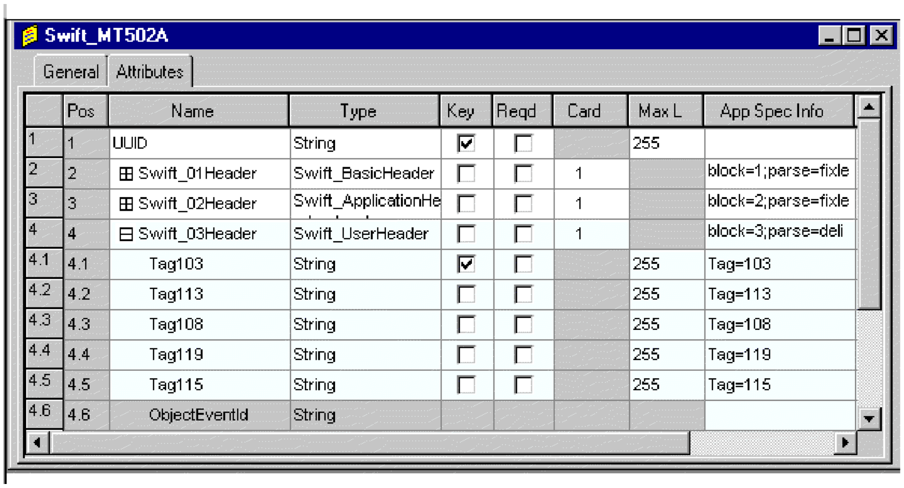
The block 3 MsgBlk BO, `Swift_UserHeader`, has the format and attributes shown in Table 25. Note that there is attribute application-specific information for this business object: the Tag parameter. For Tag parameters see Table 21.

Note: Only attribute properties of consequence are shown in Table 25. For a listing of all attribute properties, see "Sample block 3 business object definition" on page 61.

Table 25. Block 3 business object structure

Name	Type	Key	Foreign	Required	Cardinality	Application specific information	Max length
Tag103	String	Yes	No	No	1	Tag=103	6
Tag113	String	No	No	No	1	Tag=113	6
Tag108	String	No	No	No	1	Tag=108	6
Tag119	String	No	No	No	1	Tag=119	6
Tag115	String	No	No	No	1	Tag=115	6

Figure 9 shows a block 3 business object definition that has been manually created in a WebSphere development environment. Each attribute name (`Tag103`, `Tag113`, and so on,) corresponds to a field in this SWIFT message block. For further information on this SWIFT message block, see Appendix C, "SWIFT message structure,” on page 115, and *All Things SWIFT: the SWIFT User Handbook.* Specify type String for each named attribute. Note that the application-specific information for the components of this business object are SWIFT tags.



	Pos	Name	Type	Key	Reqd	Card	Max L	App Spec Info
1	1	UUID	String	☑	□		255	
2	2	⊞ Swift_01Header	Swift_BasicHeader	□	□	1		block=1;parse=fixle
3	3	⊞ Swift_02Header	Swift_ApplicationHe	□	□	1		block=2;parse=fixle
4	4	⊟ Swift_03Header	Swift_UserHeader	□	□	1		block=3;parse=deli
4.1	4.1	Tag103	String	☑	□		255	Tag=103
4.2	4.2	Tag113	String	□	□		255	Tag=113
4.3	4.3	Tag108	String	□	□		255	Tag=108
4.4	4.4	Tag119	String	□	□		255	Tag=119
4.5	4.5	Tag115	String	□	□		255	Tag=115
4.6	4.6	ObjectEventId	String					

Figure 9. Block 3 business object definition

Note: To create a block 3 business object definition for a SWIFT message, start Business Object Designer Express and then enter values for the attributes shown in “Sample block 3 business object definition” on page 61.

Sample block 3 business object definition

This section presents a sample definition of a block 3 business object for a SWIFT message of type MT502—an order to buy or sell.

```
[BusinessObjectDefinition]
Name = Swift_UserHeader
Version = 1.1.0

   [Attribute]
   Name = Tag103
   Type = String
   Cardinality = 1
   MaxLength = 255
   IsKey = true
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = Tag=103
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = Tag113
   Type = String
   Cardinality = 1
   MaxLength = 255
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = Tag=113
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = Tag108
   Type = String
   Cardinality = 1
   MaxLength = 255
   IsKey = false
```

```
IsForeignKey = false
IsRequired = false
AppSpecificInfo = Tag=108
IsRequiredServerBound = false
[End]
[Attribute]
Name = Tag119
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = Tag=119
IsRequiredServerBound = false
[End]
[Attribute]
Name = Tag115
Type = String
Cardinality = 1
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = Tag=115
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
Cardinality = 1
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

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

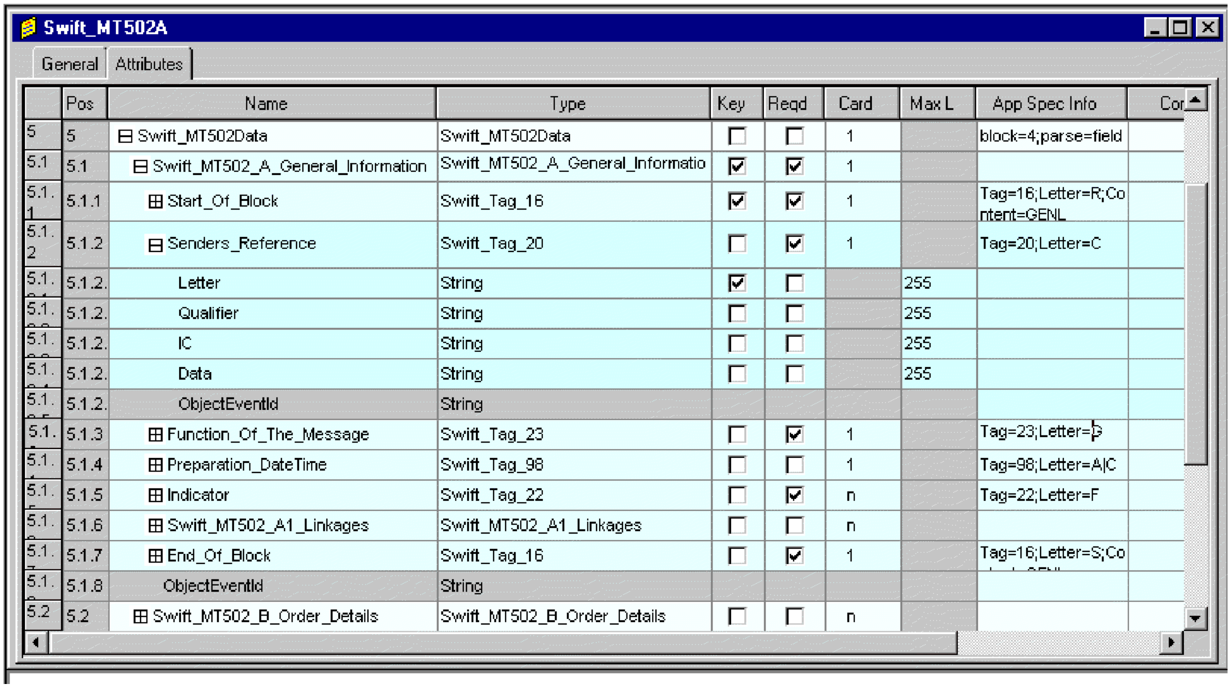
Block 4 business object structure

SWIFT block 4 contains the body of the SWIFT message. Block 4 is made up of fields of message tags and their contents on the one hand, and on the other, of sequences of message tags. This data content makes the block 4 business object structure unlike that of blocks 1, 2, and 3. The block 4 business object is the message data business object (MsgData BO).

Every tag and sequence in a SWIFT message is modeled as a child business object of the MsgData BO. Accordingly, a MsgData BO has child objects of two types: field business objects (MsgField BO) and sequence business objects (MsgSeq BO). These business objects reflect how the SWIFT data is formatted in block 4. More specifically, attributes in these business objects model the content (message tags and their content) and order (sequence) that is specified in a SWIFT message format specification. The sequence of the message tags is crucial if the business object definition is to faithfully represent the SWIFT message. For further information on MsgField BOs and MsgSeq BOs, see “Sequence and field business objects” on page 66.

As an example, view the format specification from the *SWIFT Standards Release Guide* for MT502, an order to buy or sell.Figure 10 below shows the portion of a business object definition that corresponds to MT502. The business object definition reflects the structure of the message tags and sequences in the SWIFT message:

- The Status—M (mandatory) or O (optional)—field in the SWIFT message is mapped to the Required property in the business object definition. For example, the status of SWIFT Tag 98a (shown in Figure 10) is O or optional; Figure 10 shows the corresponding business object attribute, Preparation_DateTime (of type `Swift_Tag_98`), for which the Required property is not checked.
- The Tag, Qualifier, and Content/Options fields from the SWIFT message are mapped as attribute application-specific text in the business object definition. For example, in the SWIFT message shown in Figure 10 Start of Block is Tag16R with Content of GENL. The corresponding entry shown in Figure 10 is the attribute `Start_Of_Block` of type `Swift_Tag_16` with application-specific information property parameters that identify the Tag, the Tag's letter, and Content (`Tag=16;Letter=R;Content=GENL`).
- Data formats are often indicated in the Content/Options field in a SWIFT message. For example, Figure 10 shows the sender's reference for "Mandatory Sequence A General Information" as Tag20C, with a SEME qualifier and Content consisting of data format instructions (`:4!c[/4!c]`). Figure 10 shows the corresponding attribute application-specific text: only the Tag and Letter are shown in the AppSpecInfo field (`Tag=20;Letter=C`).
- Repeating sequences in SWIFT messages are indicated by "---->" in the SWIFT Format Specifications as shown in Figure 10. Non-repeating sequences are marked "-----|". In the business object definition, a repeating sequence is assigned cardinality n. For example, the repeating sequence Tag22F shown in Figure 10 is mapped to the attribute Indicator of type `Swift_Tag_22` with a cardinality property of n.



	Pos	Name	Type	Key	Reqd	Card	Max L	App Spec Info	Cor
5	5	⊟ Swift_MT502Data	Swift_MT502Data	□	□	1		block=4;parse=field	
5.1	5.1	⊟ Swift_MT502_A_General_Information	Swift_MT502_A_General_Informatio	☑	☑	1			
5.1. 1	5.1.1	⊞ Start_Of_Block	Swift_Tag_16	☑	☑	1		Tag=16;Letter=R;Co ntent=GENL	
5.1. 2	5.1.2	⊟ Senders_Reference	Swift_Tag_20	□	☑	1		Tag=20;Letter=C	
5.1.	5.1.2.	Letter	String	☑	□		255		
5.1.	5.1.2.	Qualifier	String	□	□		255		
5.1.	5.1.2.	IC	String	□	□		255		
5.1.	5.1.2.	Data	String	□	□		255		
5.1.	5.1.2.	ObjectEventId	String						
5.1.	5.1.3	⊞ Function_Of_The_Message	Swift_Tag_23	□	☑	1		Tag=23;Letter=G	
5.1.	5.1.4	⊞ Preparation_DateTime	Swift_Tag_98	□	□	1		Tag=98;Letter=A|C	
5.1.	5.1.5	⊞ Indicator	Swift_Tag_22	□	☑	n		Tag=22;Letter=F	
5.1.	5.1.6	⊞ Swift_MT502_A1_Linkages	Swift_MT502_A1_Linkages	□	□	n			
5.1.	5.1.7	⊞ End_Of_Block	Swift_Tag_16	□	☑	1		Tag=16;Letter=S;Co	
5.1.	5.1.8	ObjectEventId	String						
5.2	5.2	⊞ Swift_MT502_B_Order_Details	Swift_MT502_B_Order_Details	□	□	n			

Figure 10. Partial block 4 business object definition

MsgData BO format

The format of a MsgData BO is summarized in the sections below.

MsgData BO name: The naming convention for the MsgData BO representing block 4 of a SWIFT message is as follows:

