

IBM WebSphere Business Integration Adapters



Adapter for SWIFT User Guide

Version 16.x

WebSphere software

IBM WebSphere Business Integration Adapters



Adapter for SWIFT User Guide

Version 16.x

Note!

Before using this information and the product it supports, read the information in Appendix E, "Notices", on page 181.

18April2003

This edition of this document applies to IBM WebSphere InterChange Server, version 4.2, WebSphere Business Integration Adapters, version 2.2.0, and to all subsequent releases and modification 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.

© Copyright International Business Machines Corporation 2002, 2003. All rights reserved.

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

Integration broker compatibility

Supported on IBM WebSphere Business Integration Adapter Framework version 2.2.0, IBM CrossWorlds Infrastructure version 4.1.1 and 4.2 (if the environment uses ISO Latin-1 data only), WebSphere MQ Integrator version 2.1.0, and WebSphere MQ Integrator Broker, version 2.1.0. See Release Notes for any exceptions.

Contents

Integration broker compatibility	iii
About this document	vii
Audience	vii
Prerequisites for this document	vii
Related documents	vii
Typographic conventions	viii
New in this release	ix
New in release 1.6.x	ix
New in release 1.5.x	ix
New in release 1.4.x	x
New in release 1.3.x	x
New in release 1.2.x	x
New in release 1.1.x	xi
Chapter 1. Overview	1
Connector architecture	2
Application-connector communication method	4
Event handling	6
Guaranteed event delivery	9
Business object requests	10
Business object mapping	10
Message processing	10
Error handling	14
Tracing	15
Chapter 2. Configuring the connector	17
Prerequisites	17
Installing the connector	18
Connector configuration	19
Enabling guaranteed event delivery	24
Queue Uniform Resource Identifiers (URI)	28
Meta-object attributes configuration	29
Startup file configuration	42
Startup	42
Chapter 3. Business objects	43
Connector business object requirements	43
Overview of SWIFT message structure	47
Overview of business objects for SWIFT	48
SWIFT message and business object data mapping	49
Chapter 4. ISO 7775 to ISO 15022 mapping	81
Production instruction meta-objects (PIMOs)	81
Creating PIMOs	88
Modifying PIMOs: Map summary	96
Chapter 5. SWIFT Data Handler	123
Configuring the SWIFT Data Handler	123
Business Object Requirements	124
Converting Business Objects to SWIFT Messages	124
Converting SWIFT Messages to Business Objects	125

Chapter 6. Troubleshooting	127
Startup problems	127
Event processing	127
Appendix A. Standard configuration properties for connectors	129
New and deleted properties	129
Configuring standard connector properties for WebSphere InterChange Server	130
Configuring standard connector properties for WebSphere MQ Integrator	142
Appendix B. Connector Configurator	151
Using Connector Configurator in an internationalized environment	151
Starting Connector Configurator	152
Choosing your broker	153
Using a connector-specific property template	154
Using Connector Configurator with ICS as the broker	157
Setting the configuration file properties (ICS)	159
Setting the configuration file properties (WebSphere MQ Integrator Broker)	164
Using standard and connector-specific properties with Connector Configurator	167
Completing the configuration	168
Appendix C. Connector feature list	169
Business object request handling features	169
Event notification features	169
General features	170
Appendix D. SWIFT message structure	173
SWIFT message types	173
SWIFT field structure	173
SWIFT message block structure	175
Appendix E. Notices	181
Programming interface information	182
Trademarks and service marks	182

About this document

IBM(R) WebSphere(R) Business Integration Adapters supply integration connectivity for leading e-business technologies and enterprise applications.

This document describes the installation, configuration, and business object development for the adapter for SWIFT.

Audience

This document is for consultants, developers, and system administrators who support and manage the WebSphere business integration system at customer sites.

Prerequisites for this document

Users of this document should be familiar with

- the WebSphere business integration system (if you are using InterChange Server as your integration broker)
- the WebSphere MQ Integrator Broker (if you are using WebSphere MQ Integrator Broker as your integration broker)
- business object development
- the WebSphere MQ application
- the SWIFT product suite and protocol

Related documents

The WebSphere business integration system documentation describes the features and components common to all installations, and includes reference material on specific collaborations and connectors.

This document contains many references to two other documents: the *System Installation Guide for Windows or for UNIX* and the *System Implementation Guide for WebSphere InterChange Server*. If you choose to print this document, you may want to print these documents as well.

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

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

- If you are using WebSphere MQ Integrator Broker as your integration broker:
<http://www.ibm.com/websphere/integration/wbiadapters/infocenter>
- If you are using InterChange Server as your integration broker:
<http://www.ibm.com/websphere/integration/wicserver/infocenter>

The documentation set consists primarily of Portable Document Format (PDF) files, with some additional files in HTML format. To read it, you need an HTML browser such as Netscape Navigator or Internet Explorer, and Adobe Acrobat

Reader 4.0.5 or higher. For the latest version of Adobe Acrobat Reader for your platform, go to the Adobe website (www.adobe.com).

Typographic conventions

This document uses the following conventions:

<code>courier font</code>	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
<i>italic, italic</i>	Indicates a variable name or a cross-reference.
<i>blue text</i>	Blue text, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click any blue text to jump to the object of the reference.
<i>ProductDir</i>	Product family is WBIA: Represents the directory where the IBM WebSphere Business Integration Adapters product is installed. The CROSSWORLDS environment variable contains the ProductDir directory path, which is IBM\WebSphereAdapters by default.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.
[]	In a syntax line, square brackets surround an optional parameter.
...	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, <code>option[,...]</code> means that you can enter multiple, comma-separated options.
< >	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in <code><server_name><connector_name>tmp.log</code> .
/, \	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All product pathnames are relative to the directory where the product is installed on your system.
<code>%text%</code> and <code>\$text</code>	Text within percent (%) signs indicates the value of the Windows text system variable or user variable. The equivalent notation in a UNIX environment is <code>\$text</code> , indicating the value of the <code>text</code> UNIX environment variable.

New in this release

New in release 1.6.x

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

The guaranteed event delivery feature has been enhanced. For further information, see "Enabling guaranteed event delivery" on page 24.

You can now associate a data handler with an input queue. For further information, see "Mapping data handlers to InputQueues" on page 31.

The connector supports SWIFTAlliance Access 5.0, but only if you deploy MQSA version 2.2 on WebSphere MQ 5.2 on and run the connector on one of the following operating systems:

- AIX 5.1
- SunOS 5.8 or Solaris 8
- Windows 2000 SP2

New in release 1.5.x

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

This release extends support for subfield parsing on the following message types:

- **Category 1**MT100, MT101, MT102, MT103, MT104, MT105, MT106, MT107, MT110, MT111, MT112, MT190, MT191, MT198, MT199
- **Category 2**MT200, MT201, MT202, MT203, MT204, MT205, MT206, MT207, MT210, MT256, MT290, MT291, MT293, MT298, MT299
- **Category 3**MT300, MT303, MT304, MT305, MT306, MT320, MT330, MT340, MT341, MT350, MT360, MT361, MT362, MT364, MT365, MT390, MT391, MT398, MT399
- **Category 4**MT400, MT405, MT410, MT412, MT416, MT420, MT422, MT430, MT450, MT455, MT456, MT490, MT491, MT498, MT499
- **Category 5**MT502, MT503, MT504, MT505, MT506, MT507, MT508, MT509, MT512, MT513, MT514, MT515, MT516, MT517, MT518, MT520, MT521, MT522, MT523, MT524, MT526, MT527, MT528, MT529, MT530, MT531, MT532, MT533, MT534, MT535, MT536, MT537, MT538, MT539, MT540, MT541, MT542, MT543, MT544, MT545, MT546, MT547, MT548, MT549, MT550, MT551, MT552, MT553, MT554, MT555, MT556, MT557, MT558, MT559, MT560, MT561, MT562, MT563, MT564, MT565, MT566, MT567, MT568, MT569, MT570, MT571, MT572, MT573, MT575, MT576, MT577, MT578, MT579, MT580, MT581, MT582, MT583, MT584, MT586, MT587, MT588, MT589, MT590, MT591, MT598, MT599

- **Category 6** MT600, MT601, MT604, MT605, MT606, MT607, MT608, MT609, MT643, MT644, MT645, MT646, MT649, MT690, MT691, MT698, MT699
- **Category 7** MT700, MT701, MT705, MT707, MT710, MT711, MT720, MT721, MT730, MT732, MT734, MT740, MT742, MT747, MT750, MT752, MT754, MT756, MT760, MT767, MT768, MT769, MT790, MT791, MT798, MT799
- **Category 8** MT800, MT801, MT802, MT810, MT812, MT813, MT820, MT821, MT822, MT823, MT824, MT890, MT891, MT898, MT899
- **Category 9** MT900, MT910, MT920, MT935, MT940, MT941, MT942, MT950, MT960, MT961, MT962, MT963, MT964, MT965, MT966, MT967, MT970, MT971, MT972, MT973, MT985, MT986, MT990, MT991, MT998, MT999

The InProgress queue is no longer required and may be disabled. For more information, see “InProgressQueue” on page 23.

The connector supports interoperability with applications via MQSeries 5.1, 5.2, and 5.3. For more information, see “Prerequisites” on page 17

The connector now has a UseDefaults property for business object processing. For more information, see “UseDefaults” on page 24.

The connector can now apply a default verb when the data handler does not explicitly assign one to a business object. For more information, see “DefaultVerb” on page 21.

The ReplyToQueue can now be dictated via the dynamic child meta-object rather than by the ReplyToQueue connector property. For more information see “JMS headers, SWIFT message properties, and dynamic child meta-object attributes” on page 37.

You can use a message selector to identify, filter and otherwise control how the adapter identifies the response message for a given request. This JMS capability applies to synchronous request processing only. For more information, see “Synchronous acknowledgment” on page 11.

New in release 1.4.x

If you are using the guaranteed event delivery feature and ICS as your integration broker, you must install release 4.1.1.2 of ICS.

New in release 1.3.x

The IBM WebSphere Business Adapter for SWIFT can dynamically transform a business object representing an ISO 7775 SWIFT message into a business object representing the corresponding ISO 15022 SWIFT message and vice versa. The adapter supports business object ISO 7775-15022 mapping for SWIFT Category 5, Securities Markets, but only with the expanded business object definitions available with this release and described in this document, and only on the Solaris platform.

New in release 1.2.x

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

- An application component specific to SWIFT
- Sample business objects
- IBM WebSphere Adapter Framework, which consists of:
 - Connector Framework
 - Development tools (including Business Object Designer and Connector Configurator))
 - APIs (including CDK)

This manual provides information about using this adapter with both integration brokers: ICS and WebSphere MQ Integrator.

Important: Because the connector has not been internationalized, do not run it against ICS version 4.1.1 if you cannot guarantee that only ISO Latin-1 data will be processed.

New in release 1.1.x

The following new features are described in this guide:

- The connector for SWIFT is now enabled for AIX 4.3.3 Patch Level 9

Chapter 1. Overview

- “Connector architecture” on page 2
- “Application-connector communication method” on page 4
- “Event handling” on page 6
- “Guaranteed event delivery” on page 9
- “Business object requests” on page 10
- “Business object mapping” on page 10
- “Message processing” on page 10
- “Error handling” on page 14
- “Tracing” on page 15

The connector for SWIFT is a runtime component of the WebSphere Business Integration Adapter for SWIFT. The connector allows the WebSphere 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 *IBM WebSphere InterChange Server System Administration Guide*, or the *IBM WebSphere Business Integration Adapters Implementation Guide for WebSphere MQ Integrator Broker*.

All WebSphere business integration adapters operate with an integration broker. The connector for SWIFT operates with both the WebSphere InterChange Server (ICS) and WebSphere MQ Integrator Broker integration brokers.

The connector for SWIFT allows the ICS or WebSphere MQ Integrator Broker to exchange business objects with applications that send or receive data in the form of SWIFT messages.

Important: The connector supports SWIFT message transformation from ISO 7775 to corresponding ISO 15022 formats and vice versa, but only with the expanded business object definitions and ISO mapping described in this document. For further information, see Chapter 3, “Business objects”, on page 43, and Chapter 4, “ISO 7775 to ISO 15022 mapping”, on page 81.

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.

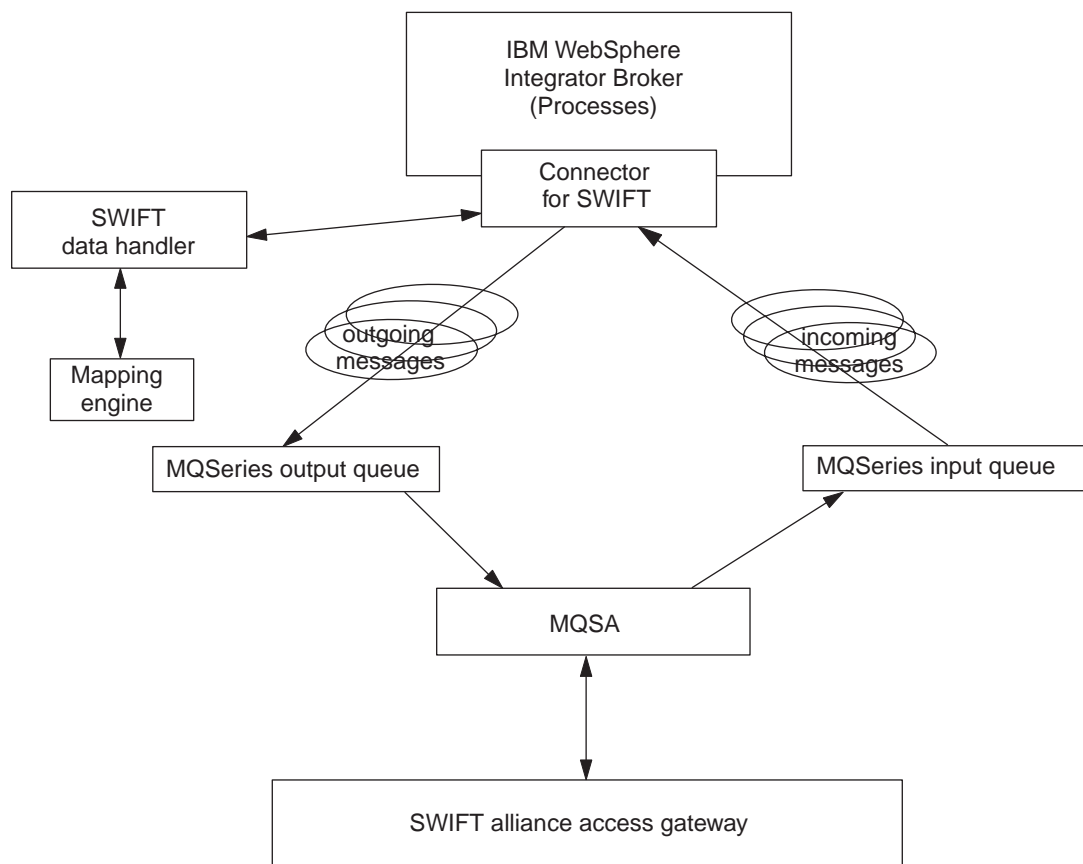


Figure 1. Connector for SWIFT architecture

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

Connector for SWIFT

The connector for SWIFT is meta-data-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 the integration broker, converts them into SWIFT messages using the same data handler, and then delivers the messages to an WebSphere MQ queue.

The type of business object and verb used in processing a message are based on the meta-data 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 the integration broker using the `gotApp1Events()` method.

SWIFT data handler and mapping engine

The connector calls the SWIFT data handler to convert business objects into SWIFT messages and vice versa. The data handler is coupled with a mapping engine, which uses a map to perform transformations between business objects representing ISO 7775 and ISO 15022 SWIFT message formats. For more on the SWIFT data handler, see Chapter 5, “SWIFT Data Handler”, on page 123. For more on mapping, see Chapter 4, “ISO 7775 to ISO 15022 mapping”, on page 81.

WebSphere MQ

The connector for SWIFT uses an MQ implementation of the Java™ 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 or WebSphere MQ. The

connector supports SWIFTAlliance Access 5.0, but only if you deploy MQSA version 2.2 on WebSphere MQ 5.2 on and run the connector on one of the following operating systems:

- AIX 5.1
- SunOS 5.8 or Solaris 8
- Windows 2000 SP2

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 7775 SWIFT message from an integration broker.
2. The connector passes the business object to the data handler.
3. If specified in the map subscription meta-object, the data handler passes the ISO 7775 object to the mapping engine.
4. Using a production instruction meta-object (PIMO), the mapping engine transforms the ISO 7775 object into an ISO 15022 business object and passes it to the data handler.
5. The data handler converts the ISO 15022 business object into an ISO 15022-compliant SWIFT message.
6. The connector dispatches the ISO 15022 SWIFT message to the WebSphere MQ output queue.
7. 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.

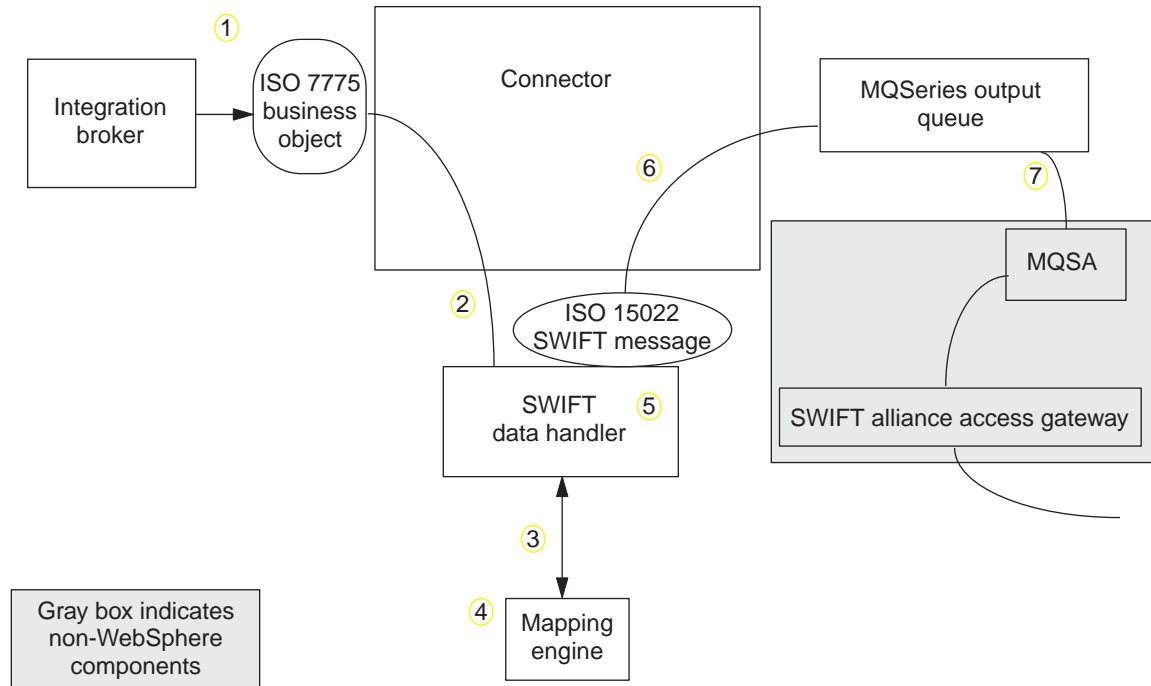


Figure 2. Application-connector communication method: Message request

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. Using the map subscription meta-object, the connector determines whether the message type is supported and, if supported, requires transformation into an ISO 7775 business object. If so, the data handler passes the ISO 15022 business object to the mapping engine.
5. Using a PIMO, the mapping engine processes the sub-fields of business object data, creating an ISO 7775-compliant business object, which is passed to the data handler.
6. The SWIFT data handler receives the ISO 7775 business object and sets the verb in it to the default verb specified in the data handler-specific meta-object.
7. 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.

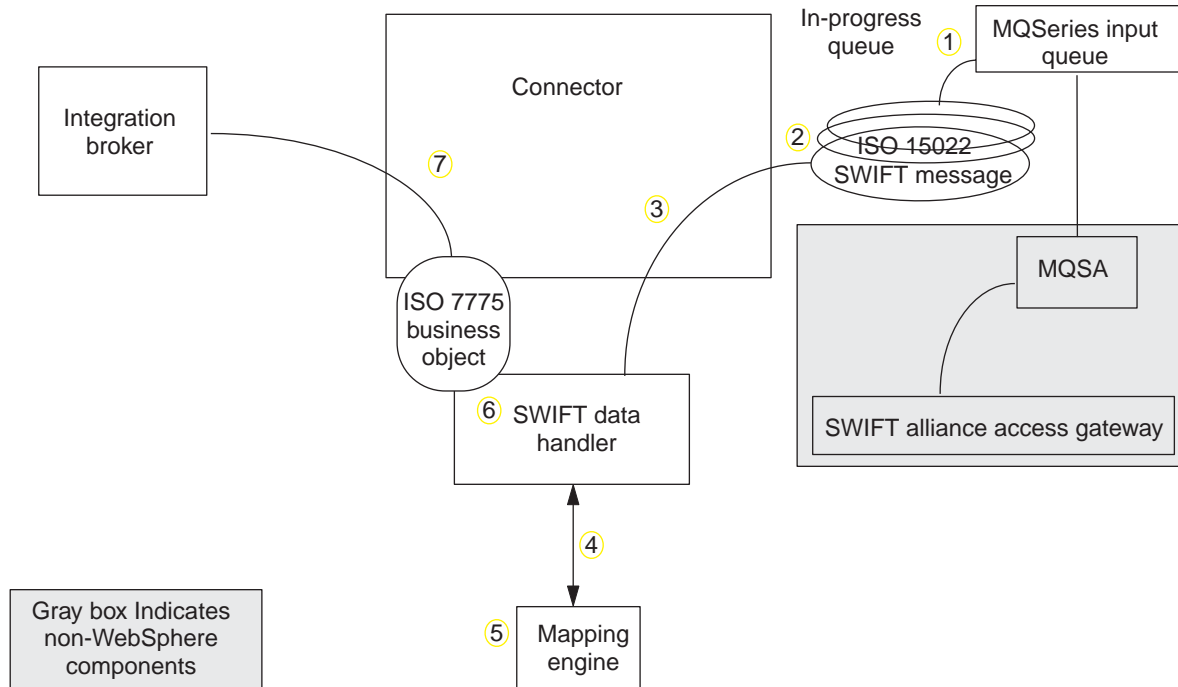


Figure 3. Application-connector communication method: Event delivery

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 the integration broker, 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 the integration broker. If the business object is successfully processed, the connector sends a report back to the requesting application including the return code from the integration broker and any business object changes. If the connector or the integration broker 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 the integration broker 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 Report	Message type Options for report message requested	<p>DATAGRAM</p> <p>You can specify one or both of the following:</p> <ul style="list-style-type: none"> • 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. <p>You can specify one of the following to control how the correlation ID of the report message is to be set:</p> <ul style="list-style-type: none"> • 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 meta-data 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 feedback	Message type Type of report	REPORT One of the following: <ul style="list-style-type: none"> • MQRO_PAN If the business object is successfully processed. • MQRO_NAN If the connector or the integration broker 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 the integration broker. This default behavior can be overridden by setting the DoNotReportBusObj property to true in the static meta-data object. If the request could not be processed, the connector populates the message body with the error message generated by the connector or the integration broker.

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 `connectorDeliveryTransport` standard property to JMS. Thus configured, the connector uses the JMS transport and all subsequent communication between the connector and the integration broker 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 `gotAppEvent()` 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 "Enabling guaranteed event delivery" on page 24.

If connector framework cannot deliver the business object to the ICS integration broker, 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 the integration broker issues a business object. Using the SWIFT data handler and mapping engine, and depending on the requirements specified in the subscription meta-object, the connector can transform an ISO 7775 object to an ISO 15022 object before converting the business object to a SWIFT message and issuing it.

Business object mapping

The map subscription meta-object determines whether mapping between ISO 7775 and ISO 15022 business objects occurs. For example, a child map subscription meta-object (MO_Swift_MapSubscriptions_In) with an attribute Map_Swift_MT523_to_MT543 indicates that the business object definition corresponding to SWIFT message type 523 must be mapped to a business object definition representing SWIFT message type 543. For more information on the map subscription meta-object, see “Production instruction meta-objects (PIMOs)” on page 81.

The transformation of business object definitions from those representing ISO 7775 messages to those representing ISO 15022 and vice versa takes place in the mapping engine. There, a production instruction meta-object (PIMO) governs the mapping process. The PIMO is a meta-object, but one designed to handle mapping only. Each PIMO specifies attribute-by-attribute processing instructions for a specific transformation, for example, from SWIFT message type 523 to message type 543. You can modify PIMOs using Business Object Designer. For more information on mapping and PIMOs, see Chapter 4, “ISO 7775 to ISO 15022 mapping”, on page 81.

Message processing

The connector processes business objects passed to it by an integration broker 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
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 <code>ReplyToQueue</code> .
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 <code>busObj</code> 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 system 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
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 *connector development 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 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 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 the integration broker for the Request operation. If an error message was expected but the message body is not populated, a generic error message is returned to the integration broker 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 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 the integration broker. 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, “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 *Connector Development Guide*.

What follows is recommended content for connector trace messages.

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

Level 2	Use this level for trace messages that log each time a business object is posted to the integration broker, either from <code>gotAppEvent()</code> or <code>executeCollaboration()</code> .
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. Configuring the connector

- “Prerequisites”
- “Installing the connector” on page 18
- “Connector configuration” on page 19
- “Enabling guaranteed event delivery” on page 24
- “Queue Uniform Resource Identifiers (URI)” on page 28
- “Meta-object attributes configuration” on page 29
- “Startup file configuration” on page 42
- “Startup” on page 42

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

Prerequisites

Prerequisite software

The following software must be installed before you install and configure the connector for SWIFT:

- WebSphere business integration system software version 4.2.x or later or WebSphere Business Integration Adapter Framework version 2.1.0
- The connector supports interoperability with applications via WebSphere MQ, WebSphere MQ 5.1, 5.2,¹ and 5.3. Accordingly, you must have one of these software releases installed.

Note: The adapter does not support Secure Socket Layers (SSL) in WebSphere MQ 5.3. For the WebSphere MQ software version appropriate to adapter framework-integration broker communication, see the Installation Guide for your platform (Windows/Unix).

- IBM WebSphere MQ Java client libraries

Note: It's advisable to download the latest MA88 libraries from IBM.

- For Windows systems: Microsoft Windows NT 4.0 Service Pack 6A or Windows 2000
- For Unix systems: Solaris 7 or AIX 4.3.3 Patch Level 7
- MQSA WebSphere MQ Interface for SWIFTAlliance 1.3

Note: The SWIFTAlliance Access gateway is a window through which SWIFT messages flow to and from remote financial applications over IP or WebSphere MQ. The connector supports SWIFTAlliance Access 5.0, but only if you deploy MQSA version 2.2 on WebSphere MQ 5.2 on and run the connector on one of the following operating systems:

- AIX 5.1
- SunOS 5.8 or Solaris 8

1. If your environment implements the convert-on-the-get methodology for character-set conversions you must download the latest MA88 (JMS classes) from IBM. The patch level should be at least 5.2.2 (for MQ Series version 5.2). Doing so may avoid unsupported encoding errors.

For client setup with an NT server, see the description in the IBM publication *WebSphere MQ Quick Beginnings for NT*.

Installing the connector

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

Installing on a UNIX system

To install the connector on a UNIX system, run the Installer for IBM WebSphere Business Integration Adapter and select the WebSphere Business Integration Adapter for SWIFT. Installer installs *all* UNIX-supported connectors.

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

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

Table 8. Installed UNIX file structure for the connector

Subdirectory of <i>ProductDir</i>	Description
connectors/SWIFT/CWSwift.jar connectors/SWIFT/CWJMCommon.jar connectors/SWIFT/start_SWIFT.sh	Connector jar files The startup script for the connector. The script is called from the generic connector manager script. When you click Install from Connector Configurator (WebSphere MQ Integrator Broker as the integration broker) or the Connector Configuration screen of System Manager (ICS as the integration broker), the installer creates a customized wrapper for this connector manager script. When the connector works with ICS, use this customized wrapper only to start and stop the connector. When the connector works with WebSphere MQ Integrator Broker, use this customized wrapper only to start the connector; use <code>mqsiremotestopadapter</code> to 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
/lib	Contains the WBIA. jar file.
/bin	Contains the CWConnEnv.sh file.

For more information on installing the connector component, refer to one of the following guides, depending on the integration broker you are using:

- *System Installation Guide for UNIX* (when ICS is used as integration broker)
- *Implementation Guide for WebSphere MQ Integrator Broker* (when MQ Integrator Broker is used as integration broker)

Installing on a Windows system

To install the connector on a Windows system, run the Installer for IBM WebSphere Business Integration Adapter and select the WebSphere Business Integration Adapter for SWIFT. Installer installs standard files associated with the connector. Table 9 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 9. Installed Windows file structure for the connector

Subdirectory of ProductDir	Description
connectors\SWIFT\CWSwift.jar	Connector jar files
connectors\SWIFT\CWJMCommon.jar	
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
\lib	Contains the WBIA. jar file.
\bin	Contains the CWConnEnv.bat file.

For more information on installing the connector component, refer to one of the following guides, depending on the integration broker you are using:

- *IBM WebSphere InterChange Server System Installation Guide for Windows* (when ICS is used as integration broker)
- *IBM WebSphere Business Integration Adapters Implementation Guide for WebSphere MQ Integrator Broker* (when MQ Integrator Broker is used as integration broker)

Connector configuration

Connectors have two types of configuration properties: standard configuration properties and connector-specific configuration properties. You must set the values of these properties before running the connector. Use one of the following tools to set a connector's configuration properties:

- Connector Configurator (if ICS is the integration broker)—Access to this tool is from the System Manager.
- Connector Configurator (if WebSphere MQ Integrator Broker is the integration broker)—Access this tool from the WebSphere Business Integration Adapter program folder. For more information see Appendix B, "Connector Configurator", on page 151

Standard connector properties

Standard configuration properties provide information that all connectors use. See Appendix A, "Standard configuration properties for connectors", on page 129 for documentation of these properties.

Important: Because this connector supports both the ICS and WebSphere MQ Integrator Broker, configuration properties for both brokers relevant to the connector.

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

Table 10. 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 21	MIME type of file	swift	No
"DefaultVerb" on page 21	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 22	WebSphere MQ server		No
"InDoubtEvents" on page 22	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 23	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
"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 the integration broker. 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 the integration broker. 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 system-specific status codes as attribute values in the meta-object:

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

Table 11 shows a sample meta-object.

Table 11. 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 pollQuantity 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 MyQueueB 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.

Enabling guaranteed event delivery

You can configure the guaranteed-event-delivery feature for a JMS-enabled connector in one of the following ways:

- If the connector uses a JMS event store (implemented as a JMS source queue), the connector framework can manage the JMS event store. For more information, see “Guaranteed event delivery for connectors with JMS event stores”.
- If the connector uses a non-JMS event store (for example, implemented as a JDBC table, Email mailbox, or flat files), the connector framework can use a JMS monitor queue to ensure that no duplicate events occur. For more information, see “Guaranteed event delivery for connectors with non-JMS event stores” on page 26.

Guaranteed event delivery for connectors with JMS event stores

If the JMS-enabled connector uses JMS queues to implement its event store, the connector framework can act as a “container” and manage the JMS event store (the JMS source queue). In a single JMS transaction, the connector can remove a message from a source queue and place it on the destination queue. This section provides the following information about use of the guaranteed-event-delivery feature for a JMS-enabled connector that has a JMS event store:

- “Enabling the feature for connectors with JMS event stores”
- “Effect on event polling” on page 26

Enabling the feature for connectors with JMS event stores

To enable the guaranteed-event-delivery feature for a JMS-enabled connector that has a JMS event store, set the connector configuration properties to values shown in Table 12.

Table 12. Guaranteed-event-delivery connector properties for a connector with a JMS event store

Connector property	Value
DeliveryTransport	JMS
ContainerManagedEvents	JMS
PollQuantity	The number of events to processing in a single poll of the event store
SourceQueue	Name of the JMS source queue (event store) which the connector framework polls and from which it retrieves events for processing Note: The source queue and other JMS queues should be part of the same queue manager. If the connector's application generates events that are stored in a different queue manager, you must define a remote queue definition on the remote queue manager. WebSphere MQ can then transfer the events from the remote queue to the queue manager that the JMS-enabled connector uses for transmission to the integration broker. For information on how to configure a remote queue definition, see your IBM WebSphere MQ documentation.

In addition to configuring the connector, you must also configure the data handler that converts between the event in the JMS store and a business object. This data-handler information consists of the connector configuration properties that Table 13 summarizes.

Table 13. Data-handler properties for guaranteed event delivery

Data-handler property	Value	Required?
MimeType	The MIME type that the data handler handles. This MIME type identifies which data handler to call.	Yes
DHClass	The full name of the Java class that implements the data handler	Yes
DataHandlerConfigMOname	The name of the top-level meta-object that associates MIME types and their data handlers	Optional

Note: The data-handler configuration properties reside in the connector configuration file with the other connector configuration properties.

If you configure a connector that has a JMS event store to use guaranteed event delivery, you must set the connector properties as described in Table 12 and Table 13. To set these connector configuration properties, use the Connector Configurator tool. Connector Configurator displays the connector properties in Table 12 on its Standard Properties tab. It displays the connector properties in Table 13 on its Data Handler tab.

Note: Connector Configurator activates the fields on its Data Handler tab only when the DeliveryTransport connector configuration property is set to JMS and ContainerManagedEvents is set to JMS.

For information on Connector Configurator, see Appendix B, “Connector Configurator”, on page 151.

Effect on event polling

If a connector uses guaranteed event delivery by setting `ContainedManagedEvents` to `JMS`, it behaves slightly differently from a connector that does not use this feature. To provide container-managed events, the connector framework takes the following steps to poll the event store:

1. Start a JMS transaction.
2. Read a JMS message from the event store.
The event store is implemented as a JMS source queue. The JMS message contains an event record. The name of the JMS source queue is obtained from the `SourceQueue` connector configuration property.
3. Call the data handler to convert the event to a business object.
The connector framework calls the data handler that has been configured with the properties in Table 13 on page 25.
4. When WebSphere MQ Integrator Broker is the integration broker, convert the business object to a message based on the configured wire format (XML).
5. Send the resulting message to the JMS destination queue. If you are using the WebSphere ICS integration broker, the message sent to the JMS destination queue is the business object. If you are using WebSphere MQ Integrator broker, the message sent to the JMS destination queue is an XML message (which the data handler generated).
6. Commit the JMS transaction.
When the JMS transaction commits, the message is written to the JMS destination queue and removed from the JMS source queue in the same transaction.
7. Repeat step 1 through 6 in a loop. The `PollQuantity` connector property determines the number of repetitions in this loop.

Important: A connector that sets the `ContainerManagedEvents` property is set to `JMS` does *not* call the `pollForEvents()` method to perform event polling. If the connector’s base class includes a `pollForEvents()` method, this method is *not* invoked.

Guaranteed event delivery for connectors with non-JMS event stores

If the JMS-enabled connector uses a non-JMS solution to implement its event store (such as a JDBC event table, Email mailbox, or flat files), the connector framework can use duplicate event elimination to ensure that duplicate events do not occur. This section provides the following information about use of the guaranteed-event-delivery feature with a JMS-enabled connector that has a non-JMS event store:

- “Enabling the feature for connectors with non-JMS event stores”
- “Effect on event polling”

Enabling the feature for connectors with non-JMS event stores: To enable the guaranteed-event-delivery feature for a JMS-enabled connector that has a non-JMS event store, you must set the connector configuration properties to values shown in Table 14.

Table 14. Guaranteed-event-delivery connector properties for a connector with a non-JMS event store

Connector property	Value
DeliveryTransport	JMS
DuplicateEventElimination	true
MonitorQueue	Name of the JMS monitor queue, in which the connector framework stores the ObjectEventId of processed business objects

If you configure a connector to use guaranteed event delivery, you must set the connector properties as described in Table 14. To set these connector configuration properties, use the Connector Configurator tool. It displays these connector properties on its Standard Properties tab. For information on Connector Configurator, see Appendix B, “Connector Configurator”, on page 151.

Effect on event polling: If a connector uses guaranteed event delivery by setting `DuplicateEventElimination` to true, it behaves slightly differently from a connector that does not use this feature. To provide the duplicate event elimination, the connector framework uses a JMS monitor queue to track a business object. The name of the JMS monitor queue is obtained from the `MonitorQueue` connector configuration property.

After the connector framework receives the business object from the application-specific component (through a call to `getAppEvent()` in the `pollForEvents()` method), it must determine if the current business object (received from `getAppEvents()`) represents a duplicate event. To make this determination, the connector framework retrieves the business object from the JMS monitor queue and compares its `ObjectEventId` with the `ObjectEventId` of the current business object:

- If these two `ObjectEventIds` are the same, the current business object represents a duplicate event. In this case, the connector framework ignores the event that the current business object represents; it does *not* send this event to the integration broker.
- If these `ObjectEventIds` are *not* the same, the business object does *not* represent a duplicate event. In this case, the connector framework copies the current business object to the JMS monitor queue and then delivers it to the JMS delivery queue, all as part of the same JMS transaction. The name of the JMS delivery queue is obtained from the `DeliveryQueue` connector configuration property. Control returns to the connector’s `pollForEvents()` method, after the call to the `getAppEvent()` method.

For a JMS-enabled connector to support duplicate event elimination, you must make sure that the connector’s `pollForEvents()` method includes the following steps:

- When you create a business object from an event record retrieved from the non-JMS event store, save the event record’s unique event identifier as the business object’s `ObjectEventId` attribute.

The application generates this event identifier to uniquely identify the event record in the event store. If the connector goes down after the event has been sent to the integration broker but before this event record’s status can be changed, this event record remains in the event store with an In-Progress status. When the connector comes back up, it should recover any In-Progress events. When the connector resumes polling, it generates a business object for the event

record that still remains in the event store. However, because both the business object that was already sent and the new one have the same event record as their ObjectEventIds, the connector framework can recognize the new business object as a duplicate and not send it to the integration broker.

- During connector recovery, make sure that you process In-Progress events *before* the connector begins polling for new events.

Unless the connector changes any In-Progress events to Ready-for-Poll status when it starts up, the polling method does not pick up the event record for reprocessing.

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 15 shows property names for queue URIs.

Table 15. SWIFT-specific connector property names for queue URIs

Property name	Description	Values
expiry	Lifetime of the message in milliseconds.	0 = unlimited.
priority	Priority of the message.	positive integers = timeout (in ms). 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
CCSID ¹ on page 29	Character set of the destination.	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. 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.

Table 15. SWIFT-specific connector property names for queue URIs (continued)

Property name	Description	Values
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 16 describes the meta-object properties.

Table 16. Static meta-object properties

Property name	Description
CollaborationName	<p>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 meta-data object must contain an attribute named <code>Swift_MTnnn_Verb</code>, where <code>nnn</code> is the Swift message type, for example, <code>Swift_MT502_Create</code>. The <code>Swift_MT502_Create</code> attribute must contain application-specific text that includes a name-value pair. For example, <code>CollaborationName=MyCustomerProcessingCollab</code>. See the “Application-specific information” on page 31 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.</p> <p>Note: This property is available only for synchronous requests.</p>
DoNotReportBusObj	<p>Optionally, you can include the <code>DoNotReportBusObj</code> 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.</p> <p>Note: This property is available only for synchronous requests.</p>
InputFormat	<p>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.</p>
OutputFormat	<p>The output format is set on messages created from the given business object. If a value for the <code>OutputFormat</code> property is not specified, the input format is used, if available. An <code>OutputFormat</code> property value defined in a dynamic child meta-object overrides the value defined in the static meta-object.</p>
InputQueue	<p>The input queue that the connector polls to detect new messages. You can use connector-specific properties to configure multiple <code>InputQueues</code> and optionally map different data handlers to each queue.</p>
OutputQueue	<p>The output queue is the queue to which messages derived from the given business object are delivered. An <code>OutputQueue</code> property value defined in a dynamic child meta-object overrides the value defined in the static meta-object.</p>

Table 16. Static meta-object properties (continued)

Property name	Description
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 the integration broker. This causes the integration broker 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.

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 meta-data through a static meta-object, the connector can optionally accept meta-data 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 17 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 17. Static meta-object structure for Swift_MT502_Create

Attribute name	Application-specific text
Swift_MT502_Create	InputFormat=ORDER_IN;OutputFormat=ORDER_OUT;OutputQueue=QueueA;Response

Table 18 shows a sample dynamic child meta-object for business object `Swift_MT_Create`.

Table 18. 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 the integration broker 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 19 shows how a dynamic child meta-object might be structured for polling.

Table 19. 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 19, 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 queue `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 20. Dynamic meta-object header attributes

Header attribute name	Mode	Corresponding JMS header
CorrelationID	Read/Write	JMSCorrelationID
ReplyToQueue	Read/Write	JMSReplyTo
DeliveryMode	Read	JMSDeliveryMode
Priority	Read	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.

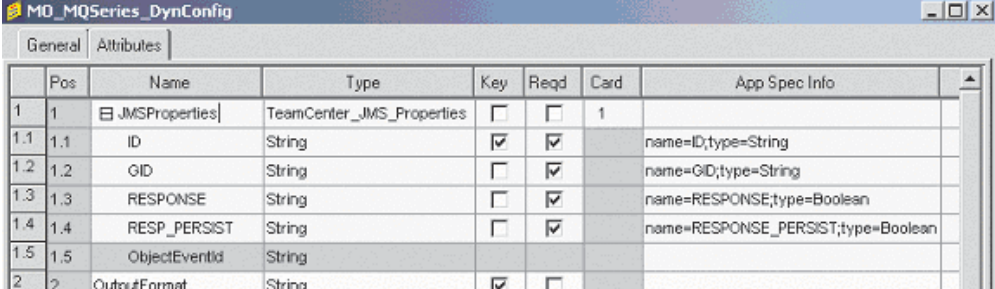
The table below shows application-specific information properties that you must define for attributes in the `JMSProperties` object.

Table 21. 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: <code>setIntProperty</code> , <code>setLongProperty</code> , <code>setStringProperty</code> , 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	Req'd	Card	App Spec Info
1	1	JMSProperties	TeamCenter_JMS_Properties	<input type="checkbox"/>	<input type="checkbox"/>	1	
1.1	1.1	ID	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		name=ID;type=String
1.2	1.2	GID	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		name=GID;type=String
1.3	1.3	RESPONSE	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		name=RESPONSE;type=Boolean
1.4	1.4	RESP_PERSIST	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		name=RESPONSE_PERSIST;type=Boolean
1.5	1.5	ObjectEventId	String				
2	2	OutputFormat	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>		

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 the integration broker 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 `ReplyToQueue` value 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 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 and UNIX systems.

Windows

To complete the configuration of the connector for Windows platforms, 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.

UNIX

To complete the configuration of the connector for UNIX 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.

Startup

For information on starting a connector, stopping a connector, and the connector’s temporary startup log file, see the startup chapter in the *System Installation Guide* for your platform.

Chapter 3. Business objects

- “Connector business object requirements”
- “Overview of SWIFT message structure” on page 47
- “Overview of business objects for SWIFT” on page 48
- “SWIFT message and business object data mapping” on page 49

The connector for SWIFT is a meta-data-driven connector. In WebSphere business objects, meta-data 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 meta-data-driven connector handles each business object that it supports based on meta-data encoded in the business object definition rather than on instructions hard-coded in the connector.

Business object meta-data 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 meta-data-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.

Important: The connector supports business object mapping between the ISO 7775-15022 message formats for SWIFT Category 5, Securities Markets, but only with the expanded business object definitions available with release 1.3 (and later) of this connector. To install the object definition files, see Chapter 2, “Configuring the connector”, on page 17. *If you use business object definitions from release 1.2 or earlier, the connector cannot perform ISO 7775-15022 business object transformations.*

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
- a business object representing a SWIFT ISO 7775 message into a business object representing the corresponding SWIFT ISO 15022 message, 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 5, “SWIFT Data Handler”, on page 123.

A review of the following WebSphere documents is strongly recommended:

- *IBM WebSphere InterChange Server Technical Introduction to IBM WebSphere InterChange Server* (when ICS is the integration broker)
- *IBM WebSphere Business Integration Adapters Implementation Guide for WebSphere MQ Integrator Broker* (when MQ Integrator Broker is the integration broker)
- *Business Object Development Guide*

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 (relevant only when ICS is the integration broker).

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 (relevant only when ICS is the integration broker).

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 (relevant only when ICS is the integration broker).

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 object 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 22 describes the name-value parameters for attribute application-specific text.

Table 22. 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 47.
parse	Yes for attributes of the top-level object only	Describes whether, and how, to parse the SWIFT message block. Values are: <code>fixlen</code> —parse as fixed length <code>delim</code> —parse as delimited text <code>field</code> —Block 4 only <code>no</code> —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 D, “SWIFT message structure”, on page 173. For further information on sequence and field business objects, see “Block 4 business object structure” on page 61.
letter= <i>a</i>	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 20A 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.

Note: The application-specific information for production instruction meta-objects (PIMOs) contains name-value pairs that indicate compute instructions. For more information, see Chapter 4, “ISO 7775 to ISO 15022 mapping”, on page 81.