```
Swift_MT<message_type>Data
```

For example:

```
Name = Swift_MT502Data
```

MsgData BO attribute names: Each attribute of the MsgData BO represents one of the following:

- a MsgSeq BO
- a MsgField BO

Accordingly, the attribute names are the same as those for MsgSeq BOs and MsgField BOs. The naming convention for MsgField BO attributes is as follows:

```
Swift_<tag_number>_<position_in_the_SWIFT_message>
```

For example:

```
Name = Swift_94_1
```

The naming convention for MsgSeq BO attributes is as follows:

```
Swift_MT<message_type>_<SWIFT_sequence_name>
```

For example:

```
Name = Swift_MT502_B
```

For further information see “Sequence business object structure” on page 67 and “Field business object definitions” on page 70.

MsgData BO attribute types: The type for MsgData attributes is as follows:

For MsgField BO attributes:

```
Swift_Tag_<tag_number>
```

For example:

```
Type = Swift_Tag_94
```

For MsgSeq BO attributes:

```
Swift_MT<message_type>_<SWIFT_sequence_name>
```

For example:

```
Type = Swift_MT502_B
```

MsgData BO attribute ContainedObjectVersion: The contained object version for the MsgData BO as well as for the its MsgSeq BO attributes is 1.1.0. For example:

```
[Attribute]
Name = Swift_MT502_B
Type = Swift_MT502_B

...

ContainedObjectVersion = 1.1.0

...

[End]
```

Note: MsgField BO attributes are simple, and have no `ContainedObjectVersion`.

MsgData BO attribute relationship: The relationship attribute property for MsgData BO and its MsgSeq BO attributes is `Containment`. For example:

```
[Attribute]
Name = Swift_MT502Data
Type = Swift_MT502Data

...

Relationship = Containment

...

[End]
```

MsgData BO attribute cardinality: The MsgData BO and its MsgSeq BO attributes have a cardinality property of `n`. MsgField BO attributes that represent repeating fields also have cardinality `n`. All others attributes have cardinality 1. For example:

```
[Attribute]
Name = Swift_16_1
Type = Swift_Tag_16

...

Cardinality = n

...

[End]
```

MsgData BO attribute IsKey: Each MsgData BO definition must contain at least one attribute defined as the key attribute (`IsKey = true`). The rule is that the first single cardinality attribute in each BO definition must be defined as key attribute.

For example:

```
[Attribute]
Name = Swift_16.1
Type = Swift_Tag_16

...

Cardinality = 1

IsKey = true

[End]
```

MsgData BO attribute AppSpecificInfo: In MsgData BO definitions, only MsgField BO attributes have application-specific information; this property is always null for MsgSeq BO attributes. The convention for application-specific information for MsgField BO attributes is as follows:

```
   Tag=nn;Letter=xx;Content=string
```

where `nn` is the SWIFT tag number of the field, *`xx`* is one or a list of supported letter options for the tag, and *`string`* is the value of the qualifier for a non-generic field as described in Table 21 on page 49. For example:

```
[Attribute]
Name = Swift_16_22
Type = Swift_Tag_16
```

```
...

AppSpecificInfo = Tag=16;Letter=S;Content=OTHRPRTY

...

[End]
```

When MsgField BO attributes appear in MsgSeq BOs and the application specific information indicates:

```
...;Union=True
```

The MsgField child object—a TagUnion business object and its child objects, TagLetterOption objects—will be populated instead of the DataField attribute. For information on TagUnion business objects, see "Field business object definitions" on page 70.

Sequence and field business objects

As noted above, the connector models sequences and tags in SWIFT messages as sequence business objects (MsgSeq BO) and field business objects (MsgField BO), respectively. Figure 11 illustrates the hierarchical relationship of these business objects.

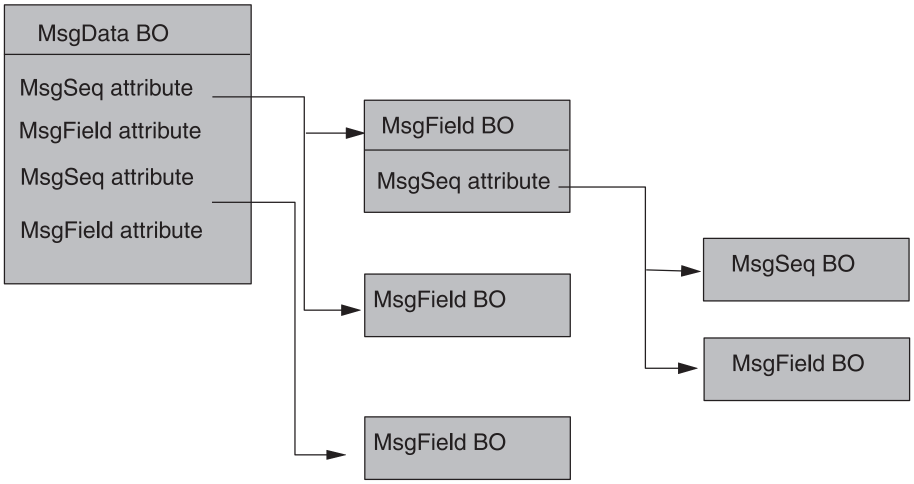
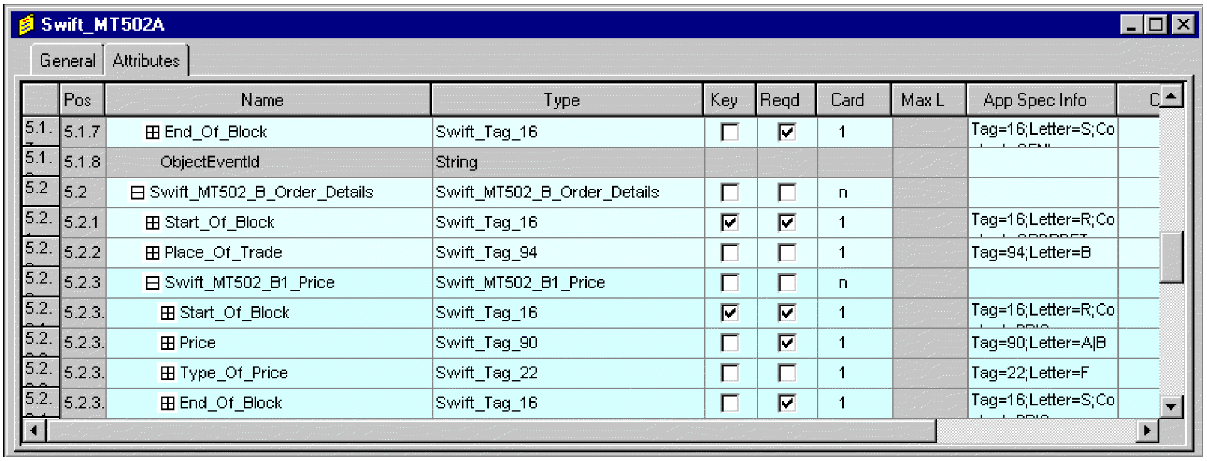


Figure 11. Field and sequence business objects in the (block 4) MsgData BO

Figure 12 shows part of a definition for a SWIFT message (MT502) that illustrates a sequence containing field and sequence attributes. The sequence attribute `Swift_MT02_B_Order_Details` not only includes several attributes of type Tag (for example, `Swift_Tag_16`, `Swift_Tag_94`), but also the subsequence `Swift_MT502_B1_Price`. This subsequence is a repeating optional sequence, and its properties reflect this (`Required= no`; `Cardinality=n`). Note that the sequences contain no application-specific information.



Swift_MT502A

General | Attributes

	Pos	Name	Type	Key	Reqd	Card	Max L	App Spec Info	C
5.1.	5.1.7	⊞ End_Of_Block	Swift_Tag_16	□	☑	1		Tag=16;Letter=S;Co	
5.1.	5.1.8	ObjectEventId	String						
5.2	5.2	⊟ Swift_MT502_B_Order_Details	Swift_MT502_B_Order_Details	□	□	n			
5.2.	5.2.1	⊞ Start_Of_Block	Swift_Tag_16	☑	☑	1		Tag=16;Letter=R;Co	
5.2.	5.2.2	⊞ Place_Of_Trade	Swift_Tag_94	□	□	1		Tag=94;Letter=B	
5.2.	5.2.3	⊟ Swift_MT502_B1_Price	Swift_MT502_B1_Price	□	□	n			
5.2.	5.2.3.	⊞ Start_Of_Block	Swift_Tag_16	☑	☑	1		Tag=16;Letter=R;Co	
5.2.	5.2.3.	⊞ Price	Swift_Tag_90	□	☑	1		Tag=90;Letter=A|B	
5.2.	5.2.3.	⊞ Type_Of_Price	Swift_Tag_22	□	□	1		Tag=22;Letter=F	
5.2.	5.2.3.	⊞ End_Of_Block	Swift_Tag_16	□	☑	1		Tag=16;Letter=S;Co	

Figure 12. A Sequence containing tag and subsequence attributes

Sequence business object structure

As shown in Figure 13, each sequence business object (MsgSeq BO) attribute indicates one of the following:

- another MsgSeq BO, or subsequence
- a MsgField BO

There is no limit to the number of subsequences that a MsqSeq BO can nest.

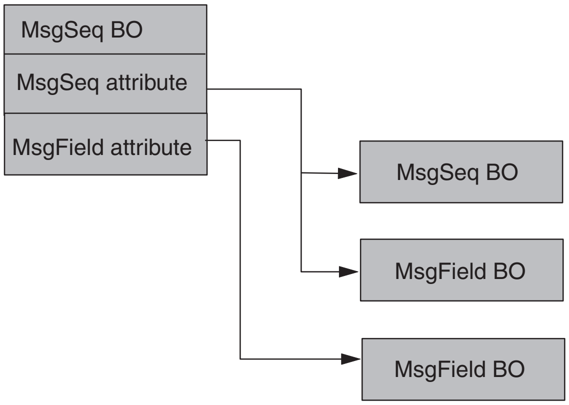
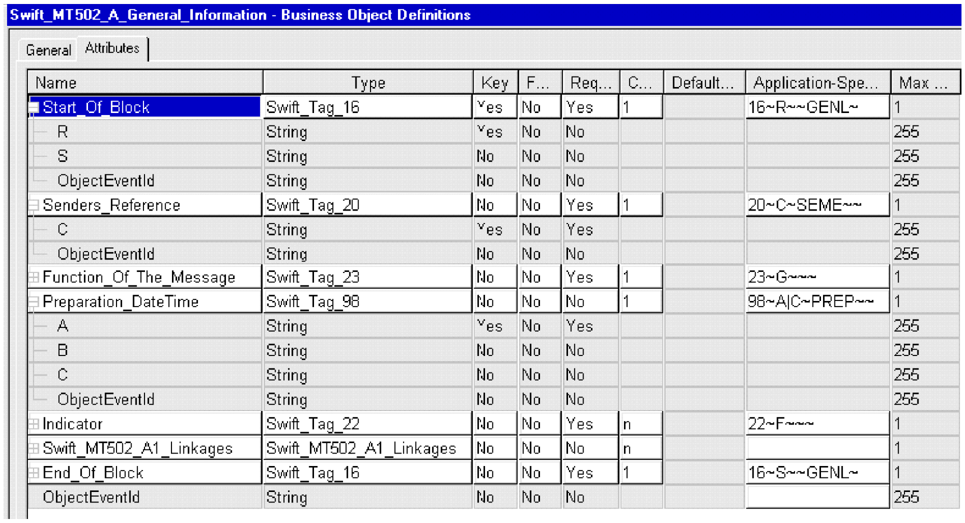


Figure 13. Field and Subsequence Business Objects in the MsgSeq BO

Figure 14 shows another excerpt of a MsgSeq BO. In this excerpt, the `Swift_Tag_` attributes represent MsgField BOs. The `Swift_MT502_A1_Linkages` attribute is for a child object that is a subsequence MsgSeq BO.



Swift_MT502_A_General_Information - Business Object Definitions

Name	Type	Key	F...	Req...	C...	Default...	Application-Spe...	Max ...
Start_Of_Block	Swift_Tag_16	Yes	No	Yes	1		16~R~~GENL~	1
R	String	Yes	No	No				255
S	String	No	No	No				255
ObjectEventId	String	No	No	No				255
Senders_Reference	Swift_Tag_20	No	No	Yes	1		20~C~SEME~~	1
C	String	Yes	No	Yes				255
ObjectEventId	String	No	No	No				255
Function_Of_The_Message	Swift_Tag_23	No	No	Yes	1		23~G~~~	1
Preparation_DateTime	Swift_Tag_98	No	No	No	1		98~A|C~PREP~~	1
A	String	Yes	No	Yes				255
B	String	No	No	No				255
C	String	No	No	No				255
ObjectEventId	String	No	No	No				255
Indicator	Swift_Tag_22	No	No	Yes	n		22~F~~~	1
Swift_MT502_A1_Linkages	Swift_MT502_A1_Linkages	No	No	No	n			1
End_Of_Block	Swift_Tag_16	No	No	Yes	1		16~S~~GENL~	1
ObjectEventId	String	No	No	No				255

Figure 14. Excerpt from a sequence business object (MsgSeq BO)

The following rules apply to sequence business objects:

- A subsequence business object is an attribute of a particular sequence business object type.
- A collection of more than one repeating field is treated as a subsequence.
- The application-specific information of a sequence attribute is always `NULL`.

For a sample sequence business object, see "Sample sequence business object definition" on page 68.

MsgSeq BO format

Like a MsgData BO, a MsgSeq BO consists of attributes that are either MsgSeq BOs or MsgField BOs. For information on the format of these attributes, see "MsgData BO format" on page 63.

Sample sequence business object definition

This section presents a sample definition of a MsgSeq BO for a SWIFT message of type MT502—an order to buy or sell. The definition is of a Mandatory Sequence A Order to Buy or Sell.

```
[BusinessObjectDefinition]
Name = Swift_MT502_A_General_Information
Version = 1.0.0

   [Attribute]
   Name = Start_Of_Block
   Type = Swift_Tag_16
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 1
   IsKey = true
   IsForeignKey = false
   IsRequired = true
   AppSpecificInfo = Tag=16;Letter=R;Content=GENL
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = Senders_Reference
   Type = Swift_Tag_20
   ContainedObjectVersion = 1.0.0
```

```
Relationship = Containment
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = Tag=20;Letter=C
IsRequiredServerBound = false
[End]
[Attribute]
Name = Function_Of_The_Message
Type = Swift_Tag_23
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = Tag=23;Letter=G
IsRequiredServerBound = false
[End]
[Attribute]
Name = Preparation_DateTime
Type = Swift_Tag_98
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
AppSpecificInfo = Tag=98;Letter=A|C
IsRequiredServerBound = false
[End]
[Attribute]
Name = Indicator
Type = Swift_Tag_22
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = n
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = Tag=22;Letter=F
IsRequiredServerBound = false
[End]
[Attribute]
Name = Swift_MT502_A1_Linkages
Type = Swift_MT502_A1_Linkages
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = n
MaxLength = 1
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]
[Attribute]
Name = End_Of_Block
Type = Swift_Tag_16
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 1
```

```
IsKey = false
IsForeignKey = false
IsRequired = true
AppSpecificInfo = Tag=16;Letter=S;Content=GENL
IsRequiredServerBound = false
[End]
[Attribute]
Name = ObjectEventId
Type = String
MaxLength = 255
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Verb]
Name = Create
[End]

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

Field business object definitions

WebSphere represents every SWIFT tag as a field business object (MsgField BO). Each MsgField BO is modeled using the SWIFT generic field structure, even if the field is non-generic. WebSphere uses two additional business object models to represent the combination of letters and options used to represent and combine SWIFT message components as subfields in business objects:

- **Tag union business object (TagUnion BO)** This is a child object of the MsgField BO. A TagUnion BO contains all possible letter options for a specific tag, and is not specific to a particular message type.
- **Tag letter option business object (TagLetterOption BO)** This is a letter option child object of the TagUnion BO that defines the content of the subfield as well as its format including delimiters.

MsgField BO format

As shown in Figure 15, each MsgField BO contains five attributes, including one and only one TagUnion BO, with the data type shown in parentheses () below:

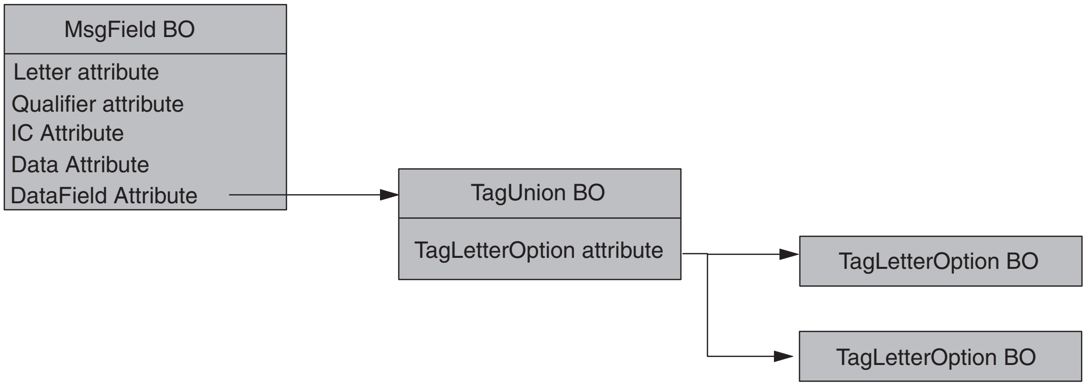


Figure 15. Attributes and business objects in the MsgField BO

The content and order of all subfields other than the SWIFT Qualifier and Issuer Code (IC) are captured in the child object of DataField, which is the TagUnion BO and its child objects, TagLetterOption BOs. The attributes and business objects shown in Figure 15 are discussed in the section below.

MsgField BO, TagUnion BO, and TagLetterOption BO names: The naming convention for a MsgField BO is as follows:

```
Swift_Tag_<N>
```

where N stands for the message number. For example:

```
Name = Swift_Tag_22
```

The naming convention for a TagUnion BO is as follows:

```
Swift_Tag_Union_<tag_number>
```

where `tag_number` is the numeric representation of tag number. For example:

```
Name = Swift_Tag_Union_20
```

The naming convention for a TagLetterOption BO is as follows:

```
Swift_Tag_Union_<tag_number>_Opt_[<letter_option>]
```

where `tag_number` is the numeric representation of tag number and `[<letter_option>]` is the letter option when a tag is associated with a letter. If the tag has no letter associated with it, then the name ends at `Opt`.For example:

```
Name = Swift_Tag_Union_20_Opt_C
```

MsgField BO, TagUnion BO, and TagLetter BO Attribute names: The names of the five attributes in a MsgField BO are as follows:

- `Letter`
- `Qualifier`
- `IC`
- `Data`
- `DataField`

The names of attributes in TagUnion BOs are as follows:

```
Swift_<tag_number>_[<letter_option>]
```

where `tag_number` is the numeric representation of the tag number and the square brackets signify that the letter is appended only when it is associated with the tag. For example:

```
Swift_20_C
```

The name of the attribute in TagLetterOption BOs is the concatenation of words in the subfield name shown in the SWIFT format specification table. The first letter of each word in the concatenated string is always capitalized, with subsequent letters in the word appearing in lowercase, regardless of how the words are spelled in the SWIFT format specification. Spaces and non-alphabetic symbols are left out of the concatenated name. If a field has no subfield, the word `Subfield` is used as an attribute name. For example, for the subfield “Proprietary Code” in 95R, the corresponding attribute name in the definition of TagLetterOption BO `Swift_Tag_Union_95_Opt_R` is as follows:

```
Name = ProprietaryCode
```

MsgField BO, TagUnion BO, and TagLetterOption BO attribute types: The type for MsgField attributes is as follows:

- Letter (String)
- Qualifier (String)
- Issuer Code (String)
- Data (String)
- DataField (*TagUnion_BO*)

For example, in a MsgField BO definition, the type for a Swift_Tag_20 attribute would be listed as follows:

```
[Attribute]
Name = DataField
Type = Swift_Tag_Union_20
```

The type for attributes in the TagUnion BO is the name of the TagLetterOption BO child object. For example, in a TagUnion BO definition for Swift_Tag_Union_20, the type for the TagLetterOption attribute is as follows:

```
[Attribute]
Name = Swift_20_C
Type = Swift_Tag_Union_20_Opt_C
```

The type for attributes in TagLetterOption BOs is always String.

MsgField BO, TagUnion BO, and TagLetterOption BO ContainedObjectVersion: The contained object version for the MsgField BO, the TagUnion BO, and the TagLetterOption BO is 1.1.0. For example:

as well as for the its MsgSeq BO attributes is 1.1.0. For example:

```
[Attribute]
Name = Swift_20_C
Type = Swift_Tag_Union_20_Opt_C

...

ContainedObjectVersion = 1.1.0

...
[End]
```

Note: MsgField BO attributes are simple, and have no ContainedObjectVersion.

MsgField BO, TagUnion BO, and TagLetterOption BO attribute cardinality: The cardinality of attributes in TagUnion BOs and TagLetterOption BOs is always set to 1. For example:

```
[Attribute]
Name = Swift_20_C
Type = Swift_Tag_Union_20_Opt_C

...

Cardinality = 1

...

[End]
```

MsgField BO, TagUnion BO, and TagLetterOption BO attribute IsKey: In each MsgField BO, the attribute Letter must be defined as the key attribute.

For example:

```
[Attribute]

Name = Letter
Type = String
IsKey = true

...

[End]
```

The first attribute of a TagUnionBO is defined as key.

The first attribute of TagLetterOption BO is defined as key.

TagLetterOption BO attribute AppSpecificInfo: The `AppSpecificInfo` attribute definition of a TagLetterOption BO provides crucial SWIFT message formatting information for business object subfields. The `AppSpecificInfo` attribute must contain the following information:

```
   Format=***;Delim=$$$
```

where

*** stands for the SWIFT subfield format specification, which excludes delimiter information

$$$ stands for one or more letters that constitute the delimiter between the current subfield and the next subfield.

When the delimiters are CrLf, the symbol string `CrLf` specifies that a carriage return is immediately followed by a line feed.

For example, the `AppSpecificInfo` attribute for a TagLetterOption BO, `Swift_Tag_Union_95_Opt_C`, might appear as follows:

```
[Attribute]

Name = CountryCode
Type = String

...

AppSpecificInfo = Format=2!a;Delim=/

...

[End]
```

For a sample object and attribute definitions, see "Sample MsgField BO, TagUnion BO, and TagLetterOption BO definitions" on page 73.

Sample MsgField BO, TagUnion BO, and TagLetterOption BO definitions

This section presents a sample definition of a MsgField BO definition that illustrates TagUnion and TagLetterOption attributes and objects.

The sample MsgField BO, `Swift_Tag_21`, is as follows:

```
[BusinessObjectDefinition]
Name = Swift_Tag_21
Version = 3.0.0

   [Attribute]
   Name = Letter
```

```
Type = String
MaxLength = 255
IsKey = true
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

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

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

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

[Attribute]
Name = DataField
Type = Swift_Tag_Union_21
ContainedObjectVersion = 3.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

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

[Verb]
Name = Create
[End]

[Verb]
Name = Delete
```

```
   [End]

   [Verb]
   Name = Retrieve
   [End]

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

Note that the DataField attribute indicates a TagUnion BO, whose name is defined by the Type attribute, Swift_Tag_Union_21. Here is that TagUnion BO, which lists as attributes all the letter options for Swift_Tag_21.

```
[BusinessObjectDefinition]
Name = Swift_Tag_Union_21
Version = 1.1.0

   [Attribute]
   Name = Swift_21
   Type = Swift_Tag_Union_21_Opt
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 0
   IsKey = true
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = Swift_21_A
   Type = Swift_Tag_Union_21_Opt_A
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 0
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = Swift_21_B
   Type = Swift_Tag_Union_21_Opt_B
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 0
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = Swift_21_C
   Type = Swift_Tag_Union_21_Opt_C
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 0
   IsKey = false
   IsForeignKey = false
```

```
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = Swift_21_D
Type = Swift_Tag_Union_21_Opt_D
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = Swift_21_E
Type = Swift_Tag_Union_21_Opt_E
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = Swift_21_F
Type = Swift_Tag_Union_21_Opt_F
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = Swift_21_G
Type = Swift_Tag_Union_21_Opt_G
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
[End]

[Attribute]
Name = Swift_21_N
Type = Swift_Tag_Union_21_Opt_N
ContainedObjectVersion = 1.0.0
Relationship = Containment
Cardinality = 1
MaxLength = 0
IsKey = false
IsForeignKey = false
IsRequired = false
IsRequiredServerBound = false
```

```
   [End]

   [Attribute]
   Name = Swift_21_P
   Type = Swift_Tag_Union_21_Opt_P
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 0
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = Swift_21_R
   Type = Swift_Tag_Union_21_Opt_R
   ContainedObjectVersion = 1.0.0
   Relationship = Containment
   Cardinality = 1
   MaxLength = 0
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

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

   [Verb]
   Name = Create
   [End]

   [Verb]
   Name = Retrieve
   [End]

[End]
```

Note that `IsKey = true` for the first attribute in the TagUnion BO above, `Swift_21`.

The attribute `Swift_21_A` indicates a child object TagLetterOption BO. This child object's name is defined by the attribute's `Type` attribute, `Swift_Tag_Union_21_Opt_A`. Here is that TagLetterOption BO:

```
[BusinessObjectDefinition]
Name = Swift_Tag_Union_21_Opt_A
Version = 1.0.0

   [Attribute]
   Name = ReferenceOfTheIndividualAllocation
   Type = String
   MaxLength = 255
   IsKey = true
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = Format=16x
   IsRequiredServerBound = false
   [End]

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

   [Verb]
   Name = Create
   [End]

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

Note that the only attribute of this TagLetterOption BO, ReferenceOfTheIndividualAllocation, is a concatenation of the corresponding SWIFT subfield name for this tag option, with the first letter of each word in uppercase. The Qualifier and Issuer Code subfields are excluded from the attribute of the TagLetterOption BOs. The IsKey property is also true for this attribute.