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 D, “SWIFT message structure”, on page 173, 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:

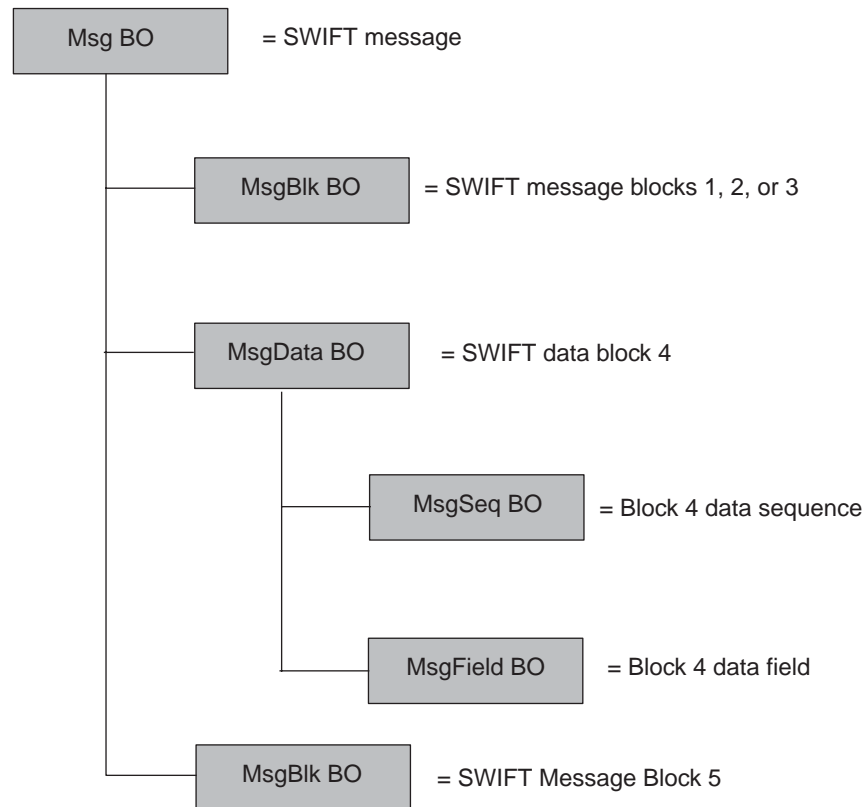


Figure 5. Business objects map to SWIFT message components

- **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 49.
- **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 53.
- **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 61.
- **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 66.

- **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 69.

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 49.
 - Swift_ApplicationHeader is the name of a MsgBlk BO. For further information, see “Block 1 business object structure” on page 53, “Block 2 business object structure” on page 55, and “Block 3 business object structure” on page 59.
 - Swift_MT502Data is the name of a MsgData BO. For further information, see “Block 4 business object structure” on page 61.
 - Swift_MT502_B1 is the name of a MsgSeq BO. For further information, see “Sequence business object structure” on page 66.
 - Swift_Tag_22 is the name of a MsgField BO. For further information, see “Field business object definitions” on page 69.
- **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 44 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 Adapter for SWIFT supports two kinds of mapping:

- **SWIFT-message-to-WebSphere-business-object** The sections below describe the data mapping that occurs between SWIFT messages and WebSphere business objects.
- **Business-object-to-business-object** The mapping used to transform a business object that represents an ISO 7775 SWIFT message into a business object that represents the corresponding ISO 15022 SWIFT message, and vice versa. For further information, see Chapter 4, “ISO 7775 to ISO 15022 mapping”, on page 81.

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 23, 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 23. For a listing of all attribute properties, see “Sample top-level Business Object (Msg BO)

definition” on page 51.

Table 23. 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_ApplicationHeader	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_MTMessageType

where:

BOPrefix = an attribute of the meta-object (MO). For further information on the meta-object, see “Static meta-object” on page 29.

_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 22 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 22. 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 51. 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-Specific...
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 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 24. 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 24. For a listing of all attribute properties, see “Sample block 1 business object definition” on page 54.

Table 24. 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
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 22 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 D, “SWIFT message structure”, on page 173, 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.

The screenshot shows a window titled "Swift_MT502A" with two tabs: "General" and "Attributes". The "Attributes" tab is active, displaying a table with the following data:

	Pos	Name	Type	Key	Reqd	Card	Max L	App Spec Info	Comments
2	2	Swift_01Header	Swift_BasicHeader	<input type="checkbox"/>	<input type="checkbox"/>	1		block=1;parse=fixle	
2.1	2.1	BlockIdentifier	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		2		
2.2	2.2	ApplicationIdentifie	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		1		
2.3	2.3	ServiceIdentifier	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		2		
2.4	2.4	LTIdentifier	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		12		
2.5	2.5	SessionNumber	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		4		
2.6	2.6	SequenceNumber	String	<input type="checkbox"/>	<input type="checkbox"/>		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 and then enter values for the attributes shown in “Sample block 1 business object definition” on page 54.

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 25. 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 25. For a listing of all attribute properties, see “Sample block 2 business object definition” on page 56.

Table 25. 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

Table 25. Block 2 business object structure (continued)

Name	Type	Key	Required	Cardinality	Default	Max length
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 25 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 22 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_) are populated during SWIFT-to-business-object conversion. For further information on this SWIFT message block, see Appendix D, "SWIFT message structure", on page 173, 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	<input type="checkbox"/>	<input type="checkbox"/>	1		block=1;parse=fixle
3	3	Swift_02Header	Swift_ApplicationHe	<input type="checkbox"/>	<input type="checkbox"/>	1		block=2;parse=fixle
3.1	3.1	BlockIdentifier	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		2	
3.2	3.2	IOIdentifier	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		1	
3.3	3.3	MessageType	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		3	
3.4	3.4	I_ReceiverAddress	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		12	
3.5	3.5	I_MessagePriority	String	<input type="checkbox"/>	<input checked="" type="checkbox"/>		1	
3.6	3.6	I_DeliveryMonitorin	String	<input type="checkbox"/>	<input type="checkbox"/>		1	
3.7	3.7	I_ObsolescencePe	String	<input type="checkbox"/>	<input type="checkbox"/>		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 and then enter values for the attributes shown in "Sample block 2 business object definition" on page 56.

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 = 0
  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 26. Note that there is attribute application-specific information for this business object: the Tag parameter. For Tag parameters see Table 22.

Note: Only attribute properties of consequence are shown in Table 26. For a listing of all attribute properties, see “Sample block 3 business object definition” on page 60.

Table 26. 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 D, “SWIFT message structure”, on page 173, 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	Req'd	Card	Max L	App Spec Info
1	UUID	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>		255	
2	Swift_01Header	Swift_BasicHeader	<input type="checkbox"/>	<input type="checkbox"/>	1		block=1;parse=fixde
3	Swift_02Header	Swift_ApplicationHe	<input type="checkbox"/>	<input type="checkbox"/>	1		block=2;parse=fixde
4	Swift_03Header	Swift_UserHeader	<input type="checkbox"/>	<input type="checkbox"/>	1		block=3;parse=deli
4.1	Tag103	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>		255	Tag=103
4.2	Tag113	String	<input type="checkbox"/>	<input type="checkbox"/>		255	Tag=113
4.3	Tag108	String	<input type="checkbox"/>	<input type="checkbox"/>		255	Tag=108
4.4	Tag119	String	<input type="checkbox"/>	<input type="checkbox"/>		255	Tag=119
4.5	Tag115	String	<input type="checkbox"/>	<input type="checkbox"/>		255	Tag=115
4.6	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>			

Figure 9. Block 3 business object definition

Note: To create a block 3 business object definition for a SWIFT message, start Business Object Designer and then enter values for the attributes shown in “Sample block 3 business object definition” on page 60.

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.

Figure 10 shows a portion of the format specification from the *SWIFT Standards Release Guide* for MT502, an order to buy or sell.

Status	Tag	Qualifier	Generic Field Name	Detailed Field Name	Content/Options	No.
Mandatory Sequence A General Information						
M	16R			Start of Block	GENL	<u>1</u>
M	20C	SEME	Reference	Sender's Reference	:4!c//16x	<u>2</u>
M	23G			Function of the Message	4!c[4!c]	<u>3</u>
O	98a	PREP	Date/Time	Preparation Date/Time	A or C	<u>4</u>
----->						
M	22F	4!c	Indicator	(see qualifier description)	:4!c[8c]/4!c	<u>5</u>

-----> Repetitive Optional Subsequence A1 Linkages						
M	16R			Start of Block	LINK	<u>6</u>
O	22F	LINK	Indicator	Linkage Type Indicator	:4!c[8c]/4!c	<u>7</u>
O	13A	LINK	Number Identification	Linked Transaction	:4!c/3!c	<u>8</u>
M	20C	4!c	Reference	(see qualifier description)	:4!c//16x	<u>9</u>
M	16S			End of Block	LINK	<u>10</u>

Figure 10. SWIFT message type 502 format specification

Figure 11 shows the corresponding portion of a business object definition, which 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 11 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 11 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 11 shows the corresponding attribute application-specific text: only the Tag and Letter are shown in the AppSpecInfo field (Tag=20;Letter=C). The SWIFT data handler also parses the field's data content—the formatting information (:4!c[/4!c]) is

included in the business object definition in ways that support mapping between ISO 7775 and ISO 15022 message formats.

- Repeating sequences in SWIFT messages are indicated by “---->” in the SWIFT Format Specifications as shown in Figure 10. Nonrepeating 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	Req'd	Card	Max L	App Spec Info	Cor
5	5	Swift_MT502Data	Swift_MT502Data	<input type="checkbox"/>	<input type="checkbox"/>	1		block=4;parse=field	
5.1	5.1	Swift_MT502_A_General_Information	Swift_MT502_A_General_Information	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1			
5.1.1	5.1.1	Start_Of_Block	Swift_Tag_16	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1		Tag=16;Letter=R;Content=GENL	
5.1.2	5.1.2	Senders_Reference	Swift_Tag_20	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1		Tag=20;Letter=C	
5.1.2.1	5.1.2.1	Letter	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>		255		
5.1.2.2	5.1.2.2	Qualifier	String	<input type="checkbox"/>	<input type="checkbox"/>		255		
5.1.2.3	5.1.2.3	IC	String	<input type="checkbox"/>	<input type="checkbox"/>		255		
5.1.2.4	5.1.2.4	Data	String	<input type="checkbox"/>	<input type="checkbox"/>		255		
5.1.2.5	5.1.2.5	ObjectEventId	String						
5.1.3	5.1.3	Function_Of_The_Message	Swift_Tag_23	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1		Tag=23;Letter=J	
5.1.4	5.1.4	Preparation_DateTime	Swift_Tag_98	<input type="checkbox"/>	<input type="checkbox"/>	1		Tag=98;Letter=A C	
5.1.5	5.1.5	Indicator	Swift_Tag_22	<input type="checkbox"/>	<input checked="" type="checkbox"/>	n		Tag=22;Letter=F	
5.1.6	5.1.6	Swift_MT502_A1_Linkages	Swift_MT502_A1_Linkages	<input type="checkbox"/>	<input type="checkbox"/>	n			
5.1.7	5.1.7	End_Of_Block	Swift_Tag_16	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1		Tag=16;Letter=S;Content=GENL	
5.1.8	5.1.8	ObjectEventId	String						
5.2	5.2	Swift_MT502_B_Order_Details	Swift_MT502_B_Order_Details	<input type="checkbox"/>	<input type="checkbox"/>	n			

Figure 11. 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 66 and “Field business object definitions” on page 69.

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 22 on page 47. 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 69.

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 12 illustrates the hierarchical relationship of these business objects.

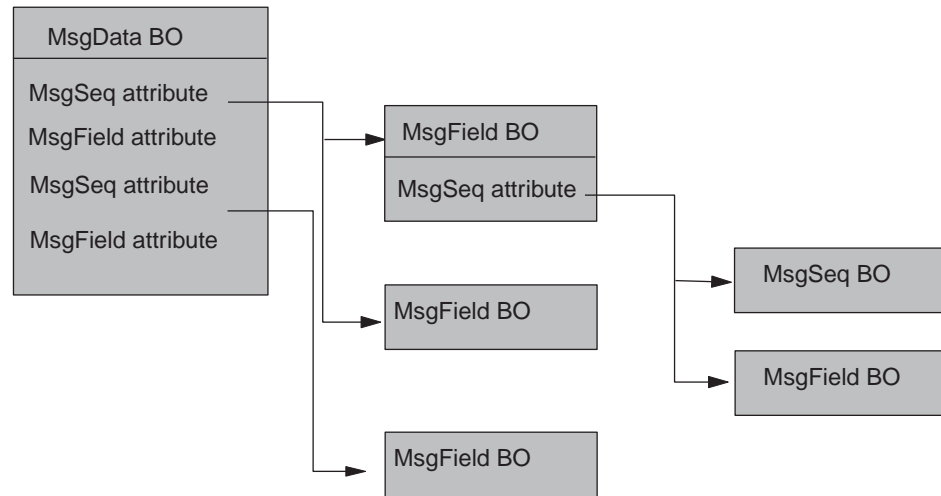


Figure 12. Field and sequence business objects in the (block 4) MsgData BO

Figure 13 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	
5.1.	5.1.7	End_Of_Block	Swift_Tag_16	<input type="checkbox"/>	<input checked="" type="checkbox"/>	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	<input type="checkbox"/>	<input type="checkbox"/>	n			
5.2.	5.2.1	Start_Of_Block	Swift_Tag_16	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1		Tag=16;Letter=R;Co	
5.2.	5.2.2	Place_Of_Trade	Swift_Tag_94	<input type="checkbox"/>	<input type="checkbox"/>	1		Tag=94;Letter=B	
5.2.	5.2.3	Swift_MT502_B1_Price	Swift_MT502_B1_Price	<input type="checkbox"/>	<input type="checkbox"/>	n			
5.2.	5.2.3.	Start_Of_Block	Swift_Tag_16	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1		Tag=16;Letter=R;Co	
5.2.	5.2.3.	Price	Swift_Tag_90	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1		Tag=90;Letter=A B	
5.2.	5.2.3.	Type_Of_Price	Swift_Tag_22	<input type="checkbox"/>	<input type="checkbox"/>	1		Tag=22;Letter=F	
5.2.	5.2.3.	End_Of_Block	Swift_Tag_16	<input type="checkbox"/>	<input checked="" type="checkbox"/>	1		Tag=16;Letter=S;Co	

Figure 13. A Sequence containing tag and subsequence attributes

Sequence business object structure

As shown in Figure 14, 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 MsgSeq BO can nest.

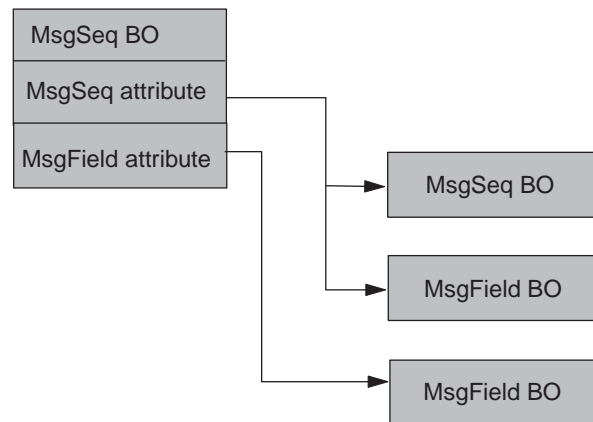


Figure 14. Field and Subsequence Business Objects in the MsgSeq BO

Figure 15 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									
General Attributes									
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 15. 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 16, each MsgField BO contains five attributes, including one and only one TagUnion BO, with the data type shown in parentheses () below:

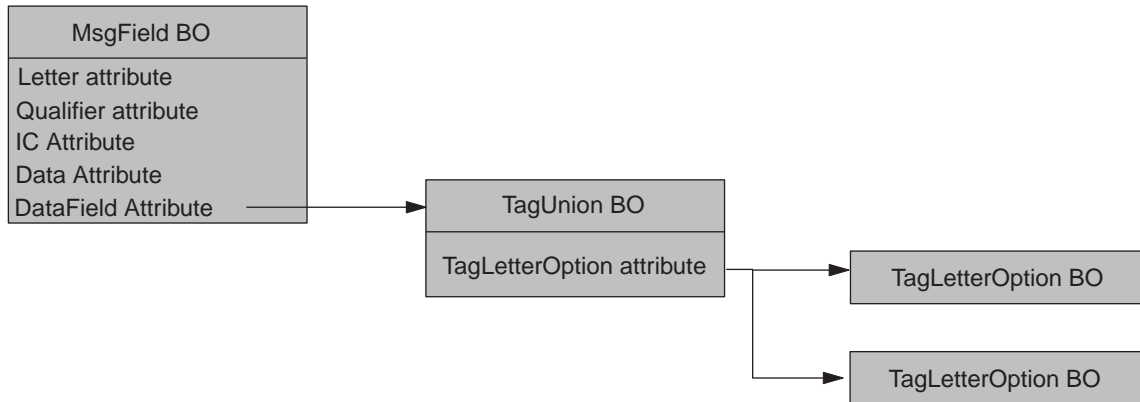


Figure 16. Attributes and business objects in the MsgField BO

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 16 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 17.

```
[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 17. SWIFT field definition

Chapter 4. ISO 7775 to ISO 15022 mapping

- “Production instruction meta-objects (PIMOs)”
- “Creating PIMOs” on page 88
- “Modifying PIMOs: Map summary” on page 96

The IBM WebSphere Business Adapter for SWIFT dynamically transforms a business object representing an ISO 7775 SWIFT message into a business object representing the corresponding ISO 15022 SWIFT message and vice versa. The transformation is performed by a mapping engine. The mapping engine is governed by a production instruction meta-object (PIMO). This chapter describes PIMOs, how they map business object attributes, and how to create or modify a PIMO.

Important: The adapter supports business object ISO 7775-15022 mapping for SWIFT Category 5, Securities Markets, but only with the expanded business object definitions available with this release and described in Chapter 3, “Business objects”, on page 43, and only on Solaris platforms. The mapping capability does not support transformation of business objects released with 1.2 or earlier releases of the adapter for SWIFT.

Production instruction meta-objects (PIMOs)

A PIMO is a hierarchical meta-object that supports the transformation of business objects from one format to another. PIMOs specify not only the attribute-to-attribute mapping but also the computation instructions required to perform the transformation. The mapping and the computation instructions constitute meta-data that is used by the mapping engine.

Map_objects.txt contains 20 PIMOs in their entirety. Each PIMO contains all of the meta-data required to transform a business object representing an ISO 7775- or 15022-formatted SWIFT message to a business object representing the corresponding ISO 15022- or 7775-formatted SWIFT message. For more on these PIMOs and how to create or modify them, see “Creating PIMOs” on page 88.

PIMO structure and syntax

As Figure 18 shows, a simple PIMO has two mandatory child objects, Port and Action, and an optional child object, Declaration.

The screenshot shows the 'Business Object Designer' window with the title '[Map_Swift_MT520_to_MT540]'. The 'Attributes' tab is selected, displaying a table with the following data:

	Pos	Name	Type	Key	Card	Default	App S
1	3	Action	Map_Swift_MT520_to_MT540_Action	<input type="checkbox"/>	1		
2	2	Declaration	Map_Swift_MT520_to_MT540_Declaration	<input type="checkbox"/>	1		
3	4	ObjectEventId	String				
4	1	Port	Map_Swift_MT520_to_MT540_Port	<input checked="" type="checkbox"/>	1		
4.1	1.1	IPort	Swift_MT520	<input checked="" type="checkbox"/>	1		
4.2	1.3	ObjectEventId	String				
4.3	1.2	OPort	Swift_MT540	<input type="checkbox"/>	1		
4.4	1.4			<input type="checkbox"/>			

Figure 18. Simple PIMO

The attribute structure is as follows:

Port This child object always contains an input port and output port.

- **IPort** The input port
- **OPort** The output port

Declaration This optional child object contains attributes that name local variables.

Action This attribute describes objects that contain compute instructions. The instructions, which reside in the attribute's application-specific information, are used to process Declaration variables and Port objects. Action child objects represent computational sub-tasks listed in the order of execution:

- Action1
- Action2
- ...
- Actionn

In fact, as shown in Figure 19, the PIMOs that are used to map SWIFT business objects are hierarchical objects that contain many nested levels of PIMOs, each with their own Port, Action, and Declaration objects as well as discrete mapping and computing instructions. At every level, however, the Port, Action and Declaration attributes exhibit the same syntax, structure, and function, which are described in the sections below.

Note: A simple PIMO can have multiple nested levels of Port, Declaration, and Action objects. A complex PIMO has more than one Port object on the same level. The PIMOs available with this release are simple PIMOs.

Pos	Name	Type	Key	Foreign	App Spec
1	Port	Map_Swift_MT520_to_MT540_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
2	Declaration	Map_Swift_MT520_to_MT540_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	
3	Action	Map_Swift_MT520_to_MT540_Action	<input type="checkbox"/>	<input type="checkbox"/>	
3.1	Action1	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	type=compute;opCode=move;target=OPort.Swift_01Header;IPort.Swift_01Header
3.2	Action2	String	<input type="checkbox"/>	<input type="checkbox"/>	type=compute;opCode=move;target=OPort.Swift_02Header;IPort.Swift_02Header
3.3	Action3	String	<input type="checkbox"/>	<input type="checkbox"/>	type=compute;opCode=move;target=OPort.Swift_03Header;IPort.Swift_03Header
3.4	Action4	String	<input type="checkbox"/>	<input type="checkbox"/>	type=compute;opCode=move;target=OPort.Swift_02Header.MessageType;TargetMTNum
3.5	Action5	String	<input type="checkbox"/>	<input type="checkbox"/>	type=nativeStatic;class=com.crossworlds.DataHandlers.swift.Swift_Map_Uilities;method=replaceMTN
3.6	Action6	Map_Swift_MT520_A_to_MT540_A	<input type="checkbox"/>	<input type="checkbox"/>	type=delegate;IPort.Swift_MT520Data.Swift_MT520_A;OPort.Swift_MT540Data.Swift_MT540_A
3.6.1	Port	Map_Swift_MT520_A_to_MT540_A_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3.6.2	Action	Map_Swift_MT520_A_to_MT540_A_Action	<input type="checkbox"/>	<input type="checkbox"/>	
3.6.2.1	Action1	Map_Swift_Tag_20_to_20C	<input checked="" type="checkbox"/>	<input type="checkbox"/>	type=delegate;IPort.Swift_20_1;OPort.Swift_20_1
3.6.2.2	Port	Map_Swift_Tag_20_to_20C_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
3.6.2.3	Declaration	Map_Swift_Tag_20_to_20C_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	
3.6.2.4	Action	Map_Swift_Tag_20_to_20C_Action	<input type="checkbox"/>	<input type="checkbox"/>	
3.6.2.5	Action1	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	type=compute;opCode=move;target=OPort.Letter;Letter
3.6.2.6	Action2	String	<input type="checkbox"/>	<input type="checkbox"/>	type=compute;opCode=move;target=OPort.Qualifier;Qualifier
3.6.2.7	Action3	String	<input type="checkbox"/>	<input type="checkbox"/>	type=compute;opCode=move;target=OPort.DataField.Swift_20_C.Reference;IPort.DataField.Swift_20.Tr
3.6.2.8	ObjectEve	String			
3.6.2.9	ObjectEventId	String			
3.6.2.10	ObjectEventId	String			
3.6.2.11	ObjectEventId	String			
3.6.3	ObjectEventId	String			

Figure 19. Excerpt of expanded PIMO

Port

The Port object type describes the specific transformation the mapping engine undertakes and has the following naming syntax:

PimoName>_Port

In Figure 18, for example, the Port is of type *Map_Swift_MT520_to_MT540_Port*, which names the transformation of a business object representing an ISO 7775 SWIFT message to an object representing an ISO 15022 SWIFT message.

Port contains as child objects IPort (input port) and OPort (output port). The types of the IPort and OPort attributes describe the parameters to be mapped: an IPort is used to pass an input parameter (for example, *Swift_MT_520*), and an OPort is used to pass an output parameter (for example *Swift_MT_540*). The parameters are passed by way of reference to their object types.

The IPort and OPort object may be a primitive type, such as *int*, *float*, or *String*, or a business object, such as *Swift_MT520*. The computation instructions contained in an Action child object refer to, and act on, these IPort and OPort object types.

Note: IPort and OPort cannot specify a meta-object as a type.

By convention, the Port and IPort attributes are marked as key (*IsKey* = *true*).

Declaration

The Declaration object type describes the objects to be transformed and has the following naming syntax:

*PimoName*_Declaration

The attributes in Declaration child objects specify local variables for the computing instructions that are detailed in Action objects for the Port. The attribute type declares the type of variable.

Variables can be constant or variable. The keyword `final` in a Declaration object's `AppSpecificInfo` designates a constant variable. If a Declaration object's `AppSpecificInfo` is blank, the variable is not constant. In the PIMO excerpt shown in Figure 20, for example, `Qualifier` and `Letter` are declared constant variables; `Var Boolean` and the `Map_Swift_Tag35E_to_70E_Declaration` itself are variable.

	Pos	Name	Type	Key	Foreign	Reqd	Card	Max L	Default	App Spec Info	Comments
1	1	Port	Map_Swift_Tag_35E_to_70E_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
2	2	Declaration	Map_Swift_Tag_35E_to_70E_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
2.1	2.1	Letter	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	E	final	
2.2	2.2	Qualifier	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	DECL	final	
2.3	2.3	Var_Boolean	Boolean	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>					
2.4	2.4	ObjectEventId	String								
3	3	Action	Map_Swift_Tag_35E_to_70E_Action	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
4	4	ObjectEventId	String								
5	5			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255			

Figure 20. Declarations in a PIMO

By convention, the first variable in the Declaration object is marked as key (`IsKey = true`).

Note: A Declaration object is optional; when Action objects require no local variable, the Declaration attribute can be omitted. The data type of the Declaration child object cannot be a meta-object.

Action

The Action object describes the computing instructions that the mapping engine performs. The top-level Action object has the following naming syntax:

PimoName_Action

The computing instructions specified in Action objects act on the `IPort` and `OPort` objects and use the variables specified in Declaration objects. For each Port and Declaration, the Action objects aggregate, in sequential order, all of the computing instructions for the map engine. No space or period is allowed in the Action type names.

Actions can be one of the following types:

- Compute
- Delegate
- Native
- Scenario

Compute action: A Compute type indicates that the Action can be performed by a simple operation. All compute type Actions use the keyword `opCode` to specify the

name of operation. The keyword Target designates the receiving party of the operation. The syntax for a compute type Action is specified in its AppSpecificInfo and is as follows:

opCode=<opCode>;target=<variable>;<parameters>

where *opCode*, *variable*, and *parameters* are described in Table 27.

Table 27. Compute action opCodes

opCode	Example	Description
+	type=compute;opCode=+;target=Var_B;Var_1;Var_2	Addition. Var_2 is added to Var_1 and the sum assigned to the target, Var_B; applies to string and numeric type variables.
-	type=compute;opCode=-;target=Var_B;Var_1;Var_2	Subtraction. Var_2 is subtracted from Var_1 and the difference is assigned to the target, Var_B. Applies to numeric type variables.
move	type=compute;opCode=move;target=0Port;Var_1	Copy the value of Var_1 to the target, 0Port. Applies to all variable types.
index	type=compute;opCode=index;target=var_4;Var_1;Var_2;Var_3	Looks for first occurrence of string Var_2 in the string Var_1, starting at the position Var_3.
substring	type=compute;opCode=substring;target=result;sourceString;index1;index2	Returns Var_4 to the target, 0Port. Var_4 can be -1 if Var_2 is not found. Applies to string variables only. Assign target the result of the substring of sourceString from index1 to index2. When index2 exceeds the string length of sourceString, index2 is deemed the same length as that of sourceString.
append	type=compute;opCode=append;target=result;source1String;index1;source2String;index2	Append source2String to source1String at index. The result is assigned to the target, result.
size	type=compute;opCode=size;target=0Port;Var;	Assign target the length of variable Var when Var is of type String. When Var is of type containment, assign target the number of instances of Var.

Delegate action: Delegate Action is specified when more than one Compute Action is required. A Delegate Action relies on nested PIMOs to complete the Action, and in this sense is analogous to a function call. The syntax of a Delegate Action object is specified in its AppSpecificInfo and is as follows:

type=delegate;<var1>;<var2>

where type indicates the Delegate Action type and *var1* is passed to the IPort of a child PIMO, and *var2* is passed to the OPort of the child PIMO. The relative position of the variables corresponds to the sequence of Ports in the invoked PIMO. (The invoked PIMO is the PIMO to which Action is delegated.) Looping

occurs if one or both of the IPort and OPort objects in the invoked PIMO is of cardinality n. The loop syntax of the delegation is as follows:

- If both the IPort and OPort objects in the invoked PIMO have cardinality n, then for each instance of these Port objects, an object is created and passed on to the invoked PIMO's IPort and OPort. Looping occurs as many times as the number of object instances, and delegation is passed by reference along with each instance.
- If the IPort object in the invoked PIMO has cardinality n and the OPort object does not have cardinality n, then for each instance of the IPort object, an object is created and passed on to the IPort of the invoked PIMO along with a reference to the type of the OPort object. Looping occurs as many times as the number of IPort object instances, and delegation is passed by reference along with each instance.

For example, in Figure 21, Action6 is of type Delegate. The computing tasks for this action are too complex for specification using Compute Action. In fact, the computing tasks require two levels of Delegate Action. The first Delegate Action is to a nested PIMO, Map_Swift_MT520_A_to_MT540_A_Port. The Action of this Port is also of type Delegate, with its variables passed to the IPort and OPort of Map_Swift_Tag_20_to_20C_Port. The Action of this nested PIMO is of the compute type, and the results are passed back up to the invoking PIMOs, just like a function call.

Pos	Name	Type	App Spec Info
1	Port	Map_Swift_MT520_to_MT540_Port	
2	Declaration	Map_Swift_MT520_to_MT540_Declaration	
2.1	MTNum	String	final
2.2	TargetMTNum	String	
2.3	SourceMTNum	String	
2.4	ObjectEventId	String	
3	Action	Map_Swift_MT520_to_MT540_Action	
3.1	Action1	String	type=compute;opCode=move,target=OPort_Swift_01Header;Port_Swift_01Header
3.2	Action2	String	type=compute;opCode=move,target=OPort_Swift_02Header;Port_Swift_02Header
3.3	Action3	String	type=compute;opCode=move,target=OPort_Swift_03Header;Port_Swift_03Header
3.4	Action4	String	type=compute;opCode=move,target=OPort_Swift_02Header.MessageType;TargetMTNum
3.5	Action5	String	type=nativeStatic;class=com.crossworlds.DataHandlers.swift.Swift_Map_Utils;method=replaceMTNumber;target=OP
3.6	Action6	Map_Swift_MT520_A_to_MT540_A	type=delegate;Port_Swift_MT520Data.Swift_MT520_A;OPort_Swift_MT540Data.Swift_MT540_A
3.6.1	Port	Map_Swift_MT520_A_to_MT540_A_Port	
3.6.1	IPort	Swift_MT520_A	
3.6.1	OPort	Swift_MT540_A	
3.6.1	ObjectEventId	String	
3.6.2	Action	Map_Swift_MT520_A_to_MT540_A_Action	
3.6.2	Action1	Map_Swift_Tag_20_to_20C	type=delegate;Port_Swift_20_1;OPort_Swift_20_1
3.6.2	Port	Map_Swift_Tag_20_to_20C_Port	
3.6.2	Declaration	Map_Swift_Tag_20_to_20C_Declaration	
3.6.2	Action	Map_Swift_Tag_20_to_20C_Action	
3.6.2	Action1	String	type=compute;opCode=move,target=OPort.Letter;Letter
3.6.2	Action2	String	type=compute;opCode=move,target=OPort.Qualifier;Qualifier
3.6.2	Action3	String	type=compute;opCode=move,target=OPort.DataField.Swift_20_C.Reference;Port.DataField.Swift_20.TransactionRefer
3.6.2	ObjectEve	String	

Figure 21. Delegate action in a PIMO

Native action: Native Action is used to invoke components implemented in Java. In particular, a Native Action object supports calls to the public static method of a Java class, which allows you to develop more complicated processing functions in Java. The syntax of a Native Action object is specified in its AppSpecificInfo and is as follows:

```
type=nativeStatic;class=<className>;method=<methodName>;
target=return;var1;var2, varn
```

where *type* indicates the Native Action type, *className* indicates the name of a Java class, *methodName* indicates the name of the static Java method that provides the functionality for the action, and the return value of the function call is passed to the target variables as follows: *var1* is passed to the first variable of the method, and *var2* will be passed to the second variable of the method, and so on.

Note: The order and type of variables in the method call must match the that of the parameters in the method.

Scenario action: A Scenario Action is a construct that makes possible conditional computing and branching in PIMOs.

A Scenario Action is a child object that consists of two to three attributes:

Scenario (Action Object)

- Scenario (attribute)
- TrueAction (attribute)
- FalseAction (attribute)

The Scenario attribute is always a boolean expression, and the TrueAction or FalseAction (or both) attributes contain conditional computing instructions that are processed after the boolean expression is evaluated.

The syntax of a Scenario Action object is specified in its AppSpecificInfo and is as follows:

type=scenario

The syntax of a Scenario attribute is specified in its AppSpecificInfo and is as follows:

Boolean_Expression

where *Boolean_Expression* corresponds to one of the entries in Figure 19.

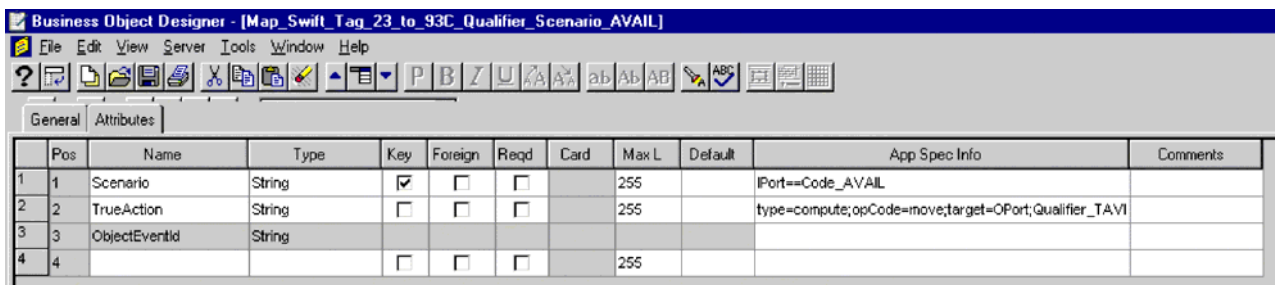
Table 28. Scenario attribute boolean expressions

Boolean symbol	Example	Description
==	IPort==LetterA	Equal. When applied to string type variables, equality means that the lexical content of the two strings is the same.
>	Var_A>Var_B	Greater than. When applied to String type variables, string length is compared.
>=	Var_A>=Var_B	Greater than or equal to. When applied to String type variables, string length is compared.
&&	(IPort==LetterA)&& (IPort==LetterC)	And
	(IPort==LetterA) (IPort==LetterC)	Or
<		Less than. When applied to String type variables, string length is compared.
<=		Less than or equal to. When applied to String type variables, string length is compared.

Note: PIMOs support parentheses in the Boolean expression.

A TrueAction attribute indicates the computing instruction that the map engine will process if the Boolean expression (in the Scenario attribute) evaluates to true; likewise, a FalseAction attribute indicates the action if the Boolean expression evaluates to false. Actions specified in the TrueAction and FalseAction can be Compute, Delegate and Native types, and follow the syntax of the Compute, Delegate and Native type Actions, respectively.

For example, Figure 22 shows the attributes of a Scenario Action. If the IPort Object is equal to Code_AVAL, then the computing instruction `type=compute;opCode=move;target=OPort;Qualifier_TAV1` will be processed by the map engine.



Pos	Name	Type	Key	Foreign	Reqd	Card	Max L	Default	App Spec Info	Comments
1	Scenario	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		IPort==Code_AVAL	
2	TrueAction	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255		type=compute;opCode=move;target=OPort;Qualifier_TAV1	
3	ObjectEventId	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255			
4			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255			

Figure 22. Scenario action in a PIMO

Object references in actions: Actions make reference to other PIMO objects, whether Port or Declaration objects. By convention, the application-specific information of Action objects follows the path name convention that uses a period to denote the attribute names at different levels in the hierarchy of the PIMO.

For example, consider an IPort of type `Swift_Tag_20` (the type shown in parentheses), which has the following hierarchy:

IPort (`Swift_Tag_20`)

- Letter (String)
- Qualifier (String)
- IC (String)
- Data (String)
- DataField (`Swift_Tag_Union_20`)
 - `Swift_20` (`Swift_Tag_Union_20_Opt`) Subfield (String)
 - `Swift_20_C` (`Swift_Tag_Union_20_Opt_C`) Reference (String)

Then the reference to the `Swift_20_C` attribute is `IPort.DataField.Swift_20_C`.

Creating PIMOs

This section describes how to create a PIMO using Business Object Designer. Before proceeding, review the previous sections of this chapter, and make sure you have access to the *Swift User Handbook* and *Business Object Designer*, with which you should be familiar. For more information, see the *Business Object Development Guide*. You should also have access to `Map_Objects.txt`, which contains the PIMOs shipped with this release as well as thousands of sub-maps and objects that may be of use.

Note: The example used to illustrate the process of creating PIMOs is a sample PIMO residing in Map_Objects.txt. For a closer look at any of the screenshots presented below, launch Business Object Designer and open Map_Swift_MT520_to_MT540 from your repository.

Getting started

- 1. Launch Business Object Designer.
- 2. Choose **Server>Connect** and login to your server.
This provides access to your repository. To load the repository, see Chapter 2, “Configuring the connector”, on page 17 and the system installation guide for your operating environment.
- 3. Choose **File>New**.
This displays the New Business Object Dialog.
- 4. Enter the Business Object name as follows:
Map_Swift_MT<SourceNN>_to_MT<DestinationNN>

where *SourceNN* is the SWIFT message type number that corresponds to the business object you want to transform and *DestinationNN* is the SWIFT message type number that corresponds to the destination business object. This is the name of a new PIMO representing the mapping between the source message type and the destination message type as shown in Figure 23.

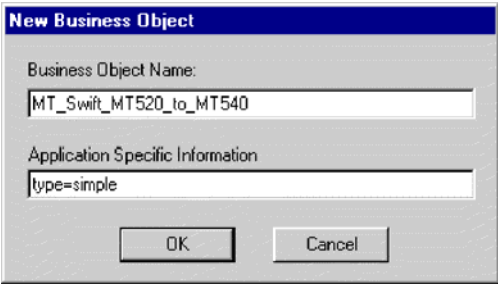


Figure 23. Naming a new PIMO

- 5. Enter type=simple in the Application Specific Information field.

Defining port

- 1. Enter **Port** in the Attribute Name field as shown in Figure 24.

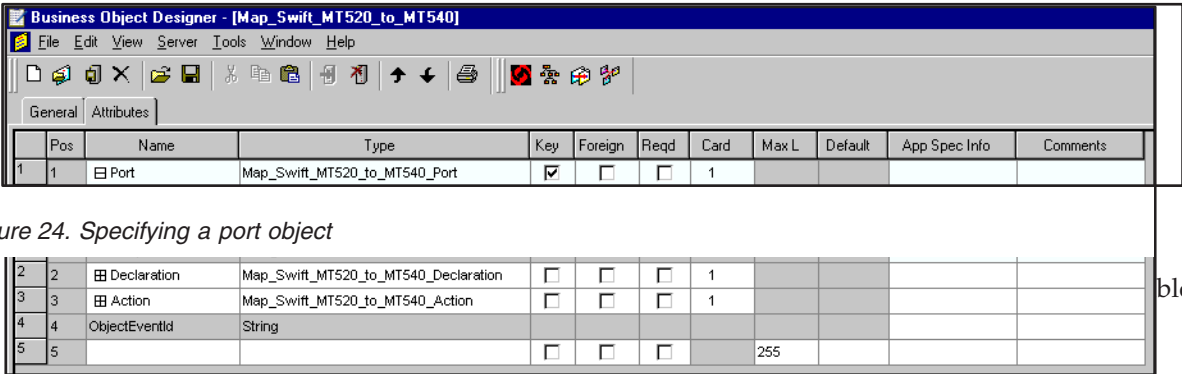


Figure 24. Specifying a port object

PimoName_Port

In the example shown in Figure 24, the type is as follows:

Map_Swift_MT520_to_MT540_Port

3. In the Port attribute row, check the Key property and enter **1** in the Cardinality field as shown in Figure 24.

Defining port child objects

The Port object must have two child objects, IPort and OPort, which correspond to the business objects representing the source and destination SWIFT messages.

1. Select ObjectEventId under Port and, from the Edit menu, choose **Insert Above**.
2. Enter **IPort** and **OPort** as names of new attributes under the Port object.
3. Right click the IPort Type field, and from the pop-up menu choose the object from those available in your repository.

The type syntax for the IPort is as follows:

Swift_<IPortSourceNN>

In the example shown in Figure 25 the IPort type is Swift_MT520.

4. Right click the OPort Type field, and from the pop-up menu choose the object from those available in your repository.

In the example shown in Figure 25 the OPort type is Swift_MT540.

5. In the IPort attribute row, check the Key property and enter **1** in the Cardinality field for both IPort and OPort as shown in Figure 25.

	Pos	Name	Type	Key	Foreign	Reqd	Card	Max L	Default	App Spec Info	Comments
1	1	Port	Map_Swift_MT520_to_MT540_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.1	1.1	IPort	Swift_MT520	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.2	1.2	OPort	Swift_MT540	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.3	1.3	ObjectEventId	String								

Figure 25. Specifying IPort and OPort objects

Defining declaration

A Declaration object at the top-level of a PIMO specifies high-level variables as required by Action objects at this level. As shown in Figure 26, the Declaration variable applies to the top-level Action object discussed below in “Defining action” on page 91. Following the naming convention for a Declaration object, the type of this variable is Map_Swift_MT520_to_MT540_Declaration.

1. Enter **Declaration** in the Name field under ObjectEventId.
2. Enter *PimoName_Declaration* in the Type field.

In the example, the Declaration type is Map_Swift_MT520_to_MT540_Declaration

3. Enter three attributes of type String with specifications as follows:
 - **MTNum** A constant (enter final in the App Spec Info field) with the Key property checked and a Default value of **MT<DestinationNN>**. In the example, the default is the destination SWIFT message type, **MT540**.
 - **TargetMTNum** A variable whose default value is the *DestinationNN*. In the example, the default value is 540.
 - **SourceMTNum** A variable whose default value is the *SourceNN*. In the example, the default value is 520.

	Pos	Name	Type	Key	Foreign	Reqd	Card	Max L	Default	App Spec Info	Comments
1	1	Port	Map_Swift_MT520_to_MT540_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.1	1.1	IPort	Swift_MT520	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.2	1.2	OPort	Swift_MT540	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.3	1.3	ObjectEventId	String								
2	2	Declaration	Map_Swift_MT520_to_MT540_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
2.1	2.1	MTNum	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	MT540	final	
2.2	2.2	TargetMTNum	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	540		
2.3	2.3	SourceMTNum	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	520		
2.4	2.4	ObjectEventId	String								

Figure 26. Specifying a Declaration for a Top-Level PIMO

For examples of sequence- and field-level Declarations, see “Representing sequence objects” on page 91 and “Representing field objects” on page 93.

Defining action

To define the top-level Action Object for the PIMO:

1. Enter Action in the Name field under ObjectEventId.
2. Enter Map_Swift_MT<SourceNN>_to_MT<DestinationNN>Action in the Type field.

In the example, the Action type is Map_Swift_MT520_to_MT540_Action, as shown in Figure 27.

	Pos	Name	Type	Key	Foreign	Reqd	Card	Max L	Default	App Spec Info	Comments
1	1	Port	Map_Swift_MT520_to_MT540_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.1	1.1	IPort	Swift_MT520	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.2	1.2	OPort	Swift_MT540	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
1.3	1.3	ObjectEventId	String								
2	2	Declaration	Map_Swift_MT520_to_MT540_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
3	3	Action	Map_Swift_MT520_to_MT540_Action	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1				
4	4	ObjectEventId	String								
5	5			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255			

Figure 27. Specifying an action for a top-level PIMO

To create all other Action objects for a PIMO mapping SWIFT MT520 to MT 540, you must become familiar with the actual tag- and field-level maps for these messages. This information is available in the *SWIFT User Handbook, Category 5, Securities Markets Message Usage Guidelines*. The PIMO models the SWIFT tag and field maps.

The sections below describe how to extract this information and create Action objects that represent SWIFT sequences and fields.

Representing sequence objects

Review the maps in the *SWIFT User Handbook, Category 5, Securities Markets Message Usage Guidelines* Appendix for SWIFT MT520 to MT 540 and identify the following:

- The number of sequences in the source message type (MT520)
- The number of sequences in the destination message type (MT 540)

Use the following naming convention for representing a sequence in a PIMO:
Map_Swift_MT<SourceNN>_<X>_to_MT<DestinationNN>_<X>

where *X* stands for the sequence letter and is always in uppercase.

Note: PIMO sequence object names can also reflect two special cases:

- The source message tag has a sequence but the destination does not:
– Map_Swift_MT<SourceNN>_<X>_to_MT<DestinationNN>
- The destination message tag has a sequence but the source does not:
– Map_Swift_MT<SourceNN>_to_MT<DestinationNN>_<X>

As shown in Figure 28, the child objects representing some of the sequences make use of Delegate Action because the mapping requires multiple Compute Action objects.

Pos	Name	Type	Key	Foreign	Req'd	Card	
1	Port	Map_Swift_MT520_to_MT540_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
2	Declaration	Map_Swift_MT520_to_MT540_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
3	Action	Map_Swift_MT520_to_MT540_Action	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
3.1	Action1	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=compute;opCode=move;target=OPort.Swift_01Header;IPort.Swift_01Header
3.2	Action2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=compute;opCode=move;target=OPort.Swift_02Header;IPort.Swift_02Header
3.3	Action3	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=compute;opCode=move;target=OPort.Swift_03Header;IPort.Swift_03Header
3.4	Action4	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=compute;opCode=move;target=OPort.Swift_02Header.MessageType;TargetMTNurr
3.5	Action5	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=nativeStatic;class=com.crossworlds.DataHandlers.swift.Swift_Map_Uilities;metho
3.6	Action6	Map_Swift_MT520_A_to_MT540_A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_A;OPort.Swift_MT540Data.Swift_M
3.7	Action7	Map_Swift_MT520_A_to_MT540_A1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_A;OPort.Swift_MT540Data.Swift_M
3.8	Action8	Map_Swift_MT520_A_to_MT540_B	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_A;OPort.Swift_MT540Data.Swift_M
3.9	Action9	Map_Swift_MT520_A_to_MT540_B1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_A;OPort.Swift_MT540Data.Swift_M
3.10	Action10	Map_Swift_MT520_A_to_MT540_E1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_A;OPort.Swift_MT540Data.Swift_M
3.11	Action11	Map_Swift_MT520_B_to_MT540_C	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_B;OPort.Swift_MT540Data.Swift_M
3.12	Action12	Map_Swift_MT520_C_to_MT540_E1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_C;OPort.Swift_MT540Data.Swift_M
3.13	Action13	Map_Swift_MT520_C_to_MT540_B	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_C;OPort.Swift_MT540Data.Swift_M
3.14	Action14	Map_Swift_MT520_C_to_MT540_C	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_C;OPort.Swift_MT540Data.Swift_M
3.15	Action15	Map_Swift_MT520_B_to_MT540_F	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.Swift_MT520Data.Swift_MT520_B;OPort.Swift_MT540Data.Swift_M
3.16	Action16	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=compute;opCode=move;target=OPort.Swift_05Trailer;IPort.Swift_05Trailer
3.17	Action17	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=compute;opCode=move;target=OPort.Swift_BlockS;IPort.Swift_BlockS
3.18	ObjectEventId	String					
4	ObjectEventId	String					

Figure 28. Specifying sequence objects with delegate action

When using Delegate Action, you must pass the IPort and OPort object paths. For example, Figure 28 shows that mapping computations from Swift_MT520_A (the child object of Swift_MT_520Data) to Swift_MT540_A (the child object of Swift_MT_540Data). Accordingly, the naming convention is as follows:

IPort.<Source_Object_Path>
OPort.<Destination_Object_Path>

In the example, the corresponding entries are:

IPort.Swift_MT520Data.Swift_MT520_A
OPort.Swift_MT540Data.Swift_MT540_A

For each delegated sequence shown in Figure 28, you must define the Port, Declaration, and Action objects for the parent. Declaration is mandatory at the

level of any Compute Action that refers to variables. Figure 29 shows how some of the Action and sub-Action objects are defined.

Pos	Name	Type	Key	Foreign	Reqd	Card	Max L	
3.9	Action9	Map_Swift_MT520_A_to_MT540_B1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate; Port.Swift_MT520Data.Swift_MT520_A;OPort.Swift_MT540
3.9.1	Port	Map_Swift_MT520_A_to_MT540_B1_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
3.9.2	Action	Map_Swift_MT520_A_to_MT540_B1_Action	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
3.9.2	Action1	Map_Swift_Tag_35D_to_98A	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate; Port.Swift_35_2;OPort.Swift_98_3
3.9.2	Action2	Map_Swift_Tag_35D_to_13A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate; Port.Swift_35_1;OPort.Swift_13_2
3.9.2	Action3	Map_Swift_Tag_33V_to_90B	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate; Port.Swift_33_1;OPort.Swift_90_2
3.9.2	Port	Map_Swift_Tag_33V_to_90B_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
3.9.2	Declaration	Map_Swift_Tag_33V_to_90B_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
3.9.2	Action	Map_Swift_Tag_33V_to_90B_Action	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
3.9.2	Action1	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	type=compute;opCode=move;target=OPort.Letter;Letter
3.9.2	Action2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	type=compute;opCode=move;target=OPort.Qualifier;Qualifier
3.9.2	Action3	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	type=compute;opCode=move;target=OPort.DataField.Swift_90_B.Currency
3.9.2	Action4	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	type=compute;opCode=move;target=OPort.DataField.Swift_90_B.Price; Port
3.9.2	Action5	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		255	type=compute;opCode=move;target=OPort.DataField.Swift_90_B.AmountTy
3.9.2	ObjectEve	String						
3.9.2	ObjectEventI	String						
3.9.2	ObjectEventId	String						
3.9.2	ObjectEventId	String						
3.10	Action10	Map_Swift_MT520_A_to_MT540_E1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate; Port.Swift_MT520Data.Swift_MT520_A;OPort.Swift_MT540
3.11	Action11	Map_Swift_MT520_B_to_MT540_C	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate; Port.Swift_MT520Data.Swift_MT520_B;OPort.Swift_MT540
3.12	Action12	Map_Swift_MT520_C_to_MT540_E1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate; Port.Swift_MT520Data.Swift_MT520_C;OPort.Swift_MT540
3.13	Action13	Map_Swift_MT520_C_to_MT540_B	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate; Port.Swift_MT520Data.Swift_MT520_C;OPort.Swift_MT540

Figure 29. Sub-action objects for a sequence object

Representing field objects

Just as SWIFT sequences can contain other sequences as well as field tags, sequence objects in PIMOs make reference to sequence and field objects. Field objects are child objects of sequence Action objects. This section describes how to create field objects that map to other field objects via parent sequence Action objects.

The previous section showed how to define sequence objects, including Map_Swift_20_to_20C. Figure 30 shows how to create the PIMO entries that correspond to the field and sub-field mapping for this sequence.

Note: The *SWIFT User Handbook* shows that no Code Words are required for the Map_Swift_20_to_20C mapping.

	Pos	Name	Type	Key	Card	Ma	Defa	App Spec Info
1	1	Port	Map_Swift_Tag_20_to_20C_Port	<input checked="" type="checkbox"/>	1			
1.1	1.1	IPort	Swift_Tag_20	<input checked="" type="checkbox"/>	1			
1.2	1.2	OPort	Swift_Tag_20	<input type="checkbox"/>	1			
1.3	1.3	ObjectEventId	String					
2	2	Declaration	Map_Swift_Tag_20_to_20C_Declaration	<input type="checkbox"/>	1			
2.1	2.1	Letter	String	<input checked="" type="checkbox"/>		255	C	final
2.2	2.2	Qualifier	String	<input type="checkbox"/>		255	SEME	final
2.3	2.3	ObjectEventId	String					
3	3	Action	Map_Swift_Tag_20_to_20C_Action	<input type="checkbox"/>	1			
3.1	3.1	Action1	String	<input checked="" type="checkbox"/>		255		type=compute;opCode=move;target=OPort.Letter;Letter
3.2	3.2	Action2	String	<input type="checkbox"/>		255		type=compute;opCode=move;target=OPort.Qualifier;Qualifier
3.3	3.3	Action3	String	<input type="checkbox"/>		255		type=compute;opCode=move;target=OPort.DataField.Swift_20_C.Reference;IPort.DataField.Swift_20
3.4	3.4	ObjectEventId	String					
4	4	ObjectEventId	String					
5	5			<input type="checkbox"/>		255		

Figure 30. Field action objects for a sequence

As shown in Figure 30, you must specify Port (IPort and OPort) objects, Declaration variables, and Actions for Field objects:

Port The type for IPort and OPort is the source and destination tag, stripped of tag letters. In this case, they are the same, Swift_Tag_20. IPort, by convention is Key (IsKey = true).

Declaration The tag letter variable is specified as Letter. In this example, the default value is C. The keyword final designates this variable as a constant, and, by convention, the first variable is Key (IsKey = true). Qual_<QUALIFIER> represents the qualifier. (The qualifier is always in uppercase.) In the example the default value is SEME.

Action The sub-Action objects are as follows:

Action1 type=compute;opCode=move;target=OPort.Letter;Letter The value of the Letter variable is moved to the Letter attribute of OPort (which is Swift_Tag_20)

Action2 type=compute;opCode=move;target=OPort.Qualifier;Qualifier The value of the Qualifier variable is moved to the Qualifier attribute of OPort (which is Swift_Tag_20).

Action3
type=compute;opCode=move;target=OPort.DataField.Swift_20_C.Reference;
IPort.DataField.Swift_20.TransactionReferenceNumber The value of
IPort.DataField.Swift_20.TransactionReferenceNumber is moved to
OPort.DataField.Swift_20_C.Reference.

Figure 31 shows a field map that involves code word mapping and a Delegate Action.

	Pos	Name	Type	Key	Foreign	Reqd	Card	App Spec Info
1	1	Port	Map_Swift_Tag_12_to_13A_Port	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
1.1	1.1	IPort	Swift_Tag_12	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
1.2	1.2	OPort	Swift_Tag_13	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
1.3	1.3	ObjectEventId	String					
2	2	Declaration	Map_Swift_Tag_12_to_13A_Declaration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
2.1	2.1	Letter	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		final
2.2	2.2	Qualifier	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		final
2.3	2.3	code_574	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		final
2.4	2.4	ObjectEventId	String					
3	3	Action	Map_Swift_Tag_12_to_13A_Action	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	
3.1	3.1	Action1	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=compute;opCode=move;target=OPort.Letter;Letter
3.2	3.2	Action2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		type=compute;opCode=move;target=OPort.Qualifier;Qualifier
3.3	3.3	Action3	Map_Swift_Tag_12_13A_Code_REQU	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1	type=delegate;IPort.DataField.Swift_12.MTNumber;OPort.DataField.Swift_13_A.NumberId
3.4	3.4	ObjectEventId	String					
4	4	ObjectEventId	String					
5	5			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

Figure 31. Field mapping with code words

For Action3, the Delegate Action, you must define the Port (IPort and OPort) Declaration, and sub-Action objects as shown in Figure 32.

	Pos	Name	Type	Key	Foreign	Reqd	Card	Default	App Spec Info
4	3	Action	Map_Swift_Tag_12_to_13A_Action	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
4.1	3.1	Action1	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			type=compute;opCode=move;target=OPort.Letter;Letter
4.2	3.2	Action2	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			type=compute;opCode=move;target=OPort.Qualifier;Qualifier
4.3	3.4	ObjectEventId	String						
4.4	3.3	Action3	Map_Swift_Tag_12_13A_Code_REQU	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=delegate;IPort.DataField.Swift_12.MTNumber;OPort.DataField.Swift_13_A.NumberId
4.4.1	3.3.4	ObjectEventId	String						
4.4.2	3.3.1	IPort	Map_Swift_Tag_12_to_13A_Code_REQU_P	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
4.4.3	3.3.1.1	IPort	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
4.4.4	3.3.1.1	OPort	String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>			
4.4.5	3.3.1.1	ObjectEventId	String						
4.4.6	3.3.2	Declaration	Map_Swift_Tag_12_to_13A_Code_REQU_D	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
4.4.7	3.3.2.1	Code_577	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		577	final
4.4.8	3.3.2.1	Code_572	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		572	final
4.4.9	3.3.2.1	Code_573	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		573	final
4.4.10	3.3.2.1	Code_574	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		574	final
4.4.11	3.3.2.1	Code_576	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		576	final
4.4.12	3.3.2.1	Code_571	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		571	final
4.4.13	3.3.2.1	Code_535	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		535	final
4.4.14	3.3.2.1	Code_536	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		536	final
4.4.15	3.3.2.1	Code_537	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		537	final
4.4.16	3.3.2.1	ObjectEventId	String						
4.4.17	3.3.3	Action	Map_Swift_Tag_12_to_13A_Code_REQU_A	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		
4.4.18	3.3.3.1	ObjectEventId	String						
4.4.19	3.3.3.1	Action6	Map_Swift_Tag_12_to_13A_Code_REQU_S	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=scenario
4.4.20	3.3.3.1.1	ObjectEventId	String						
4.4.21	3.3.3.1.1	Scenario	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			IPort==Code_577
4.4.22	3.3.3.1.1	TrueAction	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			type=compute;opCode=move;target=OPort;Code_577
4.4.23	3.3.3.1.1	Action5	Map_Swift_Tag_12_to_13A_Code_REQU_S	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=scenario
4.4.24	3.3.3.1.1	Action4	Map_Swift_Tag_12_to_13A_Code_REQU_S	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	1		type=scenario

Figure 32. Code word declaration for a field map

The code words are declared and assigned constant default values. A Scenario Action (Action6) specifies an equality boolean expression and a contingent Compute Action to execute if the expression evaluates to false.

Modifying PIMOs: Map summary

The file `Map_Objects.txt` contains PIMOs for maps as follows:

From ISO 7775 to ISO 15022

- MT520 to MT540 Receive Free
- MT521 to MT541 Receive Payment Against
- MT522 to MT542 Deliver Free
- MT523 to MT543 Deliver Against Payment
- MT530 to MT544 Receive Free Confirmation
- MT531 to MT 545 Receive Against Payment Confirmation
- MT532 to MT546 Deliver Free Confirmation
- MT533 to MT547 Deliver Against Payment Confirmation
- MT571 to MT535 Statement of Holdings
- MT573 to MT537 Statement of Pending Transactions

From ISO 15022 to ISO 7775

- MT536 to MT572 Statement of Transactions
- MT540 to MT520 Receive Free
- MT541 to MT521 Receive Payment Against
- MT542 to MT522 Deliver Free
- MT543 to MT523 Deliver Against Payment
- MT544 to MT530 Receive Free Confirmation
- MT545 to MT 531 Receive Against Payment Confirmation
- MT546 to MT532 Deliver Free Confirmation
- MT547 to MT533 Deliver Against Payment Confirmation
- MT548 to MT534 Settlement Status and Processing Advice

If you need to modify one or more of these PIMOs:

- Review “Production instruction meta-objects (PIMOs)” on page 81 and “Creating PIMOs” on page 88
- Find the PIMO you want to modify in the tables below, which summarize the default values and mapping relationships for each PIMO

Note: Each table below shows the mapping relationships and default values for the ISO 7775-to-15022 direction and for the reverse (ISO 15022-to-7775) direction.

- Launch Business Object Designer and open the PIMO you wish to modify, saving a backup copy of the unmodified original copy.

MT520 to MT540 receive free

Table 29. MT520 to MT540 receive free

Data element	MT520		MT540			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Delivery Date	A	30	B	98a	SETT	MT540 default values: 98A
TRN	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Date and Place of Trade	A	31P	B B	98a 94B	TRAD TRAD	MT540 default values: 98A
Identification of Securities	A	35B	B	35B		
Next Coupon	A	35a	B1 B1	98A 13a	COUP COUP	MT520->540: If the incoming tag is 35a, it is mapped to 98A; If the incoming tag is 35c, it is mapped to 13A. For MT540->520: User defines whether 98A is mapped to 35D or 13A to 35C.
Book Value	A	33V	B1	90a	MRKT	MT540 default values: 90B
Instructing Party	A	82a	E1 E1	95a 97a	SAFE	MT520 default values: 82D MT540 default values: 95Q DEAG, 97A (See Field Specs in SWIFT UHB)
Quantity of Securities	B	35A	C	36B	SETT	
Safekeeping Account	B	83a	C	97a	SAFE	MT520 default values: 83D MT540 default values: 97A
Certificate Numbers	B	35E	C	13B	CERT	MT520 default values: 35E->70E (F)
Deliverer of Securities	C	87a	E1 E1	95a 97a	DEAG SAFE	MT520 default values: 87D MT540 default values: 95Q, 97A
Beneficiary of Securities	C	88a	E1 E1	95a 97a	BUYR SAFE	MT520 default values: 88D MT540 default values: 95Q, 97A
Deliverer's Instructing Party	C	85a	E1 E1	95a 97a	SELL SAFE	MT520 default values: 85D MT540 default values: 95Q, 97A
Registration Details	C	77D	E1	70a	REGI	MT540 default values: 70D
Declaration Details	C	77R	E1	70a	DECL	MT540 default values: 70E

Table 29. MT520 to MT540 receive free (continued)

Data element	MT520		MT540			Comments
	Seq	Field tag	Seq	Field tag	Qualifier	
Sender to Receiver Information	C	72	B C	70E 13B	SPRO CERT	

Table 30. MT520 to MT540 code word mapping

MT520	MT540
Tag 72 (C)	Tag 13B (C)
MSG579	CERT
Tag 35A	Tag 36B
FMT	FAMT
BON	UNIT
CER	
CPN	
MSC	
OPC	
OPS	
PRC	
PRS	
RTE	
SHS	
UNT	
WTS	

MT521 to MT541 receive payment against

Table 31. MT521 to MT541 receive payment against

Data element	MT521		MT541			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Settlement Date	A	30	B	98a	SETT	MT541 default values: 98A
TRN	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Date and Place of Trade	A	31P	B B	98a 94B	TRAD TRAD	MT541 default values: 98A
Identification of Securities	A	35B	B	35B		
Next Coupon	A	35a	B1 B1	98A 13a	COUP COUP	MT521->541: If the incoming tag is 35a, it is mapped to 98A; If the incoming tag is 35c, it is mapped to 13A. For MT541->521: User defines whether 98A is mapped to 35D or 13A to 35C.

Table 31. MT521 to MT541 receive payment against (continued)

Data element	MT521		MT541			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Book Value	A	33V	B1	90a	MRKT	
Instructing Party	A	82a	E1 E1	95a 97a	SAFE	MT521 default values: 82D MT541 default values: 95Q REAG, 97A (See Field Specs in SWIFT UHB)
Quantity of Securities	B	35A	C	36B	SETT	
Safekeeping Account	B	83a	C	97a	SAFE	MT521 default values: 83D MT541 default values: 97A
Certificate Numbers	B	35E	C	13B	CERT	521.B.35E is mapped to 541.F.70E (with DECL as qualifier) instead of to 541.C.13B
Deliverer of Securities	C	87a	E1 E1	95a 97a	SAFE	MT521 default values: 87D MT541 default values: 95Q DEAG, 97A (See Field Specs in SWIFT UHB)
Beneficiary of Securities	C	88a	E1 E1	95a 97a	BUYR SAFE	MT521 default values: 88D MT541 default values: 95Q, 97A
Deliverer's Instructing Party	C	85a	E1 E1	95a 97a	SELL SAFE	MT521 default values: 85D MT541 default values: 95Q, 97A
Registration Details	C	77D	E1	70a	REGI	MT541 default values: 70D
Declaration Details	C	77R	E1	70a	DECL	MT541 default values: 70E
Account for Payment	C	53a	C	97a	CASH	MT521 default values: 53C MT541 default values: 97A
Account with Institution	C	57a	E2 E2	95a 97A	ACCW CASH	MT521 default values: 57D MT541 default values: 95Q

Table 31. MT521 to MT541 receive payment against (continued)

Data element	MT521		MT541			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Beneficiary of Money	C	58a	E2 E2	95a 97A	BENM CASH	MT521 default values: 58D MT541 default values: 95Q
Deal Price	C	33T	B	90a	DEAL	MT541 default values: 90B
Deal Amount	C	32M	E3	19A	DEAL	
Accrued Interest	C	34a	E3	19A	ACRU	MT521 default values: 34G
Taxes Added	C	71E	E3	19A		MT 541 default values: TRAX (See Field Specs in SWIFT UHB)
Broker's Commission	C	71F	E3	19A		MT 541 default values: LOCO (See Field Specs in SWIFT UHB)
Other Charges or Fees	C	71G	E3	19A		MT 541 default values: CHAR (See Field Specs in SWIFT UHB)
Settlement Amount	C	32B	C E3	19A 19A	SETT SETT	
Account(s) for Charges	C	71D	E2	97A 70E	CHAR DECL	
Sender to Receiver Information	C	72	B C	70E 13B	SPRO CERT	

Table 32. MT521 to MT541 code mapping

MT521	MT541
Old Tag: 72 (C)	New Tag: 13B (C)
Old Code	Qualifier
MSG579	CERT
Old Tag: 35A	New Tag: 36B
Old Code	Qualifier
FMT	FAMT

Table 32. MT521 to MT541 code mapping (continued)

MT521	MT541
BON	UNIT
CER	
CPN	
MSC	
OPC	
OPS	
PRC	
PRS	
RTE	
RTS	
SHS	
UNT	
WTS	

MT522 to MT542 deliver free

Table 33. MT522 to MT542 deliver free

Data element	MT522		MT542			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Delivery Date	A	30	B	98a	SETT	
Transaction Reference Number	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Date and Place of Trade	A	31P	B B	98a 94B	TRAD TRAD	MT542 default values: 98A
Identification of Securities	A	35B	B	35B	-	
Next Coupon	A	35a	B1 B1	98A 13a	COUP COUP	MT522->542: If the incoming tag is 35a, it is mapped to 98A; If the incoming tag is 35c, it is mapped to 13A. For MT542->522: User defines whether 98A is mapped to 35D or 13A to 35C.
Book Value	A	33V	B1	90a	MRKT	
Instructing Party	A	82a	E1 E1	95a 97a	SAFE	MT522 default values: 82D MT542 default values: 95Q DEAG, 97A SAFE (See Field Specs in SWIFT UHB)
Quantity of Securities	B	35A	C	36B	SETT	

Table 33. MT522 to MT542 deliver free (continued)

Data element	MT522		MT542			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Safekeeping Account	B	83a	C	97a	SAFE	MT522 default values: 83D Default values for MT542: 97A
Certificate Numbers	B	35E	C	13B	CERT	
Receiver of Securities	C	87a	E1 E1	95a 97a	REAG SAFE	MT522 default values: 87D MT542 default values: 95Q, 97A
Beneficiary of Securities	C	88a	E1 E1	95a 97a	BUYR SAFE	MT522 default values: 88D MT542 default values: 95Q, 97A
Registration Details	C	77D	E1	70a	REGI	MT542 default values: 70D
Declaration Details	C	77R	E1	70a	DECL	MT542 default values: 70E
Sender to Receiver Information	C	72	B C	70E 13B	SPRO CERT	13B CERT: not implemented

Table 34. MT 522 to MT 542 code mapping

MT522	MT542
Old Tag: 72 (C)	New Tag: 13B (C)
MSG579	CERT
Old Tag: 35A	New Tag: 36B
FMT	FAMT
BON	UNIT
CER	
CPN	
MSC	
OPC	
OPS	
PRC	
PRS	
RTE	
RTS	
SHS	
UNT	
WTS	

MT523 to MT543 deliver against payment

Table 35. MT523 to MT543 deliver against payment

Data element	MT523		MT543			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Settlement Date	A	30	B	98a	SETT	
Transaction Reference Number	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Date and Place of Trade	A	31P	B B	98a 94B	TRAD TRAD	MT543 default values: 98A
Identification of Securities	A	35B	B	35B	-	
Next Coupon	A	35a	B1 B1	98A 13a	COUP COUP	MT523->543: If the incoming tag is 35a, it is mapped to 98A; If the incoming tag is 35c, it is mapped to 13A. For MT543->523: User defines whether 98A is mapped to 35D or 13A to 35C.
Book Value	A	33V	B1	90a	MRKT	MT543 default values: 90B
Instructing Party	A	82a	E1 E1	95a 97a	SAFE	MT523 default values: 82D MT543 default values: 95Q DEAG, 97A (See Field Specs in SWIFT UHB)
Quantity of Securities	B	35A	C	36B	SETT	
Safekeeping Account	B	83a	C	97a	SAFE	MT523 default values: 83D MT543 default values: 97A
Certificate Numbers	B	35E	C	13B	CERT	
Receiver of Securities	C	87a	E1 E1	95a 97a	REAG SAFE	MT523 default values: 95Q MT543 default values: 97A
Beneficiary of Securities	C	88a	E1 E1	95a 97a	BUYR SAFE	MT523 default values: 88D; MT543 default values: 95Q, 97A
Registration Details	C	77D	E1	70a	REGI	MT543 default values: 70D
Declaration Details	C	77R	E1	70a	DECL	MT543 default values: 70E

Table 35. MT523 to MT543 deliver against payment (continued)

Data element	MT523		MT543			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Account for Payment	C	53a	C	97a	CASH	MT523 default values: 53D MT543 default values: 97A
Account With Institution	C	57a	E2 E2	95a 97a	ACCW CASH	MT523 default values: 57D MT543 default values: 95Q, 97R
Beneficiary of Money	C	58a	E2 E2	95a 97a	BENM CASH	MT523 default values: 58D MT543 default values: 95Q, 97A
Deal Price	C	33T	B	90a	DEAL	MT543 default values: 90B
Deal Amount	C	32M	E3	19A	DEAL	
Accrued Interest	C	34a	E3	19A	ACRU	
Taxes Deducted	C	71E	E3	19A		MT543 default values: TRAX (See Field Specs in SWIFT UHB)
Broker's Commission	C	71F	E3	19A		MT543 default values: LOCO (See Field Specs in SWIFT UHB)
Other Charges or Fees	C	71G	E3	19A		MT543 default values: CHAR (See Field Specs in SWIFT UHB)
Settlement Amount	C	32B	C E3	19A 19A	SETT SETT	
Account(s) For Changes	C	71D	E2	70E	DECL	
Sender to Receiver Information	C	72	B C	70E 13B	SPRO CERT	

Table 36. MT523 to MT543 code word mapping

MT523	MT543
Old Tag: 72 (C)	New Tag: 13B (C)
MSG579	CERT
Old Tag: 35A	New Tag: 36B
FMT	FAMT

Table 36. MT523 to MT543 code word mapping (continued)

MT523	MT543
BON	UNIT
CER	
CPN	
MSC	
OPC	
OPS	
PRC	
PRS	
RTE	
SHS	
UNT	
WTS	

MT530 to MT544 receive free confirmation

Table 37. MT530 to MT544 receive free confirmation

Data element	MT530		MT544			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
TRN	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Delivery Date	A	30	B	98a	ESET	
Identification of Securities	A	35B	B	35B		
Next Coupon	A	35a 35C	B1 B1	98A 13a	COUP COUP	MT530->544: If the incoming tag is 35a, it is mapped to 98A; If the incoming tag is 35c, it is mapped to 13A. For MT544->530: User defines whether 98A is mapped to 35D or 13A to 35C.
Book Value	A	33V	B1	90a	MRKT	SWIFT documentation erroneously lists 35V as MT530 tag.
Instructing Party	A	82a	E1 E1	95a 97a	SAFE	MT530 default values: 82D MT544 default values: 95Q, 97A (See Field Specs in SWIFT UHB)
Quantity of Securities	B	35A	C	36B	ESTT	

Table 37. MT530 to MT544 receive free confirmation (continued)

Data element	MT530		MT544			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Safekeeping Account	B	83a	C	97a	SAFE	MT530 default values: 83D MT544 default values: 97A
Certificate Numbers	B	35E	C	13B	CERT	
Deliverer of Securities	C	87a	E1 E1	95a 97a	DEAG SAFE	MT530 default values: 87D MT544 default values: 95Q 97A
Beneficiary of Securities	C	88a	E1 E1	95a 97a	BUYR SAFE	MT530 default values: 88D MT544 default values: 95Q, 97A
Deliverer's Instructing Party	C	85a	E1 E1	95a 97a	SELL SAFE	MT530 default values: 85D MT544 default values: 95Q, 97A
Registration Details	C	77D	E1	70a	REGI	MT544 default values: 70D
Declaration Details	C	77R	E1	70a	DECL	MT544 default values: 70E
Other Charges	C	71C	E3	19A		MT544 default values: CHAR (See Field Specs in SWIFT UHB)
Own Charges	C	71B	E3	19A		MT544 default values: CHAR (See Field Specs in SWIFT UHB)
Sender to Receiver Information	C	72	B C1	70E 13B	SPRO CERT	

Table 38. MT530 to MT544 code word mapping

MT530	MT544
Old Tag: 72 (C)	New Tag: 23G (A)
REVERSAL	RVSL
Old Tag: 72 (C)	New Tag: 22a (A) Qualifier PARS
PARTIAL	PAIN
COMPLETE	PARC
Old Tag: 72 (C)	New Tag: 13B (C)
MSG579	CERT
REGOPEN	

MT531 to MT 545 receive against payment confirmation

Table 39. MT531 to MT 545 confirmation of receive against payment

Data element	MT531		MT545			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
TRN	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Settlement Date	A	30	B	98a	ESET	MT545 default values: 98A
Identification of Securities	A	35B	B	35B		
Date and Place of Trade	A	31P	B B	98a 94B	TRAD TRAD	
Next Coupon	A	35a 35C	B1 B1	98A 13a	COUP COUP	MT531->545: If the incoming tag is 35a, it is mapped to 98A; If the incoming tag is 35c, it is mapped to 13A. For MT545->531: User defines whether 98A is mapped to 35D or 13A to 35C.
Book Value	A	33V	B1	90a	MRKT	SWIFT documentation erroneously lists 35V as MT531 tag.
Instructing Party	A	82a	E1 E1	95a 97a	REAG SAFE	MT531 default values: 82D MT545 default values: 95Q, 97A
Quantity of Securities	B	35A	C	36B	ESTT	
Safekeeping Account	B	83a	C	97a	SAFE	
Certificate Numbers	B	35E	C	13B	CERT	
Deliverer of Securities	C	87a	E1 E1	95a 97a	DEAG SAFE	MT531 default values: 87D MT545 default values: 95Q, 97A
Beneficiary of Securities	C	88a	E1 E1	95a 97a	BUYR SAFE	MT531 default values: 88D MT545 default values: 95Q, 97A
Deliverer's Instructing Party	C	85a	E1 E1	95a 97a	SELL SAFE	MT531 default values: 85D MT545 default values: 95Q, 97A
Registration Details	C	77D	E1	70a	REGI	MT545 default values: 70D
Declaration Details	C	77R	E1	70a	DECL	MT545 default values: 70E

Table 39. MT531 to MT 545 confirmation of receive against payment (continued)

Data element	MT531		MT545			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Account for Payment	C	53a	C	97a	CASH	MT531 default values: 53C MT545 default values: 97A
Account with Institution	C	57a	E2 E2	95a 97A	ACCW CASH	MT531 default values: 57D MT545 default values: 95Q
Beneficiary of Money	C	58a	E2 E2	95a 97A	BENM CASH	MT531 default values: 58D MT545 default values: 95Q
Special Concessions	C	33S	E3	19A	SPCN	
Deal Price	C	33T	B	90a	DEAL	MT545 default values: 90B
Deal Amount	C	32M	E3	19A	DEAL	
Accrued Interest	C	34a	E3	19A	ACRU	MT531 default values: 34G
Settlement Amount	C	32B	C E3	19A 19A	ESTT ESTT	
Other Charges	C	71C	E3	19A	CHAR	
Own Charges	C	71B	E3	19A	CHAR	
Exchange Rate	C	36	E3	92B	EXCH	
Net Proceeds	C	34A	E3	19A	POST	
Sender to Receiver Information	C	72	B C1	70E 13B	SPRO CERT	

Mapping of code words (MT 531 ➔ MT 545):

Table 40. MT531 to MT545 code word mapping

MT531	MT545
Old Tag: 72 (C)	New Tag: 23G (A)
REVERSAL	RVSL
Old Tag: 72 (C)	New Tag: 22a (A) Qualifier PARS
PARTIAL	PAIN
COMPLETE	PARC
Old Tag: 72 (C)	New Tag: 13B (C)
MSG579	CERT -
REGOPEN	

MT532 to MT 546 deliver free confirmation

Table 41. MT532 to MT 546 deliver free confirmation

Data element	MT532		MT546			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Transaction Reference Number	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Delivery Date	A	30	B	98a	ESET	MT546 default values: 98A
Identification of Securities	A	35B	B	35B	-	
Next Coupon	A	35a	B1 B1	98A 13a	COUP COUP	MT532->546: If the incoming tag is 35a, it is mapped to 98A; If the incoming tag is 35c, it is mapped to 13A. For MT546->532: User defines whether 98A is mapped to 35D or 13A to 35C.
Book Value	A	33V	B1	90a	MRKT	
Instructing Party	A	82a	E1 E1	95a 97a	SAFE	MT532 default values: 82D MT546 default values: 95Q DEAG, 97A SAFE (See Field Specs in SWIFT UHB)
Quantity of Securities	B	35A	C	36B	ESTT	
Safekeeping Account	B	83a	C	97a	SAFE	MT532 default values: 83D MT546 default values: 97A
Certificate Numbers	B	35E	C	13B	CERT	
Receiver of Securities	C	87a	E1 E1	95a 97a	REAG SAFE	MT532 default values: 87D MT546 default values: 95Q, 97A
Beneficiary of Securities	C	88a	E1 E1	95a 97a	BUYR SAFE	MT532 default values: 88D MT546 default values: 95Q, 97A
Registration Details	C	77D	E1	70a	REGI	MT546 default values: 70D
Declaration Details	C	77R	E1	70a	DECL	MT546 default values: 70E

Table 41. MT532 to MT 546 deliver free confirmation (continued)

Data element	MT532		MT546			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Other Charges	C	71C	E3	19A		MT546 default values: CHAR (See Field Specs in SWIFT UHB)
Own Charges	C	71B	E3	19A		MT546 default values: CHAR (See Field Specs in SWIFT UHB)
Sender to Receiver Information	C	72	B C	70E 13B	SPRO CERT	

Table 42. MT532 to MT546 code word mapping

MT 532	MT546
Old Tag: 72 (C)	New Tag: 13B (C)
MSG579	CERT
Old Tag: 35A	New Tag: 36B
FMT	FAMT
BON	UNIT
CER	
CPN	
MSC	
OPC	
OPS	
PRC	
PRS	
RTE	
RTS	
SHS	
UNT	
WTS	

MT533 to MT547 deliver against payment confirmation

Table 43. MT533 to MT547 deliver against payment confirmation

Data element	MT533		MT547			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Transaction Reference Number	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Settlement Date	A	30	B	98a	ESET	MT547 default values: 98A

Table 43. MT533 to MT547 deliver against payment confirmation (continued)

Data element	MT533		MT547			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Date and Place of Trade	A	31P	B B	98a 94B	TRAD TRAD	
Identification of Securities	A	35B	B	35B	-	
Next Coupon	A	35a 35C	B1 B1	98A 13a	COUP COUP	MT533->547: If the incoming tag is 35a, it is mapped to 98A; If the incoming tag is 35c, it is mapped to 13A. For MT547->533: User defines whether 98A is mapped to 35D or 13A to 35C.
Book Value	A	33V	B1	90a	MRKT	
Instructing Party	A	82a	E1 E1	95a 97a		MT533 default values: 82D MT547 default values: 95Q DEAG, 97A SAFE (See Field Specs in SWIFT UHB)
Quantity of Securities	B	35A	C	36B	ESTT	
Safekeeping Account	B	83a	C	97a	SAFE	MT533 default values: 83D MT547 default values: 97A
Certificate Numbers	B	35E	C	13B	CERT	Note the Code level mapping
Receiver of Securities	C	87a	E1 E1	95a 97a	REAG SAFE	MT533 default values: 87D MT547 default values: 95Q, 97A
Beneficiary of Securities	C	88a	E1 E1	95a 97a	BUYR SAFE	MT533 default values: 88D MT547 default values: 95Q, 97A
Registration Details	C	77D	E1	70a	REGI	
Declaration Details	C	77R	E1	70a	DECL	
Account for Payment	C	53a	C	97a	CASH	MT533 default values: 53D MT547 default values: 97A

Table 43. MT533 to MT547 deliver against payment confirmation (continued)

Data element	MT533		MT547			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Account With Institution	C	57a	E2 E2	95a 97a	ACCW CASH	MT533 default values: 57D MT547 default values: 95Q, 97A
Beneficiary of Money	C	58a	E2 E2	95a 97a	BENM CASH	MT533 default values: 58D MT547 default values: 95Q, 97A
Special Concessions	C	33S	E3	19A	SPCN	
Deal Price	C	33T	B	90a	DEAL	MT547 default values: 90B
Deal Amount	C	32M	E3	19A	DEAL	
Accrued Interest	C	34a	E3	19A	ACRU	MT533 default values: 34G
Settlement Amount	C	32B	C E3	19A 19A	ESTT ESTT	
Other Charge(s)	C	71C	E3	19A		MT547 default values: CHAR (See Field Specs in SWIFT UHB)
Own Charge(s)	C	71B	E3	19A		MT547 default values: CHAR (See Field Specs in SWIFT UHB)
Exchange Rate	C	36	E3	92B	EXCH	
Net Proceeds	C	34A	E	19A	POST	
Sender to Receiver Information	C	72	B C	70E 13B	SPRO CERT	

Table 44. MT533 to MT547 code word mapping

MT533	MT547
Old Tag: 72 (C)	New Tag: 13B (C)
MSG579	CERT
Old Tag: 35A	New Tag: 36B
FMT	FAMT

Table 44. MT533 to MT547 code word mapping (continued)

MT533	MT547
BON	UNIT
CER	
CPN	
MSC	
OPC	
OPS	
PRC	
PRS	
RTE	
RTS	
SHS	
UNT	
WTS	

MT534 to MT548 settlement status and processing advice

Table 45. MT534 to MT548 settlement status and processing advice

Data element	MT534		MT548			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Transaction Reference Number	----	20	A	20C	SEME	
Related Reference	----	21	A1	20C	RELA	
MT and Date of Original Instruction	----	11a	A1	13A	LINK	Date of original instruction can not be specified anymore.
Settlement Date	----	30	B	98a	SETT	
Date Problem Occurred	----	31S	----	----		
Further Identification	----	23	A2	25D		MT548 default values: SETT (See Field Specs in SWIFT UHB)
			A2a	24B		MT548 default values: PEND (See Field Specs in SWIFT UHB)
			B	22H	REDE	
Safekeeping Account	----	83a	B	97a	SAFE	MT534 default values: 83D MT548 default values: 97A
Quantity of Securities	----	35A	B	36B	SETT	
Identification of Securities	----	35B	B	35B		

Table 45. MT534 to MT548 settlement status and processing advice (continued)

Data element	MT534		MT548			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Receiver/Deliverer of Securities	----	87a	B1	95a	REAG or DEAG	MT534 default values: 87D MT548 default values: 96A
			B1	97a	SAFE	MT548 default values: 97A
Receiver/Deliverer's Instructing Party	----	82a	B1	95a		MT534 default values: 8D MT548 default values: 95A (See Field Specs in SWIFT UHB)
			B1	97a	SAFE	MT548 default values: 97A
Narrative	----	79	B	70E	SPRO	
Sender to Receiver Information	----	72	B	70E	SPRO	

Table 46. MT534 to MT548 code word mapping

MT534	MT548
Old Tag: 23	New Tag: 24B (A2a) Qualifier: PEND
COLLATER	COLL
CPFUTURE	CFUT
CPLACK	CLAC
CPMONEY	CMON
FUTURE	FUTU
LACK	LACK
MONEY	MONY
REGOPEN	REGO
MONSE	LACK and MONY
NODEL	NDEL
REFUS	REFS
INCAD	INCA

MT571 to MT535 statement of holdings

Table 47. MT571 to MT535 statement of holdings

Data element	MT571		MT535			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Page Number/Continuation Indicator	A	28	A	28E		