Note: A TagUnion BO contains both generic and non-generic fields. A non-generic field has no subfields.

The TagLetterOption BO can represent simple and complex SWIFT field and subfield formatting. Here is a business object definition for Swift_Tag_Union_22_Opt, a TagLetterOption BO whose attributes and application-specific information specify the subfield formatting for SWIFT Field 22, a function for a Common Reference between a sender and receiver. Notice that the AppSpecificInfo for Function specifies the format and the delimiter with which to parse the data in the SWIFT message. CommonReference is the concatenation of the subfield name. The AppSpecificInfo for CommonReference corresponds to that shown in Figure 16.

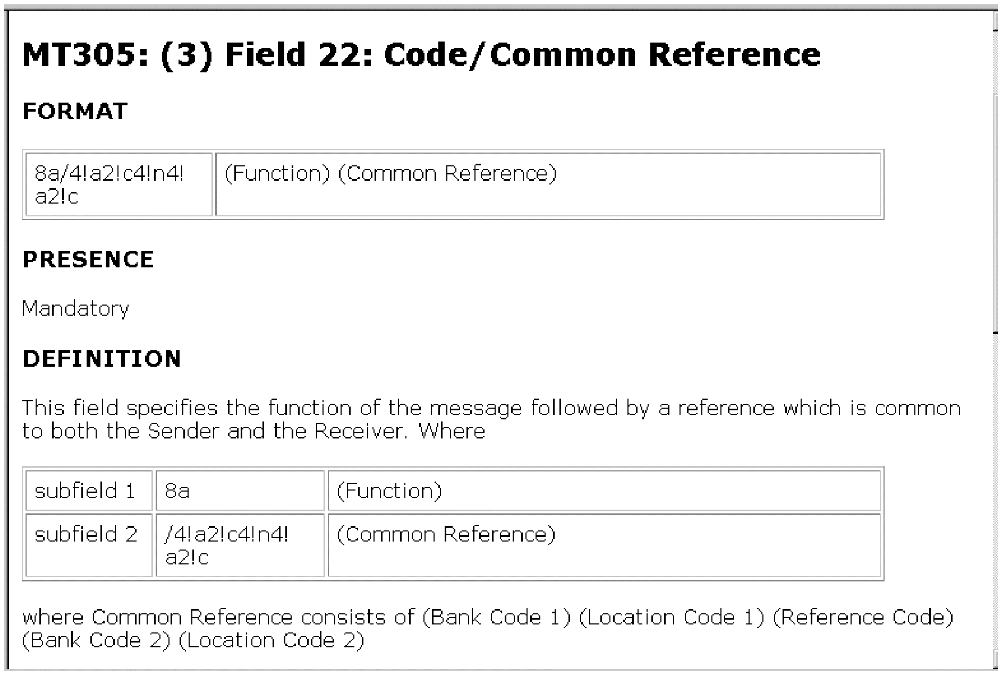
```
[BusinessObjectDefinition]
Name = Swift_Tag_Union_22_Opt
Version = 1.0.0

   [Attribute]
   Name = Function
   Type = String
   MaxLength = 255
   IsKey = true
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = Format=8a;Delim=/
   IsRequiredServerBound = false
   [End]

   [Attribute]
   Name = CommonReference
   Type = String
   MaxLength = 255
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   AppSpecificInfo = Format=4!a2!c4!n4!a2!c
   IsRequiredServerBound = false
   [End]
   [Attribute]
   Name = ObjectEventId
   Type = String
   MaxLength = 255
   IsKey = false
   IsForeignKey = false
   IsRequired = false
   IsRequiredServerBound = false
   [End]

   [Verb]
   Name = Create
   [End]

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



MT305: (3) Field 22: Code/Common Reference

FORMAT

8a/4!a2!c4!n4!a2!c	(Function) (Common Reference)

PRESENCE

Mandatory

DEFINITION

This field specifies the function of the message followed by a reference which is common to both the Sender and the Receiver. Where

subfield 1	8a	(Function)
subfield 2	/4!a2!c4!n4!a2!c	(Common Reference)

where Common Reference consists of (Bank Code 1) (Location Code 1) (Reference Code) (Bank Code 2) (Location Code 2)

Figure 16. SWIFT field definition

Chapter 4. SWIFT Data Handler

- "Configuring the SWIFT data handler"
- "Configuring the data handler child meta-object"
- "Converting business objects to SWIFT messages" on page 82
- "Converting SWIFT messages to business objects" on page 83

The SWIFT data handler is a data-conversion module whose primary roles are to convert business objects into SWIFT messages and SWIFT messages into business objects. Both default top-level data-handler meta-objects (connector and server) support the `swift` MIME type and therefore support use of the SWIFT data handler.

This chapter describes how the SWIFT data handler processes SWIFT messages. It also discusses how to configure the SWIFT data handler.

Configuring the SWIFT data handler

To configure a SWIFT data handler for use with the connector, you must do the following:

- Make sure that the class name of the SWIFT data handler is specified in the connector properties.
- Enter the appropriate values for the attributes of the SWIFT data handler child meta-object.

Note: For the SWIFT data handler to function properly, you must also create or modify business object definitions so that they support the data handler. For more information, see "SWIFT field structure" on page 115.

Configuring the connector meta-object

To configure the connector to interact with the SWIFT data handler, make sure that the connector-specific property `DataHandlerClassName` has the value `com.crossworlds.DataHandlers.swift.SwiftDataHandler`.

You must set the value of this property before running the connector. Doing so will enable the connector to access the SWIFT data handler when converting SWIFT messages to business objects and vice versa. For further information, see "Connector-specific properties" on page 20.

Configuring the data handler child meta-object

For the SWIFT data handler, WebSphere delivers the default meta-object `MO_DataHandler_Default`. This meta-object specifies a child attribute of type `MO_DataHandler_Swift`. Table 26 describes the attributes in the child meta-object, `MO_DataHandler_SWIFT`.

Table 26. Child Meta-Object Attributes for the SWIFT Data Handler

Attribute Name	Description	Delivered Default Value
BOPrefix	Prefix used by the default NameHandler class to build business object names. The default value must be changed to match the type of the business object. The attribute value is case-sensitive.	Swift
DefaultVerb	The verb used when creating business objects.	Create
ClassName	Name of the data handler class to load for use with the specified MIME type. The top-level data-handler meta-object has an attribute whose name matches the specified MIME type and whose type is the SWIFT child meta-object.	com.crossworlds. DataHandlers.swift. SwiftDataHandler
DummyKey	Key attribute; not used by the data handler but required by the integration broker.	1

The Delivered Default Value column in Table 26 lists the value that WebSphere provides for the default value of the associated meta-object attribute. You must ensure that all attributes in this child meta-object have a default value that is appropriate for your system and your SWIFT message type. Also, make sure that at least the ClassName and BOPrefix attributes have default values.

Note: Use Business Object Designer Express to assign default values to attributes in this meta-object.

Business object requirements

The SWIFT data handler uses business object definitions when it converts business objects or SWIFT messages. It performs the conversion using the structure of the business object and its application-specific text. To ensure that business object definitions conform to the requirements of the SWIFT data handler, follow the guidelines described in Chapter 3, "Business objects," on page 45.

Converting business objects to SWIFT messages

To convert a business object to a SWIFT message, the SWIFT data handler loops through the attributes in the top-level business object in sequential order. It generates populated blocks of a SWIFT message recursively based on the order in which attributes appear in the business object and its children.

Attributes without a block number, or with values unrecognized by the parser properties, are ignored. Also ignored is block 0, the UUID header that is added by the MQSA.

The parse=value application-specific information property is used to determine how to format strings. This property parses the business object as follows:

- parse=no; The attribute MUST be of type String and is formatted as {*block number*:*attribute value*}The *block number* is the value of the block=*block value* application-specific text property.
- parse=fixlen; The attribute must be a single cardinality container. It is formatted as {*block number*:*attr0 value* a*ttr1 value*....*attrn value*}where *attrn value* is the attribute value of the *n*th attribute. All CxIgnore and CxBlank attributes are IGNORED.
- parse=delim; The attribute must be a single cardinality container. It is formatted as {*block number*:[*Tag*:*attr1 data*]...[*Tag*:*attr1 data*]}where: *Tag* is the value of the Tag property of attribute application-specific text *attrn data* is the value of the attribute. All CxIgnore and CxBlank attributes are IGNORED.

- `parse=field;` This setting can be used only on Block 4 messages. Fields are printed out in loop through non-CxIgnore and non-CxBlank attributes of the business object.
  - If `appText == NULL` and the attribute is a container, call `printBO(childBO)`. Handle multiple cardinality if required.
  - If `appText != NULL`, call `printFieldObj()`, which handles multiple cardinality and calls `printFieldBO()` to write out a tag.
- All fields are formatted as generic or non-generic fields. The tag number is determined by the value of the Tag business object attribute. All non-CxIgnore attributes of the tag business object are printed out. For more on generic or non-generic fields, see Appendix C, "SWIFT message structure," on page 115.

Converting SWIFT messages to business objects

All SWIFT messages as well as compliance with SWIFT formats and syntax, are validated by SWIFT before being processed by the SWIFT data handler. The SWIFT data handler performs validation of business object structure and compliance only.

The SWIFT data handler extracts data from a SWIFT message and sets corresponding attributes in a business object as follows:

1. The SWIFT parser is called to extract the first 4 blocks (UUID + blocks 1 through 3). For block 2, the SWIFT application header, only the input attributes are extracted.
2. The SWIFT data handler is called to extract the name of the business object from block 2 of the SWIFT message.
3. The SWIFT data handler creates an instance of the top-level object.
4. Based on the application-specific information parameters, the data handler processes SWIFT message blocks. The blocks are parsed in one of four different ways
   - `parse=no;` The block data is treated as type `String` and not parsed out.
   - `parse=fixlen;` The block data is parsed as a fixed-length structure, based on the values of the maximum length attributes of the block business object.
   - `parse=delim;` The block data is parsed as {*n*:*data*} delimited format.
   - `parse=field;` This setting is used only on block 4 data. Fields are parsed as generic and non-generic.
5. For block 4 data (`parse=field;`) the data handler either matches the field returned from the parser to a tag business object attribute, or finds the sequence business object that the field belongs to.
   a. If the application specific information of the attribute is `NULL`, the child business object is a sequence. The data handler checks if the first required attribute of the child business object matches the field:
      - If it does match, the data handler assigns the attribute multiple cardinality and populates the sequence for the child business object.
      - If it does not match, the data handler skips to the next attribute of the parent business object.

   b. If application-specific information is not NULL, the child is a tag business object. If the field matches the application-specific information, it is handled with the multiple cardinality and extracted, with the data handler setting the letter and data attributes of the tag business object.
6. If a non-NULL field is returned, the field is written to a log and an exception is thrown.

7. The data handler parses block 5 of the SWIFT message. The application-specific information for this block is always `block=5; parse=no` and is of type `String`. Block 5 is treated as a single string.

Chapter 5. Troubleshooting

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

Startup problems

Problem	Potential solution / explanation
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/jms/JMSException...`	Connector cannot find file `jms.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: com/ibm/mq/jms/MQConnectionFactory...`	Connector cannot find file `com.ibm.mqjms.jar` in the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following message is reported: `Exception in thread "main" java.lang.NoClassDefFoundError: javax/naming/Referenceable...`	Connector cannot find file `jndi.jar` from the IBM WebSphere MQ Java client libraries. Ensure that variable `MQSERIES_JAVA_LIB` in `start_connector.bat` points to the IBM WebSphere MQ Java client library folder.
The connector reports `MQJMS2005: failed to create MQQueueManager for ':'`	Explicitly set values for the following properties: `HostName`, `Channel`, and `Port`.

Event processing

Problem	Potential solution / explanation
The connector delivers all messages with an MQRFH2 header.	To deliver messages with only the MQMD WebSphere MQ header, append `?targetClient=1` to the name of output queue URI. For example, if you output messages to queue `queue://my.queue.manager/OUT`, change the URI to `queue://my.queue.manager/OUT?targetClient=1`. See Chapter 2, “Installing and configuring the connector,” on page 17 for more information.
The connector truncates all message formats to 8 characters upon delivery regardless of how the format has been defined in the connector meta-object.	This is a limitation of the WebSphere MQ MQMD message header and not the connector.

Appendix A. Standard configuration properties for connectors

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

Not every connector makes use of all these standard properties. When you select an integration broker 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 the integration broker.

- **Server restart**
  The change takes effect only after you stop and restart the application-specific component and the integration broker.
- **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 27 provides a quick reference to the standard connector configuration properties. Not all the connectors make use of all these properties, and property settings may differ from integration broker to integration broker, as standard property dependencies are based on `RepositoryDirectory`.

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

Table 27. 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 IDL
AgentTraceLevel	0-5	0	Dynamic	
ApplicationName	Application name	Value specified for the connector application name	Component restart	
BrokerType	ICS	ICS		
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	truetrue	Dynamic	Repository directory is <REMOTE>

Table 27. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
ControllerTraceLevel	0-5	0	Dynamic	Repository directory is <REMOTE>
DeliveryQueue		*CONNECTORNAME*/DELIVERYQUEUE	Component restart	JMS transport only
DeliveryTransport	IDL or JMS	IDL	Component restart	
DuplicateEventElimination	true or false	false	Component restart	JMS transport only: Container Managed Events must be <NONE>
EnableOidForFlowMonitoring	true or false	false	Component restart	
FaultQueue		*CONNECTORNAME*/FAULTQUEUE	Component restart	JMS transport only
jms.FactoryClassName	CxCommon.Messaging.jms .IBMMQSeriesFactory or any Java class name	CxCommon.Messaging. jms.IBMMQSeriesFactory	Component restart	JMS transport only
jms.MessageBrokerName	crossworlds.queue. manager	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>
Locale	en_US, ja_JP, ko_KR, zh_CN, zh_TW, fr_FR, de_DE, it_IT, es_ES, pt_BR **Note:** This is a subset of the supported locales.	en_US	Component restart	
LogAtInterchangeEnd	true or false	false	Component restart	
MaxEventCapacity	1-2147483647	2147483647	Dynamic	Repository Directory is <REMOTE>
MessageFileName	Path or filename	InterchangeSystem.txt	Component restart	

Table 27. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
MonitorQueue	Any valid queue name	CONNECTORNAME/MONITORQUEUE	Component restart	JMS transport only: DuplicateEvent Elimination must be true
OADAutoRestartAgent	true or false	false	Dynamic	Repository Directory 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 is 0-23, MM is 0-59)	HH:MM	Component restart	
PollFrequency	A positive integer in milliseconds no (to disable polling) key (to poll only when the letter p is entered in the connector's Command Prompt window)	10000	Dynamic	
PollQuantity	1-500	1	Agent restart	JMS transport only: Container Managed Events is specified
PollStartTime	HH:MM(HH is 0-23, MM is 0-59)	HH:MM	Component restart	
RepositoryDirectory	Location of metadata repository		Agent restart	Set to <REMOTE>
RequestQueue	Valid JMS queue name	CONNECTORNAME/REQUESTQUEUE	Component restart	Delivery Transport is JMS
ResponseQueue	Valid JMS queue name	CONNECTORNAME/RESPONSEQUEUE	Component restart	Delivery Transport is JMS:
RestartRetryCount	0-99	3	Dynamic	
RestartRetryInterval	A sensible positive value in minutes: 1 - 2147483547	1	Dynamic	
SourceQueue	Valid JMS queue name	CONNECTORNAME/SOURCEQUEUE	Agent restart	Only if Delivery Transport is JMS and Container Managed Events is specified
SynchronousRequestQueue	Valid JMS queue name	CONNECTORNAME/ SYNCHRONOUSREQUESTQUEUE	Component restart	Delivery Transport is JMS

Table 27. Summary of standard configuration properties (continued)

Property name	Possible values	Default value	Update method	Notes
SynchronousRequestTimeout	0 - any number (millisecs)	0	Component restart	Delivery Transport is JMS
SynchronousResponseQueue	Valid JMS queue name	*CONNECTORNAME*/ SYNCHRONOUSRESPONSEQUEUE	Component restart	Delivery Transport is JMS
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 the integration broker to send administrative messages to the connector.

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

The default value is CONNECTORNAME/ADMINOUTQUEUE.

AgentConnections

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.

BrokerType

Identifies the integration broker that you are using, which is ICS.

CharacterEncoding

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

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

By default, a subset of supported character encodings only is displayed in the drop 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 using Connector Configurator Express in this guide.

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 the integration broker, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

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

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

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

ContainerManagedEvents

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

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

The default value is No value.

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

- PollQuantity = 1 to 500
- SourceQueue = CONNECTORNAME/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.

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 ICS, the connector controller blocks the request to the application-specific component. When the application-specific component becomes operational, the controller forwards the request to it.