Table 47. MT571 to MT535 statement of holdings (continued)

Data element	MT571		MT535			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Transaction Reference Number	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Safekeeping Account	A	83a	A	97a	SAFE	MT571 default values: 83D MT535 default values: 97A
Statement Period	A	67A	A	98a	STAT	MT535 default values: 98A
Date Prepared	A	30	A	98a	PREP	MT535 default values: 98C
Statement Basis	A	26F	A	22F	STBA	
Quantity of Securities	B	35H	B	93B	AGGR	
Quantity of Securities	B1	60B	B1	93C		MT535 default values: B0RR (See Field Specs in SWIFT UHB)
Further Identification	B1	23	B1 B1	93C 70C	B0RR SUBB	The qualifier of the balance field will specify the status or characteristic Additional information can be given in the narrative
Sender to Receiver Information	B1	72	B1a	70C	SUBB	
Identification of Securities	B	35B	B	35B		
Price per Unit	B	33B	B	90a		MT535 default values: 90B MRKT (See Field Specs in SWIFT UHB)
Accrued Interest	B	34a	B B	99A 19A	DAAC ACRU	
Exchange Rate	B	36	B	92B	EXCH	
Value	B	32H		19A	HOLD	
Sender to Receiver Information	B	72	B	70E	HOLD	
Number of Repetitive Parts	C	18A	----	----		
Final Value	C	34E	C	19A	HOLP	
Sender to Receiver Information	B	72	B1 B1 B1	90a 98a 94B	PRIC PRIC	MT535 default values: 90A MRKT, 98A (See Field Specs in SWIFT UHB)

Table 48. MT571 to MT535 code word mapping

Old tag: 26F (A)	New tag: 22F (A) qualifier: STBA
ACTUA	SETT
TRADE	TRAD
CONTR	----
BOOKD	----
Old Tag: 23 (B1)	New Tag: 93C (B1) All Qualifiers
NA	NAVL
AD	AVAL
Old Tag: 23 (B1)	New Tag: 93C (B1)
REGIS	REGO
DEPRJ	----
REGRJ	----
DENOM	REGO
PLEDG	COLI or COLO
MARGE	COLI or COLO
COLLA	COLI or COLO
LOAND	LOAN
BORRO	BORR
TRNSH	TRAN
REINV	PECA
DIVID	PECA
SPLIT	PECA
TENDE	PECA
REDEM	PECA
EXCHS	PECA
AVAIL	TAVI
MERGE	PECA
LIQUI	PECA
BANKR	PECA
PENDD	PEND
PENDR	PENR
SEE72	----
Old Tag: 72 (C)	New Tag: 90a (B) All Qualifiers
DISCOUNT	DISC
PREMIUM	PREM
Old Tag: 72 (C)	New Tag: 98a (B)
VALUDATE (date)	PRIC
Old Tag: 72 (C)	New Tag: 94B (B)
VALUDATE (source)	PRIC

MT572 to MT536 statement of transactions

Table 49. MT572 to MT536 statement of transactions

Data element	MT572		MT536			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Page Number/ Continuation Indicator	A	28	A	28E		
Transaction Reference Number	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA	
Safekeeping Account	A	83a	A	97a	SAFE	MT572 default values: 83D MT536 default values: 97A
Statement Period	A	67A	A	69a	STAT	MT536 default values: 69A
Date Prepared	A	30	A	98a	PREP	MT536 default values: 98A
Identification of Securities	B	35B	B1	35B		
Opening Balance	B	60A	B1	93B	FIOP	
Safekeeping Account	B	83a	----	----		The functionality of the new message does not allow you to specify more than one safekeeping account per message (which is specified in sequence A)
Opening Balance	B	60B	----	----		
Quantity of Securities	B	35A	B1a2	36B	PSTA	
Transaction Details	B1a	66A	B1a2 B1a2 B1a2 B1a2 B1a2	98a 22H 22F 22F 25D	ESET REDE TRAN CAEV and SETR MOVE	MT536 default values: 98A High level transaction type Detailed transaction type
Transaction Details	B1a	66A	B1a1 B1a1	20C 20C	RELA PREV	Account Owner's Reference Sender's Reference
Counterparty	B1a	87a	B1a2a B1a2a	95a 97a	DEAG and REAG SAFE	MT572 default values: 87D MT536 default values: 97A
Settlement Date	B1a	30	B1a	98a	SETT	MT536 default values: 98A
Price Per Unit	B1a	33B	B1	90a	MRKT	MT536 default values: 90B
Accrued Interest	B	34a	B1a2 B1b2	99A 19A	DAAC ACRU	MT572 default values: 34G
Amount	B	34a	B1b	19A	PSTA	MT572 default values: 34G

Table 49. MT572 to MT536 statement of transactions (continued)

Data element	MT572		MT536			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Sender to Receiver Information	B1a	72	B1a2 B1a2 B1 B1	70E 22a 98a 94B	TRDE TRAN PRIC PRIC	
Closing Balance	B	62B	----	----		
Closing Balance	B	62A	B1	93B	FICL	
Number of Repetitive Parts	C	18A	----	----		
Sender to Receiver Information	C	72	A B1a2 B1 B1	17B 22a 98a 94B	ACTI TRAN PRIC PRIC	

Table 50. MT572 to MT536 code word mapping

MT572	MT536
Old tag: 66A (B)	New tag: 22H (B1b) qualifier: REDE
N	RECE
T	DELI
Old tag: 66A (B)	New tag: 22F (B1b) qualifier: SETR
10 (correction)	----
11 (receipt for deposit)	TRAD (trade)
12 (regular trade)	TRAD (trade)
13 (forward trade)	----
14 (new issue)	----
16 (delivery)	TRAD (trade)
27 (depository transfer)	OWNE (external own account transfer)
28 (deposit transfer)	OWNI (internal own account transfer)
29 (opening buy)	----
30 (opening sell)	----
31 (closing buy)	----
32 (closing sell)	----
34 (assigned)	----
Old tag: 66A (B)	New tag: 22F (B1b1) qualifier: CAEV
17 (bonus securities)	BONU (bonus issue)
18 (stock dividends)	DVSE (stock dividend)
19 (split)	SPLF (stock split)
20 (reverse split)	SPLR (reverse stock split)
21 (exchange)	EXOF (exchange offer) MRGR (merger)
22 (conversion)	CONV (conversion)
	EXWA (warrant exercise)
23 (redemption)	BPUT (put redemption)
24 (new issue for conversion of maturing debentures)	REDM (redemption)

Table 50. MT572 to MT536 code word mapping (continued)

MT572	MT536
25 (drawing by lot)	DRAW (drawing)
26 (modification of identification of security)	----
33 (exercise)	EXWA (warrant exercise)
Old tag: 66A (B)	New tag: 22F (B1b) qualifier: MOVE
35 (reversal)	REVE
Old tag: 66A (B)	New tag: 22F (B1b) all qualifiers
72 (see field 72)	All other codes
Old tag: 72 (B,C)	New tag: 22F (B1b) qualifier: TRAN
LOANDOUT	BOLE
BORROWED	BOLE
COLLATER	COLL
Old tag: 72 (B,C)	New tag: 98a (B)
VALUDATE (date)	PRIC
Old tag: 72 (B)	New tag: 94B (B)
VALUDATE (source)	PRIC
Old tag: 72 (C)	New tag: 17B (A) qualifier: ACTI
NOTRANS	N

MT573 to MT537 statement of pending transactions

Table 51. MT573 to MT537 statement of pending transactions

Data element	MT573		MT537			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Page Number/ Continuation Indicator	A	28	A	28E		
Transaction Reference Number	A	20	A	20C	SEME	
Related Reference	A	21	A1	20C	RELA PREV	
Safekeeping Account	A	83a	A	97a	SAFE	MT573 default values: 83A MT537 default values: 97A
Statement Period	A	67A	A	98a	STAT	MT537 default values: 98A
Date Prepared	A	30	A	98a	PREP	MT537 default values: 98C
Identification of Securities	B,C	35B	B2b	35B		

Table 51. MT573 to MT537 statement of pending transactions (continued)

Data element	MT573		MT537			
	Seq	Field tag	Seq	Field tag	Qualifier	Comments
Safekeeping Account	B,C	83a	----	----		The functionality of the new message does not allow you to specify more than one safekeeping account per message (which is specified in sequence A)
Transaction Reference Number	B,C	20	B2a	20C	RELA	
Further Identification	B,C	23	B B1	25D 24B	SETT	MT537 default values: 24B PEND (See Field Specs in SWIFT UHB)
MT and Date of the Original Instruction	B,C	11a	B1	13A	LINK	MT573 default values: 11R
Related Reference	B,C	21	B2a	20C	RELA	
Quantity of Securities	B,C	35A	B2b	36B	PSTA	
Settlement Date	B,C	30	B2b	98a	SETT	
Settlement Amount	B,C	32B	B2b	19A	PSTA	
Counterparty	B,C	87a	B2b1 B2b1	95a 97a	REAG or DEAG SAFE	MT573 default values: 87D MT537 default values: 95Q, 97A
Sender to Receiver Information	B,C	72	B2b	70E	TRDE	
Sender to Receiver Information	D	72	A	17B	ACTI	

Table 52. MT573 to MT537 code word mapping

MT573	MT537
Old tag: 23 (B,C)	New tag: 24B (B1) qualifier: PEND
COLLATER	COLL
CPFUTURE	CFUT
CPLACK	CLAC
CPMONEY	CMON
FUTURE	FUTU
INCAD	INCA
LACK	LACK
MONEY	MONY
MONSE	LACK and MONY
NODEL	NDEL
REFUS	REFS

Table 52. MT573 to MT537 code word mapping (continued)

MT573	MT537
REGOPEN	REGO ----
SEE72	
Old tag: 23 (B,C)	
CERTD	New tag: 24B (B1) qualifier: PENF
COLLATER	REGO
CPLACK	COLL
CPMONEY	CLAC
FAIL	CMON
INCAD	---- (see other codes)
LACK	INCA
MONEY	LACK
REGOPEN	MONY
MONSE	REGO
NODEL	LACK and MONY
REFUS	NDEL
Old tag: 23 (B,C)	New message type: MT 548 new tag: 24B (A2a) qualifier: REJT
LATEI	LATE
CPLAT	LATE
MDATE	MDAT
Old tag: 23 (B,C)	New tag: 24B (B1), qualifier: NMAT
CUNMATCH	NMAT
UNMATCH	CMIS
DSECU	DSEC
DDATE	DDAT
DTRAN	DTRA
DMONE	DMON
DQUAN	DQUA
Old tag: 72 (D)	New tag: 17B (A)
Old Code	Qualifier: ACTI
NOPENDGS	N

Chapter 5. SWIFT Data Handler

- “Configuring the SWIFT Data Handler”
- “Configuring the Data Handler Child Meta-Object”
- “Converting Business Objects to SWIFT Messages” on page 124
- “Converting SWIFT Messages to Business Objects” on page 125

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 173.

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 53 describes the attributes in the child meta-object, `MO_DataHandler_SWIFT`.

Table 53. 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 53 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 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 43.

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 attr1 value....attrn value}where attrn value is the attribute value of the nth 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 `printB0(childB0)`. Handle multiple cardinality if required.
 - If `appText != NULL`, call `printFieldObj()`, which handles multiple cardinality and calls `printFieldB0()` 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 D, “SWIFT message structure”, on page 173.

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 6. 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/JMSEException...	Connector cannot find file <code>jms.jar</code> from the IBM WebSphere MQ Java client libraries. Ensure that variable <code>MQSERIES_JAVA_LIB</code> in <code>start_connector.bat</code> 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 <code>com.ibm.mqjms.jar</code> in the IBM WebSphere MQ Java client libraries. Ensure that variable <code>MQSERIES_JAVA_LIB</code> in <code>start_connector.bat</code> 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 <code>jndi.jar</code> from the IBM WebSphere MQ Java client libraries. Ensure that variable <code>MQSERIES_JAVA_LIB</code> in <code>start_connector.bat</code> points to the IBM WebSphere MQ Java client library folder.
The connector shuts down unexpectedly during initialization and the following exception is reported: java.lang.UnsatisfiedLinkError: no mqjbnd01 in shared library path	Connector cannot find a required runtime library (<code>mqjbnd01.dll</code> [NT] or <code>libmqjbnd01.so</code> [Solaris]) from the IBM WebSphere MQ Java client libraries. Ensure that your path includes the IBM WebSphere MQ Java client library folder.
The connector reports MQJMS2005: failed to create MQQueueManager for ':'	Explicitly set values for the following properties: <code>HostName</code> , <code>Channel</code> , and <code>Port</code> .

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 <code>?targetClient=1</code> to the name of output queue URI. For example, if you output messages to queue <code>queue://my.queue.manager/OUT</code> , change the URI to <code>queue://my.queue.manager/OUT?targetClient=1</code> . See Chapter 2, "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

- “Configuring standard connector properties for WebSphere InterChange Server” on page 130
- “Configuring standard connector properties for WebSphere MQ Integrator” on page 142

Connectors have two types of configuration properties:

- Standard configuration properties
- Connector-specific configuration properties

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

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

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

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

New and deleted properties

The following are the standard properties that have been either added or deleted in the 2.2 release of the adapters.

- **New properties**

CharacterEncoding
Local
JVMMinHeapSize
JVMMaxHeapSize
JVMMaxStackSize
WireFormat
MaxEventCapacity
DuplicateEventElimination
jms.NumConcurrentRequests
ContainerManagedEvents
jms.Messagebrokername (replaces jms.BrokerName)

- **Deleted properties**

RequestTransport
PingFrequency
TraceLevel
AgentProxyType

MaxThreadPoolSize
Anonymous Connections
GW Name
Agent URL
Listener Port
Certificate Location
LogFileName
TraceFileName
jms.BrokerName

Configuring standard connector properties for WebSphere InterChange Server

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

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

For general information about how connectors work with InterChange Server, see the *Technical Introduction to IBM WebSphere InterChange Server*.

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

You configure connector properties from Connector Configurator, which you access from System Manager.

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

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

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

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

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

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

Property Name	Possible values	Default value	Update method	Notes
AdminInQueue	<i>valid JMS queue name</i>	<i>CONNECTORNAME /ADMININQUEUE</i>		
AdminOutQueue	<i>valid JMS queue name</i>	<i>CONNECTORNAME/ADMINOUTQUEUE</i>		
AgentConnections	1-4	1	server restart	multi-threaded connector only
AgentTraceLevel	0-5	0	dynamic	
ApplicationName	<i>application name</i>	<i>the value that is specified for the connector name</i>	component restart	value required
BrokerType	ICS, WMQI			ICS is required if your broker is ICS
CharacterEncoding	ascii7, ascii8, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437	ascii7	component restart	
ConcurrentEventTriggeredFlows	1 to 32,767	no value	component restart	
ContainerManagedEvents	JMS or no value	JMS		guaranteed event delivery
ControllerStoreAndForwardMode	true or false	true	dynamic	
ControllerTraceLevel	0-5	0	dynamic	
DeliveryQueue		<i>CONNECTORNAME/DELIVERYQUEUE</i>	component restart	JMS transport only
DeliveryTransport	MQ, IDL, or JMS	IDL	component restart	
FaultQueue		<i>CONNECTORNAME/FAULTQUEUE</i>	component restart	
DuplicateEventElimination	True/False	False	component restart	JMS transport only, Container Managed Events must be <NONE>
JvmMaxHeapSize	heap size in megabytes	128m	component restart	
JvmMaxNativeHeapSize	size of stack in kilobytes	128k	component restart	
JvmMinHeapSize	heap size in megabytes	1m	component restart	
jms.MessageBrokerName	If FactoryClassName is IBM, use crossworlds.queue.manager. If FactoryClassName is Sonic, use localhost:2506.	crossworlds.queue.manager	server restart	JMS transport only

Property Name	Possible values	Default value	Update method	Notes
jms.FactoryClassName	CxCommon.Messaging. jms.IBMMQSeriesFactory or CxCommon.Messaging. jms.SonicMQFactory or any Java class name	CxCommon.Messaging. jms.IBMMQSeriesFactory	server restart	JMS transport only
jms.NumConcurrentRequests	positive integer	10	component restart	JMS transport only
jms.Password	Any valid password		server restart	JMS transport only
jms.UserName	Any valid name		server restart	JMS transport only
Locale	en_US , ja_JP, ko_KR, zh_C, zh_T, fr_F, de_D, it_I, es_E, pt_BR Note: These are only a subset of supported locales.	en_US	component restart	
LogAtInterchangeEnd	true or false	false	component restart	
MaxEventCapacity	1-2147483647	2147483647	dynamic	Repository Directory value must be <REMOTE>
MessageFileName	<i>path/filename</i>	<i>Connectorname.txt</i> or <i>InterchangeSystem.txt</i>	component restart	
MonitorQueue	any valid queue name	CONNECTORNAME/MONITORQUEUE	component restart	JMS transport only, Duplicate Event Elimination must be True
OADAutoRestartAgent	true or false	false	dynamic	
OADMaxNumRetry	<i>a positive number</i>	1000	dynamic	
OADRetryTimeInterval	<i>a positive number in minutes</i>	10	dynamic	
PollEndTime	HH:MM	HH:MM	component restart	
PollFrequency	<i>a positive integer in milliseconds</i>	10000	dynamic	
PollQuantity	no (to disable polling) key (to poll only when the letter p is entered in the connector's Command Prompt window) 1-500	1	component restart	Number of items to poll from application
PollStartTime	HH:MM(HH is 0-23, MM is 0-59)	HH:MM	component restart	
RepositoryDirectory	location where repository is located	<REMOTE>	component restart	<REMOTE> for ICS broker
RequestQueue	<i>valid JMS queue name</i>	CONNECTORNAME/REQUESTQUEUE	component restart	
ResponseQueue	<i>valid JMS queue name</i>	CONNECTORNAME/RESPONSEQUEUE	component restart	
RestartRetryCount	0-99	3	dynamic	
RestartRetryInterval	<i>a sensible positive value in minutes</i>	1	dynamic	

Property Name	Possible values	Default value	Update method	Notes
SourceQueue	<i>valid MQSeries queue name</i>	CONNECTORNAME/SOURCEQUEUE	component restart	Valid only if delivery transport is JMS and Container Managed Events is specified.
SynchronousRequestQueue		CONNECTORNAME/ SYNCHRONOUSREQUESTQUEUE	component restart	
"SynchronousResponseQueue" on page 148		CONNECTORNAME/ SYNCHRONOUSRESPONSEQUEUE	component restart	
SynchronousRequestTimeout		0	component restart	
"WireFormat" on page 149	CwXML, CwBO	cwxml	agent restart	CwXML for non-ICS broker; CwBO if Repository Directory is <REMOTE>

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

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

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

AgentTraceLevel

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

ApplicationName

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

BrokerType

Identifies the integration broker type that you are using. If you are using an ICS connector, this setting must be 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 ASCII for this property. If you previously configured the value of this property to `ascii7` or `ascii8`, you must reconfigure the connector to use either ASCII or one of the other supported values. To determine whether a specific connector is written in Java or C++, see the installing and configuring chapter of its adapter guide.

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

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

The default value is `ascii`.

ConcurrentEventTriggeredFlows

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

Setting this property to a value greater than 1 allows a connector controller for a source application to simultaneously map multiple event business objects, and to simultaneously deliver them to multiple collaboration instances. Setting this property to enable concurrent mapping of multiple business objects can speed delivery of business objects to a collaboration, particularly if the business objects use complex maps. Increasing the arrival rate of business objects to collaborations can improve overall performance in the system.

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

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

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

ContainerManagedEvents

Setting this property to JMS 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 can also be set to no value.

Notes:

1. When `ContainerManagedEvents` is set to JMS, you must also configure the following properties to enable guaranteed event delivery: `PollQuantity` = 1 to 500, `SourceQueue` = `SOURCEQUEUE`. In addition, you must configure a data handler with the `MimeType`, `DHClass`, and `DataHandlerConfigMOName` (optional) properties. To set those values, use the Data Handler tab of Connector Configurator. The fields for the values under the Data Handler tab will be displayed only if you have set `ContainerManagedEvents` to JMS.
2. When `ContainerManagedEvents` is set to JMS, the connector does *not* call its `pollForEvents()` method, thereby disabling that method's functionality.

The default value is JMS.

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

ControllerStoreAndForwardMode

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

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

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

The default is true.

ControllerTraceLevel

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

DeliveryQueue

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

The default value is DELIVERYQUEUE.

DeliveryTransport

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

If ICS is the broker type, the value of the DeliveryTransport property can be MQ, IDL, or JMS, and the default is IDL.

If WMQI is the broker type, JMS is the only possible Delivery Transport value.

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

WebSphere MQ and IDL

Use WebSphere MQ rather than IDL for event delivery transport, unless you have compelling reasons not to license and maintain two separate products. WebSphere MQ offers the following advantages over IDL:

- Asynchronous communication – WebSphere MQ allows the application-specific component to poll and persistently store events even when the server is not available.
- Server side performance – WebSphere MQ provides faster performance on the server side. In optimized mode, WebSphere MQ stores only the pointer to an event in the repository database, while the actual event remains in the WebSphere MQ queue. This saves the overhead of having to write potentially large events to the repository database.
- Agent side performance – WebSphere MQ provides faster performance on the application-specific component side. Using WebSphere MQ, the connector's polling thread picks up an event, places it in the connector's queue, then picks up the next event. This is faster than IDL, which requires the connector's polling thread to pick up an event, go over the network into the server process, store the event persistently in the repository database, then pick up the next event.

JMS

Enables communication between the connector controller 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," display in Connector Configurator. The first two of these properties are required for this transport.

Important: There may be a memory limitation if you use the JMS transport mechanism for a connector in the following environment:

- AIX 5.0
- WebSphere MQ 5.3.0.1
- InterChange Server (ICS) as the Integration broker

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

- The LDR_CNTRL environment variable in the CWSHaredEnv.sh script.

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

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

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

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

Notes:

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

DuplicateEventElimination

Setting this property to true enables a JMS-enabled connector to ensure that duplicate events are not delivered to the delivery queue. To make use of this feature, during connector development a unique event identifier must be set as the business object's ObjectEventId attribute in the application specific code.

This property can also be set to false.

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

FaultQueue

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

The default value is CONNECTORNAME/FAULTQUEUE.

JvmMaxHeapSize

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

The default value is 128m.

JvmMaxNativeStackSize

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

The default value is 128k.

JvmMinHeapSize

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

The default value is 1m.

jms.FactoryClassName

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

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

jms.MessageBrokerName

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

The default is `crossworlds.queue.manager`.

jms.NumConcurrentRequests

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

The default value is 10.

jms.Password

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

There is no default.

jms.UserName

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

There is no default.

Locale

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

A locale name has the following format:

```
ll_TT.codeset
```

where:

ll a two-character language code (usually in lower case)

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

The default is en_US.

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

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

LogAtInterchangeEnd

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

MaxEventCapacity

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

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

MessageFileName

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

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

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

OADAutoRestartAgent

Specifies whether the Object Activation Daemon (OAD) automatically attempts to restart the application-specific component after an abnormal shutdown. The properties "OADMaxNumRetry" on page 140 and "OADRetryTimeInterval" on page 140 are related to this property. This property is required for automatic restart.

The default value is false.

OADMaxNumRetry

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

The default value is 1000.

OADRetryTimeInterval

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

The default is 10.

PollEndTime

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

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

PollFrequency

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

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

The default is 10000.

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

PollStartTime

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

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

RequestQueue

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

The default value is REQUESTQUEUE.

RepositoryDirectory

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

When the integration broker is ICS, this value must be set to <REMOTE> because the connector uses the InterChange Server repository to obtain its connector-definition information

ResponseQueue

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

RestartRetryCount

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

The default is 3.

RestartRetryInterval

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

The default is 1.

SourceQueue

Designates the JMS source queue for the connector framework in support of guaranteed event delivery for JMS-enabled connectors that use a JMS event store. For further information, see “ContainerManagedEvents” on page 135.

The default value is SOURCEQUEUE.

SynchronousRequestQueue

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

SynchronousResponseQueue

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

SynchronousRequestTimeout

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

The default value is 0.

TraceFileName

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

WireFormat

Message format on the transport.

Possible values are:

- CwXMLif the broker is not ICS.
- CwB0if the value of RepositoryDirectory is <REMOTE>.

Configuring standard connector properties for WebSphere MQ Integrator

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

Important: Not all properties are applicable to all connectors that use WebSphere MQ Integrator Broker. For information specific to a connector, see its adapter user guide.

You configure connector properties from Connector Configurator.

Note: Connector Configurator runs only on the Windows system. Even if you are running the connector on a UNIX system, you must still have a Windows machine with this tool installed. Therefore, to set connector properties for a connector that runs on UNIX, you must run Connector Configurator on the Windows computer and copy the configuration files to the UNIX computer using FTP or some other file transfer mechanism. For more information about Connector Configurator, see Appendix B, "Connector Configurator."

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

Standard connector properties

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

Name	Possible values	Default value
AdminInQueue	<i>valid JMS queue name</i>	CONNECTORNAME/ADMININQUEUE
AdminOutQueue	<i>valid WebSphere MQ queue name</i>	CONNECTORNAME/ADMINOUTQUEUE
AgentTraceLevel	0-5	0
ApplicationName	<i>application name</i>	AppNameConnector
BrokerType	WMQI	WMQI
CharacterEncoding	ASCII, SJIS, Cp949, GBK, Big5, Cp297, Cp273, Cp280, Cp284, Cp037, Cp437 Note: These are only a subset of supported values.	ASCII
ContainerManagedEvents	JMS or no value	JMS
DeliveryQueue	<i>valid WebSphere MQ queue name</i>	CONNECTORNAME/DELIVERYQUEUE
DeliveryTransport	JMS	JMS
DuplicateEventElimination	true, false	
FaultQueue	<i>valid WebSphere MQ queue name</i>	CONNECTORNAME/FAULTQUEUE
jms.FactoryClassName		CxCommon.Messaging.jms.IBMMQSeriesFactory
jms.MessageBrokerName	If FactoryClassName is IBM, use crossworlds.queue.manager. If FactoryClassName is Sonic, use localhost:2506.	crossworlds.queue.manager
jms.NumConcurrentRequests		10
jms.Password		
jms.UserName		
Locale	en_US , ja_JP, ko_KR, zh_C, zh_T, fr_F, de_D, it_I, es_E, pt_BR Note: These are only a subset of supported locales.	en_US
MessageFileName	<i>path/filename</i>	<i>InterchangeSystem.txt</i>
PollEndTime	HH:MM	HH:MM
PollFrequency	<i>milliseconds/key/no</i>	10000
PollStartTime	HH:MM	HH:MM
RepositoryDirectory	<i>path/directory name</i> Note: Typically you must change this value from the default to whatever path and directory name was actually used when you installed the connector files.	C:\crossworlds\Repository
RequestQueue	<i>valid WebSphere MQ queue name</i>	CONNECTORNAME/REQUESTQUEUE
ResponseQueue		RESPONSEQUEUE
RestartRetryCount	0-99	3
RestartRetryInterval	<i>an appropriate integer indicating the number of minutes between restart attempts</i>	1

Name	Possible values	Default value
SourceQueue	<i>valid WebSphere MQ queue name</i>	CONNECTORNAME/SOURCEQUEUE
SynchronousRequestQueue	<i>valid WebSphere MQ queue name</i>	
SynchronousResponseQueue	<i>valid WebSphere MQ queue name</i>	
SynchronousTimeout	<i>an appropriate integer indicating the number of minutes the connector waits for a response to a synchronous request</i>	0
WireFormat	CwXML	CwXML

AdminInQueue

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

The default value is CONNECTORNAME/ADMININQUEUE.

AdminOutQueue

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

AgentTraceLevel

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

ApplicationName

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

BrokerType

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

CharacterEncoding

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

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

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

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

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

The default value is `asci i`.

ContainerManagedEvents

Setting this property to `JMS` enables 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 can also be set to no value.

Notes:

1. When `ContainerManagedEvents` is set to `JMS`, you must also configure the following properties to enable guaranteed event delivery: `PollQuantity = 1` to `500`, `SourceQueue = SOURCEQUEUE`. In addition, you must configure a data handler with the `MimeType`, `DHClass`, and `DataHandlerConfigMOName` (optional) properties.
2. When `ContainerManagedEvents` is set to `JMS`, the connector does *not* call its `pollForEvents()` method, thereby disabling that method's functionality.

The default value is `JMS`.

DeliveryQueue

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

The default value is `CONNECTORNAME/DELIVERYQUEUE`.

DeliveryTransport

Specifies the transport mechanism for the delivery of events. The property defaults to the value `JMS`, indicating that the Java Messaging Service (JMS) is used for communication with WebSphere MQ Integrator. This property *must* be set to `JMS` when WebSphere MQ Integrator Broker is the integration broker. Otherwise, the connector cannot start.

DuplicateEventElimination

Setting this property to `true` enables a JMS-enabled connector to ensure that duplicate events are not delivered to the delivery queue. To make use of this feature, during connector development a unique event identifier must be set as the business object's `ObjectEventId` attribute in the application specific code.

This property can also be set to `false`.

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

FaultQueue

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

The default value is `CONNECTORNAME/FAULTQUEUE`.

jms.FactoryClassName

Specifies the class name to instantiate for a JMS provider.

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

jms.MessageBrokerName

Specifies the broker name to use for the JMS provider.

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

A locale name has the following format:

ll_TT.codeset

where:

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

The default is `en_US`.

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

Attention:

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

MessageFileName

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

PollEndTime

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

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

PollFrequency

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

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

The default is `10000`.

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

PollStartTime

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

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

RepositoryDirectory

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

The default value is `C:\crossworlds\repository`. You must change this to the directory path that you are using for the `\repository` directory for your connector. Typically that path is established when you install the adapter product; for

example, C:\WebSphereAdapters\repository. The value must be a directory path. Do not use <REMOTE> as the RepositoryDirectory value for a connector that is not using ICS as the broker.

RequestQueue

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

The default value is CONNECTORNAME/REQUESTQUEUE.

ResponseQueue

Designates the JMS response queue, which delivers a response message from the connector framework to the integration broker.

RestartRetryCount

Specifies the number of times the connector attempts to restart itself. The default value is 3, indicating that the connector tries to restart 3 times. For instance, if a connector is unable to log in to an application it fails to start, but with this property set to the value 3 the connector tries a total of three times to start. When used in conjunction with the “RestartRetryInterval” property, this behavior enables a connector to make several attempts at communicating with an application that might not reliably have a connection available all the time.

RestartRetryInterval

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

SourceQueue

Designates the 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 135.

The default is CONNECTORNAME/SOURCEQUEUE.

SynchronousRequestQueue

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

SynchronousResponseQueue

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

SynchronousTimeout

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

The default value is 0.

WireFormat

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

Appendix B. Connector Configurator

Before you can use a connector, you must create a connector configuration file that sets the properties for the connector, designates the business objects and any meta-objects that it supports, and sets logging and tracing values that the connector will use at runtime. The configuration file may also contain properties for the use of messaging and data handlers required by your connector.

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

When you complete a connector configuration file, the file is saved as an XML document. You will save the XML document either as a project in System Manager (if ICS is your broker) or as a file with a *.cfg extension in a directory folder (if WebSphere MQ Integrator Broker is your broker, or if you are using the file as a local configuration file for ICS).

This appendix describes how to use Connector Configurator to:

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

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

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

Using Connector Configurator in an internationalized environment

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

Connector Configurator supports non-English characters in:

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

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

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

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

Starting Connector Configurator

Connector Configurator can be started and run in either of two modes:

- Launched from System Manager
- Independent of System Manager (stand-alone mode)

Running Configurator from System Manager

When you run Connector Configurator in conjunction with System Manager, you can

- Save connector configuration files (XML documents with the extension *.cfg) to a directory that you specify, and
- Save connector configuration files as components of System Manager projects. If you are using ICS as your broker, this is a mandatory step before you deploy your configuration into the ICS.

Note: When you save a configuration file as a component of a System Manager project, the file is stored in the designated project as an XML document file with the extension *.con. It is not advisable to open the *.con file and edit it directly; instead, make any changes by opening the component in System Manager.

To run Connector Configurator with System Manager, do any of the following:

- In System Manager, right-click on the Connector folder of the Integration Components Library (to create a new configuration), or right-click on a connector configuration component within the Connector folder (to edit an existing configuration), or
- From the System Manager menu, choose Tools>Connector Configurator, or
- With System Manager already running, from Start>Programs choose IBM WebSphere InterChange Server>IBM WebSphere Business Integration Toolset>Development>Connector Configurator.

For details about using projects in System Manager and deploying to InterChange Server, see the *Implementation Guide for WebSphere InterChange Server*.

Running Configurator independently of System Manager

When you run Connector Configurator without connecting to System Manager, you can save a connector configuration file (an XML document with the extension *.cfg) to a directory that you specify, but you cannot save or open a System Manager project.

When you are creating a connector for use with a broker other than ICS, you do not need to connect to System Manager at any point in order to use the file. If you are creating a connector configuration for use with ICS as the broker, you may still find it useful on occasion to run Connector Configurator independently, and then connect to System Manager when you are ready to save the configuration file as a component of a System Manager project.

Choosing your broker

Connector Configurator can be used to configure connectors either for use with ICS as the broker, or with WebSphere MQ Integrator Broker (also referred to as WMQI) as the broker.

Before you begin to configure the connector, you must choose the mode of Connector Configurator that is appropriate for your broker. The mode that you choose determines the properties that Connector Configurator will include in the configuration file. Choosing a broker is a mandatory step when you begin the process of creating a completely new configuration file. After a configuration file has been created, you can optionally change the designated broker mode, using a standard configuration property. (This makes it possible to use an existing configuration file as a starting point for creating a configuration file that will be used with a different broker. However, be aware that revising a configuration file for use with a different broker typically involves changing other configuration properties as well, and not just the broker mode property.)

To choose a broker when you create a new configuration file (mandatory):

- In the Connector Configurator home menu, choose File>New>Connector Configuration. The New Connector Dialog displays.
- In the Integration Broker field, choose either WMQI connectivity (for WebSphere Integrator Broker) or ICS connectivity, according to the broker you are using.
- Complete the remaining fields of the New Connector dialog, as described later in this chapter for your specific broker.

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

- Open the existing configuration file in Connector Configurator.
- Select the Standard Properties tab.
- In the Broker Type field of the Standard Properties tab, choose the value that is appropriate for your broker. If you change the existing value, the available tabs and field selections of the properties screen will immediately refresh, to show only those tabs and fields that appropriate for a configuration using the broker you have selected.

After you have chosen your broker type, you can complete the remaining Connector Configurator tasks for configuring your connector. When you save the connector configuration file, Connector Configurator will save it in the broker mode that you have already selected. The title bar of Connector Configurator always displays the broker mode (such as ICS or WMQI) that Connector Configurator is currently using.

After you have completed the configuration file and set its properties, it will need to be deployed to the appropriate location for your connector.

- If you are using ICS as your broker, save the configuration in a System Manager project, and use System Manager to load the file into InterChange Server.
- If you are using WebSphere MQ Integrator Broker as your broker, manually copy the configuration file to its appropriate location, which must match exactly the configuration file location specified in the startup file for your connector.

For further information about deployment, see the *Implementation Guide for WebSphere InterChange Server* (for using the connector with ICS as the broker), or the *Implementation Guide for WebSphere MQ Integrator Broker* (for using the connector with MQ Integrator as the broker).

Using a connector-specific property template

To create a configuration file for your connector, you can start with a previously created connector configuration file (*.cfg), a connector definition file (*.txt) or a repository file (*.in or *.out), if any of these already exists for your connector. For instructions on using such existing files, see “Using an existing file” on page 158.

If none of those files exist, or if they are too dissimilar to the configuration requirements of your connector, you can start instead by creating a template for the connector-specific properties of your connector. You’ll create properties in the template, define general characteristics and values for those properties, and specify any dependencies between the properties. Then you’ll save the template and use it as the base for creating a new connector configuration file.

Creating a template of connector-specific properties

To create a template:

1. Choose File>New>Connector-Specific Property Template.
2. The Connector-Specific Property Template dialog appears, with the following fields

- 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 for creating a new configuration file from a template.

- Find Template, and Template Name

The names of all currently available templates are displayed in the Template Name display. Look for an existing template that would make a good starting point for your new connector template (such as a template whose property definitions are a subset of the properties used by your connector).

To see the connector-specific property definitions that are contained 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.

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

Choose a template from the Template Name display, enter that template name in the Find Name field (or highlight your choice in Template Name), and choose Next.

Specifying general characteristics

The Properties - Connector-Specific Property Template dialog appears. The dialog has tabs for General characteristics of the defined properties and for Value restrictions. The General display has the following fields:

- Edit properties

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

A child property is a property that is an attribute of another property--the "parent" property. The parent property can obtain values, or child properties, or both. These property relationships are commonly referred to as "hierarchical" properties. Later, when you create a configuration file from these properties, Connector Configurator will identify hierarchical property sets with a plus sign in a box at the left of any parent property.

- Property type

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

- Flags

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

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

Specifying values

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

1. Choose 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 necessary changes. Note that the changes will not be accepted until and unless you also open the Property Value dialog for the property, described in the next step.
4. Right-click the box in the left-hand corner of the Value display panel. A Property Value dialog displays. Depending on the type of the property, the dialog 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, and it displays a table with three columns:

The Value column shows the value that you entered in the Property Value dialog, 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.

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 choose EditValue.

Setting dependencies

After you have finished making changes in both the General and the Value tabs, choose Next. The Dependencies dialog 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. To designate a property as being dependent and 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, choose one of the following:
 - == (equal to)
 - /= (not equal to)
 - > (greater than)
 - < (less than)
 - >= (greater than or equal to)
 - <=(less than or equal to)
4. In the Conditional Value field, enter the value that is required in order for the dependent property to be included in the template.
5. With the dependent property highlighted in the Available Properties display, click an arrow to move it to the Dependent Property display.
6. Click Finish. Connector Configurator stores the information you have entered as an XML document, under \data\app in the \bin directory where you have installed Connector Configurator.

Creating a configuration file from a connector-specific template

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

1. Choose File > New>Connector Configuration.
2. The New Connector dialog appears, with the following fields:
 - Name
Enter the name of the connector. Names are case-sensitive. The name you enter must be unique, must end with the word “connector”, and must be consistent with the file name for a connector that is installed on the system; for example, enter PeopleSoftConnector if the connector file name is PeopleSoft.jar.

Important: Connector Configurator does not check the spelling of the name that you enter. You must ensure that the name is correct.
 - System Connectivity
Choose ICS or choose WMQI (for WebSphere MQ Integrator Broker) connectivity.
 - Select Connector-Specific Property Template
Type the name of the template that has been designed for your connector. The names of all available templates are displayed 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.

After you have chosen the template you want to use, choose OK.

3. A configuration screen will display for the connector that you are configuring. The title bar of the configuration screen shows the broker that you are using and the name that you have given to the connector. You can fill in all the field values to complete the definition now, or you can save the file and complete the fields later.

When you are using the configuration screen, you can, if you wish, add additional connector-specific properties, as described under “Setting application-configuration properties (ICS)” on page 160. Any such additions become part of the configuration file that you are creating, but do not affect the template that you used in creating the file.

4. To save the file, choose File > Save > to File or File > Save > Save to the project. To save to a project, you must be using ICS as the broker, and System Manager must be running. If you save as a file, the Save File Connector dialog displays. Choose *.cfg as the file type, verify in the File Name field that the name is spelled correctly and has the correct case, navigate to the directory where you want to locate the file, and choose Save. The status display in the message panel of Connector Configurator indicates that the configuration file was successfully created.

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

5. To complete the connector definition, enter values in the fields for each of the tabs of the Connector Configurator window, as described for your broker later in this chapter.

Using Connector Configurator with ICS as the broker

To use Connector Configurator to configure a connector that will be used with ICS, first select ICS as the broker mode in which you are running Connector Configurator, as described under “Choosing your broker” on page 153.

In a typical ICS implementation, the configuration file that you create with Connector Configurator is not put into use until after you have deployed it to the ICS server. You will perform that deployment (described in the *Implementation Guide for WebSphere InterChange Server*) after you have finished using Connector Configurator to complete the connector configuration file.

Completing a configuration file

This topic assumes that you already have a starting point for your connector configuration, either from an existing file (a connector definitions file, a repository file, or a *.cfg file) or from an existing project in System Manager. If you do not, see “Creating a template of connector-specific properties” on page 154.

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

The title of the configuration screen displays the type of the broker and the name of the connector as specified in the file. Make sure the title indicates the

appropriate type for your broker—either ICS or WebSphere MQ Integrator Broker (for WMQI). If it does not, change the broker value before you configure the connector. To do so:

1. Under the Standard Properties tab, select the value field for the BrokerType property. In the drop-down menu, select the value WMQI or ICS.
2. The Standard Properties tab refreshes to display properties associated with the selected broker. When you save the file, you retain this broker selection. You can save the file now or proceed to complete the remaining configuration fields, as described in “Setting the configuration file properties (WebSphere MQ Integrator Broker)” on page 164.
3. When you have finished making entries in the configuration fields, choose *File>Save>To Project* or *File>Save>To File*.

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

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

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

Using an existing file

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

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

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

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

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

1. In Connector Configurator, choose *File > Open > From File*.
2. In the Open File Connector dialog, choose one of the following file types to see the available files:
 - Configuration (*.cfg)
 - ICS Repository (*.in, *.out)

Choose this option if a repository file was used to configure the connector in an ICS environment. A repository file may include multiple connector definitions, all of which will display when you open the file.

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

Using an existing System Manager project

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

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

Setting the configuration file properties (ICS)

The topics in this section apply if you are using InterChange Server as the integration broker. If you are using WebSphere MQ Integrator Broker as the integration broker, see “Setting the configuration file properties (WebSphere MQ Integrator Broker)” on page 164. When you create and name a new connector configuration file, or when you open an existing connector configuration file, Connector Configurator displays a configuration screen with tabs for the categories of required configuration values.

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

1. Standard Properties
2. Connector-Specific Properties
3. Supported Business Objects
4. Associated Maps
5. Resources
6. Trace/Log File values
7. Messaging (where applicable)
8. Data handlers (applicable for connectors that use JMS messaging with guaranteed event delivery)

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

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

Standard properties differ from connector-specific properties as follows:

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

default values and some do not; you can modify some of the default values. The installation and configuration chapter of each adapter guide describes the application-specific properties and the recommended values.

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

Setting standard connector properties (ICS)

To change the value of a standard property:

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

Setting application-configuration properties (ICS)

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

1. Right click in the top-left portion of the grid. A pop-up menu bar will appear. Select Add to add a property or Add Child to add a child property for a property.
2. Enter a value for the property or child property.
3. To encrypt a property, click the Encrypt box.
4. Choose to save or discard changes, as described for Setting Standard Connector Properties.

The Update Method displayed for each property indicates whether a component or 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 (ICS)

Application-specific properties can be encrypted by clicking 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 choose OK. If the entered value is correct, the value is decrypted and displays. The adapter guide for each connector contains a list and description of each property and its default value.

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

Update method (ICS)

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

Specifying supported business object definitions (ICS)

This topic assumes that you have already created or acquired the intended business objects, created or acquired maps for them, and have saved both the business object definitions and map definitions into System Manager projects.

Before you can make use of a connector (and before you can bind the connector with a collaboration's ports), you must make selections under the Supported Business Objects tab to specify the business objects that the connector will use. You must specify both generic business objects and corresponding application-specific business objects, and you must specify associations for the maps between the business objects.

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

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

Business object name

These instructions assume that you started Business Object Designer with System Manager running.

To designate that a business object definition is supported by the connector:

1. Click in an empty field of the Business Object Name list. A drop-down list displays, showing all the business object definitions that exist in the System Manager project.
2. Click on a business object to add it.
3. Set the Agent Support (described below) for the business object.

4. In the File menu of the Connector Configurator window, choose 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 window, choose Delete Row. The business object is removed from the list display.
3. From the File menu, choose Save to Project.

Note that deleting a business object from the supported list does not affect the code of the connector, nor does it remove the business object definition itself from System Manager. It does, however, change the connector definition and make the deleted business object unavailable for use in this implementation of this connector.

Agent support

Indicating Agent Support for a business object means that 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, put a check in the Agent Support box. Note that the Connector Configurator window does not validate your Agent Support selections.

Maximum transaction level

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

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

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

Note: For this release, maximum transaction level of a connector is always Best Effort.

Associated maps (ICS)

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

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

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

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

The Associated Maps tab displays the following fields:

- Business Object Name

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

- Associated Maps

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

- Explicit

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

Explicit binding is required only when more than one map exists for a particular supported business object. When InterChange Server 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 not a map that is the superset of the others, the server will not be able to bind the business object to a single map, and you will need to set the binding explicitly.

To explicitly bind a map:

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

Resources (ICS)

The Resource tab allows you to set a value that determines whether and to what extent the connector agent will handle multiple processes concurrently using connector agent parallelism. Not all connectors support this feature, and use of this feature is not usually advised for connector agents that were designed in Java to be multi-threaded, since it is usually more efficient to use multiple threads than multiple processes.

Setting trace/log file values (ICS)

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

To change the logging and tracing values:

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

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

Configuring messaging

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

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

Setting the configuration file properties (WebSphere MQ Integrator Broker)

The topics in this section apply if you are using WebSphere MQ Integrator (also referred to as WMQI) as the integration broker.

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

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

1. Standard Properties
2. Connector-Specific Properties
3. Supported Business Objects
4. Trace/Log File values
5. Data Handlers (where applicable)

Note: For connectors that use JMS messaging, an additional category may display, for configuration of data handlers that convert the data to business objects. For information about the values to use in the Data Handlers category, see the *Connector Development Guide for C++* or the *Connector Development Guide for Java*.

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

Standard properties differ from connector-specific properties as follows:

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

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

Setting standard connector properties

To change the value of a standard property:

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

Setting application-configuration properties

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

1. Click in the field whose name or value you want to set.

2. Enter a name or value.
3. To encrypt a property, click the Encrypt box.
4. Choose to save or discard changes, as described for Setting Standard Connector Properties.

The Update Method displayed for each property indicates whether a component or 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 clicking 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 choose OK. If the entered value is correct, the value is decrypted and displays. The adapter guide for each connector contains a list and description of each property and its default value.

Update method

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

Specifying supported business object definitions

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

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

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

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

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

To specify supported business objects:

1. Select the Supported Business Objects tab and choose Load. The Open Message Set ID File(s) dialog displays.
2. Navigate to the directory where you have placed the message set file for the connector and select the appropriate message set file (*.set) or files.
3. Choose Open. The Business Object Name field displays the business object names contained in the *.set file; the numeric message set ID for each business object is listed in its corresponding Message Set ID field. Do not change the message set IDs. These names and numeric IDs are saved when you save the configuration file.

4. When you add business objects to the configuration, you must load their message set files. If you attempt to load a message set that contains a business object name that already exists in the configuration, or if you attempt to load a message set file that contains a duplicate business object name, Connector Configurator detects the duplicate and displays the Load Results dialog. The dialog shows the business object name or names for which there are duplicates. For each duplicate name shown, click in the Message Set ID field, and choose the Message Set ID that you wish to use.

Setting trace/log file values

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

To change the logging and tracing values:

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

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

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

Using standard and connector-specific properties with Connector Configurator

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 within your configuration file; Connector Configurator already has those definitions, and it incorporates them into your configuration file as soon as you create the file. For standard properties, your only task is to use Connector Configurator to set the values of the properties.

For connector-specific properties, however, you will need to both define the properties and set their values. Connector Configurator provides the interface for performing both of these tasks.

Completing the configuration

After you have created a configuration file for a connector and modified it, make sure that the connector can locate the configuration file when the connector starts up. To do so, open the startup file used for the connector, and verify that the location and file name used for the connector configuration file match exactly the name you have given the file and the directory or path where you have placed it.

Appendix C. Connector feature list

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

Business object request handling features

Table 54 details the business object request handling features supported by the connector.

Table 54. Business object request handling features

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

Event notification features

Table 55 details the event notification features supported by the connector.

Table 55. Event notification features

Category	Feature	Support	Notes
Connector Properties	Event distribution	No	
	PollQuantity	Full	

Table 55. Event notification features (continued)

Category	Feature	Support	Notes
Event Table	Event status values	N/A	Connector retrieves messages based on the priority specified in their message header (range 0-9). See WebSphere MQ documentation for more information.
	Object key	N/A	
	Object name	N/A	
	Priority	Full	
Misc.	Archiving	Full	Connector can deliver copies of messages to different queues depending on whether the message was unsubscribed, was successfully processed, or resulted in errors.
	CDK method gotApplEvent	Full	
	Delta event notification	No	
	Event sequence	Full	Support is entirely dependent on application receiving connector request.
	Future event processing	No	
	In-Progress event recovery	Full	
	Physical delete event	N/A	
	RetrieveAll	N/A	Support is entirely dependent on application receiving connector request.
	Smart filtering	No	
	Verb stability	N/A	

General features

Table 56 details the general features supported by the connector.

Table 56. General features

Category	Feature	Support	Notes
Business Object Attributes	Foreign key	No	Support is entirely dependent on application receiving connector request.
	Foreign Key attribute property	N/A	
	Key	No	
	Max Length	Partial	
Connection Lost	Meta-data-driven design	Full	May be used by the data handler chosen for the connector.
	Required	No	
	Connection lost on poll	Full	
	Connection lost on request processing	Full	
Connector Properties	Connection lost while idle	No	Depends on the data handler established for the connector.
	ApplicationPassword	Full	
	ApplicationUserName	Full	
	UseDefaults	Partial	
Message Tracing	General messaging	Full	
	generateMsg()	Full	
	Trace level 0	Full	
	Trace level 1	Full	
	Trace level 2	Full	
	Trace level 3	Full	

Table 56. General features (continued)

Category	Feature	Support	Notes
Misc.	Trace level 4	Full	
	Trace level 5	Full	
	CDK method LogMsg	Full	
	Java Package Names	Full	
	Logging messages	Full	
	NT service compliance	Full	
Special Value	Transaction support	Full	
	CxBlank processing	Partial	Depends on the data handler chosen for the connector
	CxIgnore processing	N/A	Support is entirely dependent on application receiving connector request.

Appendix D. SWIFT message structure

- “SWIFT message types”
- “SWIFT field structure”
- “SWIFT message block structure” on page 175

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	BANKDEFFXXX	U	3	003}
(a)	(b)	(c)	(d)	(e)	(f)	(g)

- a) 2: = Block ID (always 2)
- b) I = Input
- c) 100 = Message type
- d) BANKDEFFXXX = 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:	0	100	1200	970103	BANKBEBBAXXX2222123456	970103	1201	N}
(a)	(b)	(c)	(d)	(e)		(f)	(g)	(h)

- a) 2: = Block ID (always 2)
- b) 0 = 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 173.

The content of the text block is a collection of fields. For more on SWIFT fields, see “SWIFT field structure” on page 173. 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* = Maximum 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 E. 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 RTP Laboratory
3039 Cornwallis Road
P.O. BOX 12195

Raleigh, NC 27709-2195
U.S.A

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

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

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

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

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

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

Programming interface information

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

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

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

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

Trademarks and service marks

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

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

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

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

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

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

IBM WebSphere InterChange Server V4.2, IBM WebSphere Business Integration Toolset V4.2, IBM WebSphere Business Integration Adapters V2.2, IBM WebSphere Business Integration Collaborations V4.2.





Printed in U.S.A.