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

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

The default is `true`.

ControllerTraceLevel

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

DeliveryQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default value is `CONNECTORNAME/DELIVERYQUEUE`.

DeliveryTransport

Specifies the transport mechanism for the delivery of events. Possible values are `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 `IDL`.

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 WebSphere 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.

EnableOidForFlowMonitoring

When you set this property to `true`, the adapter framework will mark the incoming **ObjectEventId** as a foreign key for the purpose of flow monitoring.

The default is `false`.

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).

The default value is 128m.

JvmMaxNativeStackSize

The maximum native stack size for the agent (in kilobytes).

The default value is 128k.

JvmMinHeapSize

The minimum heap size for the agent (in megabytes).

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 (see 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.

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 using Connector Configurator Express in this guide.

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

http://www.ibm.com/software/websphere/wbiadapters/infocenter, or
http://www.ibm.com/websphere/integration/wicserver/infocenter

LogAtInterchangeEnd

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

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

MaxEventCapacity

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

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

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

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

The default value is false.

OADMaxNumRetry

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

The default value is 1000.

OADRetryTimeInterval

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

The default is 10.

PollEndTime

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

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

PollFrequency

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 0-59 seconds.

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

RequestQueue

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

The default value is CONNECTOR/REQUESTQUEUE.

RepositoryDirectory

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

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

ResponseQueue

Applicable only if `DeliveryTransport` is JMS.

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

RestartRetryCount

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

The default is 3.

RestartRetryInterval

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

The default is 1.

SourceQueue

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

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see “ContainerManagedEvents” on page 92.

The default value is `CONNECTOR/SOURCEQUEUE`.

SynchronousRequestQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is `CONNECTORNAME/SYNCHRONOUSREQUESTQUEUE`

SynchronousResponseQueue

Applicable only if `DeliveryTransport` is JMS.

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

The default is `CONNECTORNAME/SYNCHRONOUSRESPONSEQUEUE`

SynchronousRequestTimeout

Applicable only if `DeliveryTransport` is JMS.

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

The default value is 0.

WireFormat

This is the 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.

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 WebSphere 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 maps for use with collaborations as well as specify messaging, logging and tracing, and data handler parameters, as required.

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

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

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

For **connector-specific properties**, however, you need first to define the properties and then set their values. You do this by creating a connector-specific property template for your particular adapter. There may already be a template set up in your system, in which case, you simply use that. If not, follow the steps in "Creating a new template" on page 103 to set up a new one.

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

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 and work with connector configuration files, irrespective of your broker.

To do so:

- From **Start>Programs**, click **IBM WebSphere Business Integration Server Express> Toolset Express>Development>Connector Configurator Express**.
- Select **File>New>Configuration File**.

You may choose to run Connector Configurator Express independently to generate the file, and then connect to System Manager to save it in a System Manager project (see "Completing a configuration file" on page 107.)

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.

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.
2. Click the Standard Properties tab to see which properties are included in this configuration file.

Creating a connector-specific property template

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

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

- To create a new template, see "Creating a new template" on page 103.
- To use an existing file, simply modify an existing template and save it under the new name.

Creating a new template

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

To create a template:

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

Specifying general characteristics

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

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

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

Specifying values

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

1. Click the **Value** tab. The display panel for Value replaces the display panel for General.

2. Select the name of the property in the **Edit properties** display.
3. In the fields for **Max Length** and **Max Multiple Values**, 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 top left-hand corner of the value table and click **Add**. 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 - Connector-Specific Property Template** dialog box appears.

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

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

   == (equal to)

   != (not equal to)

   > (greater than)

   < (less than)

   >= (greater than or equal to)

   <=(less than or equal to)
4. In the **Conditional Value** field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the **Available Properties** display, click an arrow to move it to the **Dependent Property** display.
6. Click **Finish**. Connector Configurator 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 configuration file:

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

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

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

     The default broker is ICS. You cannot change this value.
   - **Select Connector-Specific Property Template**

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

     Select the template you want to use and click **OK**.
3. A configuration screen appears for the connector that you are configuring. The title bar shows the integration broker and connector 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.
4. To save the file, click **File>Save>To File** or **File>Save>To Project**. To save to a project, System Manager must be running.
   If you save as a file, the **Save File Connector** dialog box appears. Choose `*.cfg` as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and click **Save**. The status display in the message panel of Connector Configurator 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.
5. 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 chapter.

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`).

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

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

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

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

1. In Connector Configurator 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.
   - All files (*.*)

     Choose this option if a `*.txt` file was delivered in the adapter package for the connector, or if a definition file is available under another extension.
3. In the directory display, navigate to the appropriate connector definition file, select it, and click **Open**.

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

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

Completing a configuration file

When you open a configuration file or a connector from a project, the Connector Configurator 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 application-specific configuration properties" on page 108
- "Specifying supported business object definitions" on page 109
- "Associated maps" on page 110
- "Setting trace/log file values" on page 111

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 112.

Setting the configuration file properties

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

Standard properties differ from connector-specific properties as follows:

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

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

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

Setting standard connector properties

To change the value of a standard property:

1. Click in the field whose value you want to set.
2. Either enter a value, or select one from the drop-down menu if it appears.

3. After entering all the values for the standard properties, you can do one of the following:
   - To discard the changes, preserve the original values, and exit Connector Configurator 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 application-specific configuration properties

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

1. Right-click in the top left portion of the grid. A pop-up menu bar will appear. Click **Add** to add a property. To add a child property, right-click on the parent row number and click **Add child**.
2. Enter a value for the property or child property.
3. To encrypt a property, select the **Encrypt** box.
4. Choose to save or discard changes, as described for “Setting standard connector properties” on page 107.

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

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

Encryption for connector properties

Application-specific properties can be encrypted by selecting the **Encrypt** check box in the **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

Refer to the descriptions of update methods found in the *Standard configuration properties for connectors* appendix, under “Setting and updating property values” on page 87.

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.

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 objects and application-specific business objects, 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 Integration Component Library (ICL) projects. For more information on ICL projects, see the *User Guide for WebSphere Business Integration Server Express*.

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 agent.

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

To indicate that the business object is supported by the connector agent, check the **Agent Support** box. The Connector Configurator 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.

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 InterChange Server Express. This list appears when you select the **Associated Maps** tab.

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

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

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

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

- **Business Object Name**

  These are the business objects supported by this connector, as designated in the **Supported Business Objects** tab. If you designate additional business objects under the Supported Business Objects tab, they will be reflected in this list after you save the changes by choosing **Save to Project** from the **File** menu of the Connector Configurator 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.
- **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.
3. In the **File** menu of the Connector Configurator Express window, click **Save to Project**.
4. Deploy the project to InterChange Server Express.
5. Reboot the server for the changes to take effect.

Resources

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

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

Setting trace/log file values

When you open a connector configuration file or a connector definition 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.

     Note: You can only use the STDOUT option from the **Trace/Log Files** tab for connectors running on the Windows platform.
   - To File:
     Writes logging or tracing messages to a file that you specify. To specify the file, click the directory button (ellipsis), navigate to the preferred location, provide a file name, and click **Save**. (If your connector is not running on the Windows platform on which you have installed Connector Configurator Express, you must first map a drive to a location on the system where you want the file.) 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.

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

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

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

Completing the configuration

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

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

Using Connector Configurator Express in a globalized environment

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

Connector Configurator Express supports non-English characters in:

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

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

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

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

Appendix C. SWIFT message structure

This appendix describes SWIFT message structure.

SWIFT message types

SWIFT messages consist of five blocks of data including three headers, message content, and a trailer. Message types are crucial to identifying content.

All SWIFT messages include the literal “MT” (Message Type). This is followed by a 3-digit number that denotes the message type, category, and group. Consider the following example, which is an order to buy or sell via a third party:

MT502
The first digit (5) represents the category. A category denotes messages that relate to particular financial instruments or services such as Precious Metals, Syndications, or Travelers Checks. The category denoted by 5 is Securities Markets.

The second digit (0) represents a group of related parts in a transaction life cycle. The group indicated by 0 is a Financial Institution Transfer.

The third digit (2) is the type that denotes the specific message. There are several hundred message types across the categories. The type represented by 2 is a Third-Party Transfer.

Each message is assigned unique identifiers. A 4-digit session number is assigned each time the user logs in. Each message is then assigned a 6-digit sequence number. These are then combined to form an ISN (Input Sequence Number) from the user’s computer to SWIFT or an OSN (Output Sequence Number) from SWIFT to the user’s computer. It is important to remember that terminology is always from the perspective of SWIFT and not the user.

The Logical Terminal Address (12 character BIC), Day, Session and Sequence numbers combine to form the MIR (Message Input Reference) and MOR (Message Output Reference), respectively.

For a full list of SWIFT message types, see *All Things SWIFT: the SWIFT User Handbook*.

SWIFT field structure

This section discusses the SWIFT field structure. A field is a logical subdivision of a message block A, which consists of a sequence of components with a starting field tag and delimiters.

A field is always prefaced by a field tag that consists of two digits followed, optionally, by an alphabetic character. The alphabetic character is referred to as an option. For example, 16R is a tag (16) with an option (R) that indicates the start of a block; 16S is a tag (16) with an option (S) that indicates the end of a block. A field is always terminated by a field delimiter. The delimiter depends on the type of field used in a message block.

There are two types of fields used in SWIFT messages: generic and non-generic. The type of field used in a SWIFT message block is determined by the Message Type. What follows is a discussion of these SWIFT field structures. For more on generic and non-generic fields and how to distinguish between them, see Part III, Chapter 3 of the *SWIFT User Handbook*

Note: The symbol CRLF shown below is a control character and represents carriage return/line feed (0D0A in ASCII hex, 0D25 in EBCDIC hex).

Non-generic fields

The structure of non-generic fields in SWIFT message blocks is as follows:

`:2!n[1a]:` *`data content`*`<CRLF>`

where:

: = mandatory colon

2!n = numeric character, fixed length

[1a] = one optional alphabetic character, letter option

: = mandatory colon

data content = the data content, which is defined separately for every tag

<CRLF> = field delimiter

The following is an example of a non-generic field:

```
:20:1234<CRLF> :32A:...<CRLF>
```

Note: In some cases (such as with the tag 15A...*n*), the data content is optional.

Generic fields

The structure of generic fields in SWIFT messages is as follows:

`:2!n1a::4!c'/'[8c]'/'`*`data content`*

where

:2!n1a: = same format as non-generic fields, except that 1a is mandatory

: = mandatory second colon (required in all generic fields)

4!c = qualifier

'/' = first delimiter

[8c] = issuer code or Data Source Scheme (DSS)

'/' = second delimiter

data content = See Part III, Chapter 3 of the *SWIFT User Handbook* for the format definition

Note: Non-generic fields and generic fields cannot share the same field tag letter option letter. In order to distinguish between them easily, a colon is defined as the first character of the column Component Sequence. Generic fields are defined in the same section (Part III, Chapter 3 of the *SWIFT User Handbook*) as the non-generic fields.

The following character restrictions apply to generic field data content:

- Second and subsequent lines within the data content must start with the delimiter `CRLF`.
- Second and subsequent lines within the data content must never start with a colon (:) or a hyphen (-).
- The data content must end with the delimiter `CRLF`.

SWIFT message block structure

The connector supports SWIFT Financial Application (FIN) messages. They have the following structure:

{1: *Basic Header Block*} {2: *Application Header Block*} {3: *User Header Block*} {4: *Text Block or body*} {5: *Trailer Block*}

These five SWIFT message blocks include header information, the body of the message, and a trailer. All blocks have the same basic format:

`{`*n*`:...}`

The curly braces ({}) indicate the beginning and end of a block. *n* is the block identifier, in this case a single integer between 1 and 5. Each block identifier is associated with a particular part of the message. There is no carriage return or line feed (`CRLF`) between blocks.

Blocks 3, 4, and 5 may contain sub-blocks or fields delimited by field tags. Block 3 is optional. Many applications, however, populate block 3 with a reference number so that when SWIFT returns the acknowledgement, it can be used for reconciliation purposes.

Note: For further information on SWIFT message blocks, see Chapter 2 of the *SWIFT User Handbook FIN System Messages Document*.

{1: Basic Header Block}

The basic header block is fixed-length and continuous with no field delimiters. It has the following format:

```
{1:    F    01   BANKBEBB   2222   123456}
(a)   (b)  (c)     (d)      (e)      (f)
```

a) `1:` = Block ID (always 1)

b) Application ID as follows:

- `F` = FIN (financial application)
- `A` = GPA (general purpose application)
- `L` = GPA (for logins, and so on)

c) Service ID as follows:

- 01 = FIN/GPA
- 21 = ACK/NAK

d) BANKBEBB = Logical terminal (LT) address. It is fixed at 12 characters; it must not have X in position 9.

e) 2222 = Session number. It is generated by the user's computer and is padded with zeros.

f) 123456 = Sequence number that is generated by the user's computer. It is padded with zeros.

{2: Application Header Block}

There are two types of application headers: Input and Output. Both are fixed-length and continuous with no field delimiters.

The input (to SWIFT) structure is as follows:

```
{2:    I     100    BANKDEFFXXXX    U       3       003}
(a)   (b)    (c)      (d)          (e)     (f)      (g)
```

a) 2: = Block ID (always 2)

b) I = Input

c) 100 = Message type

d) BANKDEFFXXXX = Receiver's address with X in position 9/ It is padded with Xs if no branch is required.

e) U = the message priority as follows:

- S = System
- N = Normal
- U = Urgent

f) 3 = Delivery monitoring field is as follows:

- 1 = Non delivery warning (MT010)
- 2 = Delivery notification (MT011)
- 3 = Both valid = U1 or U3, N2 or N

g) 003 = Obsolescence period. It specifies when a non-delivery notification is generated as follows:

- Valid for U = 003 (15 minutes)
- Valid for N = 020 (100 minutes)

The output (from SWIFT) structure is as follows:

```
{2:   O      100   1200   970103BANKBEBBAXXX2222123456   970103  1201   N}
(a)  (b)      (c)   (d)           (e)                      (f)    (g)   (h)
```

a) 2: = Block ID (always 2)

b) O = Output

c) 100 = Message type

d) 1200 = Input time with respect to the sender

e) The Message Input Reference (MIR), including input date, with Sender's address

f) 970103 = Output date with respect to Receiver

g) 1201 = Output time with respect to Receiver

h) N = Message priority as follows:
- S = System
- N = Normal
- U = Urgent

{3: User Header Block}

This is an optional block and has the following structure:

```
{3:  {113:xxxx}  {108:abcdefgh12345678}     }
(a)      (b)             ( c)
```

a) 3: = Block ID (always 3)

b) 113:xxxx = Optional banking priority code

c) This is the Message User Reference (MUR) used by applications for reconciliation with ACK.

Note: Other tags exist for this block. They include tags (such as 119, which can contain the code ISITC on an MT521) that may force additional code word and formatting rules to validate the body of the message as laid down by ISITC (Industry Standardization for Institutional Trade Communication). For further information, see *All Things SWIFT: the SWIFT User Handbook*.

{4: Text Block or body}

This block is where the actual message content is specified and is what most users see. Generally the other blocks are stripped off before presentation. The format, which is variable length and requires use of CRLF as a field delimiter, is as follows:

```
{4:CRLF
:20:PAYREFTB54302 CRLF
:32A:970103BEF1000000,CRLF
:50:CUSTOMER NAME CRLF
AND ADDRESS CRLF
:59:/123-456-789 CRLF
BENEFICIARY NAME CRLF
AND ADDRESS CRLF
-}
```

The symbol CRLF is a mandatory delimiter in block 4.

The example above is of type MT100 (Customer Transfer) with only the mandatory fields completed. It is an example of the format of an ISO7775 message structure. Block 4 fields must be in the order specified for the message type in the appropriate volume of the *SWIFT User Handbook*.

Note: The ISO7775 message standard is gradually being replaced by the newer data dictionary standard ISO15022. Among other things, the new message standard makes possible generic fields for block 4 of a SWIFT message structure. For further information, see “SWIFT field structure” on page 115.

The content of the text block is a collection of fields. For more on SWIFT fields, see “SWIFT field structure” on page 115. Sometimes, the fields are logically grouped into sequences. Sequences can be mandatory or optional, and can repeat. Sequences also can be divided into subsequences. In addition, single fields and groups of consecutive fields can repeat. For example, sequences such as those in the SWIFT Tags 16R and 16S may have beginning and ending fields. Other sequences, such as Tag 15, have only a beginning field. In yet other message types, no specific tags mark the start or end of a field sequence.

The format of block 4 field tags is:

`:`*nna*`:`

nn = Numbers

a = Optional letter, which may be present on selected tags

For example:

`:20:` = Transaction reference number

`:58A:` = Beneficiary bank

The length of a field is as follows:

nn = Maximum length

nn`!` = Fixed-length

nn`-`*nn* = Minimum and maximum length

nn `*` *nn* = Maimum number of lines times maximum line length

The format of the data is as follows:

n = Digits

d = Digits with decimal comma

h = Uppercase hexadecimal

a = Uppercase letters

c = Uppercase alphanumeric

e = Space

x = SWIFT character set

y = Uppercase level A ISO 9735 characters

z = SWIFT extended character set

Some fields are defined as optional. If optional fields are not required in a specific message, do not include them because blank fields are not allowed in the message.

`/,word` = Characters "as is"

`[...]` = Brackets indicate an optional element

For example:

`4!c[/30x]` This is a fixed 4 uppercase alphanumeric, optionally followed by a slash and up to 30 SWIFT characters.

`ISIN1!e12!c` This is a code word followed by a space and a 12 fixed uppercase alphanumeric.

Note: In some message types, certain fields are defined as conditional. For example, when a certain field is present, another field may change from optional to mandatory or forbidden. Certain fields may contain sub-fields, in which case there is no `CRLF` between them. Validation is not supported.

Certain fields have different formats that depend on the option that is chosen. The option is designated by a letter after the tag number, for example:

:32A:000718GBP1000000,00 Value Date, ISO Currency, and Amount

:32B:GBP1000000,00 ISO Currency and Amount

Note: The SWIFT standards for amount formats are: no thousand separators are allowed (10,000 is not allowed, but 10000 is allowed); use a comma (not a decimal point) for a decimal separator (1000,45 = one thousand and forty-five hundredths).

`:58A:NWBKGB2L` Beneficiary SWIFT address

`:58D:NatWest Bank` Beneficiary full name and address

```
Head Office

London
```

{5: Trailer Block}

A message always ends in a trailer with the following format:

```
{5: {MAC:12345678}{CHK:123456789ABC}
```

This block is for SWIFT system use and contains a number of fields that are denoted by keywords such as the following:

MAC Message Authentication Code calculated based on the entire contents of the message using a key that has been exchanged with the destination and a secret algorithm. Found on message categories 1,2,4,5,7,8, most 6s and 304.

CHK Checksum calculated for all message types.

PDE Possible Duplicate Emission added if user thinks the same message was sent previously

DLM Added by SWIFT if an urgent message (`U`) has not been delivered within 15 minutes, or a normal message (`N`) within 100 minutes.

Appendix D. Notices

Notices

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

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

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

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

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

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

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

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

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

IBM Burlingame Laboratory Director IBM Burlingame Laboratory 577 Airport Blvd., Suite 800 Burlingame, CA 94010 U.S.A

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

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

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

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

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

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

COPYRIGHT LICENSE

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

Programming interface information

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

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

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

Note: 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.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

System Manager includes software developed by the Eclipse Project (http://www.eclipse.org/)

IBM WebSphere Business Integration Server Express V4.3.1 and IBM WebSphere Business Integration Server Express Plus V4.3.1

Printed in USA